# MULTIAGENT SYSTEMS:
# GAMES AND LEARNING FROM STRUCTURES

JING YAN

A DISSERTATION SUBMITTED TO
THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILMENT OF THE REQUIREMENTS
FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

GRADUATE PROGRAM IN
COMPUTER SCIENCE AND ENGINEERING
YORK UNIVERSITY
TORONTO, ONTARIO

OCTOBER 2013

# Abstract

Multiple agents have become increasingly utilized in various fields for both physical robots and software agents, such as search and rescue robots, automated driving, auctions and electronic commerce agents, and so on. In multiagent domains, agents interact and coadapt with other agents. Each agent's choice of policy depends on the others' joint policy to achieve the best available performance. During this process, the environment evolves and is no longer stationary, where each agent adapts to proceed towards its target. Each micro-level step in time may present a different learning problem which needs to be addressed. However, in this non-stationary environment, a holistic phenomenon forms along with the rational strategies of all players; we define this phenomenon as structural properties.

In our research, we present the importance of analyzing the structural properties, and how to extract the structural properties in multiagent environments. According to the agents' objectives, a multiagent environment can be classified as self-interested, cooperative, or competitive. We examine the structure from these three general multiagent environments: self-interested random graphical game playing, distributed cooperative team playing, and competitive group survival. In each scenario, we analyze the structure in each environmental setting, and demonstrate the structure learned as a comprehensive representation: structure of players' action influence, structure of constraints in teamwork communication, and structure of inter-connections among strategies. This structure represents macro-level knowledge arising in a multiagent system, and provides critical, holistic information for each problem domain. Last, we present some open issues and point toward future research.

# Acknowledgements

I wish to express my sincere gratitude to my supervisor, Nick Cercone, for his support and guidance during the completion of this dissertation. He graciously allowed me the freedom to explore my interests, with understanding and patient encouragement, as well as intellectual support.

To my dissertation supervisor committee members, Dr. Yves Lesperance and Dr. Parke Godfrey, I express my thanks for their insights, comments, suggestions and for their commitment to helping me be a thorough researcher and better writer, making my dissertation clear and strong. To my External examiner Dr. Corey Butz, I appreciate that he made this special trip here for my defence and providing me with his advice and showing me how to improve myself to be a better and more precise researcher and writer. To my Internal examiner Dr. Henry Kim, and lastly to the Dean's representative Dr. Suprakash Datta, I express my thanks for their insights and comments, and for their suggestions.

To my parents, for their continued and unwavering support in helping and encouraging me to realize my goals. For their love, understanding, support, and patience without which this work would not have been possible.

To my best friends, Mark, Jim, Erich, for their invaluable support, encouragement, tolerance and understanding; all of you always gave me your time and attention in my hardest times.

To colleagues and collaborators, Matt and Albert, for providing me feedback, discussion and technical support to make my work interesting and bringing balance and richness to my doctoral experience. To Dr. Anestis Topsis, I express my appreciation for his encouragement during my writing process. And to many others, all of whom gave me support and made my Ph.D dream possible. Thank you!

It is not the mountain we conquer - but ourselves.

SIR EDMUND HILLARY

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

What are the differences between single agent learning versus multiagent learning? Single agent learning is a process where one single agent improves its performance through its own experience; whereas, multi-agent learning describes a process where multiple agents perform in the same environment, and each agent learns along with other agents to achieve each individual's agenda. Mature research exists in single agent learning, such as supervised learning (regression modelling and classification), unsupervised learning (clustering), and reinforcement learning (learning from rewards given). In contrast, multiagent learning, a relatively new field, mostly extends from a single agent reinforcement learning perspective, and the goal is to learn how to perform and achieve the highest rewards under coordination/constraints of other agents who perform jointly and simultaneously. Another branch of multiagent learning research, influenced by game theorists, focuses on strategy selection, which studies how multiple players play in a game and choose best strategies to achieve their goals. Strategy selection is a critical process to analyze the best strategy, for example, the minmax-Q (Littman, 1994) strategy describes a process which is to choose the maximum payoff in the worst/minimum situation. The purpose of analyzing this difference is to examine what the important breakthrough is in both communities, and what may be missing in addressing/solving some bottleneck problems.

## 1.1 Rationale

Multiple agents become increasingly important in various applications for both physical robots and software agents, such as, robot soccer, search and rescue robots, automated driving, auctions and electronic commerce agents, and so on. The merits of game theory influence computer science researchers in non-human-player game playing. An agent, a non-human player, observes the environment and chooses an action to perform. Commonly, agents have goals, assumptions, algorithms for learning and reasoning, and conventions. Learning through single agent tasks has been studied extensively in the reinforcement learning field, in which an agent acts alone in a stationary environment. In multiagent domains, agents interact with others, and co-adapt with others, then act on the best choice available. Since all the agents are evolving, the environment is no longer stationary, and this dynamic brings in a difficult learning problem that violates the basic stationary assumption of traditional techniques for behaviour learning. Each agent's choice of policy depends on the others' joint policy, which also aims to achieve the best available performance. Our work focuses on understanding the dynamics in a multigent system, in order to improve the strategic decision-making and learning process of agent behaviours, whose target is to select the best strategies, and adapt to unforeseen difficulties and changes in the environment.

## 1.2 Objective

An agent can learn through experience from its own actions and associated effects, while learning from observation of other agents' experience. Note that an agent should effectively associate similar patterns and build knowledge, instead of merely keeping a record of the reward history for all agents. By using this knowledge, the agent can efficiently explore the strategy space. Exploration vs. exploitation is a critical choice in the agent learning process.

A complex non-stationary environment provides a dynamic learning domain. This domain is composed of other agents' diverse states. Thus, the complexity of the domain grows with the increase of the number of agents. Our intention is to seek an answer to the following questions:

> How can an agent perform robustly in the various types of multiagent environment, so that each agent can efficiently observe other agents behaviours, and learn from its observation in order to act (or adapt) effectively in the complex non-stationary environment? What is the macro-level phenomenon of the whole system, and how can this understanding of phenomenon improve individual performance?

Ultimately, through a learning period and a series of actions, agents can achieve top-ranked performance.

In traditional machine learning, the single agent learning process is designed to achieve one determined goal through exploration, with or without supervision. A rational decision improves performance at each step. However, when a single agent explores in a multiagent environment, a new dynamic occurs wherein every agent in this environment acts to optimize its own rewards at the state-of-art priority (interest). Therefore, there may exist a current temporary goal arising in the situation which may not lie in the same direction of a simple agent's objective learning curve. Furthermore, unknown factors in the environment add another level of difficulty to an agent who tries to maximize its performance. Assuming each agent is rational, we can model other agents' behaviours and predict their behaviours with sufficient confidence. However, if the assumption of rationality does not hold, this problem can be seen as a new problem where the original model with a certain objective function does not hold; thus, a new dynamic model is required to tackle this problem.

In addition to this dynamic occurrence arising in multiagent learning, another critical issue is how an agent balances between exploration and exploitation in the environment. The objective of exploration is to obtain maximum information about the environment; however, exploitation, as a result of exploration, is the ultimate required action. Thus, we include both these factors in the objective function which measures the satisfactory of agents' performance. While performing, this learning system is a closed system within a certain limited time; if it fails to reach satisfactory performance, the system changes to open stage and modifies required learning parameters. This whole process repeats in a cycle, and the learning process is formulated continuously.

## 1.3 Contributions

We provide the following contributions to the literature: 1) present the importance of analyzing the structural properties in multiagent problems; 2) provide a novel structure learning algorithm (MDRLSA) to learn a compact representation in random graphical games; 3) introduce an adaptive teamwork algorithm (SE-adaptive) for cooperative agents who choose an optimal level of teamwork in varying density of constraint structures; and 4) present a competitive multiagent simulation platform (ALGAE), and learn a Bayesian Network structure representation that is revealed among agents' strategies.

In one aspect, multiagent learning research focuses on learning from individual agent's past experience or modeling other agents' behaviors to improve performance. In the other aspect, research on multiagent systems addresses particular problems from a system perspective, with more focus on a number of agents' interactions, and provides optimal solutions. However, in a non-stationary multiagent environment, each agent adapts to proceed towards its target. Each micro-level step in time may present a different learning problem which needs to be addressed. In this non-stationary environment, a holistic phenomenon forms along with the rational strategies of all players; we define this phenomenon as structural properties. In this dissertation, we present how to extract the structural properties in multiagent problems.

A multiagent environment can be classified as self-interested, cooperative, or competitive according to agents' goal. Here, a self-interested environment differs from a competitive, where all agents are not competing with others as a general-sum game as in a competitive environment. Thus, we examine the structure from these three general multiagent environments: self-interested random graphical game playing, distributed cooperative team playing, and competitive group competition. In each scenario, we analyze the structure in each environmental setting and demonstrate a structure learned as a comprehensive representation: structure of players' action influence, structure of constraints in teamwork communication, and structure of inter-connections among strategies. This structure represents macro-level knowledge arising in a multiagent system, and provides critical, holistic information for each problem domain.

## 1.4 Outline

In the following chapters, we first review the related work in multiagent learning, framework and systems. Then, before we present our perspective on how to tackle multiagent systems, we analyze the characteristics of various scales of a multiagent learning problem from a holistic perspective.

After revealing the important characteristics which exist among multiple player games in Chapter 3, we further explore the structural connection of mutual influence between players' action choices in game-playing in Chapter 4. In Chapter 5, we explore team-playing games, how structure matters to achieve the best exploration strategy in various network connections, in order to balance the time consumption and overall payoff. In Chapter 6, we present a simulation of multiagent systems in a competitive environment, artificial life, where we analyze what we can learn from survivors' fitness through a graphical representation: Bayesian Networks. In our last chapter, we present some open issues and point toward future research.

# Chapter 2

# Background: Multiagent Learning, Framework and Systems

A multiagent system (MAS) (Stone and Veloso, 2000; Wooldridge, 2008; Shoham and Leyton-Brown, 2009) has a broad set of definitions; each definition leads to different constraints to solve MAS tasks. The goal of machine learning is to build intelligent programs which can solve problems after a learning and evolving process. This intelligent program is often called an "agent".

An agent is a computational application that is designed to automate certain tasks with a guiding intelligence, to achieve a result. A multiagent environment is one in which more than one agent acts while agents interact with one another to perform tasks. Moreover, agents may or may not know everything about the environment. An agent learns by interacting in its environment and by observing the effect of these interactions. This learning, while performing in the environment, is the key to accumulating experience and forming knowledge through performance.

## 2.1 General Multiagent Learning Approaches

Multiagent learning (MAL) (Shoham et al., 2007; Stone and Veloso, 2000; Panait and Luke, 2005) has a long history in the game theory field, as well as in the machine learning community. In MAL, agents are given feedback about their behaviors as rewards or penalties in a given situation. Thus, reward-based methods are widely used in this field, including two major streams: reinforcement learning (RL) (Sutton and Barto, 1998) which estimates value functions, and evolutionary computation (EC) which directly learn behaviors using stochastic search methods. The similarities and differences between these two classes of learning methodology have generated a rich literature, and some address both classes, such as the bucket-brigade algorithm (Holland, 1985), the Samuel system (Grefenstette et al., 1990), and the recent Stochastic Direct Reinforcement policy gradient algorithm (Moody et al., 2004).

Evolutionary Computation is a family of mechanisms inspired by biological evolution such as reproduction, mutation, recombination, natural selection and survival of the fittest. Candidate solutions to the given problem play the role of individuals in a population, and the cost function (also calls "fitness function") determines the environment within which the solutions "live". Then evolution of the population takes place to select and continue to refine the population until time is exhausted, or an optimal solution is discovered.

Coevolutionary algorithms (CEAs) naturally apply evolutionary computation to refine multi-agent behaviors. In a CEA, the fitness of an individual is both subjective and context-sensitive, based on its iteration with other individuals in the population. In competitive coevolution, individuals benefit at the expense of their components, but in cooperative coevolution (CCE), individuals succeed and fail together in collaboration. Generally, cooperative coevolution algorithms (CCEAs) solve a problem starting by decomposing the problem, and then assigning each subcomponent to a separate population of individuals (Potter and De Jong, 2000).

Before we get into the learning process for multiple agents, we first examine how a single agent learns and evolves in an environment.

## 2.2 Single Agent Learning

One interesting problem arising along with this agent reinforcement learning process is the trade-off between exploration and exploitation (Sutton and Barto, 1998). Once an agent learns a certain action which has performed well, should an agent exploit this action since it is known to receive a decent reward? Or should the agent explore other possibilities in order to seek a better reward? Obviously, exploring is definitely a good tactic sometimes, but without a balance between exploration and exploitation, agents will not learn successfully. The common way to achieve a good balance is to try a variety of actions while progressively favoring those producing the most reward. In this section, we examine the most influential work in RL: temporal difference learning and Q-learning.

### 2.2.1 Markov Decision Process

An agent learning process can be separated into the following steps:

- Observe the surrounding environment;
- Decide an action (or "strategy") according to certain criteria;
- Perform the action;
- Agent receives feedback, rewards or penalty, from the environment;
- Information about experience is recorded. In detail, the experience includes the environment situation, the action chosen, and the feedback received.

Eventually, an agent can learn an optimal decision policy which performs the best in a certain environment, by performing actions and evaluating the results related. Markov decision processes are the foundation for research in single agent learning. A Markov decision process (MDP) (Sutton and Barto, 1998;Bellman, 1957) is a 4-tuple, $(S, A, T, R)$, where,

- $S$ is the finite set of the states;
- $A$ is the finite set of actions;
- $T : S \times A \times S \to [0, 1]$ is a transition function, which defines a probability distribution over next states as a function of the current state and the agent's action:
$$\forall s \in S, \forall a \in A, \sum_{s' \in S} T(s, a, s') = 1;$$
- $R : S \times A \to \mathbb{R}$ is a reward function, which defines the reward received when selecting an action from the given state.

At time $t$, the agent receives the reward $r^t = R(s^t, a^t)$, and the agent observes a new state $s^{t+1}$, which is drawn from the probability distribution specified by

$T(s^t, a^t, s^{t+1})$.

In general, the transition function $T$ and the reward function $R$ are not known in advance. Thus, the goal of a learning agent in an MDP is to learn a policy $\pi$ to maximize its long-term reward $R$ based on the only samples received. A policy $\pi$ is defined to map the probability of selecting an action from a particular state. Formally, $\pi \in S \times A \to [0,1]$, where $\forall s \in S, \sum_{a\in A} \pi(s,a) = 1$.

Two common ways to formulate the long-term reward are the discounted reward function and the average reward function. Define $V^\pi(s)$ as a policy's state value function, and $E(r^t \mid s^0 = s, \pi)$ as the expected reward received at time $t$ given the initial state $s$ and the agent follows the policy $\pi$. The average reward is formed as:

$$V^\pi(s) = \lim_{T\to\infty} \sum_{t=0}^{T} \frac{1}{T} E(r^t \mid s^0 = s, \pi), \tag{2.1}$$

which is under a common assumption that the MDP is a unichain. The unichain assumption is that the Markov chain induced by every stationary policy (perhaps randomized) has only one ergodic class of states and, perhaps, some transient states.[1]

The discounted reward is described as follows:

$$V^\pi(s) = \sum_{t=0}^{\infty} \gamma^t E(r^t \mid s^0 = s, \pi), \gamma \in [0,1). \tag{2.2}$$

$\gamma$ is a discount factor, which accumulates the immediate reward with probability $\gamma$ instead of a larger future utility. Temporal difference learning describes a class of algorithms that adopt this discounted reward formulation.

[1]An MDP is unichain if and only if, for all policies, there exists an ergodic set of states (i.e., any state in the set can be reached with non-zero probability from any other state in the set), and all states outside this set are transient (i.e., after some finite point in time it will never be visited again).

The Markov decision process is under the Markov assumption, which generally, requires that the next state and reward to the agent depend only on the current state and agent's action. Formally, we state this property of MDP as follows.

**Definition 1** *A decision process is Markovian if and only if, the sequence of states ($s_t \in S$), actions ($a_t \in A$), and the rewards ($r_i^t \in \mathbb{R}$), satisfies*

$$Pr\{s^t = s, r_i^t = r_i \mid s^{t-1}, a^{t-1}, \ldots, s^0, a^0\} = Pr\{s^t = s, r_i^t = r_i \mid s^{t-1}, a^{t-1}\}.$$

*An agent's selection of actions is Markovian if and only if,*

$$Pr\{a^t = a \mid s^t, s^{t-1}, a^{t-1}, \ldots, s^0, a^0\} = Pr\{a^t = a \mid s^t\};$$

*that is, only if the agent's next action depends only on the current state.*[2]

We also refer to a Markovian process as stationary, and in the multiagent framework of stochastic games, this property does not hold in a non-stationary environment.

### 2.2.2 Q-learning

Q-learning is the most significant breakthrough as an off-policy Temporal Difference (TD) control algorithm. The simplest, one-step Q-learning is defined as follows:

$$Q(s^t, a^t) \leftarrow Q(s^t, a^t) + \alpha[r^{t+1} + \gamma \max_a Q(s^{t+1}, a) - Q(s^t, a^t)], \qquad (2.3)$$

[2]Definition 1, 2, 3, 4, 5, 6, are adopted from the formulation presented in Bowling, 2003, and Definition 1, 6 are based on Sutton and Barto, 1998, Bellman, 1957.

where $\alpha$ is the learning rate, $0 < \alpha < 1$; when $\alpha$ is set to 0, it means that the $Q$-value is never updated and nothing is learnt; while $\alpha$ is set to 0.9, it means that learning can occur quickly. $Q(s^t, a^t)$ is the expected value of performing action $a$ in state $s$; and $\max_a Q(s,a)$ is the maximum reward received and then follows the optimal policy. The Q-learning algorithm is shown in Algorithm 1.

**Algorithm 1:** Q-learning: An off-policy TD control algorithm

```
Initialize $Q(s,a)$ arbitrarily;
**repeat** for each episode:
    Initialize $s$;
    **repeat** for each step of episode:
        Choose $a$ from $s$ using policy derived from $Q$;
        Take action $a$, observe $r$, $s'$;
        $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$;
        $s \leftarrow s'$;
    **until** $s$ *is terminal*;
**until**;
```

Take a discrete, grid game, "cat-mouse-cheese", as an example. The traditional rules of the Cat and Mouse game are:

a. Both the cat and mouse have 8 degrees of movement. Up, down, left and right, as well as the four diagonals.

b. The mouse scores reward $r_{ch}$ for getting the cheese. The mouse gets the cheese when it is in the same square as the cheese.

c. The mouse gets punishment $r_c$ for being caught by a cat, by simply both being in the same square.

d. If the mouse gets the cheese, a new piece is placed randomly while the cat and mouse keep their positions.

e. The game is over when the cat catches the mouse. The scores are then updated and a new game can begin.

In this general cat and mouse game, the mouse, as the intelligent agent, performs while learning in a $n \times n$ grid; on the other hand, cat is not a learning agent. In each learning episode, the mouse/cat action space is $a \in [0, 7]$, $\|a\| = 8$; while the observation space is the cat's grid position $[c_x, c_y]$ and the mouse's grid position $[m_x, m_y]$ and if the mouse is in hole: $flag$. The performance measure for this game is the cumulative reward for the mouse. Using the off-policy Q-learning algorithm, the mouse can be trained to learn strategies to gain optimal cumulative reward in this $n \times n$ grid environment. However, when the cat is also a learning agent, can this mouse continue to survive with optimal performance using the same strategies? The answer is probably no. This scenerio leads to a different issue: a multiagent learning problem.

In the next section, we present a general framework for multi-agent learning.

## 2.3 General Framework of MAL

In a multiagent learning framework, multiple agents process three different categories of activities: perception, reasoning and action (see Figure 2.1). First, each agent observes other agents and collect information in the environment, called "perception". Second, agents conduct reasoning according to their own preferences and knowledge to decide an optimal strategy; thereafter, agents perform their actions and receive feedback respectively.

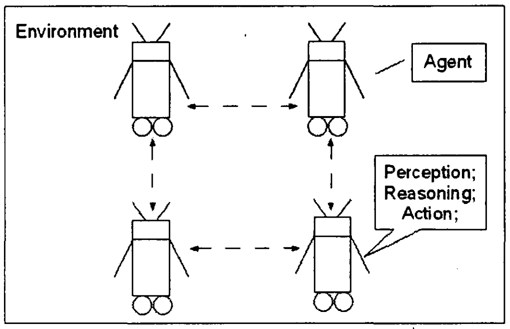


Figure 2.1: Multiagent Framework

Stochastic games are defined as multiple agents with a multiple states framework, which can be viewed as a synthesis of Markov decision processes and matrix games. MDPs model a single agent, multiple states model, which have been explored prominently in the field of reinforcement learning (see Section 2.2.1). On the other hand, matrix games describe a multiagent system with single state model, which are the foundational concepts in the game theory field. Since stochastic games share concepts with these two simpler frameworks, it is useful to consider them independently to analyze the core concepts while addressing the critical issues existing in stochastic games only. Figure 2.2 illustrates the relations among these three concepts. In Section 2.2, we discuss MDP as a single agent reinforcement learning; then, we examine matrix games, a multiagent, single-state learning process.

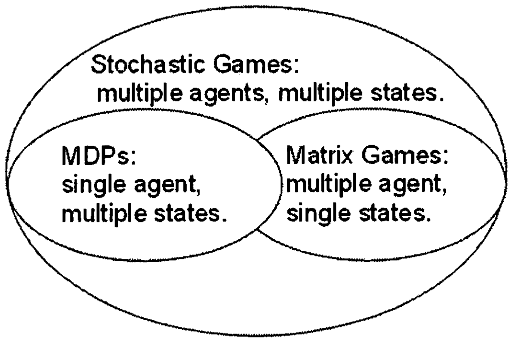


Figure 2.2: Stochastic Games include MDPs and Matrix Games

### 2.3.1 Matrix Games

Matrix games were first examined in the field of game theory to model strategic interactions of many decision makers (von Neumann and Morgenstern, 1944; Osborne and Rubinstein, 1994). Mathematically, a matrix game (or strategic game) is a tuple $(A, R)$, where $A = A_i \times \cdots \times A_n$ is the action space for each player; player $i$ chooses an action $A_i$, and receives the payoff $R_i$, $i \in [1, n]$, which depends on all the players' actions. $R$ is normally written as $n$-dimensional matrices, and each entry in the reward matrices corresponds to the joint actions taken. The learning process in matrix games means that agents repeatedly play the same matrix game, which is also called a repeated game. Agents learn through experience from observation of other agents' behaviors and their rewards, to maximize its own reward.

#### Examples

As follows, we list several matrix games and the payoff function matrices. Note that $R_1$ is the payoff matrix for player 1 and $R_2$ is for player 2. In each game matrix, the row represents player 1, and the column represents player 2.

- **(a) Rock-Paper-Scissors**

  Two players with each having three options: "Rock", "Paper" and "Scissors", and the rules are: "Rock" loses to "Paper", "Paper" loses to "Scissors", and "Scissors" loses to "Rock"; otherwise, it is a tie. The winner gains one dollar from the loser, while the loser loses one dollar. For example, player 1 plays $P$ while player 2 plays $S$, and the reward is $-1$ for player 1 and 1 for player 2.

$$R_1 = \begin{bmatrix} & R & P & S \\ R & 0 & -1 & 1 \\ P & 1 & 0 & -1 \\ S & -1 & 1 & 0 \end{bmatrix}, \ R_2 = \begin{bmatrix} & R & P & S \\ R & 0 & 1 & -1 \\ P & -1 & 0 & 1 \\ S & 1 & -1 & 0 \end{bmatrix}.$$

- **(b) Coordination Game**

  Two players simply both desire to agree on their action choice, but with no preferences between them.

$$R_1 = \begin{bmatrix} & A & B \\ A & 1 & 0 \\ B & 0 & 1 \end{bmatrix}, \ R_2 = \begin{bmatrix} & A & B \\ A & 1 & 0 \\ B & 0 & 1 \end{bmatrix}.$$

- **(c) Stackelberg Stage Game**

  The players of this game are a leader and a follower and they compete on reward quantity; the leader moves first and then the follower moves sequentially.

$$R_1 = \begin{bmatrix} & Left & Right \\ Up & 1 & 3 \\ Down & 2 & 4 \end{bmatrix}, R_2 = \begin{bmatrix} & Left & Right \\ Up & 0 & 2 \\ Down & 1 & 0 \end{bmatrix}.$$

Matrix games can be classified according to their payoff function. If one agent's gain is other agents' loss, we call this type of game as *general-sum games.* For example game $(a)$, the sum of player 1's gain and player 2's loss equals zero, we also call this *zero-sum game.* Another common type of matrix game is *team game*, i.e., game $(b)$, in which all agents have the same payoff function. In other words, one agent's best interest is the best interest of all others. Game $(c)$ looks similar to the general-sum game and team game, but it is neither of them.

What we can learn in game $(c)$ is as follows: imagine a repeated version of this game, and assume that the column player (secondary player: follower) is paying attention to the row player's (first player: leader) strategy and the rewards after each move. The two players will end up in a repeated (Down, Left) play and (Up, Right) play, since this is a way that benefits both. We conclude from this example: that learning and teaching happens at the same time: the row player has taught the column player to play in a way that benefits both most. Or, we can see this as an adaptation rather than a learning process. Note that the concept of strategy is not the same as a move. A move refers to an action taken by a player at the certain point during the game; while a strategy means a complete algorithm for playing the game which then tells a player what to do throughout the game.

Nevertheless, the learning agent's goal is to learn a strategy that maximizes its reward, using either pure strategies or mixed strategies. A pure strategy provides a complete set of how a player plays a game; while a mixed strategy is a probability of each pure strategy. An arbitrary finite matrix game may not have a pure strategy Nash equilibrium, but it always has a mixed strategy Nash equilibrium (Nash, 1951). Therefore, in our research, we focus on mixed strategies, and the definition is given as below.

A *mixed strategy* refers to a joint strategy $\sigma$ for all $n$ players. One player $i$'s strategy $\sigma_i$, specifies a probability distribution over all actions $A$, and its reward function $R_i$ is defined over mixed strategy as follows:

$$R_i(\sigma) = \sum_{a \in A} R_i(a) \Pi_{i=1}^{n} \sigma_i(a). \tag{2.4}$$

$R_i(a)$ is the reward received by player $i$ when playing action $a$, and $\sigma_i(a)$ is the probability distribution of playing action $a$.

In matrix games, one player's optimal strategy can only be evaluated if the other players' strategies are known. So, this is an opponent-dependent solution, also called *best-response.* We use $< \sigma_i, \sigma_{-i} >$ to represent the joint strategy where player $i$ follows $\sigma_i$ while others follow $\sigma_{-i}$. $\sigma_{-i}$ refers to a joint strategy for all the players except player $i$.

**Definition 2** *For a matrix game, the best-response function for player $i$, $BR_i(\sigma_{-i})$, is the set of all strategies that are optimal given the other player(s) play the joint strategy $\sigma_{-i}$. Formally, $\sigma_i^\star \in BR_i(\sigma_{-i})$, if and only if,*

$$\forall \sigma_i \in PD(A_i) \quad R_i(< \sigma_i^\star, \sigma_{-i} >) \geq R_i(< \sigma_i, \sigma_{-i} >)$$

*where $PD(A_i)$ is the set of all probability distributions over the set $A_i$ (the set of all mixed strategies for player i).*[3]

One most critical notion in matrix game and game theory is a best-response equilibrium, also called *Nash Equilibrium* (Nash, 1950).

**Definition 3** *For a matrix game, a Nash equilibrium is a collection of strategies for all players, $\sigma_i$, with*

$$\sigma_i \in BR_i(\sigma_{-i}).$$

*Therefore, no player can do better by changing strategies given that the other players continue to follow the equilibrium strategy.*

All matrix games have a Nash equilibrium, and there may be more than one. In *zero-sum* games, one appealing feature is that there is a unique Nash equilibrium, and this equilibrium corresponds to the games' *minmax* solution. In other words, this mixed strategy maximizes the worst-case expected reward. This solution can be found in a linear program as illustrated in Eq. 2.5.

$$\begin{aligned} \text{Maximize:} \quad & \min_{a_2 \in A_2} \textstyle\sum_{a_1 \in A_1} \sigma(a_1) R(< a_1, a_2 >), \\ \text{Subject to:} \quad & \textstyle\sum_{a_1 \in A_1} \sigma(a_1) = 1, \\ & \sigma(a_1) \geq 0, \quad \forall a_1 \in A_1. \end{aligned} \tag{2.5}$$

This solution is player 1's equilibrium strategy, where this linear program has $||A_1||$ parameters. Player 2's strategy can be solved similarly. In Rock-Paper-Scissors game, there is a unique Nash equilibrium in which each player selects their actions

[3]Definitions 2, 3, 4, 5 are based on Nash, 1950.

with equal probability 1/3 (as mixed strategy Nash equilibrium). But, if one player simply adopts this equilibrium strategy, will the player win the competition of a tournament? The answer is no, because a Nash equilibrium provides a rational strategy, not necessary a best benefit one. Furthermore, in a general matrix game, finding a Nash equilibrium is known to be NP-hard, yet is still an open question (Gilboa and Zemel, 1988; Conitzer and Sandholm, 2008).

### 2.3.2 Stochastic Games

Stochastic games are an extension of a combination of matrix games and MDPs, which include multiple agents with multiple stages. Formally, a stochastic game (Shapley, 1953) can be represented as a tuple: $(n, S, A, T, R)$, where:

- $n$ is the number of agents;
- $S$ is a set of stages;
- $A$ is a set of actions, $A = A_1, \cdots, A_n$; $A_i$ is player $i$'s action. (We assume that each player has the same strategy space in all games. This is a notational convenience, not a substantive restriction.)
- $T$ is a transition function specifying the probability of the next stage game to be played based on the game just played and the action taken in it: $S \times A \times S \rightarrow [0, 1]$, such that,
$$\forall s \in S, \forall a \in A, \quad \sum_{s' \in S} T(s, a, s') = 1.$$
- $R$ is the reward function, $R = R_1, \cdots, R_n$. $R_i$ is the immediate reward function of player $i$ for at the stage $S$: $S \times A \rightarrow R$. Note that each player has its own independent reward function.

When $n = 1$, stochastic games are MDPs; when $\|S\| = 1$, they are matrix games or repeated games. The goal for player $i$ in a stochastic game is to learn a policy that maximizes long-term reward, same as for MDPs. A policy for player $i$, $\pi_i$ is a mapping that defines the probability of selecting an action from a particular stage. Formally, $\pi_i \in S \times A \rightarrow [0, 1]$, where

$$\forall s \in S, \ \sum_{a \in A} \pi_i(s, a) = 1.$$

We use $\pi$ to refer to a joint policy for all the players, and $\Pi_i$ refers to the set of all possible stochastic policies available to player $i$, while $\Pi = \Pi_1 \times \cdots \times \Pi_n$ refers to the set of joint policies for all the players. $\pi_{-i}$ refer to a particular joint policy of all the players except player $i$, and $\Pi_{-i}$ refers to the set of such joint policies. Finally, the notion $< \pi_i, \pi_{-i} >$ refers to the joint policy where player $i$ follows $\pi_i$ while the other players follow their policy from $\pi_{-i}$.

Next, similar to MDPs, we need to define how to aggregate the set of the immediate rewards received in each stage for each agent in order to quantify the value of a policy. For finitely repeated games, we can simply use the sum or average reward which is the typical approach. For infinitely repeated games, the most common approaches are to use either the limit average or the sum of discounted rewards. The limit average reward function $V$ of player $i$ in stochastic games is defined similarly to MDPs, as follows,

$$V_i^\pi(s) = \lim_{T \to \infty} \sum_{t=0}^{T} \frac{1}{T} E(r_i^t \mid s^0 = s, \pi), \tag{2.6}$$

where $E(r_i^t \mid s^0 = s, \pi)$ as the expected reward to player $i$ received at time $t$ given the initial state $s$ and the agents follow the policy $\pi$. Similarly, the sum of discounted award function is defined with discount factor $\gamma, \gamma \in [0, 1)$, as,

$$V_i^\pi(s) = \sum_{t=0}^{\infty} \gamma^t E(r_i^t \mid s^0 = s, \pi). \tag{2.7}$$

Notice that this reward function for each agent $i$ is dependent on the joint policy of the other agents. As in MDPs, we can also define $Q$-values for a given agent for a particular joint policy. For the discounted reward framework, $Q$-values can be formulated as,

$$Q_i^\pi(s, a) = R_i(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V_i^\pi(s').$$

On the other hand, similar to matrix games, there is a best-response in stochastic games. Notice that a policy for a player can only be evaluated in the context of all the players' policies.

**Definition 4** *For a stochastic game, the best-response function for player i, $BR_i(\pi_{-i})$, is the set of all policies that are optimal given the other player(s) play the joint policy $\pi_{-i}$. Formally, $\pi_i^\star \in BR_i(\pi_{-i})$, if and only if,*

$$\forall \pi_i \in \Pi_i, \quad \forall s \in S, \qquad V_i^{<\pi_i^\star, \pi_{-i}>}(s) \geq V_i^{<\pi_i, \pi_{-i}>}(s),$$

*where $PD(A_i)$ is the set of all probability distributions over the set $A_i$ (the set of all mixed strategies for player i).*

We can also define the most critical notion: a best-response equilibrium or *Nash Equilibrium*, similar to matrix games in game theory.

**Definition 5** *For a stochastic game, a Nash equilibrium is a collection of policies, one for each player,* $\pi_i$*, such that,*

$$\pi_i \in BR_i(\pi_{-i}).$$

*Therefore, no player can do better by changing policies given that the other players continue to follow the equilibrium policy.*

Stochastic games can be classified the same way as matrix games. Team games are the ones where all the agents receive the same reward function. General-sum games are the ones where one player's gain means other players' loss. Zero-sum games refer to the sum of total rewards equals to zero. Like matrix games, zero-sum stochastic games have a unique Nash equilibrium.

In stochastic games, the Markov assumption still holds, but it has a different form, given in Definition 6.

**Definition 6** *A multiagent decision problem is Markovian if and only if, the sequence of states* ($s^t \in S$)*, actions* ($a^t \in A$)*, and the rewards* ($r_i^t \in R$)*, satisfies*

$$Pr\{s^t = s, r_i^t = r_i \mid s^{t-1}, a^{t-1}, \ldots, s^0, a^0\} = Pr\{s^t = s, r_i^t = r_i \mid s^{t-1}, a^{t-1}\}.$$

*That is, if the next state and rewards depend only on the previous state and all of the agents' actions, but not on the history of states and actions.*

From the game's perspective, stochastic games are Markovian, but from a single agent's perspective, the process is no longer stationary or Markovian (versus "behavior strategy"[4]). It is because the transition probabilities associated with a single agent's action from a state are not stationary and change over time as the other agents' action choices change. This property is critical to single-agent reinforcement learning research, and this violation of basic assumptions require new techniques to be developed to learn effective policies in stochastic games.

## 2.4 Multiagent Systems and Related Work

In the evolutionary computing community, multiagent learning research focuses on refining multiagent behaviours through each generation by assessing the fitness of the individual. In competitive coevolution, individuals benefit at the expense of their opponent; in cooperative coevolution, individuals succeed or fail together while in collaboration. This process refines the population until the sufficient level of fitness for individuals is discovered. As team learning goes, both the homogeneous (Haynes and Sen, 1996b; Haynes et al., 1995a; Haynes et al., 1995b) and heterogeneous forms (Luke and Spector, 1996; Andre and Teller, 1999; Haynes and Sen, 1996a; Haynes and Sen, 1997a; Haynes and Sen, 1997b; Potter et al., 2001) are promoted, from the perspective of allowance of different roles or behaviours for each agent in the group who has successful achievement. However, in this setting, one critical feature in multiagent learning has not been achieved: where a single agent's individual learning process has not been addressed to solve the issues arising when emergent, unforeseen changes occur.

This dynamic feature in a multiagent setting has been addressed in reinforcement learning communities. In the MAL literature, the RL community extends

[4]A behavior strategy is defined if $\pi_t = f(h_t)$ where $h_t$ is the history up to time t; a makovian or stationary strategy is a special case of behavior strategy when $h_t = \phi$.

Bellman-style single-agent reinforcement learning techniques to a multiagent setting, in particular Q-learning (Watkins and Dayan, 1992), which learns the utility of performing actions in states for controlling and prediction purposes. This technique has performed well in: a) zero-sum repeated games (Littman, 1994; Littman and Szepesvari, 1996), b) common-pay-off (or "team") repeated games (Claus and Boutilier, 1998; Kapetanakis and Kudenko, 2004; Wang and Sandholm, 2002), but not so well in c) general-sum stochastic games (Hu and Wellman, 1998; Littman, 2001; Greenwald and Hall, 2003).

In multiagent reinforcement learning settings, research takes on stochastic games and focuses on individuals who learn simultaneously and converge to optimal results (Bowling, 2005; Claus and Boutilier, 1998; Hu and Wellman, 2003). The important research of GIGA-WOLF (Generalized Infinitesimal Gradient Ascent - Win or Learn Fast) proves the no-regret and convergence criteria theoretically and experimentally in general-sum games. In general, optimal payoff is the common interest for the multiagent learning process. (Here, optimal payoff refers to no-regret.)

Nevertheless, scalability with the number of agents is a critical problem for multiagent learning. Multiagent learning involves multiple agents' behaviours in order to solve a common task, thus the search space can grow exponentially according to the number of agents and the complexity of agent behaviour. The evaluation criteria for learning methods should be standardized with respect to their scalability. In a general-sum learning process, especially with partially observed stochastic games, research usually involves studies in two-agent scenarios, with two or three actions for each agent. When scaled up to include more agents, current methods are unlikely to work in practice. In cooperative multiagent systems, to optimize a global objective has been addressed as the Distributed Constraint Optimization Problems

(DCOPs) with promising results. Research on finding globally optimal DCOP algorithms has been provided, such as ADOPT (Asynchronous Distributed Constraint Optimization) where it proves that DCOPs are NP-hard (Modi et al., 2005). Thus, when the number of agents increases, regarding both computational and communication requirement, the scalability needs to be improved. Furthermore, modelling the uncertainty in the multiagent systems, rich model such as Decentralized Partially Observable MDPs (DEC-POMDPs) gives promising results but mostly limits their applications with two or three agents (Bernstein et al., 2000). Velagapudi et al. have scaled DEC-POMDPs with hundreds of agents by given *coordination locals* (CLs) as heuristic information (Velagapudi et al., 2011). Interactive POMDP (IPOMDP) (Gmytrasiewicz and Doshi, 2005; Rossi, 2013) can explicitly model and predict the other agents intention (i.e., mixed strategy) under partial observability. However, both DEC-POMDPs and DCOPs assume that agents act in a static environment, and solving IPOMDP is prohibitively expensive due to computational difficulties that policy space grows exponentially with the length of planning horizon, where dynamic factors havenot been encountered and modeled.

## 2.5 Summary

In this chapter, we described the single agent learning process and examine most critical techniques Q-learning in the reinforcement learning field. Thereafter, we introduce MDPs and matrix games, since stochastic games can be seen as a merging of both. Through detailed analysis of MDPs and matrix games, we present the general framework for multiagent learning, and some important concepts in stochastic games and in game theory. Last, we examine multiagent systems and related work.

---

In the next chapter, we analyze the structure characteristics and the dynamics in large scale multiagent systems.

# Chapter 3

# Methodology

Multiagent research derives from two perspectives: learning and systems. In one aspect, multiagent learning research focuses on learning from individual agent's past experience or modeling other agents' behaviors to improve performance. In the other aspect, research on multiagent system addresses particular problems from a system perspective, with more focus on a number of agents' interactions, and provides optimal solutions. However, in a non-stationary multiagent environment, each agent adapts to proceed towards its target. Each micro-level step in time may present a different learning problem which needs to be addressed. In this chapter, we present our research methodology on how to solve multiagent problems, and illustrate the common characteristics and dynamics in large scale multiagent systems.

## 3.1 Research Methodology and Framework

We propose a novel multiagent environmental description, and a system process to describe the interaction among agents. This design differs from the description in

Chapter 2, this new unified framework includes more functional features for each agent to perform adaptively in dynamic environment with guidance from macro-level influence of structures, as well as micro-level individual learning and modeling. In this non-stationary environment, a holistic phenomenon forms along with the rational strategies of all players; we define this phenomenon as structural properties. The macro-level influence forms as a holistic phenomenon of mutual influence, constraints or strategies, while all the players perform intelligently in this environment. These structure connections can play crucial role where the collective intelligence feature appears.

A multiagent environment can be classified as self-interested, cooperative, or competitive according to an agents' goal. Here, a self-interested environment differs from a competitive, where all agents are not competing with others as a general-sum game as in a competitive environment. Thus, we examine the structure from these three general multiagent environments: self-interested random graphical game playing, distributed cooperative team playing, and competitive group competition. In each scenario, we analyze the structure in each environmental setting and demonstrate a structure learned as a comprehensive representation: structure of players' action influence, structure of constraints in teamwork communication, and structure of inter-connections among strategies. This structure represents macro-level knowledge arising in a multiagent system, and provides critical, holistic information for each problem domain.

## 3.2 Understanding the Multiagent Problem: Structure and Dynamics

Before we present our perspective on how to tackle multiagent systems, we focus on understanding the common characteristics in multiple players games. More specifically, we seek to understand the structure formed by a large number of agents in a multiagent system. The structure of a multiagent system reveals characteristics, which provides critical information for solving multiagent system problems.

How does each player perform efficiently in a multiagent environment? To find an answer to this question, we explore which diagram can describe multiagent system procedures. Firstly, we analyze that common characteristics underlying network models which are naturally formed (socially or biologically), by inspecting all connections existing among nodes. Then, a "preferential pub choosing" example is given to demonstrate large scale agent networks. In contrast to large scale multiagent networks, we introduce another interesting phenomenon appearing in group population: replicator dynamics.

### 3.2.1 Large Scale of Agent System's Structure Characteristics

A network graph model, $G = (V, E)$, is composed of a collection of $N$ vertices (or nodes) $V$, and lists of edges $E$. Each vertex represents an individual player; while every edge connects a pair of nodes that are neighbours. One typical example of a large scale multiagent environment is social networks. Social networks are formed by a number of persons, where each has ties with others. In social networks, each player performs in this complex network, and the structure of a social network plays a role. Each large scale network is formed from a large number of individuals, where each individual is unique microscopically, however, they emerge with common macroscopic characteristics. In this section, we introduce three distinct visual characteristics that exist in large networks: heavy-tailed degree distribution (Broder et al., 2000), small diameter (Travers and Milgram, 1969), and high clustering of connectivity (Watts and Strogatz, 1998; Watts, 1999; Strogatz, 2001; Boccaletti et al., 2006).

### Heavy-tailed degree distribution

In large universal networks, a large number of connections exist and each node has influence on various numbers of neighbors. A mathematical model is needed to differentiate the influential nodes from all the nodes in the network. Plot a histogram between the number of connections each node holds and the number of same influential nodes which exist together. The relations follow a power-law distribution, rather than follow a bell-shaped normal distribution. The power-law distribution is also called heavy tail distribution, known as the "80-20" rule. The statistical characteristic of heavy-tailed is that it is linearly on a log-log scale.

$$y = a * x^k,$$

$$y = probability(x) \propto x^k.$$

To illustrate the characteristics of social networks, we illustrate how to choose a pub as an example. At scenario one, each customer chooses a pub uniformly at random, ignoring how many others are currently there at each pub. In Figure 3.1 (a), it shows that the distribution of number of customers and the number of pubs which contain the same customer volume, which follows the 'bell' curve. At scenario two, each customer chooses a pub to go which is more popular. Figure 3.1 (b) shows the customer numbers and the number of pubs with the same capacity follows a heavy tail distribution. Figure 3.1 gives a demonstration for this pub phenomenon and presents the heavy tail characteristic. This example demonstrates the "rich get richer" phenomenon, which is also shown as heavy tail distribution.

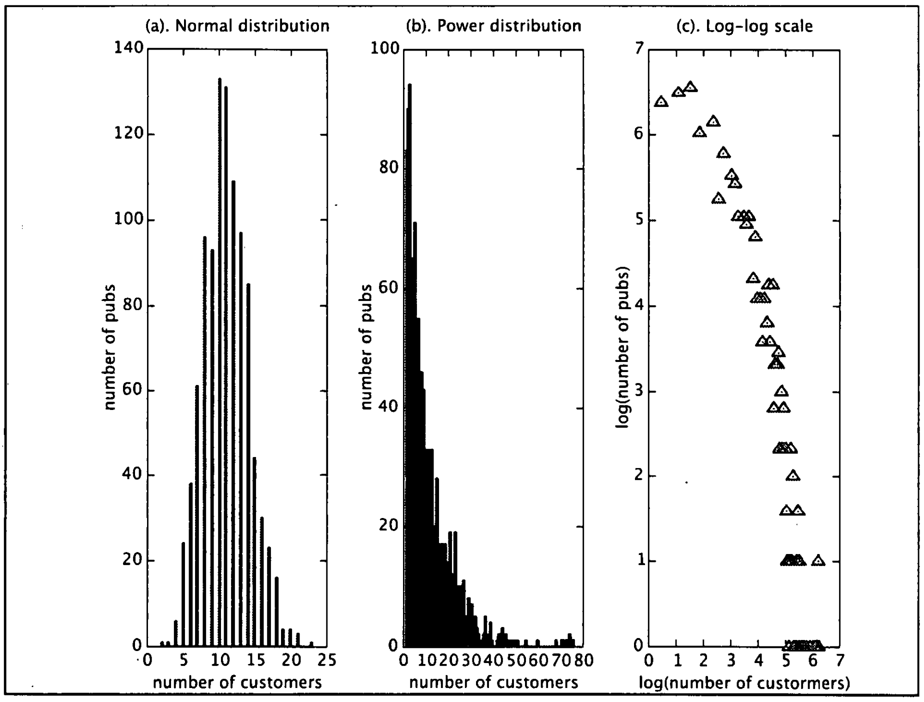


Figure 3.1: Pub choice (Artificial data): (a). Normal distribution: each customer chooses a pub uniformly at random, ignoring how many others are currently there; (b). Power law distribution: customer chooses a pub proportionally to its current popularity count; (c). Log-log scale of number of customers and number of pubs.

## Small diameter

The distance between two vertices is the length of the shortest path connecting them. The diameter of a network is the average distance between pairs. It measures how near or far typical individuals are from each other. According the definition of diameter, in a $N$ node graph $G$, the smallest diameter of $G$ is 1, which suggests a fully connected network and all $N(N-1)/2$ edges exists. The largest diameter exists in a chain graph, which is linear in $N$. In large scale networks, small diameter exists, considering the large population size, also known as small world of "six degrees of separation", ($\log(N)$ or $\log(\log(N))$).

$$diameter(G) = \frac{\sum_{i,j} d(i,j)}{N(N-1)/2},$$

where $i$, $j$ are any node in $G$.

### High clustering

A clustering coefficient is a measure of how densely tied together edges are in a graph. Locally, the clustering coefficient of node $v_i$ describes as the fraction of pairs (or friends that are also friends). Formally, let $k_i$ is the degree of $v_i$, which also means the number of neighbours of $v_i$. Thereafter, the maximum number of edges $e_{jk}$ among node $v_i$'s neighbourhood $N_i$ is $\frac{1}{2}k_i(k_i-1)$, where $v_j, v_k \in N_i$, and $e_{jk} \in E$. That is, every neighbour node of $v_i$ is connected with every other neighbour node of $v_i$. Let $c(v_i)$ denote the fraction of these actual edges that exist, which stand for friends of $v_i$ that are also friends:

$$c(v_i) = \frac{2\|e_{jk}\|}{k_i(k_i-1)}, 0 < c(v_i) \leq 1.$$

The clustering coefficient of graph $G$ is defined as the average of $c(v_i)$ over all the nodes $v_i$ in $G$:

$$C(G) = \frac{1}{N}\sum_{i=1}^{N} c(v_i).$$

Let $p$ be the edge density of the graph $G$:

$$p = \frac{\|E\|}{N(N-1)/2}.$$

When $C(G) \ll p$, we say graph $G$ is highly clustered.

When speculating about the connections among all the players, large scale social networks form those three common characteristics. However, when a group of diversified players co-exist in a game, another interesting phenomenon appears in the population: replicator dynamics.

### 3.2.2 Replicator Dynamics

Replicator dynamics presents an evolutionary selection phenomenon appearing in variant population evolutionary processes.

Assume that, $1, 2, \ldots, N$ types exist in a population distribution. Given that $P^0$ : $p_1^0, p_2^0, \ldots, p_n^n$, and $\Pi^0$ : $\pi_1^0, \pi_2^0, \ldots, \pi_n^n$, where each type $i$ has its initial distribution of $p_i^0$ and payoff $\pi_i^0$. At each time step $t$, all individual types are rational to be updated to choose the highest payoff. Thus, at time $t+1$, the proportion of each type $i$ is updated as follows:

$$p_i^{t+1} = \frac{p_i^t * \pi_i^t}{\sum_{i=1}^{N} p_i^t * \pi_i^t}. \tag{3.1}$$

The replicator equation Eq. 3.1 describes the fitness function to incorporate the distribution of population types and provides the essence of selection.

Furthermore, Fisher's fundamental theorem (Fisher, 1930) states that the rate of increase in fitness of any organism is proportional to its genetic variance in fitness at that time. Higher variance genetically increases the rate of adaptation. Accordingly, designing high variance in a multiagent system for agent payoff distribution utilizes the higher rate of fitness and payoff.

Take the following example, to demonstrate the replicator dynamics which appears within these three groups of individuals, with different variance in the payoff evolutionary processes. Given three groups $G_1$, $G_2$, $G_3$, each group has four types of individuals with initial distribution $P^0$, and each individual type's payoff $\Pi^0$. For example, Figure 3.2 shows that one random group population is generated and presented, where four types of individuals co-exist in this size 100 population. Each has a proportion of $p_i$ percentage in the population, where $i = 1, 2, 3, 4$. Respectively, each type of individual has its fitness $P_i$-fitness.

In the following, we present three groups of population and illustrate their evolution process. Assuming all groups start with the same level of overall fitness (mean of each group), we analyze the overall fitness changes over time as appearing different group distribution and fitness variance. According to Eq. 3.1, after one time step, each group's distribution changes, as well as the overall fitness of the group. The fitness and the adaptation gain calculation is as follows.

- Given:

  Group 1: $P^0 = [1/4, 1/4, 1/4, 1/4], \Pi^0 = [10, 20, 30, 40]$.

  Group 2: $P^0 = [1/4, 1/4, 1/4, 1/4], \Pi^0 = [5, 15, 35, 45]$.

  Group 3: $P^0 = [1/4, 1/4, 1/4, 1/4], \Pi^0 = [0, 10, 40, 50]$.

- Each group's mean, variance and overall group fitness are:

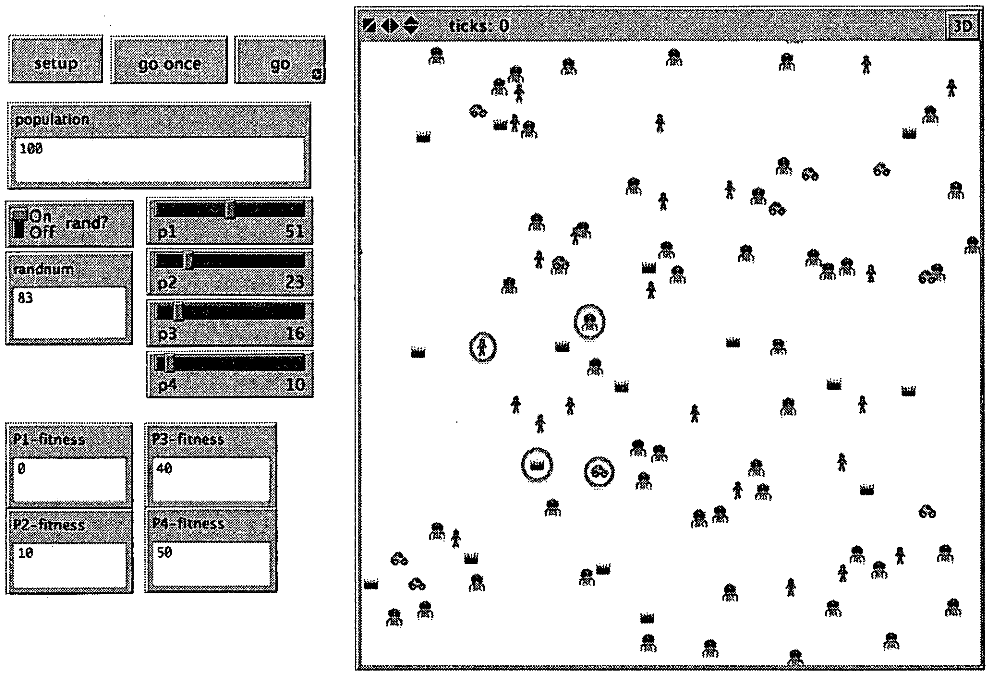


Figure 3.2: Replicator dynamics simulation

Group 1:

$$mean(G_1) = 25, var(G_1) = 166.67,$$

$$fitness^0(G_1) = \sum_{i=1}^{n=4} p_i^0 * \pi_i^0 = 1/4 * 10 + 1/4 * 20 + 1/4 * 30 + 1/4 * 40 = 25;$$

Group 2:

$$mean(G_2) = 25, var(G_2) = 333.33,$$

$$fitness^0(G_2) = \sum_{i=1}^{n=4} p_i^1 * \pi_i^1 = 1/4 * 5 + 1/4 * 15 + 1/4 * 35 + 1/4 * 45 = 25;$$

Group 3:

$$mean(G_3) = 25, var(G_3) = 566.67,$$

$$fitness^0(G_3) = \sum_{i=1}^{n=4} p_i^1 * \pi_i^1 = 1/4 * 0 + 1/4 * 10 + 1/4 * 40 + 1/4 * 50 = 25;$$

- After one time step, each group's distribution of four types individuals updates, as follows:

  Group 1:

$$p_1^1 = \frac{1/4 * 10}{25} = 1/10, \quad p_2^1 = \frac{1/4 * 20}{25} = 2/10,$$

$$p_3^1 = \frac{1/4 * 30}{25} = 3/10, \quad p_4^1 = \frac{1/4 * 40}{25} = 4/10.$$

  Group 2:

$$p_1^1 = \frac{1/4 * 5}{25} = 5/100, \quad p_2^1 = \frac{1/4 * 15}{25} = 15/100,$$

$$p_3^1 = \frac{1/4 * 35}{25} = 35/100, \quad p_4^1 = \frac{1/4 * 45}{25} = 45/100.$$

  Group 3:

$$p_1^1 = \frac{1/4 * 0}{25} = 0, \quad p_2^1 = \frac{1/4 * 10}{25} = 1/10,$$

$$p_3^1 = \frac{1/4 * 40}{25} = 4/10, \quad p_4^1 = \frac{1/4 * 50}{25} = 5/10.$$

- Thus, the new overall fitness of each group and its adaptation gain is as follows:

  Group 1:

$$fitness^1(G_1) = \sum_{i=1}^{n=4} p_i^1 * \pi_i^1 = 1/10 * 10 + 2/10 * 20 + 3/10 * 30 + 4/10 * 40 = 30,$$

$$Gain(G_1) = fitness^1(G_1) - fitness^0(G_1) = 30 - 25 = 5.$$

Group 2:

$$fitness^1(G_2) = \sum_{i=1}^{n=4} p_i^1 * \pi_i^1 = 5/100*5+15/100*15+35/100*35+45/100*45 = 35,$$

$$Gain(G_2) = fitness^1(G_2) - fitness^0(G_2) = 35 - 25 = 10.$$

Group 3:

$$fitness^1(G_3) = \sum_{i=1}^{n=4} p_i^1 * \pi_i^1 = 0 * 5 + 1/10 * 10 + 4/10 * 40 + 5/10 * 50 = 42,$$

$$Gain(G_3) = fitness^1(G_3) - fitness^0(G_3) = 42 - 25 = 17.$$

As these calculations show, the three groups of individuals, $G_1$, $G_2$, $G_3$, start with the same level of fitness; then one time step selection, (according to the simple rule: each type is rational to choose the highest payoff), results in three different fitness levels: 30, 35 and 42, respectively. Group 3 with the largest variance level ends with the highest fitness level.

In all, replicator dynamics reveals diversified players' evolutionary process. The higher variance exists in a group, the higher fitness a population leads to. However, after a number of selections, $t \to \infty$, the whole population reaches an equilibrium, regardless of the initial distribution of each type. The equilibrium only depends on the evolutionary updating rule. In this example, since each type's rational choice is to move to a higher payoff, the highest payoff is the principal criterion in the game.

## 3.3 Summary

In this chapter, we state that our research methodology of a new unified framework provides macro-level holistic information formed among agents. This structure property exists and is important to solving multiagent problems. Furthermore, we illustrate the characteristics which form in large scale networks: heavy-tailed, small diameter and high clustering. This analysis provides insights into how to solve problems arising in multiagent systems by taking account of their important features. Replicator dynamics demonstrates that the evolutionary selection rule appears in diversified populations. That is, high variance among the population leads to higher fitness in the evolutionary process. More importantly, these features provide us with guidance about how to design a multiagent system in agent-based simulations, utilizing these structural characteristics and features to simplify the problem solving-process in large-scale networks.

# Chapter 4

# Structure in Graphical Games

After revealing the important characteristics that exist among multiple-player games in Chapter 3, we further explore the structure connection of mutual influence between players' action choices in game playing. Much multiagent system and learning research have been performed from both machine learning and game theoretic perspectives. However, characterizing a multiagent system as a multiple players game, little research on how to abstract structure among players' actions has been performed. In this chapter, we provide a structure learning algorithm, "Multi-Descendent Regression Learning Structure Algorithm" (MDRLSA), to extract the action connections from graphical games. Knowing the influence between players' action choice can provide a compact representation for player's utility function, as well as reduce the search space for each player's learning process.

## 4.1 Graphical Games

Graphical games (Kearns et al., 2001) are a representation of multiplayer games to capture direct influence among players. A graphical game is described as an undirected graph $G$ in which players are represented as vertices, and each edge identifies influence between two vertices. In many natural settings, a player, vertex $v$, has payoffs that are specified by the action of $v$ and those neighbours of $v$ who have influence over $v$. In normal form representation, each player's payoff is given by a complete matrix with all players' action choice. However, each player's neighbour set is usually a small subset of the complete player set. Rather than give the entire population's normal form game, a graphical structure gives a direct and visual representation of the relationship among all the players. Graphical games are a suitable representation when sparse strong influences exist, whereas when there exists a large number of weak influences on each player, congestion games (Rosenthal, 1973) are applicable.

In our research, we generate multiplayer random graphical games represented in normal form using GAMUT (Nudelman et al., 2004). GAMUT is a suite of game generators designated for testing game-theoretic algorithms. A set of random graphical games are generated in GAMUT as experimental data input. For example, Figure 4.1 describes a six player random graphical game. In this game, each player has a choice action representing from 1 to 6, and total connections among players are represented as 10 edges. Each edge is a randomly selected connection between two players which determines/influences the payoff received for each player. In order to compare among different games, we normalize each game's payoff between 0 and 1. A strategy set $[2, 2, 1, 1, 1, 1]$ represents all players' action choice at one stage of the game, which indicates that player 1 chooses action 2, while player 2 chooses action 2, and player 3, 4, 5, 6 all choose action 1. These action combinations (also called a "strategy"), gives payoffs for player 1 to player 6 as follows respectively: $[0.95, 0.19, 0.34, 0.13, 0.55, 0.77]$. In Figure 4.1, normal form representation of this 6-player game states total number of 46656 ($6^6$) action profiles and the corresponding utilities for each player.

```
# Players: 6
# Actions:     6 6 6 6 6 6
# players:     6
# actions:     [6]
# graph:  RandomGraph
# graph_params:    [ -nodes 6 -edges 10 ]
# Graph Params:
# { nodes:     6, edges: 10, sym_edges:      true, reflex_ok:      false }
[1 1 1 1 1 1] :    [ 0.9888893423258294 0.14518868307974256 0.34666415138496487 0.038023128819446826 0.23594170400592485 0.8738306301416572 ]
[2 1 1 1 1 1] :    [ 0.6493444991678932 0.4155071673789125 0.34666415138496487 0.6590567860324859 0.4222770159171085 0.8738306301416572 ]
[3 1 1 1 1 1] :    [ 0.15193090776347526 0.12172733762196959 0.34666415138496487 0.06639426478458674 0.6214207116085547 0.8738306301416572 ]
[4 1 1 1 1 1] :    [ 0.26835599384705083 0.40252547595207655 0.34666415138496487 0.3413822595203671 0.7823505773305243 0.8738306301416572 ]
[5 1 1 1 1 1] :    [ 0.27927189630690374 0.9420454036400943 0.34666415138496487 0.1561862143425638 0.42474057576059127 0.8738306301416572 ]
[6 1 1 1 1 1] :    [ 0.16448162057531782 0.12410679110797923 0.34666415138496487 0.30514189249923224 0.4003220019315194 0.8738306301416572 ]
[1 2 1 1 1 1] :    [ 0.8218136574744752 0.02561315741805365 0.34666415138496487 0.18312778713997557 0.5239682518208068 0.7784223673233747 ]
[2 2 1 1 1 1] :    [ 0.9571011063043914 0.19445471692083255 0.34666415138496487 0.1308573184007137 0.554193377169171 0.7784223673233747 ]
[3 2 1 1 1 1] :    [ 0.21233316527055482 0.19483193314310926 0.34666415138496487 0.36152466372865794 0.2782392490110991 0.7784223673233747 ]
[4 2 1 1 1 1] :    [ 0.3499919411739944 0.34847870983594015 0.34666415138496487 0.37912818965244816 0.2541553162085356 0.7784223673233747 ]
[5 2 1 1 1 1] :    [ 0.7256479919065064 0.09740757528133273 0.34666415138496487 0.7004412054712835 0.06608667126949293 0.7784223673233747 ]
[6 2 1 1 1 1] :    [ 0.07736411265241327 0.8019652943445121 0.34666415138496487 0.38621656995429077 0.7295648258788623 0.7784223673233747 ]
```

Figure 4.1: Data sample from a 6-player random graphical game

In $n$-player games, assuming that each player has the same number of $a$ actions, normal-form representation requires $a^n$ entries of action profiles to describe multiple players' utilities. However, as the number of players $n$ increases, the action profiles size grows exponentially. Thus, a compact, visual representation to capture how every player's action choice influence others' utilities is interesting and critical. In the following section, we introduce a multi-gradient descendent regression model to learn a graphical structure representation of multiple players games from the normal form representation.

## 4.2 Learning Structure Algorithm: MDRLSA

In this section, we present a novel graphical structure learning algorithm for multiagent graphical games, called "Multi-Descendent Regression Learning Structure Algorithm" (MDRLSA). The MDRLSA uses a regression model to learn a player's utility function. Our hypothesis is that each player's utility function can be represented as a linear function of all players' individual action choices. Thus, the algorithm proceeds in two steps. In the first step, given all players' action profiles as input, define parameters $\theta$ and fit $\theta$ to all players' utility profiles $Y = [y_1 y_2 \dots y_{n_p}]$, where $n_p$ is the total number of players. The hypothesis $h_{\theta_k}(x)$ is given as Eq. 4.1 linear model

$$\begin{aligned} h_{\theta_k}(x) &= \theta_k^{\intercal} x = \theta_{0k} + \theta_{1k}x_1 + \cdots + \theta_{jk}x_j, \qquad (4.1) \\ \vec{\theta_k} &= \begin{bmatrix} \theta_{0k} \\ \theta_{1k} \\ \vdots \\ \theta_{jk} \end{bmatrix}, k \in [1, n_p]. \end{aligned}$$

To simplify the explanation, we assume all $n_p$ players have the same number of action choices, denoted by $n_a$. We apply multi-gradient descendent to achieve the objective of linear regression for each player $k$, which is to minimize its cost function, $J$, Eq. 4.2:

$$\begin{aligned} J(\vec{\theta_k}) &= \frac{1}{2m} \sum_{i=1}^{m} \left( h_{\theta_k}(x^{(i)}) - y_k^{(i)} \right)^2, \qquad (4.2) \\ \Theta &= [\vec{\theta_1} \dots \vec{\theta_k} \dots \vec{\theta_{n_p}}]. \end{aligned}$$

Here, $m$ is the total count of action profile, which is $n_p^{n_a}$; and $\theta_k^{\intercal}$ is the *transpose* of $\theta_k$. $x_j$ indicates a player's action choice, where $x_j = 1$ indicates taking action $x_j$ and $x_j = 0$ indicates action $x_j$ is not chosen. In $n_p$ players game, $[x_1, x_2, \ldots, x_{n_a}]$ describes the first payer's action profile, and $[x_{n_a+1}, x_{n_a+2}, \ldots, x_{n_a+n_a}]$ represents the second player's action profile, and so on. Here, $j \in [1, n_p * n_a]$, $n_p \times n_a$ is the total number of bits used to represent a strategy profile. As in 6 player game and each has 6 action choices, the action profile number $m$ is $6^6 = 46656$, and $[x_1, x_2, \ldots, x_{36}]$ stands for each player's action choice towards $a_1, a_2, \ldots, a_6$. $X = [x_1, x_2, \ldots, x_{36}]$ includes each player's action profile, see Figure 4.2.

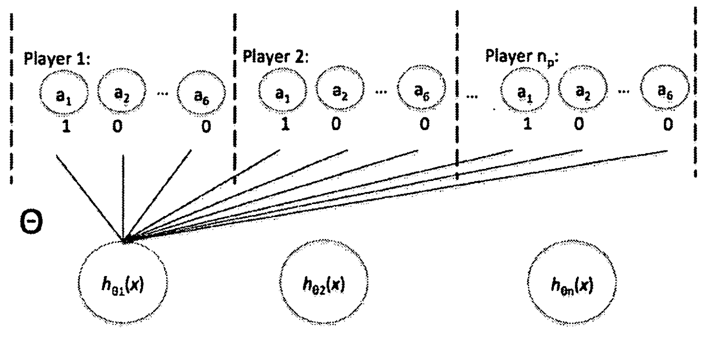


Figure 4.2: $n_p$ player' action mapping

The objective is to minimize the cost function value $J(\theta)$ by adjusting the $\theta_j$ values. In batch gradient descent, each iteration simultaneously update $\theta_j$ for all $j$ in Eq. 4.3:

$$\theta_{jk} := \theta_{jk} - \alpha \frac{1}{m} \sum_{i=1}^{m} \left( h_\theta(x_k^{(i)} - y_k^{(i)}) \right) x_{jk}^{(i)}. \tag{4.3}$$

In this experiment, we randomly initialize the initial parameters $\Theta$ to 0, the learning rate $\alpha$ to 0.01 and number of iteration as 400. This procedure is designed to observe the performance of our hypothesis linear regression model, in order to

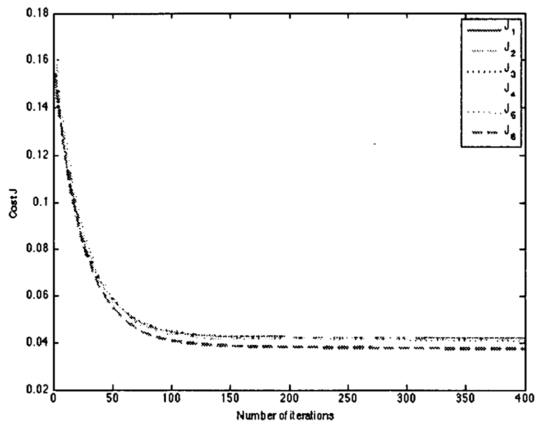


Figure 4.3: Convergence of gradient descent with learning rate $\alpha = 0.01$

avoid the model over-fitting or under-fitting the data. Figure 4.3 shows the cost function $J(\theta_k)$ as each player's utility loss function is decreasing as the number of iteration increases. This decrease of cost function $J$ proves that our linear model hypothesis Eq. 4.1 is correct description between players' action choice and their utilities. However, this batch gradient descent learning is quite slow. Thus, in large number of data entries, we choose normal equation Eq. 4.4 to optimize $\theta$ instead. As we can see in Eq. 4.4, until converge as in gradient descent, Eq. 4.4 includes no loop in the program, and learning rate $\alpha$ is not required.

$$\vec{\theta_k} = (X^{\intercal}X)^{-1}X^{\intercal}\vec{y_k}, \tag{4.4}$$

where

$$\vec{y_k} = \begin{bmatrix} y_{0k} \\ y_{1k} \\ \vdots \\ y_{jk} \end{bmatrix}.$$

In the second step, according the parameter $\Theta$, we map the coefficiency between each player's action choice and the given utility. Algorithm 2 describes the graphical structure learning algorithm MDRLSA, which indicates whether the influence between players are related or independent. The graph parameter is represented in a binary $n \times n$ matrix: each entry value '1' indicated related, whereas '0' indicates independent.

## 4.3 Results and Analysis

We develop the Multi-Descendent Regression Learning Structure Algorithm in Matlab (see Appendix A), which provides a graphical structure representation among players actions' influence for multiagent graphical games.

Taking one random graph game with six players, six actions and ten number of influence edges as an example, the learned player influence among them is shown in

**Algorithm 2:** MDRLSA

```
Data:                                                        // read game file;
    $X$ := action_profile;
    $U$ := utility_profile;
Result: graph_param;
begin
    Step 1: calculate $\Theta$ parameter;
    Initialize:
    $\Theta = [\,]$;
    for each player $i$ do
        $y = U(:, i)$;
        $\Theta = [\Theta, normalEqn(X, y)]$;
    end
    Step 2: map coefficiency graph_param;
    Initialize:
    $\epsilon = 0.00001$
    $data$ := 1 $\rightarrow n_a$ rows in $\Theta$
    $[m, n]$ := size(data);                              // n is number of player;
    $graph\_param$ := ones(n,n);                         // set a $n \times n$ matrix to all 1s;
    $temp\_coef$ := zeros(n,n);                          // Set a $n \times n$ matrix to all 0s;
    for each column player $i$ do
        $a$ := data(:,i);                                // $i^{th}$ column in $data$;
        for each row player $j$ do
            $player\_coef = a((j-1)*(m/n)+1 : ((j-1)*(m/n)+n))$;
                                         // $j^{th}$ row player's all action coefficient in a;
            $a\_norm = player\_coefmean(player\_coef)$;
                                   // normalize $player\_coef$ between 0 and 1 as $a\_norm$;
            $temp\_coef(j, i) = sum(abs(a\_norm - ones(n, 1)))$;
            // calculate the sum of all the absolute difference between a_norm and all true
            $n \times 1$ connections;
            if $temp\_coef(j,i) < \epsilon$ then
                // any choice of $j^{th}$ row player's action's influence on $i^{th}$ column player
                small than $\epsilon$;
                graph_param(j,i) = 0;
                                                         // flag unrelated player j & i as 0;
            end
        end
    end
end
```

the following matrix:

$$graph\_param = \begin{array}{c} player \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix} \end{array} \tag{4.5}$$

Figure 4.4 illustrates the conversion of Eq. 4.5 into graphical structure representation.

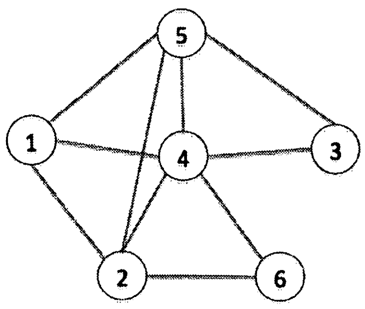


Figure 4.4: 6 player graphical game structure

In GAMUT, graph structure is represented as a list of neighbors of all nodes. By definition, graph $G$ is undirected; thus, two nodes connected by one edge have mutual influence between them. Moreover, each player's action choice results in its utility. That is, each player is its own factor. Thus, each player does not have its own node in the neighbor list, as well as one edge's two nodes only appear once in the neighbor lists. A 6-player random graphical game's structure is shown in GAMUT as lists of each node's neighbors in the following way:

$$\begin{aligned} node\ 1: &\ \ 2,4,5 \\ node\ 2: &\ \ 4,5,6 \\ node\ 3: &\ \ 4,5 \\ node\ 4: &\ \ 5,6 \\ node5: & \\ node6: & \end{aligned}$$

Comparing the structure shown in Eq. 4.5 with the benchmark generated by GAMUT, MDRLSA learns an accurate underlying structure for a 6-player random graphical game.

We test on a set of random graphical games with different number of players, actions and number of influence edges generated from GAMUT (see Table 4.1). The program runs in Matlab on Mac OS X, with Processor 2.8GHz Intel Core i7, Memory 8GB 1067MHz. The run time shown in Table 4.1 is one single run of MDRLSA. With the same graphical game, the run time varies in the scale of $10^{-1}$ of a random run time. For instance, a 5-player, 3-action game with 5-influence edges, the run time differs in 0.001 seconds. Given the listed random generated graphical games in the table, the structures learned are shown in Figure 4.5. MDRLSA shows a robust promising results of learning structure representation efficiently and effectively in random graphical games. The runnning time increases linearly to the number of strategy profiles.

| Player Number | Action Number | Number of Influence (Edges) | Runtime (Seconds) | Accuracy(%) | Normal Form Profile Entries |
|---|---|---|---|---|---|
| 4 | 3 | 3 | 0.0055 | 100 | 81 |
| 4 | 4 | 4 | 0.0071 | 100 | 256 |
| 5 | 3 | 5 | 0.0117 | 100 | 243 |
| 5 | 4 | 6 | 0.0098 | 100 | 1024 |
| 5 | 5 | 7 | 0.0318 | 100 | 3125 |
| 6 | 4 | 5 | 0.0311 | 100 | 4096 |
| 6 | 5 | 8 | 0.1007 | 100 | 15625 |
| 6 | 6 | 10 | 0.3078 | 100 | 46656 |

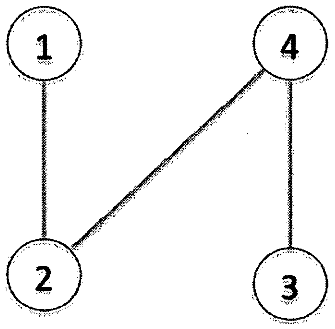
Table 4.1: Random graphical games experimental details

## 4.4 Comparison

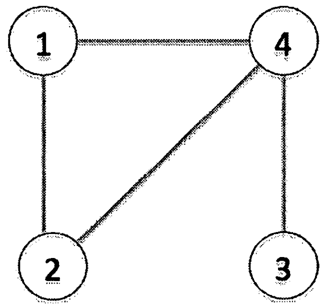
Duong et al., 2009 give a structure learning algorithm for graphical game structure learning. Their approach comes from a game theoretical perspective, which constructs a loss function and focuses on minimizing the loss of utility function in strategy choice. However, our approach comes from a machine learning perspective, and focuses on revealing the coefficiency between all the action choices and the outcome utility. Through the correlated coefficiency, the relative neighbour influence is identified. Both approaches are tested on GAMUT generated games. However, without demonstrating this in the same programming languages, we cannot evaluate the run time efficiency difference on both methodologies. Comparing both approaches, there are methodological advantages despite lack of theoretic analysis. Our approach is intuitive, straightforward and simple.

## 4.5 Concluding Remarks

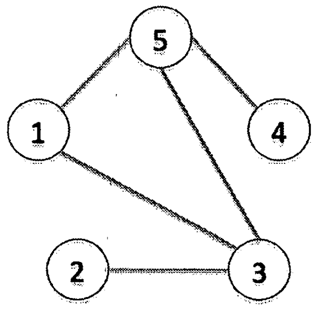
In this chapter, we demonstrate the structure properties in a self-interested multiagent environment. In simulated GAMUT random graphical games, we present the Multi-Descendent Regression Learning Structure Algorithm to learn a compact representation among agents' action influence towards each other. The Multi-



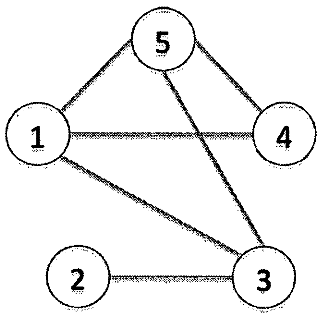
(a) 4 players, 3 actions, 3 edges



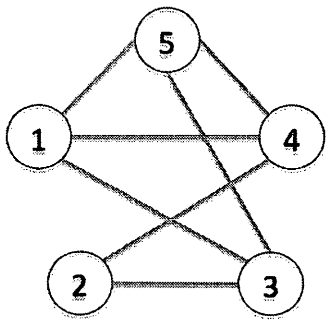
(b) 4 players, 4 actions, 4 edges



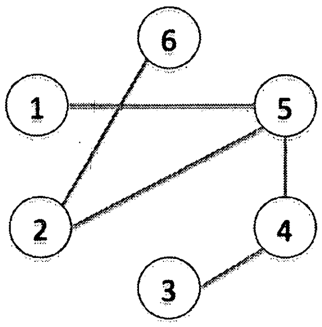
(c) 5 players, 3 actions, 5 edges



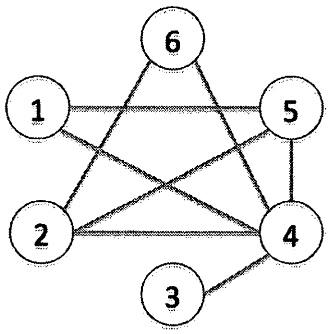
(d) 5 players, 4 actions, 6 edges



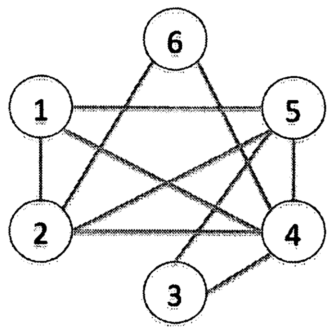
(e) 5 players, 5 actions, 7 edges



(f) 6 players, 4 actions, 5 edges



(g) 6 players, 5 actions, 8 edges



(h) 6 players, 6 actions, 10 edges

Figure 4.5: MDRLSA learned graphical structures

Descendent Regression Learning Structure Algorithm tests on a set of randomly generated graphical games. Experiments demonstrate that MDRLSA is suited to various graphical game applications, and provides promising results. The structure representation compared with normal form utility matrices reduces search space and identifies the mutual action influence among agents. MDRLSA can learn a good representation for games and MAS where there are sparse strong influences in individual player payoff.

# Chapter 5

# Structure in Teamwork

Communication among agents through connections in a network is one important aspect in a multiagent system. In a cooperative environment, agents communicate with neighbours and achieve the goal of higher total payoff overall. When exclusively acquiring information from neighbours, the time consumption increases exponentially to the scale of network inter-connection complexity. Our research exploits the influence of utilizing the structure information while performing, which can improve the optimal exploration strategy in a cooperative environment, so that the time consumption and overall payoff achievement is balanced.

In a multiagent system, incomplete information of strategy payoff is common in real applications. For instance, in the mobile ad hoc network domain, a group of cooperative agents explore in a distributed manner within limited time steps in a given environment to maximize the overall total payoff, with some uncertainty of local payoff. In this setting, each agent explores its surroundings to maximize the overall payoff and moves toward equilibrium. However, how to use a lower level of optimization among agents that leads to high overall payoff with limited time steps is a critical issue between the trade-off in exploration and exploitation. Our experiments demonstrate that, in a densely connected network, single agent's optimization can achieve 98% of a double agent's optimization, but the difference of overall payoff is achieved by higher level optimization with a large trade-off of time consumption. Thus, knowing the agent's connection structure, it is helpful to choose an optimal strategy for overall performance within the time constraint.

## 5.1 Ad hoc Networks

Our research chooses the mobile ad hoc network (MANET) domain as an example. A wireless ad hoc network is a self-configuring routable networking environment on top of a link-layer ad hoc network. A number of wireless sensor agents are free to move independently in any direction and change links to other agents frequently. Agents communicate with their neighbours according to preset topology connections, and forward traffic unrelated to their own use. The signal strength between agent communication varies in different locations. In the field, the primary challenge in building a MANET is to equip each device to continuously maintain the information required to route traffic properly while performing its tasks. Thus, our goal is to let each agent freely move to an optimal position in order to maximize the overall signal strength between all connected agent pairs, and maintain the information forwarding routes.

## 5.2 Rationale and Hypothesis

Scalability with the number of agents is a critical problem for multiagent learning. Multiagent learning involves multiple agents' behaviours in order to solve a common task, thus the search space can grow exponentially according to the number of agents and the complexity of agent behaviour. The evaluation criteria for learning methods should be standardized with respect to their scalability. In cooperative multiagent systems, to optimize a global objective has been addressed as DCOPs with promising results. Research on finding globally optimal DCOP algorithms has been provided, such as ADOPT where this proves that DCOPs are NP-hard (Modi et al., 2005). Thus, when the number of agents increases, regarding both computational and communication requirements, the scalability needs to be improved.

Our hypothesis is that exploiting a *k-optimal* strategy to obtain payoff is associated with the topology network density. In Modi et al., 2005, a class of 'incomplete' algorithms, *k-optimal* has been provided. A *k-optimal* algorithm defines that at every time step, a number of *k* agents coordinate their action choices to reach equilibrium, where no single agent's change of its action choice can improve the overall performance, such as achieving higher total payoff for all agents. As the number of *k* increases, the computation of reaching *k* equilibrium grows exponentially. That is, choosing 1-*optimal*, 2-*optimal* or *k-optimal* algorithm, this choice influences the performance to balance agents' total payoff and time consumption, and is directly associated according to agents' connection network density. Thus, in MANET domain, large number of sensors (agents) are required. Thus, 1-*optimal*, 2-*optimal* algorithms are applied to explore agents' total payoff with time constraints, and no triplet teamwork is chosen for relatively large numbers of agents.

## 5.3 An Adaptive Teamwork Algorithm: Static Estimated (SE)-Adaptive

In this section, we introduce SE-Adaptive, which can run either 1-*optimal* or 2-*optimal* algorithm per agent, improving the overall performance of the team. An agent running SE-Adaptive can decide whether to use 1-*optimal* or 2-*optimal* algorithm, depending on the number of neighbors it has.

Our hypothesis is that the connection density rate is an important potential factor in determining the level of teamwork required. We conducted three sets of experiments which include same number of rounds and variable settings: one set tested the full graph (where each agent has $n-1$ connections), while another tested chain graphs. In sparse topology, single agent movements are optimal, whereas in dense topology, high levels of team movement are rewarding. Thus, we choose the most sparse topology chain, as well as the most densely structured, fully connected network, and a third type of "hybrid" graphs as agents teamwork constraint (communication) structures. "Hybrid" graphs are constructed such that half ($\lfloor \frac{1}{2} \rfloor$) of the agents form a connected clique, and the remainder of the agents form a chain connected to one agent in the clique.

We define connection density rate, denoted by $r$, as the fraction of an agent's connection over the maximum connection number: $r = \frac{agent\_num\_connection}{max\_num\_connections}$. Algorithm 3 describes how the SE-Adaptive performs on connected agents topologies with different densities. A heuristic value $r = \frac{1}{3}$ is learned empirically from the set of experiments described above. When $r \leq \frac{1}{3}$, an agent chooses $k = 1$ individual movements as the optimal strategy; otherwise, agents move together with another agent as an optimal pair to obtain maximum rewards, where $k = 2$.

**Algorithm 3:** SE-ADAPTIVE

```
Data: connected agents networks with assignments;
Result: optimal gain and assignments overall the team;
for each neighbor i do
    Send variable assignment and reward matrices to $i$;
    if $r \leq \frac{1}{3}$ then
        Find max gain and perform 1-optimal algorithm:
        $g, a \leftarrow$ getMaxGainAndAssignment();
        Send Bid $g$ to all neighbors;
    else
        Find max gain and perform 2-optimal algorithm:
        $g', a' \leftarrow$ getMaxGainAndAssignment();
        Send Bid $g'$ to all neighbors;
    end
    Receive variable assignment and reward matrices from $i$;
end
```

## 5.4 Experiment Results and Analysis

In this section, we provide SE-Adaptive algorithm to perform on connected agents topology with different densities. The number of agents ranged from 10 to 15. Comparing the performance of each test on 1-*optimal* and 2-*optimal* algorithms, Figure 5.1 and Figure 5.2 show the results of using different numbers of agents running on different graph topologies for 50 rounds. SE-Adaptive performs as well as the best of 1-*optimal* and 2-*optimal* algorithms on chain and complete graphs. SE-Adaptive agents in the clique use 2-*optimal* algorithm and SE-Adaptive agents in the chain use 1-*optimal* algorithm, outperforming both algorithms individually.

In Figure 5.1 and Figure 5.2, the $x$-axis is the number of agents and the $y$-axis shows the reward value of net gain. We run three sets of experiments on full, chain and hybrid topologies for 30 independent trials. Here, hybrid topology is defined as half ($\lfloor \frac{1}{2} \rfloor$) fully connected graph and half ($\lceil \frac{1}{2} \rceil$) chain graph. In each trial, an agent could explore up to 50 positions in 50 rounds (i.e., variable settings), and obtain its rewards. In general, an agent will not explore all 50. In Figure 5.1 and Figure 5.2, the average of 30 trials rewards shows rewards obtained in three different algorithms. $k$=1 means 1-*optimal*: solo exploration; $k$=2 means 2-*optimal*: pairwise exploration; and SE-Adaptive means a combination of choices of solo and pairwise.

Figure 5.1 illustrates a group of fully-connected and chain-connected agents. In various sizes of group density for agent numbers 10-15, the performance of $k = 1$, $k = 2$, and SE-Adaptive algorithms, Figure 5.1 shows that pairwise exploration ($k = 2$) gains higher rewards compared with solo exploration ($k = 1$), and the SE-Adaptive algorithm chooses to move pairwise and gain the same rewards as $k$=2. Whereas, on a chain topology, solo exploration performs the best on a sparse chain topology, while the SE-Adaptive algorithm gives a similar optimal performance, both with high time efficiency. On the other hand, pairwise exploration can give better performance in a dense topology but performs poorly in a sparse topology; both situations have low time efficiency.

Figure 5.2 combines a maximum dense topology and a minimum connection topology in one graph, where the SE-Adaptive algorithm can choose a level of teamwork based on the neighbour connections and gives the best performance among the three. In a group of agents, agent numbers 10-19, the result shows that the average of net gain obtained by SE-Adaptive algorithm is higher than the average net

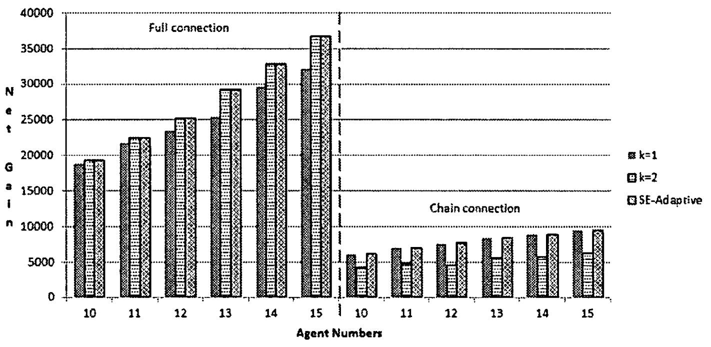


Figure 5.1: Full and chain topology, agent number: 10-15

gain of using either $k$=1 or $k = 2$ only one at time alone, but with much less time consumption.

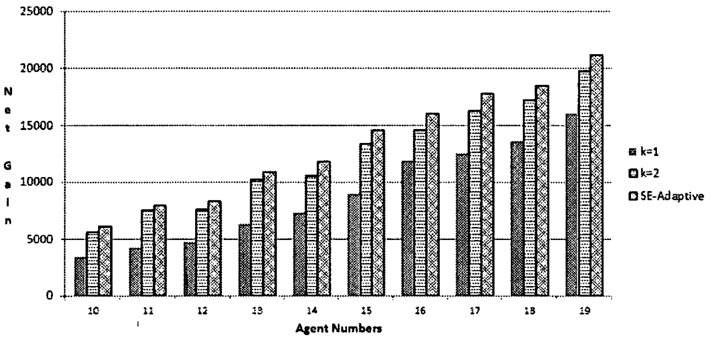


Figure 5.2: Hybrid topology, agent number: 10-19

The implementation and demonstration is tested on existing platform DCEE Python simulation code[1]. The runtime of each test on the 1-*optimal* algorithm on all three types of topologies is on the scale of minutes, agent numbers ranged from 10-15; whereas, the runtime on the 2-*optimal* algorithm on dense and hybrid topologies is on a magnitude of hours, where 2-*optimal* algorithm considers all possible pairs among $n$ number of agents and that there are $O(n^2)$ such pairs, and that this makes it much

[1]`http://teamcore.usc.edu/dcop`, version 0.9.2/5/2010.

slower than the 1-*optimal* algorithm. However, applying the SE-Adaptive algorithm on dense topology reduces the runtime to a matter of minutes, the same scale as 1-*optimal* algorithm. With double agents' teamwork, 2-*optimal* considers $O(n^2)$ such pairs, but achieves worse performance for net gains in a chain connection (see Figure 5.1). SE-Adaptive identifies when to choose 1-*optimal* algorithm according to the density rate $r$. In a chain topology, SE-Adaptive chooses 1-*optimal* since every agent satisfies the criterion of $r = \frac{2}{n-1} \leq \frac{1}{3}$, when total agent number $n \geq 7$. While in hybrid topology, SE-Adaptive switches from 1-optimal to 2-optimal when the connection density rate $r > \frac{1}{3}$, where only a small number of highly clustered agents choose 2-*optimal* algorithm and benefit from teamwork, see Figure 5.2. Meanwhile, the runtime of SE-Adaptive algorithm remains on the scale of 1-*optimal.*

## 5.5 Concluding Remarks

In this chapter, we demonstrate the importance of topology structure in a cooperative multiagent environment. By analyzing the structure of agents' constraints, we present an adaptive teamwork algorithm: SE-Adaptive, to choose optimal level of movement in various density constraints (i.e., communication) networks. Experimental results of SE-Adaptive algorithm give promise to this new concept of adaptive learning according to structure density among agent neighbors. Wisely chosen teamwork gives multiagent learning an important component to achieve an optimal performance within limited time in a dynamic environment.

# Chapter 6

# Structure Learning in Artificial Life Simulation

In this chapter, we capture the merits of an agents behavior strategy using a graphical model representation, while adapting to environmental changes. Bayesian networks (BN) are applied to demonstrate the inter-relationship of the adaptation strategies for agents' evolution according to environmental changes. We present a simulation of artificial life (AL), where two groups of agents compete for resources to survive in one community. This research is not intended to predict which agents or which group of agents can survive, but rather illustrate that agents adapt to environmental changes and that the inter-connections among the characteristics of those agents with the best behavioural strategies survive.

## 6.1 Rationale for Artificial Life Paradigm

In our research, we exploit the Artificial Life concept by building a simple ecology system: ALGAE (Artificial Life Genetic Algorithm Expression). The ALGAE concept is inspired by the evolutionary epoch of Homo Sapiens where two divergent species existed in a primitive environment to survive as the primary biological goal. The initial ALGAE simulation (Yan and Cercone, 2010a) analyzed agents competing for resources in an uncertain environment, where that ALGAE simulation provided artificial gene data. There are three aspects in this section which are derived from this perspective. Firstly, we show the environmental factors which determine the living conditions of the two species who are the subject of the experiment. Secondly, we explore the key genetic combinations which favor survival, with details about the intrinsic chromosomes and their variability during the evolutionary process. These environmental control settings and group agents' energy consumption are set up through the user's interface. This feature promotes understanding of the dynamics between species survival and environmental factors. Lastly, we reveal the hidden dependencies among fittest behaviour strategies' genomic descriptors which emerge during evolution of the species. We apply Bayesian network structure learning to show these relations among the genetic factors through the evolving strategy adaptation to environmental changes.

## 6.2 ALGAE System Design

Creation of a virtual artificial environment to emulate human populations is relatively straightforward. Two populations are devised and designated as separate species. These species co-exist in a competitive relationship, the goal of which is merely survival. By reproducing, mutating, and fighting, the program refines its variables, evolving to the point where the best solution is generated. Survival of the fittest mimics the evolution of living organisms' adaptation. Genetic fitness is one index of survivability. 'Rules' exist to govern the existence and inter-relationship of the two species in this artificial 'world', called a 'dynamic simulation framework'. These rules pertain to the constraints of the environment. Specifically, they address limitations which impinge on each species survival probabilities.

As AL simulates real organisms in their functioning and characteristics, genetics represents the information systems essential to evolution of genetic information so that species both survive and adapt over time. Our model uses standard pertinent factors in the artificial environment setting. There are basic operational rules, a well-defined virtual environment or search space, and behavioural constraints on the population species.

### 6.2.1 Description

In ALGAE, resources must exist, and these resources are distributed in a two dimensional grid randomly. We postulate two types of species in this virtual world: Species type '0' and Species type '1'. They survive in the virtual environment through competition for resources and obey these rules: species mate within their own species only, when minimum amount of energy reserved; each one subsist on native honey resource as a form of nourishment; when energy levels reach zero, an individual dies. Also, ages increase until they reach the maximum possible life span, then natural death occurs. Each individual in ALGAE is called '*ALgent*'. Barriers are initially placed in their living space and the distribution changes over the time to constrict their movement. But in this current setting, we remove the barriers from the environment, because our emphasis is the impact of food resources on *ALgents*' survival, and limiting the percentage of barriers in the environment does have an impact on the group's behaviour change.

All behaviors discussed indicate that the two species compete for resources to survive. As the population evolves, the distribution of resources changes over time. We examine a population of artificial chromosomes (AChromosomes) which present each ALgent $G_i$ in both species, as below:

$$G_i = [SP, SL, VF, MD, AS, ML, AC, AA, AL, EF], i = \{0, 1\}.$$

We examine a population of AChromosomes which present each *ALgent i* as different species, as Table 6.1:

| Gene | Description | Bit Site | Gene | Description | Bit Site |
|---|---|---|---|---|---|
| SP | SPecies type | 0 | ML | Motion Loss | 9 |
| SL | Life Span | 1-2 | AC | Action Choice | 10-12 |
| VF | Vision Field | 3-4 | AA | Attack Ability | 13-14 |
| MD | Motion Direction | 5-6 | AL | Attack Loss | 15-16 |
| AS | Action Speed | 7-8 | EF | Food Efficiency | 17-18 |

Table 6.1: 19-bit chromosome descriptor

ALGAE incorporates the genetic algorithm (GA) for a population of chromosomes (bit strings representing organism characteristics) to evolve and reproduce the fittest chromosomes. During any reproduction process, parent chromosomes perform single point cross-over, bit-flip mutation, and inversion. The fitness function selects the most fit individual, whose genes are carried forward in the evolutionary timeframe. A fitness value or score is assigned to each solution, representing the abilities of an individual to 'compete'. The individual with the optimal (or near optimal) fitness score is sought. We further define fitness as survivability. Individuals in a population compete for resources and mates, and those who cannot survive are not fit, in the evolutionary sense, so will become extinct. This process iterates over a number of generations given by user's selection. The result is a chromosome comprising the 'best' genes which have evolved to foster survival fitness through the two species evolutionary process.

In ALGAE, we consider the following aspects, such as living space, food resources, competition, behaviour patterns and preferences, and physical status. The details are discussed below:

a. Artificial Environment (AEnvironment) is defined as a search space designed in a *2D* field, a rectangular region symmetric to the center, for directional movement toward a desired object.

b. Assume *resources* exist in the AEnvironment composed of $n \times m$ grids (here we use $51 \times 51$), randomly distributed and which are renewable. Plant food is available to increase energy, located available in the exploring area.

c. *Competition* is also intrinsic in an AEnvironment. Individuals attack the other species based on Attack Ability (AA). They have a certain amount of energy which is lost by movement (Motion Loss, ML), and attack (Attack Loss, AL). Species also gain energy by consumption of food (Food Efficiency, EF). Food is assigned simulating natural law with corresponding food value and vitality. Each individual is a gene disseminator, an intelligent individual, facing a complex environment, so choosing suitable adaptive behavior is very important. Appropriate behavior ensures genetic replication and thus evolution. To achieve survival and multiplication, the species member undertakes migration, looks for food, exhibits breeding behavior. Also, in order to ensure the population's evolution, ALGAE programs in mutual attacking behavior which can eliminate the genetically inferior individual.

e. Individual behavior patterns and preferences are programmed as movement modes and action modes into their genes, as follows: 1) *ALgents* can only mate with local individuals within their 'action field'. Each individual complies with its own Action Choice (AC) as preference to choose behaviours: look for food, attack/defend, or mate. In the hypothetical *AL* world, motion characteristic emulates biological drives. 2) Motion Direction (MD) choice, according to the GA aspect of ALGAE, determines that transition motions are all caused by corresponding instinctive (genetically determined) decisions. 3) the Action Choice gene mimics biological behaviour priorities.

f. Physical status such as lifespan (SL) is also genetically determined. When a certain age is reached, or energy entropy reaches a threshold, *ALgent* dies. Individual age increases along with the generation increase, surpassing the lifespan, ending in natural death. Regarding the (biological) initial age, in order to simulate the initial population subject to the process of evolution, individual age is assigned as a random number plus the biological minimum age ($SL_{MIN}$). Similarly, the initial biological energy available is stated as $Energy = 70 + random(30)$ (maximum energy is 100) to ensure a level of individual energy consumption during the initial migration.

All behaviors above indicate that the two species compete for resources to survive. As the population evolves, the distribution of resources changes over time.

SP: The first bit, indexed as '0' shown in Table 6.1, represents the SPecies type, where each type has different preference:

- preference '1' stands for 'selfish' agent, coloured in blue;
- whereas '0' stands for 'altruistic' agent, coloured in pink.

When the food resource is renewable plant, according to user's specification, resources level is set to regrow at certain rates until the maximum limit. A height threshold is specified which indicates minimum amount reserved needed to maintain plants' healthy growth. If the height level of plants is too low, regrowing rate is significant slow which can lead to resources paucity. At renewable plants' setting, 'altruistic' *ALgent* collects food resources in an environmental way, specifically, when height level is beyond threshold, otherwise, 'altruistic' type chooses not to consume resources until they reach the threshold. On the other hand, 'selfish' *ALgent* collects resources regardless of plant's regrowing status, as long as any plant exists.

When the food resource is non-renewable honey, the resource only contains two stage: exist or finished. Both *ALgent* type '1' and '0' pursue honey in the same way to fetch it when it exists.

SL: Index 1-2 bits stand for 'Life Span'.

$Age = SL_{min} + SL$;

in the initial setting, minimim lifespan, $SL_{min}$, is set as 50. Age's range is between minimum lifespan and its value add 'SL' maximum value 4.

VF: The next 2 bits describe 'Vision Field' (VF). VF size is set as the length of radius, range from 1 to 4. This describes a sector area with certain radius length.

MD: After setting up the radius of vision field, the next 2 bits are 'Motion Direction' (MD). MD gives the moving direction angles which covers 90, 180, 270, or 360 degrees, see Figure 6.1

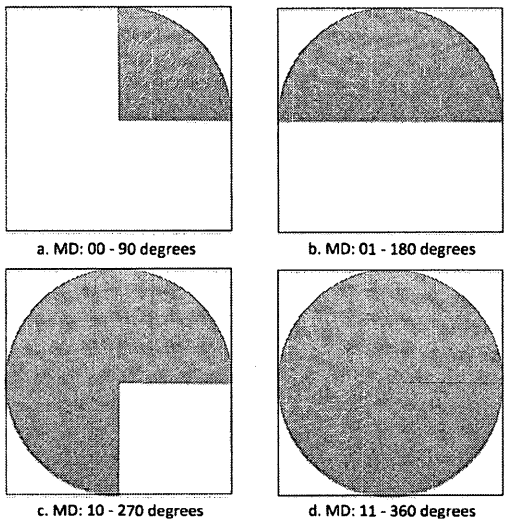


Figure 6.1: Motion direction coverage sector graph

AS: The 7-8 sequence specify individual's 'Action Speed' when moving. Each step, an agent can move the distance between 1 to 4.

ML: Motion Loss is described in bit 9, which reflects various level of energy loss during each exploration movement. Motion inevitably consumes energy, according to natural law, yet such energy consumption cannot become a decisive reason by which a species survives or perishes. When individuals move, the energy consumption will be related to and associated with 'ML' gene, that is $Loss = ML + min$. The minimum movement energy loss is given by user's specification.

AC: The next 3 bits, indexed 10 to 12, describe each agent's behaviour preference as 'Action Choice' (AC). In this action sequence, the first bit, Food Preference (FP), determines if this agent would search for food resources first when it is '0', or after interaction with other agents when it is '1'. We number the interaction choice with other agents as following: mate is 'a', and fight is 'b'. The last two bits in the AC is named 'Interaction Preference' (IP). IP gives each agent's preference of interaction with others. Specifically, '00' is choice 'a'; '01' is choice 'b'; '10' is choice 'a' then 'b'; '11' is choice 'b' then 'a'.

AA: In each behaviour interaction with other agents, agents lose various amounts of energy. During combat, the 'Attack Ability' is critical to determine the the fight result afterwards, described in 2 bits sequence from 13 to 14. The ability to attack affirms it is important whether this living thing can survive. In Nature, species have power and size; for instance, even if the tiger is injured he cannot easily be defeated by a rabbit. Here, we do not include a multitude of living things, but only two hypothetical species, who do not have the natural power which large numbers might generate. ALGAE simulates natural species, where concrete behaviour has direct correspondence with inherited genes. As in Nature, victory in conflict determines survival. The principle is explained as follows. If individual $i$ has an attack capability stronger than individual $j$, then $i$ wins. In an attack process, the energy of striking power is the most important. Individual $i$ has a striking power as computed below:

$$Attack_i = energy_i(t) + AA_i * r * 20/4,$$

where $r$ is a random float number between 0 and 1. That is, an individual's striking power is influenced by its current energy and how large of a fraction of energy boost of 20 determined by its 'Attacking Ability'.

AL: 'Attack Loss' varies from 1 to 4, given as 2 bits sequence from 15 to 16. AL gives the energy loss during a fight. After the fight, both $i$ and $j$ loss energy, but the quantity is different. In order to simulate real biological attack behavior, each individual loses a random fraction value of the 'Attack Loss' stochastically, and this individual has a further energy loss of 40, as follows:

$$Energy(t+1) = Energy(t) - AL * r - 40.$$

EF: 'Food Efficiency' is represented as the last 2 bits, which defines agent's food absorbing efficiency, scale 1 to 4:

$$Energy(t+1) = Energy(t) - e * EF,$$

where $e$ is 'energy-gain-from-food', given as basic energy provided by resources. Amount of energy each *ALgent* gained is determined by its own observation, 'EF'.

Figure 6.2 illustrates the initialized ALGAE. Two type of agents are presented: type 1 agents ('selfish') coloured in blue, and type 0 agents ('altruistic') coloured in pink. Figure 6.2(a) presents green plants as regrowing resources. Figure 6.2(b) states a non-renewable food resource (honey), represented as orange leaves.

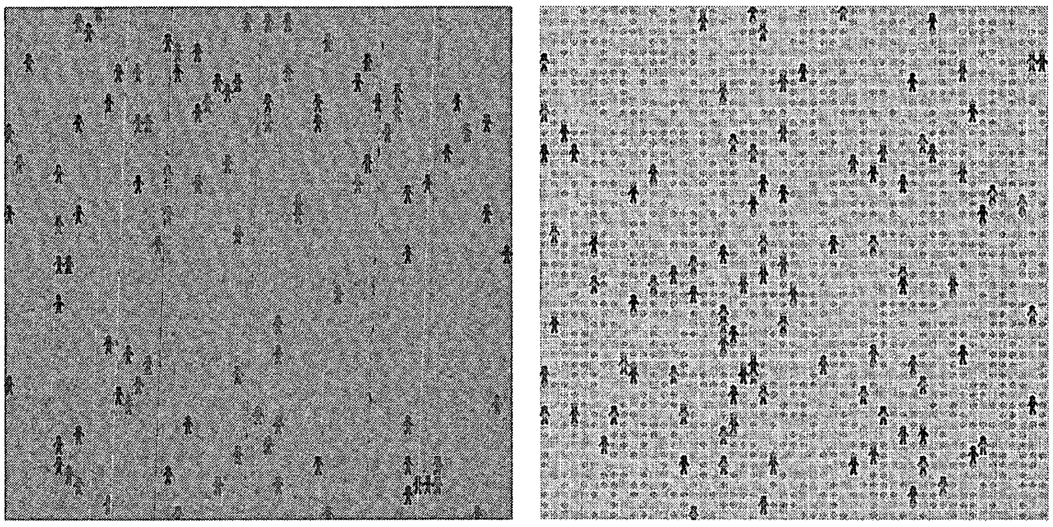

(a) Renewable plant resources (b) Non-renewable honey resources

Figure 6.2: ALGAE world

### 6.2.2 Pseudocode and Run Process

In ALGAE, the program establishes the artificial world (AWorld) environment parameters, comparable to biological evolutionary pressure. Using a graphical interface dynamic demonstration, it records the evolutionary processes for each generation (which survives). The system operation follows some basic steps and establishes parameters in relation to the environment as follows Algorithm 4.

The program iterates to mimic generational evolution over lengthy time frames. Species members experience genetic variations throughout the process, and the survivors remain to reveal which specific genes adapted. Next, we examine how these survivors' genes correlate to produce successful adaptation.

**Algorithm 4:** ALGAE

```
Initialize: AWorld environment, food resource, ALgent population;
Energy = 70 + random (30);
Age = 1;
while generation number < maxium number & both species ALgents exist do
    ALgent act;
    if first action-choice = food then
        fetch food;
        action-with-mate;
    else
        action-with-mate;
        fetch-food;
    end
    move randomly;
    Age = Age + 1;
    generation number += 1;
    death-check;
    Update: environment, food resource, ALgent population;
end
```

## 6.3 ALGAE Simulation

In this section, we describe a detailed ALGAE simulation[1], developed using Netlogo (Wilensky, 1999). The goal for this experiment is to uncover the hidden relations among AGenes by using BN to analyze the datasets of survivors' AChromosomes. This experiment has an initial run to collect the survivor genes over all generations. This is the input for BAyesian Networks ANAlysis (BANANA) (Yan and Cercone, 2010a) to analyze. The ALGAE setup and experimental results are illustrated in the following.

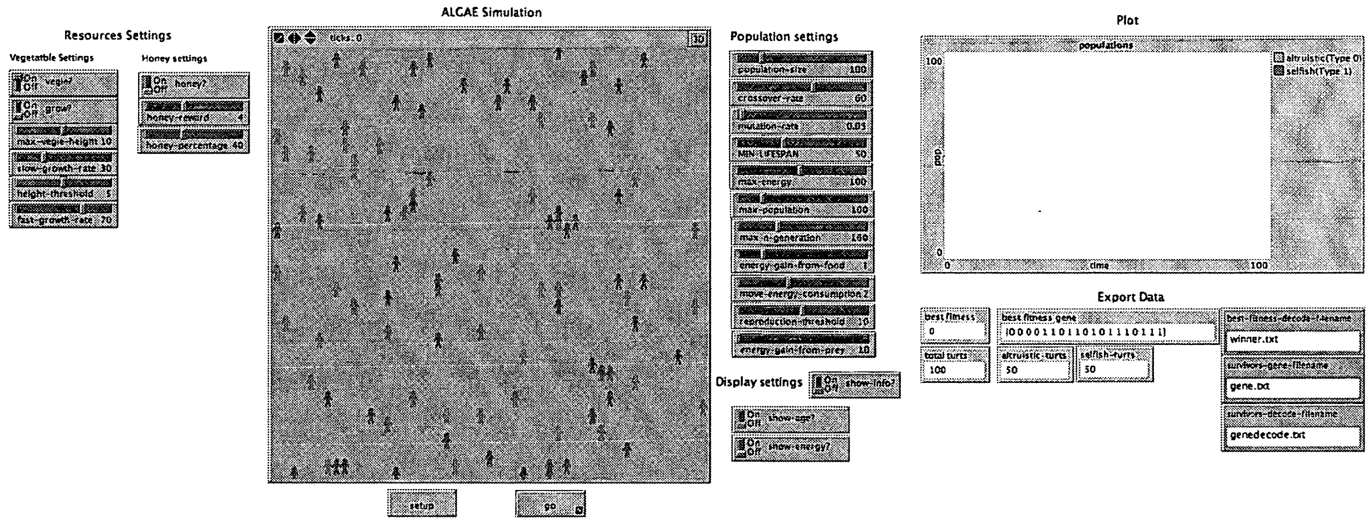
[1]ALGAE is available for download at `http://www.cse.yorku.ca/~lisayan/algae_code.nlogo`.

### 6.3.1 Experiment Setup

We first identify relevant environmental parameters which affect the species evolutionary chances: population settings and resources settings. First, define the environment AWorld size as width 51 units by height 51 units, which provides life space or living territory. Next, in the population settings, the initial species proportion indicates a population size of 100, which is a proportion of 5 percent in relation to the territory available. This setting provides ample room for migratory movements and food seeking. The minimum lifespan is set as user's specification, for example 50, which indicates that species members have some chance to act and move in their virtual world for a reasonable period of time. Also a mutation rate is set up as 0.05, and crossover rate as 60 percent. Both are carried into this simulated AWorld also. It allows for unpredictable results in gene recombination. The maximum generation number for the total run is set as 160. The initial population shows even distribution of 50 individual entities of each species, coloured red and blue, and positioned in AWorld. This positioning is purely random, and remains random for each generation run.

Last, resources settings can be specified in two groups: a) renewable plant resources; b) non-renewable honey resources. The initial setup for the ALGAE run (Figure 6.3) states the parameters for plants to grow. In Figure 6.3, for example, we give a threshold height as 5, and maximum height 10. When the plant height is beyond the threshold, plants have a fast growth rate as 70 percentage; or, plants have a slow growth rate of 30 percentage when it falls under the threshold.

In Figure 6.4, we establish the initial proportion of non-renewable food: honey. It is set as 40 percent of the living space. Each honey consumption gives an individual



On Off vegie?
On Off grow?
max-vegie-height 10
slow-growth-rate 30
height-threshold 5
fast-growth-rate 70

On Off honey?
honey-reward 4
honey-percentage 40

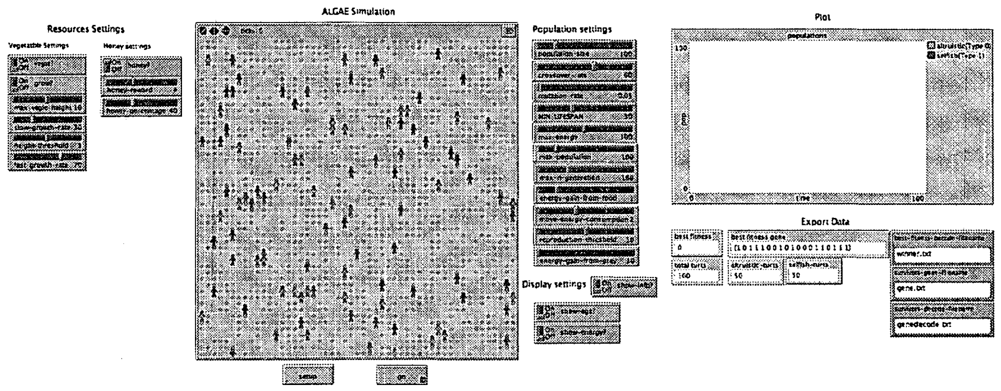
population-size 100
crossover-rate 60
mutation-rate 0.05
MIN-LIFESPAN 50
max-energy 100
max-population 100
max-n-generation 160
energy-gain-from-food 1
move-energy-consumption 2
reproduction-threshold 10
energy-gain-from-prey 10

On Off show-info?
On Off show-age?
On Off show-energy?

best fitness 0
best fitness gene [0 0 0 0 1 1 0 1 1 0 1 0 1 1 1 0 1 1 1]
total turts 100
altruistic-turts 50
selfish-turts 50
best-fitness-decode-filename winner.txt
survivors-gene-filename gene.txt
survivors-decode-filename genedecode.txt

Figure 6.3: ALGAE Interface with renewable plant resources

energy reward of 4. All types of resources allow energy level to reach the maximum level of 100.



vegie? On/Off
grow? On/Off
max-vegie-height 10
slow-growth-rate 30
height-threshold 3
fast-growth-rate 70
honey? On/Off
honey-reward 4
honey-percentage 40
population-size 100
crossover-rate 60
mutation-rate 0.05
MIN-LIFESPAN 30
max-energy 100
max-population 100
max-n-generation 100
energy-gain-from-food 2
move-energy-consumption 2
reproduction-threshold 10
energy-gain-from-prey 10
show-info? On/Off
show-ego? On/Off
show-energy? On/Off
populations: pop 0–100, time 0–100; altruistic(Type 0), selfish(Type 1)
best fitness: 0
best fitness gene: [1 0 1 1 1 0 0 1 0 1 0 0 0 1 1 0 1 1 1]
total turts: 100
altruistic-turts: 50
selfish-turts: 50
best-fitness-decode-filename: winner.txt
survivors-gene-filename: gene.txt
survivors-decode-filename: genedecode.txt

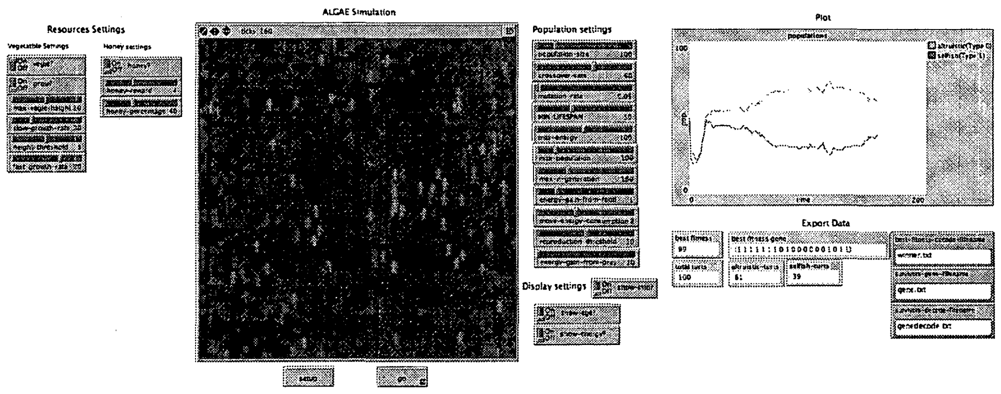
Figure 6.4: ALGAE interface with non-renewable honey resources

### 6.3.2 Experimental Results and Analysis

In this evolutionary process, two groups of living entities compete to survive. After undergoing the simulated evolutionary process, the distribution can be seen in Figure 6.5 and Figure 6.6, which presents the final stage of ALGAE simulation of two types of resources settings respectively.

In Figure 6.5, renewable plants provide resourceful energy supply for all *ALgents*. In this whole process, both type 0 and type 1 *ALgents* reach a settled amount of population while fetching resources. In the next section, we reveal the common inter-relations among all survivors' gene descriptors.

In Figure 6.6, limited resources constraints *ALgents*' survival conditions. After



population-size 100
crossover-rate 60
mutation-rate 0.01
MIN-LIFESPAN 30
max-energy 100
max-population 100
max-#-generation 150
energy-gain-from-food 1
move-energy-consumption 2
reproduction-threshold 20
energy-gain-from-prey 20

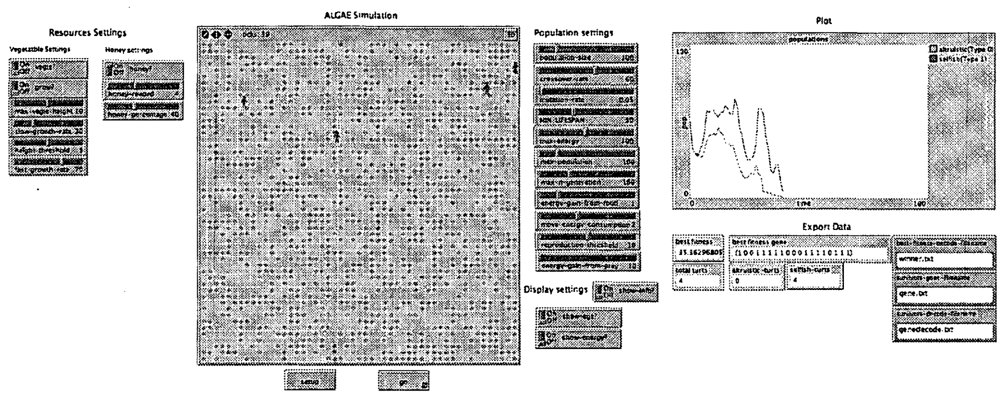
best fitness 99
best fitness gene :11111:101000001011
total turts 100
altruistic-turts 61
selfish-turts 39

Figure 6.5: ALGAE final stage in growing plants world

30 steps of exploration in the environment, population of type 1, coloured in blue, drops significantly when few resources (less then 1%) remain; whereas, population of type 0, coloured in pink, is eliminated from this evolutionary process. Grey leaves denotes food sources consumed, orange leaves denote fresh food resources. However, given the limitation of resources and random setting for behaviors, both types have equal chance to last to the end of survival. In the next section, we analyze what insights are revealed as common inter-relations among all survivors' gene descriptors in each exploration, which promote *ALgents* to survive to the very last as best fit.

### 6.3.3 Fittest Gene Discussion

In this ALGAE process, we examine the species evolution and gene expression. After running a number of experiments, we want to know which genes account for an outstanding/surviving individuals? Take the runs illustrated in Figure 6.5 and



vegis? On
grow? On
max-vegie-height 10
slow-growth-rate 30
height-threshold 3
fast-growth-rate 70
honey? On
honey-reward 4
honey-percentage 40
population-size 100
crossover-rate 60
mutation-rate 0.05
MIN-LIFESPAN 30
max-energy 100
max-population 100
max-n-generation 100
energy-gain-from-food 1
move-energy-consumption 2
reproduction-threshold 10
energy-gain-from-pray 10
show-info? On
show-age? On
show-energy? On
best fitness 35.16296805
best fitness gene [1 0 0 1 1 1 1 0 0 0 1 1 1 0 1 1 1]
total turts 4
altruistic-turts 0
selfish-turts 4
best-fitness-decode-filename winner.txt
survivors-gene-filename gene.txt
survivors-decode-filename genedecode.txt

Figure 6.6: ALGAE final stage in honey world

Figure 6.6 as examples. The final positions of surviving *ALgents* shown are quite different. The evolutionary assumption is that less fit *ALgents* are, less prone to survival in a harsh environment with other competitor species, and disappeared from the gene pool. We conclude the following characteristics of the optimal gene/individual that indicate the fittest survival:

- Lifespan is large in the fittest genes.
- Vision Field of vision is maximized in the fittest genes.
- Action Choice suggests that food resources searching first to increase the energy level before consumption promotes survival. So the fittest genes allow individuals to approach the same type of entity and obtain food resources.
- Attack capability is high in fittest gene.
- Low attack energy consumption promotes fitness.

We conducted ten different ALGAE trials and obtained ten separate datasets. It is shown that these ten different experimental results produced the ten different best genes over the same initial resource and population settings, see Table 6.2. This table shows that the results differ from trial to trial. Because the genetic

| Trial NO. | Survivor Type | Best Genes |
|---|---|---|
| 1 | 1 | 1 1 1 1 0 1 1 0 1 0 0 0 0 0 1 1 0 0 1 |
| 2 | 0 | 0 0 1 1 0 1 1 1 0 0 0 0 0 1 0 1 1 1 0 |
| 3 | 0 | 0 1 1 1 0 0 1 0 0 0 0 1 0 1 1 1 1 1 1 |
| 4 | 1 | 1 1 0 1 1 1 0 1 0 0 1 0 0 0 1 1 0 1 0 |
| 5 | 1 | 1 1 0 1 1 1 1 1 1 0 1 0 1 0 1 0 0 1 1 |
| 6 | 0 | 0 1 1 0 1 1 1 0 1 0 1 0 1 1 1 0 0 1 1 |
| 7 | 0 | 0 0 1 1 0 0 0 0 0 0 1 0 1 0 0 0 1 1 1 |
| 8 | 0 | 0 0 0 1 1 1 1 1 1 1 0 0 1 1 0 0 1 1 1 |
| 9 | 1 | 1 0 0 1 1 0 1 1 0 0 0 0 0 0 0 0 0 1 1 |
| 10 | 1 | 1 0 0 1 1 1 1 0 0 0 0 1 1 0 1 1 0 1 1 |

Table 6.2: 10 Trials Fitness Gene Profiles Datasets Logs

algorithm itself is a stochastic algorithm, any result is interesting because it reflects the pressures on genotypes over time, under variable conditions. An interesting question arises. Although each trial produced a 'best' gene, why are they different from each other? Another question is how and what can we learn from each of the 'best' genes? A third question is what is the similarity among the ten trials which could bring forward the best individual chromosome as a surviving remnant of evolution?

Table 6.2 shows variation in the composition of each best gene. Each gene has entirely different and unique attributes. ALGAE randomizes the chromosomes for each run, as well as certain environmental factors such as population distribution in relation to resources. What is interesting is that not every gene has optimal alleles or gene bits, so it is clear that survival fitness is not a question of having the best gene or chromosomes, or having proximity to resources. Rather it is the combination of genes that remain in the gene pool of a generation which is most significant to survival and adaptation. Evolutionary pressure, that is, the degree of harshness and difficulty in the struggle to survive, can be severe, such as, Figure 6.6 shows that the total number of final stage survivors for that run is only two. Nevertheless, ALGAE identifies the best chromosome for those who survive. One type of *ALgent* tends to dominate one run, and this reflects the genetic fitness of the population in terms of its genetic features. The best genes are carried forward, and this is consistent with a Darwinian concept. In the next section, we present Bayesian Network learning from survivors' chromosomes to illustrate what we can learn about gene contributions to survival from ALGAE.

## 6.4 Bayesian Networks structure representation of Evolutionary Process

We create a topological structure BN to see the implicit connections among AGenes. ALGAE is a dynamic process based on GA, so the 'survivors' genes' change with each run. Our interest lays on why those survivors have proven to possess the fittest/best genes in the AWorld. The underlying reasons can be discovered using a BN.

Given a set of variables and a dataset composed of all these variables' values, the problem is how to build a structure to present the connections among the variables. This structure learning process needs to select the arcs between them and estimate the parameters. Developing a structure gives a visual presentation to understand what underlies the knowledge or what attributes are correlated. However, to include all the information from the data into the structure, yet to keep the structure simple and condensed with only critical information, is going to result in a trade-off. The two main approaches are used to learn structure in BNs: the constraint-based and the score-based approaches. In this section, we use the E-algorithm (Yan, 2003) to construct a Bayesian Network representation for ALGAE data.

### 6.4.1 Independence Test

Given the sample data of ALGAE, we want to know whether there is an association between the 'Vision Field' levels and the type of Species, or both types of species in attack motion differ in their energy loss. This type of question is the $\chi^2$-test designed to solve (Pearson, 1900). This section, we present a simple example to illustrate how to conduct $\chi^2$-test to identify the independence between characteristics in ALGAE.

We take 'Species type' and 'Motion Direction' two variables as an example. This $2 \times 4$ contingency table (see Table 6.3) presents the observed frequencies of MD value and Species types from one trial's ALGAE data.

| | MD'1' | MD'2' | MD'3' | MD'4' | Total |
|---|---|---|---|---|---|
| Type 0 | 389 | 50 | 2158 | 5123 | 7720 |
| Type 1 | 561 | 455 | 300 | 5773 | 7089 |
| Species num | 950 | 505 | 2458 | 10896 | 14809 |

Table 6.3: Observed contingency table

Now, let us define our null hypothesis as follows: The number of species type '0' and '1' in ALGAE survived due to their 'MD' value choice range 1 to 4 is independent of their species type.

According to the null hypothesis, we can calculate the expected $2 \times 4$ contingency table (see Table 6.4) presents as follows:

| | MD'1' | MD'2' | MD'3' | MD'4' | Total |
|---|---|---|---|---|---|
| Type 0 | $E_1$ | $E_2$ | $E_3$ | $E_4$ | 7720 |
| Type 1 | $E_5$ | $E_6$ | $E_7$ | $E_8$ | 7089 |
| Species num | 950 | 505 | 2458 | 10896 | 14809 |

Table 6.4: Expected contingency table

Under the hypothesis that two variables' classifications are independent, that is we would expect the proportion of type '0' species has MD '1' is equal to the same type'0' with MD '2', '3', and '4'. So the following equations hold:

$$\begin{aligned}\frac{E_1}{950} &= \frac{E_2}{505} = \frac{E_3}{2458} = \frac{E_4}{10896} = \frac{7720}{14809} = 0.5213, \\ E_1 + E_5 &= 950, \\ E_2 + E_6 &= 505, \\ E_3 + E_7 &= 2458, \\ E_4 + E_8 &= 10896.\end{aligned} \tag{6.1}$$

Solving the Eq. 6.1 for values of $E_i$,$(i = 1, \ldots, 8)$, we obtain Table 6.5.

| | MD'1' | MD'2' | MD'3' | MD'4' | Total |
|---|---|---|---|---|---|
| Type 0 | 495 | 263 | 1281 | 5680 | 7720 |
| Type 1 | 455 | 242 | 1177 | 5216 | 7089 |
| Species num | 950 | 505 | 2458 | 10896 | 14809 |

Table 6.5: Expected frequencies on the assumption of independence

To compare the observed and expected frequencies, we calculate the $\chi^2$ value using the following equation:

$$\chi^2 = \sum \frac{(O_i - E_i)^2}{E_i} = 1775.35,$$

where $O_i$ $(i = 1, \ldots, 8)$ is the observed frequencies in Table 6.3. The degree of freedom $(df)$ of two variables in contingency table with $r$ rows and $c$ columns is computed as:

$$df = (r-1) * (c-1).$$

According to Species type and Motion Direction these two variables, the degree of freedom is 3. To assess the significance of the $\chi^2$ value 1775.35, we check the $\chi^2$ table with $df = 3$. Given the confidence value $\alpha = 0.001$, the corresponding value is 16.2662. Our $\chi^2$ value 1775.35 is much greater than 16.2662, which suggests that in less than 0.001 level of probability the null hypothesis is true. Thus, we reject the null hypothesis and conclude that SP and MD is not independent.

In the next section, we give a brief introduction to the Bayesian Network learning E-algorithm, which first conducts constraint independence tests using $\chi^2$ test. Thereafter a MDL scoring method is performed to optimize the graphical structure.

### 6.4.2 Bayesian Network Learning Algorithm: E-algorithm

The key aspect of the structure learning problem is to construct a topology network from fully observable variables. This section provides an improved BN learning algorithm: the E-algorithm first proposed in Yan, 2003 undertaken in relation to improving learning Bayesian Networks. The E-algorithm has been adapted to business applications, e.g., suggested business strategies that a business should choose, as reported in Ji et al., 2004; Yan et al., 2007. In Ji et al., 2005, the accuracy and efficiency of the E-algorithm has been established by comparing execution time of the E-algorithm against two established algorithms: I-MDL, I-B&B-MDL.

The following section briefly illustrates the E-algorithm. The E-algorithm combines both a constraint-based approach and a score-based approach, jointly applying the conditional independence (CI) test and minimum description length (MDL) metric search. First, a small number of dependence tests are used to reduce the calculation complexity and to restrict the feasible search space. Second, the improved MDL metric search boosts both time performance and efficiency of BN learning.

a. Constraint-based approach:

The constraint-based approach poses learning as a constraint satisfaction problem, which is more intuitive and follows the definition of a BN more closely. This method performs tests of CI on the data, and search for a network that is consistent with the observed dependencies and independencies (Heckerman, 1995; Pearl, 1986; Cooper and Herskovits, 1991).

As a typical metric, CI is based on information flow in information theory, thus the mutual information of two variables $X, Y$ is defined as Eq. 6.2:

$$I(X,Y) = \sum_{x,y} P(x,y) \log \frac{P(x,y)}{P(x)P(y)} = \sum P(x,y) I(x,y), \tag{6.2}$$

and conditional mutual information is defined as Eq. 6.3:

$$I(X,Y|C) = \sum_{x,y,c} P(x,y) \log \frac{P(x,y|c)}{P(x|c)P(y|c)} = \sum P(x,y) I(x,y|c), \tag{6.3}$$

where $C$ is a conditional set of nodes, $P$ denotes the instance frequency (probability) observed from a sample dataset. The mutual information can show if the two variables are dependent and if so, how close is their relationship. Hence, when $I(X,Y|C)$ is smaller than a certain threshold value $\varepsilon$, we can say that $X$ is independent of $Y$ given the set $C$, or else $X$ is independent of $Y$ if $C$ is the empty set. So we can deduce if there is a connection between two variables in view of the mutual information.

Here, the threshold value $\varepsilon$ can be given based on expert knowledge, alternatively, there is another similar method, the $\chi^2$ test (Qiang et al., 2002), which is based on a statistical hypothesis to estimate a connection between two variables. Given a degree of confidence $\sigma$, a connection between two variables can be deduced by $t$-value(threshold) which is generated by $\chi^2$ test. In our case, if the connection value $I$ is greater than or equal to $t$-value, then $X$ is independent of $Y$, which implies that there is no direct connection between these two variables. Otherwise, if the connection value $I$ is less than $t$-value, then $X$ is dependent of $Y$, which means that an arc connects $X$ and $Y$ in the resultant network.

b. Score-based approach:

The score-based method is to define a score function that evaluates how well the dependencies in a structure match the data, and search for the simplest structure which also maximizes the score. In the set of feasible solutions, a recursive search can be used to find an optimal structure that satisfies the criteria. A scoring function commonly used to learn a BN is the log-likelihood, which is simply the log of the likelihood function, that is, Eq. 6.4:

$$\begin{aligned} l(X|g,\theta_g) &= \log \prod_{i=1}^{n} p(X_i|\Pi_i, g, \theta_g) \qquad (6.4) \\ &= \sum_{i=1}^{n} \log p(X_i|\Pi_i, g, \theta_g), \qquad (6.5) \end{aligned}$$

where $\theta_g$ is a parameter of the structure $g$ in a dataset $X$ which also represents all the variables, and $\Pi_i$ is the parents set of node $X_i$. The log-likelihood is easier to analyze than the likelihood, because the logarithm turns all the products into sums. Therefore, according to Eq. 6.4, we have Eq. 6.5.

There are a couple of important points to note about the log-likelihood. The log-likelihood increases linearly with the size of a dataset. The higher scoring networks are those where a node and its parents are highly correlated. The network structure that maximizes the likelihood is often fully connected. Adding a node into a network always increases the log-likelihood. This deficiency of the log-likelihood score is not desired. Thus, a score that makes it harder to add arcs is necessary. In other words, we would like to penalize structures with too many arcs. One possible formulation of this idea is called the MDL score (Suzuki, 1993). The MDL score is a compromise between fit to data and model complexity. Adding a variable as a parent causes the log-likelihood term to increase, but so does the penalty. There will be an arc addition if its increase to the likelihood is worth it. The detailed MDL scoring function will be explained in the following section.

The space of Bayesian networks is a combinatorial space, consisting of an exceeding large number of structures. This problem is combinatorially complex; both approaches have their limitations. The general idea of the E-algorithm is quite straightforward. First, the constraint based tests are performed to get an initial network to consider, which reduces the search space. Then, a metric score function is used to find a matching structure which has the best motivated score.

The E-algorithm considers the BN structure learning as a connection elimination process starting from a fully connected graph $G_0$ among all the variables. It features three elements: 1) order-0 independence tests are used to delete weak connections and obtain a graph $G_1$; 2) order-1 conditional independence $\chi^2$ tests, which only appears in the "$\Delta$-form", are conducted and simplify $G_1$ to $G_2$. The definition of "$\Delta$-form" is as follows:

Given an arc between two nodes $X_i$ and $X_j$ in BN structure $g$, if there is another path connecting them which only includes one extra node $X_k$ , we call this acyclic subgraph an order-1 "$\Delta$-form" (Figure 6.7); if this path includes two extra nodes, we call this subgraph order-2 "$\Delta$-form".

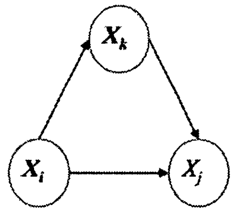


Figure 6.7: $\Delta$-form

This process reduces the search space for scoring possible structures. 3) The E-algorithm then directly evaluates the structure MDL scores. Eq. 6.6 defines a score that evaluates how well the dependencies in a structure match the data, and we can search for a structure that maximizes the score (Qiang et al., 2002; Suzuki, 1993).

$$\begin{aligned} MDL(g, X) &= \sum_{i=1}^{n} H(i, g, X) + \frac{k(g)}{2} \log n, && (6.6) \\ H(i, g, X) &= \sum_{i=1}^{n} -p(X_i \log p(X_i | \Pi_i, g), && (6.7) \end{aligned}$$

where MDL$(g, X)$ is the description length of graph $g$ for overall data variables $X$, $H(i, g, X)$ describes the empirical entropy of each node $i$ and its *sum* stands for the overall structure fitness to the observed data, and $k(g)$ is the description for the complexity of nodes (each node $i$ has the number $v_i$ values, $j$ is a parent node of $i$, $j = [1, i-1]$), as follows:

$$k(g) = \sum_{i=1}^{n} k(i, g), \tag{6.8}$$

$$k(i, g) = (v_i - 1) \sum_{j=1}^{i-1} v_j. \tag{6.9}$$

As can be seen, the problem of learning BN becomes a search problem for a structure with MDL metric. A recursive search is applied to the MDL-based search procedure. This search examines all possible local changes in the set of parent nodes, revealing that the cost of those evaluations is too high for massive datasets.

The E-algorithm, described in Algorithm 5, takes numeric data input from AL-GAE's output of the suvivors' genetic data. The learning process is to learn $n \times n$ binary matrix $G$, to represent connections among all the nodes, where '1' in the coresponding entry indicates an edge between these two nodes, and '0' indicates no connection. A fully connected graph is generated as an initial step, where all the corresponding entries are given value '1' in the matrix for graph $G$. Then, compute the conditional mutual information in light of Eq. 6.3 to remove any invalid edge by $\chi^2$-test according to a given degree of confidence level $\sigma = 0.001$. For each node $X_i$, sort its candidate parents $\Pi_i$ nodes as ascending ordering by their mutual information; then search to find a $\Pi_i$ with the minimum MDL score and confirm the local optimized structure of $X_i$. Update all the nodes, until the minimum MDL score is reached for the whole structure, and output graph matrix $G$. We test the E-algorithm in a benchmark ALARM Network dataset, and it demonstrates that it is efficient, valid and produces high accuracy for learning BN structures (Yan, 2003; Ji et al., 2005; Yan and Cercone, 2010a).

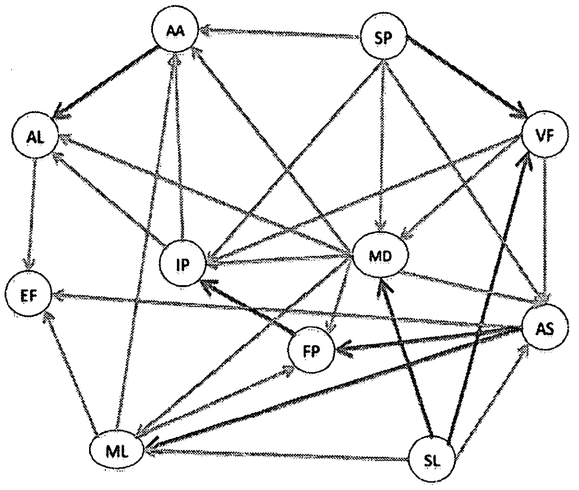
### 6.4.3 Result Discussion

We reimplement a Bayesian Network structure learning software BANANA in Matlab, (see Appendix B), where a graphical representation is constructed using the E-algorithm to reveal the inter-connections among data. We use randomly chosen 10 trial datasets from ALGAE as input, with total size of 68504 data entries. BANANA provides a graphical representation in Figure 6.8 to reveal the inter-connections among the genetic descriptors among these 10 trial simulations in ALGAE. The program runs in Matlab on Mac OS X, with Processor 2.8GHz Intel Core i7, Memory 8GB 1067MHz. The runtime for a single run is approximately 28 minutes.

The BN structure indicates the formula for the rule of survival: relationship (edges) between the genetic characteristics (nodes) will determine who lives and who does not. The ability to act rapidly (AS), obtain food (FP) or reproduce (IP) are related to ability and skill to move (MD). Species type (SP), lifespan (SL) and the visual field in the environment (VF) determines the motion direction (MD). In a competitive world, the ability to defend and resist annihilation is represented by the ability to attack (AA), and limited by entropy of energy (AL). This is tied to the power to move and act (MD, AS) and limited by loss of motion (ML). Energy can be replenished by food resources (EF), which have a scaled level of uptake efficiency. All these abilities are genetically determined in the ALGAE.

**Algorithm 5:** E-ALGORITHM

```
Input: numeric data from ALGAE;
Output: structure G matrix;
begin
    Initialize:
    G: fully connected graph matrix;
    $confidencelevel : \sigma = 0.001$;
    Step 1: mutual information test;
    for each node i do
        for each i's neighbor node j do
            if mutual-info(i, j) = true then
                remove edge i,j;
            else
                sort parent list $\Pi_i$ according to mutual-info ascendingly ;
            end
        end
    end
    Step 2: CI test, remove redundant edges;
    for each node i do
        for each i's neighbor node j do
            if CI-test(i, j, $\sigma$) = true then
                remove edge i,j;
            end
        end
    end
    Step 3: MDL test, remove redundant edges;
    for each node i do
        for each i's neighbor node j do
            if MDL-score(i, j) = true then
                remove edge i,j;
            end
        end
    end
end
```



(1) SP: SPecies type; (2) SL: Life Span; (3) VF: Vision Field;
(4) MD: Motion Direction; (5) AS: Action Speed;
(6) ML : Motion Loss; (7) FP: Food Preference; (8) IP: Interaction Preference;
(9) AA: Attack Ability; (10) AL : Attack Loss; (11) EF : Food Efficiency;

Figure 6.8: BN structure of ALGAE genetic characteristics

Chance primarily determines which genes are present at the start of a run, but it is evolutionary fitness which determines actual survival. BANANA reveals the hidden structure behind this fitness of successful genotypes. In Figure 6.8, the seven edge connections in red are the core structure for this ALGAE setting. It shows that motion and action are most important in that they impact other abilities that are inheritable and promote evolutionary fitness. Of additional importance are species type and lifespan (SP, SL). These would play a role for any organism, so in the AWorld they are to be expected to be fundamental. What is notable in this experiment is that these characteristics seem to be both necessary and sufficient to ensure fitness. Under the given constraints, these genes emerge repeatly until dominance of one species occurs. The combination of such genes, revealed by Bayesian analysis, give us insight into evolutionary processes.

## 6.5 Concluding Remarks

In this chapter, we demonstrate the structure of inter-connections among agents strategies in a competitive environment. First, we develop a competitive multiagent interaction platform: ALGAE. The process of simulated environment in ALGAE provides us to observe generations of genes evolving in an accelerated period. ALGAE also allows us to foresee the genetic recombination process and provide us insights into variations among group behaviors. Second, we develop BANANA using the E-algorithm to extract the Bayesian Network structure representation among agents strategies. BN reveals the hidden structure of relationships in *ALgent*s behaviours, and provides us a visual representation of them.

The experiment demonstrates that the reimplementation of BANANA in Matlab is robust. Testing BANANA in a real example enhanced the performance such that it learns effectively and robustly on a number of variables and connections. Applying the E-algorithm in a complete domain shows that it can learn the influence of factors to build policies in this complex environment and application.

The principles of how a survivor adapts in evolution from either optimal ancestors or weak ones, and at what point the evolutionary process can be tilted to favor certain adaptive ones, needs further exploration. How we develop a rational decision-making component for each *ALgent*, rather than mere random choice, needs further research. The knowledge revealed from Bayesian analysis need to be provided to each individual to allow actions to be chosen intelligently according to various factors in the environment. Applying this knowledge (constraints) which exists among various factors in the multiagent environment, allows an individual *ALgent* to choose an optimal action and obtain a gain of equilibrium. Thus, ALGAE and BANANA have been shown to be useful applications to extend our understanding of MAL, where dependency exists among multiple factors which influence agent strategies. These applications suggest further research into artificial intelligence in terms of heritability and evolutionary processes.

Chapter 7

# Conclusions and Future Research

Our research examines the following questions:

> How can an agent perform robustly in the various types of multiagent environments, so that each agent can efficiently observe other agents behaviours, and learn from its observations in order to act (or adapt) effectively in the complex non-stationary environment? What is the macro-level phenomenon of the whole system, and how can this understanding of the phenomenon improve individual performance?

Ultimately, through a learning period and a series of actions, agents can achieve top-ranked performance.

In this dissertation, we present the importance of analyzing the structural properties in multiagent problems. In a non-stationary multiagent environment, each agent adapts to proceed towards its target. Each micro-level step in time may present a different learning problem which needs to be addressed. However, structural properties constitute a holistic phenomenon along with the rational strategies of all players. In this chapter, we summarize our contributions and discuss some future research directions.

## 7.1 Contributions

We present the importance of analyzing the structural properties in multiagent problems. Multiagent research derives from two perspectives: learning and systems. In one aspect, multiagent learning research focuses on learning from an individual agent's past experience or modeling other agents' behaviors to improve performance. In the other aspect, research on multiagent system addresses particular problems from a system perspective, with more focus on a number of agents' interactions, and provides optimal solutions. However, in a non-stationary multiagent environment, each agent adapts to proceed towards its target. Each micro-level step in time may present a different learning problem which needs to be addressed. In this non-stationary environment, a holistic phenomenon forms along with the rational strategies of all players; we define this phenomenon as structural properties. In this dissertation, we present methods to extract some structural properties in multiagent problems.

A multiagent environment can be classified as self-interested, cooperative, or competitive according to agents' goal. Here, a self-interested environment differs from a competitive, where all agents are not competing with others as a general-sum game as in a competitive environment. Thus, we examine the structure from these three general multiagent environments: self-interested random graphical game playing, distributed cooperative team playing, and competitive group competition. In each scenario, we analyze the structure in each environmental setting and demonstrate a structure learned as a comprehensive representation: structure of players' action influence, structure of constraints in teamwork communication, and structure of inter-connections among strategies. This structure represents macro-level knowledge arising in a multiagent system, and provides critical, holistic information for each problem domain.

There are four contributions in our research.

First, we present the importance of structure, a new yet vital point to tackle the problem of how agents perform robustly in various multiagent environments. We focus on understanding the common characteristics in multi-player games, more specifically, understanding the structure formed in a large number of agents in a multiagent system. The structure reveals characteristics in a multiagent system, which provides critical information for solving multiagent system problems.

Second, after revealing the characteristics which exist within multi-player games, we further explore the structural connection exerting mutual influence between players' choices of actions in game playing. Much multiagent system and learning research has been done from both machine learning and game theoretic perspectives. However, characterizing a multiagent system as a multi-player game, little research on how to abstract structure among players' actions has been conducted. In Chapter 4, we provide a novel structure-learning algorithm, MDRLSA, to extract the action connections from graphical games. Knowing the influence between players' choice of action provides a compact representation for player's utility function, as well as reducing the search space for each player's learning process. MDRLSA can learn a good representation for games and MAS where there are sparse strong influences in individual player payoff.

Third, we analyze the importance of the structure of constraints in solving distributed team learning problem. We present the SE-Adaptive search algorithm and demonstrate in its application: the mobile ad hoc network domain. Communication among agents through connections in a network is a critical aspect in such a multiagent system. In a cooperative environment, agents communicate with neighbours and achieve the goal of higher overall payoff. When exclusively acquiring information from neighbours, the time consumption increases exponentially to the scale of network inter-connection complexity. Our research exploits how to utilize the structure information while performing can improve the optimal exploration strategy in a cooperative environment, so that the time consumption and overall payoff achievement is balanced. In MANET, a group of cooperative agents explore in a distributed manner within limited time steps in a given environment to maximize the overall total payoff, with incomplete information of strategy payoff. In this setting, each agent explores to maximize the overall payoff, and moves toward equilibrium. However, how to use a lower level of optimization among agents to reach a higher overall payoff with limited time steps is a critical issue between the trade-off in exploration and exploitation. Our experiments demonstrate that, in a densely connected network, only a single agent's optimization can achieve 98% of a two-agent optimization, although the difference of overall payoff is achieved by higher level optimization with a large trade-off in time consumption. Thus, knowing the structure of an agent's connections, it is helpful to choose an optimal strategy for overall performance within the time constraints.

Last, we demonstrate the structure of inter-connections among agents strategies in a competitive environment. We develop a competitive multiagent interaction platform: ALGAE. The process of simulating agent interaction within ALGAE provides us a way to observe generations of genes evolving in an accelerated period. ALGAE also allows us to foresee the genetic recombination process and provide us insights into variations among group behaviors. Then, we develop BANANA using the E-algorithm to extract the Bayesian Network structure representation among agents strategies. BN reveals the hidden structure of relationships in *ALgents* behaviours, and provides us a visual representation of them. This research presents the insight that the inter-connection among the characteristics of those agents with the best behavioural strategies to survive, where dependency exists among multiple factors which influence agent strategies.

## 7.2 A Unified Multiagent Framework

This section summarizes the insights gained from examing structural properties and evolutionary computing. We propose a novel multiagent environmental description, and a system process to describe the interaction among agents. This design differs from the description in Chapter 2. This new unified framework includes more functional features for each agent to perform adaptively in dynamic environments with guidance from macro-level influence of structures, as well as micro-level individual learning and modeling. The macro-level influence forms as a holistic phenomenon of mutual influence, constraints or strategies, while all the players perform intelligently in this environment. These structural connections can play crucial role where the collective intelligent features appear.

**Framework Description**

Given an environment $E$, $n$ number of agents $Ag = Ag_1, Ag_2, ..., Ag_n, (n \geq 1)$

act towards their goals $G_1, ..., G_n$. In this process, each agent $Ag_i$, $i = \{1, 2, ..., n\}$, explores $E$ and learns through its experience. In order to achieve its individual goal $G_i$, agent $Ag_i$ need to learn how to co-adapt with other agents, while its environment $E$ changes over time. Under the dynamic environment, with unknown factors arising from other agents' learning process, we give the following formal description for this learning process in such a dynamic environment:

- $Ag_i$ is represented as an entity with a set of intrinsic information, called $A$chromosome:
  - rules;
  - experience;
  - preference;
  - self-stage: {position/location, wealth, energy, constraints}

  This list of code of $A$chromosome is a compact stage for encoding information, which is a carrier for an agent to store and retrieve the learned knowledge for its planned action. We adopt this idea of using coded information (such as a chromosome) to represent the knowledge of $Ag_i$. However, the structure does not have to be a list, but could be a network structure with these properties as nodes.

- Decision-making component $DM_i$:
  - priority (ranking factors);
  - rewards comparison;
  - future influence / impact;

– knowledge known;

$$DM_i = w_c * C_i + w_f * F_i + w_p * P_i. \tag{7.1}$$

This equation refers that the decision making $DM_i$ for agent $Ag_i$ depends on a weighted combination $w$ of its current stage $C_i$, future influence/impact $F_i$ and its past experience (or knowledge known) $P_i$. Current stage $C$ is composed of action priorities and reward estimate.

Other sub-components compute a Nash equilibrium decision, for example:

– minimax-Q;

– coalition modeling.

- $E = \{e_1, e_2, ..., e_m\}, e_i = \{\|e_i\|, label_i\}$, $\|e_i\|$ is the content information described for the $i^{th}$ environment factor, and $label_i$ refers to this factor $e_i$ is hidden ($label_i = 0$), or is revealed ($label_i = 1$). (The initial $E_i$, randomly set a binary value for each label over the time steps.)

- Stage of agents at time $t$: $S = \{s_1^t, s_2^t, .., s_n^t\}$: each stage $s_i^t = \{i, k_i^t, dm_i^t, ...,$ $state_i^t\}$, $k_i^t$ is the knowledge held by agent $i$ at time $t$; $dm_i^t$ is the decision making component of agent $i$ at time $t$.

- Goal $G = \{g_i^t\}$;

Compared to game theory, this description is a learning process, not only solving a game. What there is to learn is a decision-making function which combines the learned knowledge, current rewards, and future estimate. This new framework differs from the common description, and operates differently in a multiagent learning environment. We quantify these learning dynamics in a multiagent learning problem. This non-stationary learning multiagent system can be seen as an 'open-close' system, see Figure 7.1.

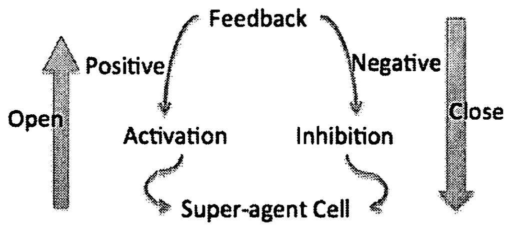


Figure 7.1: 'Open-Close' system diagram

In future research, we will explore how to utilize the holistic structural information to improve problem solving in multiagent systems.

## 7.3 Future Research

As early as 1951, fictitious play as the first learning algorithm was proposed to compute equilibria in games, and there have been numerous proposals since, regarding learning techniques in stochastic games. The MAL and MAS research has produced some inspiring results, yet, it is important to examine the foundations of MAL, and consider some relevant questions. What question exactly is MAL addressing? What is the goal for MAS? What is there to learn in stochastic games? What is the optimal design for MAS? What are the criteria by which to measure answers to these questions? How can we evaluate the success of learning rules and a system?

We further ask: do the agents know the stochastic game, including the stage game and the transition probability? More specifically, the information regarding the following: stochastic stages, transition probabilities, specific actions at each stage, actions available according to the agents, transparent (or not) for all the agents stages, action/strategies, rewards, and so on. These all are rather important factors in the whole process of agent learning. In general, this learning process can be classified as known or unknown games, observable, partially observable, or unobservable play. Furthermore, in broader settings, there is more to learn, including but not limited to learning opponents' strategies or developing the agent's own strategy for proceeding effectively against competitors.

In a multiagent system, the agents can be controlled in a centralized or decentralized manner. Is there a diagram suitable for a moment-to-moment situation? What is the tipping point for this shift from one state to another? Do the agents communicate with all their neighbors as necessary? How do we design a self-organized multiagent system to perform tasks? And how does an observed or learned macro-level phenomenon provide knowledge for micro-level individual agent learning?

In the literature, for the known, fully observable games, there are two aspects to learn in this restricted setting. One is that an agent learns the opponents' strategies as a model, so the agent can devise a best (or at least "good") response, (also known as "model-based" learning), for example, fictitious play (Brown, 1951). The other one is that an agent can learn a strategy of its own which does well against the opponents, without explicitly learning the opponents' strategies, (also known as "model-free" learning), for example, Q-learning (Watkins, 1989).

Open issues for multiagent learning research remain. Multiple agents act jointly in a common environment to achieve their own agenda, through interaction, either cooperatively or in competitive with one another. Issues of scalability, adaptive dynamics, and communication require further exploration.

### 7.3.1 Scalability

Scalability is an endemic problem for multiagent learning and multiagent systems. Multiagent systems involve multiple agent behaviors to achieve a common goal. Thus, the search space can grow exponentially according to the number of agents and the complexity of agent behavior. The evaluation criteria for learning methods should be standardized with respect to their scalability. In a general-sum learning process, especially with partially observed stochastic games, research usually involves studies in two-agent scenarios, with two or three actions for each agent. When scaled up to include more agents, current multiagent learning methods are unlikely to work in practice. A new multiagent framework will be required to incorporate diverse multiagent learning techniques with multiagent system designs.

### 7.3.2 Adaptive Dynamics

Due to the small changes caused by agents, multiagent systems can result in unpredictable global, emergent effects. How does an agent in the system recognize this phenomenon and proceed to discover an optimal strategy with the presence of such emergent effects? In a particular task, a holistic perspective of the environment should be learned. This holistic information can be stationary, given the particular problem settings. For example, the structural connections among the players can deviate how a game plays significantly. Providing such macro-level system-specific knowledge for agents can give important guidance to enable one individual to make more accurate predictions in this adaptive dynamic environment.

## 7.3.3 Communication

Communication is a principal means to effectively and immediately improve performance, and help agents accomplish their tasks. However, it can markedly increase the computation within the exploration space. The interaction can help complete tasks through passing or sharing information. However, it can also increase the complexity rapidly, in proportion to the number of agents and their idiosyncratic behaviors.

To date, much research on multiagent communication has been conducted from two perspectives: direct communication and indirect communication. Direct communication is a way for an agent to inform other agents about past experience, which can effectively improve team performance; methods include blackboards (posting and modifying information), and messages. Notably, reinforcement learning methods have presumed that the agents have access to a joint policy table to which each agent can contribute. Another perspective, indirect communication uses a third party, such as location or direction marking in the environment to pass information to others. Most indirect communication is inspired from social insects, such as ants, who utilize pheromones to mark trails, and bees send waggle dancing signals to lead others. One agent broadcasts the information in the environment, and the others can utilize and exploit it.

Yet, in a multiagent system, (just like any social system), communication is restricted by the environment. Some researchers claim that unrestricted communication in effect brings the multiagent system back to a single-agent system (Stone and Veloso, 2000). Thus, how to define the level of communication among agents and allow agents to communicate according to adaptation in the environment is still a open question which needs to be addressed.

### 7.3.4 Evaluation

In multiple agent interaction, each agent can constrain, adapt, evolve in the environment together with other agents; this diagram is not yet fully defined or understood in game theory, and brings in unknown complexity to computation. How we set up standard evaluation criteria for such complex systems and their learning process is still an open question.

## 7.4 In Closing

Seeking answers to these questions will bring us to many new research questions. Designing a robust multiagent learning system to solve real-time problems, such as emergency response, is a continuing challenge, but, exploring these questions can lead us to understand and develop more fully automatic multiagent systems. Fully automatic multiagent system can be more reflective of the ultimate goals in the field of Artificial Intelligence.

# Appendix A

# Source Code: MDRLSA

Multi-Descendent Regression Learning Structure Algorithm (MDRLSA) is a novel graphical structure learning algorithm for multiagent graphical games, using a regression model to learn a player's utility function.

Multi-Descendent Regression Learning Structure Algorithm

```
// regression_learningStructure.m
% Loading Data
if nargin < 1
    % option 1: Load Data
    load('action_profile.mat');
    load('A_profile.mat');
    load('U_profile.mat')
else
    % option 2: get matrix from reading game_file.txt file
    [U, A, action_profile] = fread_game_profile(game_file);
end
```

```
tic
%
X = action_profile;
y = U(:,1);
m = length(y);

% Add intercept term to X
X = [ones(m, 1) X];

%get number of players and each players' action number;
    [r, playerNum] = size(A);
    action_num = ceil(r^(1/playerNum));

%% Calculate the parameters from the normal equation
theta_eq = [];
for i = 1: size(U,2)
   y = U(:,i);
   theta_eq = [theta_eq, normalEqn(X, y)];
end

% Display normal equation's result
fprintf('Theta computed from the normal equations: \n');
theta_eq

%% Analyze the learning result
epsilon = 1.0000e-5
data = theta_eq(2:end,:);
[m,n] = size(data);
```

```
% give n*n matrix: 0 indicates independent; 1 indicates related;
graph_param = ones(n,n);

temp_coef = zeros(n,n);
for i = 1:n
    a = data(:,i);
    for j = 1:n
        player_coef = a((j-1)*(m/n)+1:((j-1)*(m/n)+action_num))
        a_norm = player_coef / mean(player_coef)
        temp_coef(j,i) = sum(abs(a_norm - ones(action_num,1)))
        if temp_coef(j,i) < epsilon
            % flag unrelated player j & i as 0;
            graph_param(j,i) = 0;
        end
    end
end

t = toc
graph_param
```

# Appendix B

# Source Code: BANANA

BANANA is a Bayesian Network structure learning software, where a graphical representation is constructed using the E-algorithm to reveal the inter-connections among data. Here, ALGAE provides a genetic descriptors' data for BANANA.

```
// HybridElearning.m
function DG = HybridElearning( file , a)
% this hybrid E-algorithm uses ALARM as benchmark to test BN learning
    algorithm;

D = importdata(file); %'genedecode_1.csv'
data = D.data;
% Input variable: a threshold of CI test a
% a = 0.001 ;

% Data is a variable that saves our training database.
LGObj = ConstructLGObj(data); % construct an object
LG = struct(LGObj);
```

```
Dim = LG.VarNumber;

% Step 1: Complete undirected graph H
DG = ones( Dim ); % A directed graph
for q = 1:Dim
   DG( q,q ) = 0;
   DG( q:end,q) = 0; % only upper triangle is set for 1s;
end

mutual_info = zeros(Dim, Dim);
% Step2a, test every independence relationship at first, test the mutual
    information I(xi,xj) CI test
for p = 1 : ( Dim - 1 )
   for q = ( p + 1): Dim
       [MI,R,M ] = MarginallyIndependent_MutualInformation( LGObj,p,q );
       mutual_info(p, q) = MI;
       CI = CITest_ChiTwoVar( MI,R,M,a );
       if CI == 1
          DG( p,q ) = 0; DG( q,p ) = 0;
       end
   end
end

% Step2b is optional for testing purpose;
% Step2b-1, test every independence relationship, using Expectation
    Chi-square test
chi2_testInfo = zeros(Dim, Dim);
for p = 1 : ( Dim - 1 )
```

```
    for q = ( p + 1): Dim
        [CI, chi2_testInfo(p,q)] = Chi2_Test( LGObj, p,q, a );
        if CI == 1
            DG( p,q ) = 0; DG( q,p ) = 0;
        end
    end
end

 % Step3:
    %1-variable mutual information I(xi,xj  z) CI test
    % if i->j, j->k and i->k, test I(i,kj)? and I(j,ki);
    for i = 1 : Dim
        J = find(DG(i,:)>0);
        for index1 = 1:length(J)
            j = J(index1);
            K = find(DG(j,:)>0);
            for index2 = 1: length(K)
                k = K(index2);
                if find(J == k) ~= 0
                % i->j, j->k and i->k exists, test info I(i,kj)?
                    [MI1,R1,M1 ] =
                        ConditionallyIndependent_MutualInformation(
                        LGObj,i,k,j);
                    CI1 = CITest_ChiTwoVar( MI1,R1,M1,a );
                    if CI1 == 1 % if MI <= 0.009
                        DG( i,k ) = 0; DG( k,i ) = 0;
                    end
                % test info I(j,ki)?
                [MI2,R2,M2 ] =
                    ConditionallyIndependent_MutualInformation(
                    LGObj,j,k,i);
                CI2 = CITest_ChiTwoVar( MI2,R2,M2,a );
                if CI == 1
                    DG( j,k ) = 0; DG( k,j ) = 0;
                end
            end
        end
    end
end

% Step4, using scoring test to evaluate DG; and remove the redundent
    representation;
num = length(find(DG > 0));
count = 0;
flag = 1;
% repeat Step4 a&b, until no further edges information changes.
while flag == 1
    % step 4a: removing edge (ascending order of mutual info)
    % MDL scoring test all nodes:
    % score each node j outreached by node i, through deleting edge i->j
    % MDL scoring test each edge in DG and find the mininum score graph
    [indx, indy] = find(DG > 0);
    struct_matrix = [indx indy];
    map_info_matrix = mutual_info(find(DG>0));
    % sorting orders of the edges ascendingly according to mutual_info
    between these two nodes
[new_map_info_matrix, IX] = sort(map_info_matrix);
order_edge_matrix = struct_matrix(IX,:);

[DG, remove_list] = score_descending_removedge(LGObj, DG,
    order_edge_matrix);

% step 4b: adding edges deleted in step4a rechecking (decending order
    ot mutual info)
% By avoiding the sequence prob, the first time adding all the edges
    back and retesting the scores:
% desending order of mutual_info for all the edges;
% adding from the last deleted edge to the first one;
% MDL scoring test all nodes:
% score each node j outreached by node i, through deleting edge i->j
% MDL scoring test each edge in DG and find the mininum score graph
remove_list_backup = remove_list;
[DG, remove_list, add_list] = score_ascending_addedge(LGObj, DG,
    remove_list_backup);

% step4c: sort edges according to the node ascending;
edge_matrix = sortrows(order_edge_matrix);
[DG, remove_list] = score_descending_removedge(LGObj, DG, edge_matrix);

% step 4d: adding edges deleted in step4a rechecking (decending order
    of mutual info)
% By avoiding the sequence prob, the first time adding all the edges
    back and testing again the scores:
```

```
    % desending order of mutual_info for all the edges;
    % adding from the last deleted edge to the first one;
    % MDL scoring test all nodes:
    % score each node j outreached by node i, through deleting edge i->j
    % MDL scoring test each edge in DG and find the mininum score graph
    remove_list_backup = remove_list;
    [DG, remove_list, add_list] = score_ascending_addedge(LGObj, DG,
        remove_list_backup);

    num_new = length(find(DG > 0));
    if num_new == num
        flag = 0;
    else
        num = num_new
        count = count + 1;
    end
end

end
```

# Appendix C

# ALGAE Manual

ALGAE explores the relations of group population's competition survival. The ALGAE concept is inspired by the evolutionary epoch of Homo Sapiens where two divergent species exist, Type 0 and Type 1, and compete for resources in a primitive environment where survival is the primary biological goal.

HOW ALGAE WORKS

Type 0 and Type 1 migrate randomly around the landscape. Each step costs individuals energy, and they must eat food, 'veggie' or 'honey', in order to replenish their energy - when they run out of energy, they die.

There are two main food variations to this model.

In the first variation, fix amount of food resources ('honey') is randomly located and is not renewable.

The second variation includes renewable vegetable crops('veggie') in the landscape. Once a veggie is eaten, it will only regrow according to its height at a slow growth or fast growth rate.

HOW TO USE THE PARAMETERS

1. Set the veggie? switch to TRUE to include veggie in the model, or to FALSE to only include type 1 and type 0 species.
2. Set the growth? switch for the veggie resources to TRUE to allow veggie to regrow, or to FALSE to only fixed number to the initial amount.
3. Set the honey? switch to TRUE to include honey in the model, or to FALSE to only include type 1 and type 0 species.
4. Adjust the slider parameters (see below), or use the default settings.
5. Press the SETUP button.
6. Press the GO button to begin the simulation.
7. Look at the monitors to see the current population sizes and best fitness genes and its fitness level.
8. Look at the POPULATIONS plot to watch the populations fluctuate over time.
9. Export survivors' information to data files.

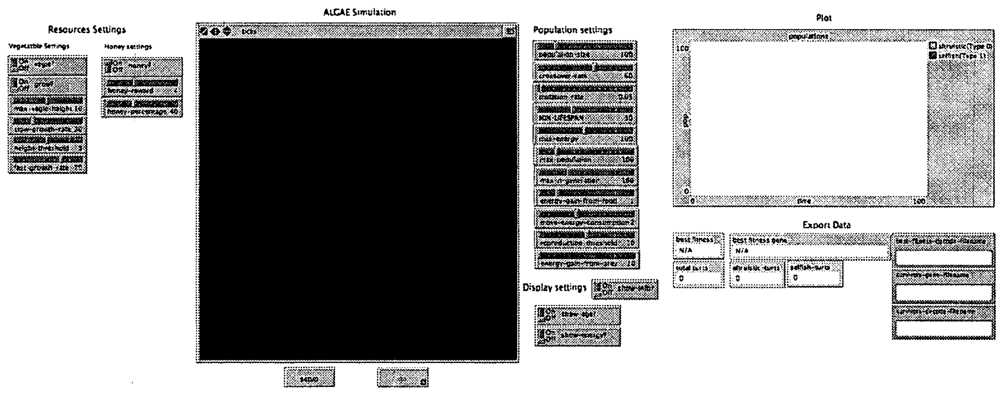
Resources Parameters:

```
max-veggie-height: The maximum height level of veggie
slow-growth-rate: The veggie slowly grows at this percentage
height-threshold: The veggie's threshold for growth rate: below the
    threshold, it grows slowly; above the threshold, it grows fast.
fast-growth-rate: The veggie grows rapidly at this percentage
honey-reward: The amount of energy reward each individual gets for every
    honey resource taken
honey-percentage: The amount of resources located in that environment

Population Parameters:
population-size: The initial size of population
crossover-rate: The crossover incidence in each chromosome pairing
mutation-rate: The probability of chromosome bits mutating
MIN-LIFESPAN: Minimum life span of an individual
max-population: Maximum population allowed in environment
energy-gain-from-food: The amount of energy individual gets for every
    resource consumed
move-enerygy-consumption: The amount of energy each agent consumes to make
    a move
reproduction-threshold: The threshold of an agent's energy required for
    reproducing at each time step
show-age?: Whether to show the age of each agent as a number
show-energy?: Whether to show the energy of each agent as a number
```

Example run:

1. Open 'ALGAE', the interface is given as:

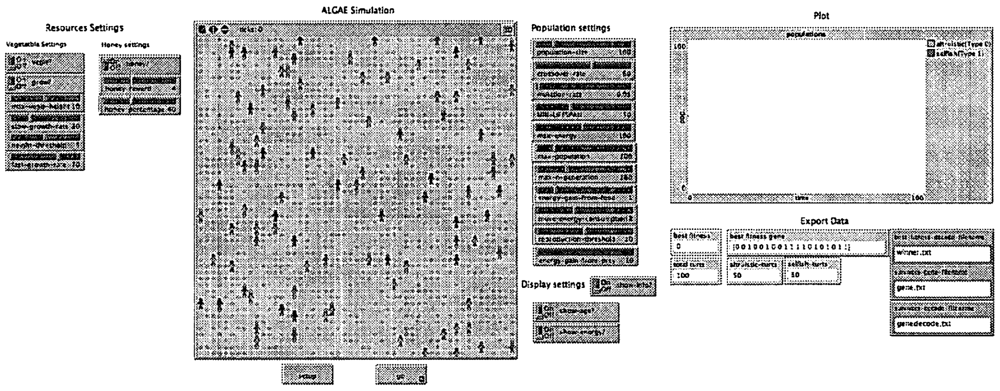


vege? On/Off
grow? On/Off
max-vegie-height 10
slow-growth-rate 30
height-threshold 3
fast-growth-rate 77
honey? On/Off
honey-reward 1
honey-percentage 40
population-size 100
crossover-rate 60
mutation-rate 0.05
MIN-LIFESPAN 30
max-energy 100
max-population 100
max-generation 100
energy-gain-from-food 1
move-energy-consumption 2
reproduction-threshold 10
energy-gain-from-prey 10
show-info? On/Off
show-age? On/Off
show-energy? On/Off
best fitness N/A
best fitness gene N/A
total turns 0
altruistic-turns 0
selfish-turns 0
best-fitness-decode-filename
survivors-gene-filename
survivors-decode-filename

Figure C.1: ALGAE interface

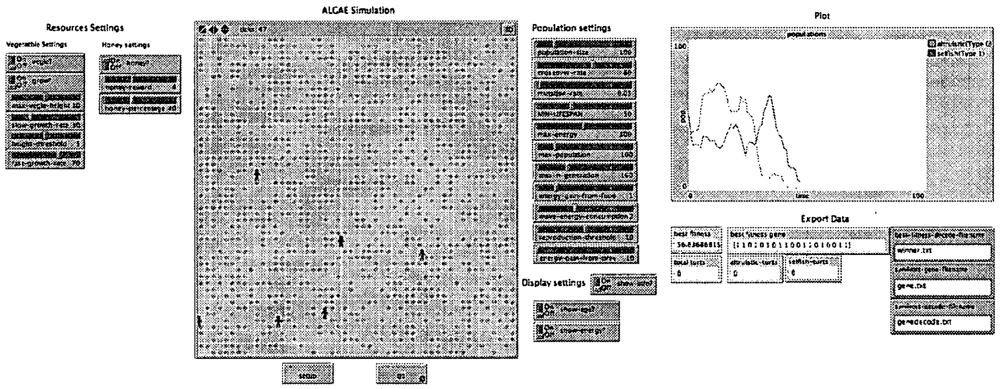
2. Adjust parameter settings. Click 'SETUP' button, ALGAE presents an initial setup accordingly.

3. Click 'RUN' button, ALGAE runs a number of generations and present the final stage when it reaches stopping criteria.



max-vegie-height 10
slow-growth-rate 20
height-threshold 1
fast-growth-rate 70
honey-reward 4
honey-percentage 40
population-size 100
crossover-rate 60
mutation-rate 0.01
MIN-LIFESPAN 10
max-energy 100
max-population 100
max-n-generation 160
energy-gain-from-food 1
move-energy-consumption 2
reproduction-threshold 10
energy-gain-from-prey 10
best fitness 0
best fitness gene [0 0 1 0 0 1 0 0 1 1 1 0 1 0 1 0 1 1]
total turts 100
altruistic-turts 50
selfish-turts 50
best-fitness-decode-filename winner.txt
genes-code-filename gene.txt
genes-decode-filename genedecode.txt

Figure C.2: ALGAE setup



max-vegie-height 10
slow-growth-rate 30
height-threshold 1
fast-growth-rate 70
honey-reward 4
honey-percentage 40
population-size 100
crossover-rate 60
mutation-rate 0.01
MIN-LIFESPAN 10
max-energy 100
max-population 100
max-n-generation 165
energy-gain-from-food 1
move-energy-consumption 2
reproduction-threshold 10
energy-gain-from-prey 10
best fitness 56.83686815
best fitness gene [1 1 0 1 0 1 0 1 1 0 0 1 1 0 1 0 0 1 1]
total turts 0
altruistic-turts 0
selfish-turts 0
best-fitness-decode-filename winner.txt
genes-code-filename gene.txt
genes-decode-filename genedecode.txt

Figure C.3: ALGAE output

# Appendix D

# ALGAE Source Code

```
breed [turts turt]
breed [honeys honey]
turtles-own [
  bits fitness energy age wealth
  preference life-span vision-radius move-angle action-speed motion-loss
      action-choice attack-ability attack-loss food-efficiency
]
patches-own [vegie value]

globals [
  winner       ;; turtle that currently has the best solution
  generation
]

to setup
  clear-all
  setup-common-variables
  if vegie? [
    ask patches [
      set vegie max-vegie-height
      color-vegie
      set value random 20
    ]
  ]
  if honey? [
    draw-grid
    distribute-honey
  ]
  setup-turts

  display-labels
  reset-ticks
end

to setup-common-variables
  set generation 0
  ask turtles [
    set energy 70 + random 30
    set age 1
    set wealth 0
    setxy random-xcor random-ycor
    ]
end
```

```
to setup-turts
  set-default-shape turtles "person"
  ;create-turts population-size [
  ask patches [
    if count turts < population-size [
      sprout-turts 1 [
        set size 2 ;; easier to see
        set bits n-values 18 [one-of [0 1]]
        ifelse count turts <= population-size / 2            ;;
            (random-float 1.0 <= cooperative-probability )
        [ set bits fput 0 bits set color red set preference 0 ]
        [ set bits fput 1 bits set color blue set preference 1 ]
        decode-turt-genes
        calculate-fitness
      ]
    ]
  ]
  update-best-turt

  save-file-var best-fitness-decode-filename
  save-file-var survivors-decode-filename
end

to save-file-var [ filename ]
  file-open filename
  file-write ( word "[preference, life-span, vision-radius, move-angle,
      action-speed, motion-loss, food-pref, inter-pref, attack-ability,
   attack-loss, food-efficiency]");, fitness]")
end

to draw-grid
  ask patches [
    set pcolor blue + 4
    sprout 1 [
      set shape "square"
      set color cyan + 2
      stamp
      die
    ]
  ]
end

to distribute-honey
  set-default-shape honeys "leaf"
  ask patches with [ random 100 < honey-percentage ] [
    sprout-honeys 1 [ set size 1 set color orange + 1 ]
  ]
end

to save-winner-decode-file
  file-open best-fitness-decode-filename
  if winner != nobody ;; [ output-print ( word [bits] of winner "\n" ) ];;
      file-write ( word [bits] of winner "\n" )
  [ ask winner [
    ;; decode " action-choice " into two parts: food-pref & interaction
        preference
    let action-queue but-first action-choice
    let food-pref ( first action-queue )
    let list-2-bits [[0 0] [0 1] [1 0] [1 1]]
    let inter-pref ( 1 + position action-queue list-2-bits )
    file-write ( word "[" preference "," (life-span - MIN-LIFESPAN) ","
        vision-radius "," (move-angle / 90) "," action-speed ","
        motion-loss "," food-pref "," inter-pref "," attack-ability ","
        attack-loss "," food-efficiency "]"); "," floor(fitness) "]")
   ]
  ]
end

to save-survivors-decode-file
  file-open survivors-decode-filename ;"genedecode.txt"
  ask turts [
    ;; decode " action-choice " into two parts: food-pref & interaction
        preference
    let action-queue but-first action-choice
    let food-pref ( first action-queue )
    let list-2-bits [[0 0] [0 1] [1 0] [1 1]]
    let inter-pref ( 1 + position action-queue list-2-bits )
    file-write ( word "[" preference "," (life-span - MIN-LIFESPAN) ","
        vision-radius "," (move-angle / 90) "," action-speed ","
        motion-loss "," food-pref "," inter-pref "," attack-ability ","
        attack-loss "," food-efficiency "]")
  ]
```

```
end

to save-survivors-gene-file
  file-open survivors-gene-filename  ;;"gene.txt"
  ask turts [ file-write ( word bits ) ]
end

to go
  ifelse (not any? turtles) or (count turts with [preference = 1] = 0 ) or
      (count turts with [preference = 0] = 0 ) or ( generation >=
      max-n-generation );; or ( honey? and not any? honeys ) or ( vegie?
      and not any? patches with [vegie > 1] ) ;; if vegie is on, check
      vegie on patch
  [ save-winner-decode-file
    save-survivors-gene-file
    save-survivors-decode-file

    file-close-all
    stop
  ]
  [ save-winner-decode-file
    save-survivors-gene-file
    save-survivors-decode-file

    file-close-all
  ]
  if vegie? [
    ask patches
    [ if grow? [ grow-vegie ]
      color-vegie
    ]
  ]
  turts-act
  display-labels
  tick
end

to turts-act
  ask turts [
    turt-act
    calculate-fitness
  ]
  ;create-next-generation           ;; without Tournament
  set generation generation + 1
  update-best-turt
end



to update-best-turt
  set winner max-one-of turts [fitness]
end

to grow-vegie
  ifelse ( vegie >= height-threshold )
  [ if fast-growth-rate >= random-float 100
    [ set vegie vegie + 1 ]
  ]
  [ if slow-growth-rate >= random-float 100
    [ set vegie vegie + 1 ]
  ]
  if vegie > max-vegie-height
  [ set vegie max-vegie-height ]
end

;; according to vegie height level, turn the patch green color in scale of
    hue
to color-vegie
  set pcolor scale-color (green - 1) vegie 0 (2 * max-vegie-height)
end

to move ;; turtle procedure
  rt random 360
  fd 1
  set age age + 1
  set energy energy - move-energy-consumption
  if energy < 0 [ die ]
end

;; turts eat vegie, change vegie height and turn the patch brown hue;
to eat-vegie-preference ;; [ selfish ]
  ifelse preference = 1 [
    if vegie > 0 [
      set vegie vegie - 1
    set energy energy + energy-gain-from-food * food-efficiency ;; gain
        energy through eating, [1*energy-gain-from-food,
        4*energy-gain-from-food]
    set wealth wealth + value
    set value 0
  ]
 ]
 [ if vegie > height-threshold [
    set vegie vegie - 1
    set energy energy + energy-gain-from-food * food-efficiency ;; gain
        energy through eating, [1*energy-gain-from-food,
        4*energy-gain-from-food]
    set wealth wealth + value
    set value 0
  ]
 ]
 if energy >= max-energy [ set energy max-energy ]          ;; maximum
    energy
end


to fight [ prey ]
 let win 0
 ask prey
  [ ;show ( word "prey energy (before):" energy )
    set energy energy - max list 10 random energy ;; prey loses energy
        minimum 10 or more if random energy is more
    ;show ( word "prey energy (after):" energy )
    if energy <= 0 [ set win 1 die ]
   ]
  ;show ( word "predator energy (before):" energy )
  let loss min list 10 random energy              ;; predator loses energy
     of maximum 10 or less (if random energy is less than 10)
  ifelse win = 1                                  ;; kill it
    [ set energy energy + energy-gain-from-prey - loss ] ;; get energy from
       prey, and lose energy fighting
    [ set energy energy - loss ] ;; get no energy from prey, and lose
       energy fighting
  if energy >= max-energy [ set energy max-energy ] ;; maximum energy
  ;show ( word "predator energy (after):" energy )
end


to attack [ prey ] ;; calculate attack strength for self and prey, and the
    stronger wins;
  let strength-prey 0
  let strength-self energy + attack-ability * random-float 1 * 20 / 4
            ;; self strength
  ask prey [ set strength-prey energy + attack-ability * random-float 1 *
     20 / 4 ] ;; prey stength
  ;; the stronger wins
  ifelse strength-self >= strength-prey [
    set energy energy - attack-loss * random-float 1 death-check
    ask prey [ set energy energy - attack-loss * random-float 1 - 40
       death-check ]
  ]
  [ set energy energy - attack-loss * random-float 1 - 40 death-check
    ask prey [ set energy energy - attack-loss * random-float 1 death-check
        ]
  ]
end

to calculate-fitness      ;; turts procedure
  set fitness energy
end

to death-check ;; energy change, need to update fitness and turt survior
    status
  if energy <= 0 [ die ]
  calculate-fitness
end

to decode-turt-genes
  let list-2-bits [[0 0] [0 1] [1 0] [1 1]]
  let list-3-bits [[0 0 0] [0 0 1] [0 1 0] [0 1 1] [1 0 0] [1 0 1] [1 1 0]
      [1 1 1]]

  set preference first bits                                        ;;
      1 bit - take item 0: first bit to describe agent's preference 1:
      selfish blue  0: altruistic pink
  set life-span ( MIN-LIFESPAN + position ( sublist bits 1 3 ) list-2-bits
      ) ;; 2 bit - take the gene from item 1 - 4 ;; obtain max life span,
      set as span; minimum-lifespan is 50
  set vision-radius ( 1 + position ( sublist bits 3 5 ) list-2-bits )   ;;
     2 bit - take the gene from item 5 - 6 ;; obtain vision field size,
     set as radius; minmum radius is 1
  set move-angle 90 * ( 1 + position ( sublist bits 5 7 ) list-2-bits ) ;;
     2 bit - take the gene from item 7 - 8 ;; obtain move direction
     degrees, set as angle [90 180 270 360]; move direction refers to 360
     degrees coverage
  set action-speed ( 1 + position ( sublist bits 7 9 ) list-2-bits )   ;; 2
     bit - take the gene from item 9 - 10 ;; obtain move action-speed, set
     as each move stepsize 1-4;
  set motion-loss ( 1 + item 9 bits )                                   ;; 1
     bit - take the gene from item 11 ;; obtain energy motion-loss: 1-2
  set action-choice sublist bits 10 13                                  ;; 3
     bit - take the gene from item 12 - 14 ;; obtain action preferece: 0-7
  ;; 1st bit: 0:eat-food first 1:eat-food later  1:mate 2: fight 00: 1 /
     01: 2 / 10: 12 / 11: 21
  set attack-ability ( 1 + position ( sublist bits 13 15 ) list-2-bits )
     ;; 2 bit - take the gene from item 15 - 18 ;; obtain attack ability:
     1-16
  set attack-loss ( 1 + position ( sublist bits 15 17 ) list-2-bits ) ;; 2
     bit - take the gene from item 19 - 21 ;; obtain attack loss: 1-8
  set food-efficiency ( 1 + position ( sublist bits 17 19 ) list-2-bits )
     ;; 2 bit - take the gene from item 22 - 23 ;; obtain food aborbing
     efficiency: 1-4
end

to action-with-mate
  let action-queue but-first action-choice
```

```
ifelse first action-queue = 0
[ ifelse last action-queue = 1
  [ let prey look-for-prey if prey != nobody [ attack prey ] ] ;; 01:
      fight ;; attack prey
  [ if count turts < max-population - 1 and energy >=
      reproduction-threshold ;; 00: mate
    [ let mate look-for-mate
      if mate != nobody and [energy] of mate >= reproduction-threshold
      [ reproduce-crossover-turt mate ]
     ]  ;; look for the best mate to reproduce
  ]
]
[ ifelse last action-queue = 1
                                                          ;; 11: fight mate
  [ let prey look-for-prey if prey != nobody [ attack prey ]
                     ;; attack prey
    if count turts < max-population - 1 and energy >=
       reproduction-threshold ;; look for the best mate to reproduce
    [ let mate look-for-mate
      if mate != nobody and [energy] of mate >= reproduction-threshold
      [ reproduce-crossover-turt mate ]
     ]
  ]
  [


      ;; 10: mate fight
    if count turts < max-population - 1 and energy >=
       reproduction-threshold ;; look for the best mate to reproduce
      [ let mate look-for-mate
        if mate != nobody and [energy] of mate >= reproduction-threshold
        [ reproduce-crossover-turt mate ]
       ]
      let prey look-for-prey if prey != nobody [ attack prey ]
                         ;; attack prey
    ]
  ]
end

to turt-act
  ifelse first action-choice = 0 ;; eat food first
  [ fetch-food action-with-mate ]
  [ action-with-mate fetch-food ]

  rt random move-angle
  fd action-speed
  set energy energy - motion-loss
  death-check

  set age age + 1
  if age >= life-span [ die ]
end

to-report look-for-honey
  let h one-of honeys in-cone vision-radius move-angle
  report h
end
```

```
to fetch-honey [ h ]
  face h
  move-to h
  set energy energy + honey-reward * food-efficiency ;; energy gain from
      honey, when honey-reward is 4, 4 - 16, [1*honey-reward,
      4*honey-reward]
  ask h [ set color gray stamp die ]
  if energy >= max-energy [ set energy max-energy ]          ;; maximum
      energy
end

to-report look-for-food-target
  let p max-one-of patches in-cone vision-radius move-angle [ vegie ]
      ;show ( word "patch here: vegie" patch-here vegie ) ;show ( word "p:
      vegie" p [vegie] of p)
  ifelse p != nobody
  [ report p ]
  [ report patch-here ]
end

to-report look-for-mate
  let mate max-one-of other breed in-cone vision-radius move-angle [
      fitness ] ;;vision-radius vision-angle ;[ let p max-one-of neighbors
      [patch-variable]
  report mate
end
```

```
to-report look-for-prey
  let prey min-one-of other breed in-cone vision-radius move-angle [
      fitness ] ;;vision-radius vision-angle ;[ let p max-one-of neighbors
      [patch-variable]
  report prey
end

to fetch-food
  if honey? [                                    ;; if honey is resources,
      fetch honey
    ifelse any? honeys-here
      [ let h one-of honeys-here fetch-honey h ] ;; preference
          food-efficiency ] ;; show h    show "honey here" ] ;, pick up can
          here
      [ let h look-for-honey                ;;  output-print ( word "patch
          here: / honey " patch-here h ) ; look for any can in the vision
          area
        if h != nobody [ fetch-honey h ]      ;; found honey in vision area,
            go and get it
        ;; [ rt random move-angle fd action-speed ]   ;; nothing found in
            the area, move randomly
      ]
    set energy energy - motion-loss
    death-check
  ]
  if vegie? [                                    ;; if vegie is resources,
      take vegie ;show ( word "patch here:" patch-here)
    let target-patch look-for-food-target        ;; move to the best vegie
        patch and eat by preference
    move-to target-patch                         ;; show ( word "new patch
        here:" patch-here)
    eat-vegie-preference
  ]
end



to reproduce-crossover-turt [ mate ]
  let parent1 self
  let parent2 mate
  let child-bits crossover ([bits] of parent1) ([bits] of parent2)

  ; create the two children, with their new genetic material
  ask parent1 [
    set energy ( 0.5 * energy )
    calculate-fitness
    hatch 1 [ set bits item 0 child-bits
      mutate
      rt random-float 360 fd 1          ;; set up offspring's initial level
      set age 1                         ;; set up wealth distribution level
          later for selfish and altruistic
      decode-turt-genes
      ifelse preference = 0
      [ set color red ]
      [ set color blue ]
      ]
    ]
  ask parent2 [
    set energy ( 0.5 * energy )
    calculate-fitness
    hatch 1 [ set bits item 1 child-bits
      mutate
      rt random-float 360 fd 1          ;; set up offspring's initial level
      set age 1                         ;; set up wealth distribution level
          later for selfish and altruistic
      decode-turt-genes
      ifelse preference = 0
      [ set color red ]
      [ set color blue ]
      ]
    ]
end


to-report crossover [bits1 bits2]
  let split-point 1 + random (length bits1 - 1)
  report list (sentence (sublist bits1 0 split-point)
                        (sublist bits2 split-point length bits2))
              (sentence (sublist bits2 0 split-point)
                        (sublist bits1 split-point length bits1))
end

;; This procedure causes random mutations to occur in a solution's bits.
;; The probability that each bit will be flipped is controlled by the
   MUTATION-RATE slider.
to mutate  ;; turtle procedure
  let p first bits
  ;; show ( word "bits:" bits )
  let temp-bits map [ifelse-value (random-float 100.0 < mutation-rate) [1
      - ?] [?]]
              but-first bits
  set bits fput p temp-bits
  ;; show ( word "mutate bits:" bits )
end


to display-labels
  ask turts [
    ifelse show-info?
    [
      if show-energy? [ set label-color black set label round energy ]
      if show-age? [ set label age ]
    ]
    [ set label "" ]
  ]
end
```

# Bibliography

Alfeld, S., Berkele, K., Desalvo, S. A., Pham, T., Russo, D., Yan, L., and Taylor, M. E. (2011). Reducing the team uncertainty penalty: Empirical and theoretical approaches. In *Proceedings of the Workshop on Multiagent Sequential Decision Making in Uncertain Domains*, AAMAS '11, pages 2–15.

Andre, D. and Teller, A. (1999). Evolving team darwin united. In *Robot Soccer World Cup II (RoboCup-98)*, pages 346–351. Springer-Verlag.

Axelrod, R. and Hamilton, W. (1981). The evolution of cooperation. *Science*, 211(4489):1390–1396.

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

Balch, T. (1997). Learning roles: Behavioral diversity in robot teams. In *1997 AAAI Workshop on Multiagent Learning*, pages 7–12. AAAI.

Balch, T. (1999). Reward and diversity in multirobot foraging. In *IJCAI-99 Workshop on Agents Learning About, From and With other Agents.*

Banerjee, B. and Peng, J. (2005). Efficient no-regret multiagent learning. In *Pro-*

ceedings of the Twentieth National Conference on Artificial Intelligence*, volume 1 of *AAAI '05*, pages 41–46. AAAI Press.

Bellman, R. (1957). *Dynamic programming.* Princeton University Press.

Bernstein, D. S., Zilberstein, S., and Immerman, N. (2000). The complexity of decentralized control of markov decision processes. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*, UAI '00, pages 32–37. Morgan Kaufmann.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics).* Springer.

Boccaletti, S., Latora, V., Moreno, Y., Chavez, M., and Hwang, D.-U. (2006). Complex networks: Structure and dynamics. *Physics Reports*, 424(4-5):175–308.

Bowling, M. (2003). *Multiagent learning in the presence of agents with limitations.* PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA.

Bowling, M. (2005). Convergence and no-regret in multiagent learning. In *Advances in Neural Information Processing Systems*, pages 209–216. MIT Press.

Bowling, M. and Veloso, M. (2002). Multiagent learning using a variable learning rate. *Artificial Intelligence*, 136:215–250.

Broder, A., Kumar, R., Maghoul, F., Raghavan, P., Rajagopalan, S., Stata, R., Tomkins, A., and Wiener, J. (2000). Graph structure in the web. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 33(1-6):309–320.

Brown, G. W. (1949). Some notes on computation of games solutions. Technical report, The RAND Corporation, Santa Monica, California.

Brown, G. W. (1951). Iterative solutions of games by fictitious play. In *Activity Analysis of Production and Allocation*, pages 367–383. Wiley.

Cheng, J., Bell, D., and Liu, W. (1997). Learning belief networks from data: An information theory based approach. In *Proceedings of the Sixth ACM International Conference on Information and Knowledge Management*, pages 325–331. ACM Press.

Cheng, J., Greiner, R., Kelly, J., Bell, D., and Liu, W. (2002). Learning Bayesian Networks from data: an information-theory based approach. *Artificial Intelligence*, 137(1-2):43–90.

Claus, C. and Boutilier, C. (1998). The dynamics of reinforcement learning in cooperative multiagent systems. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 746–752. AAAI Press.

Conitzer, V. and Sandholm, T. (2006). AWESOME: A general multiagent learning algorithm that converges in self-play and learns a best response against stationary opponents. In *Proceedings of the 20th International Conference of Machine Learning*, pages 83–90. ACM Press.

Conitzer, V. and Sandholm, T. (2008). New complexity results about Nash equilibria. *Games and Economic Behavior*, 63(2):621 – 641.

Cooper, G. F. and Herskovits, E. (1991). A Bayesian method for constructing Bayesian belief networks from databases. In Cooper, G. F. and Moral, S., editors, *Proceedings of the Seventh Conference on Uncertainty in Artificial Intelligence*, pages 86–94. Morgan Kaufmann.

Cooper, G. F. and Herskovits, E. (1992). A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 09(4):309–347.

de Campos, L. M., Fernández-Luna, J. M., and Huete, J. F. (2004). Bayesian Networks and information retrieval: an introduction to the special issue. *Information Processing & Management*, 40(5):727 – 733.

Duong, Q., Vorobeychik, Y., Singh, S., and Wellman, M. P. (2009). Learning graphical game models. In *Proceedings of the 21st International Jont Conference on Artifical Intelligence*, pages 116–121. Morgan Kaufmann.

Ficici, S. G. and Pollack, J. B. (2000). A game-theoretic approach to the simple coevolutionary algorithm. In *PPSN VI: Proceedings of the 6th International Conference on Parallel Problem Solving from Nature*, pages 467–476. Springer-Verlag.

Fisher, R. (1930). *The Genetical Theory of Natural Selection.* Clarendon Press, Oxford.

Friedman, N., Linial, M., Nachman, I., and Peér, D. (2000). Using Bayesian networks to analyze expression data. *Journal of Computational Biology*, 7:601–620.

Friedman, N., Nachman, I., and Peér, D. (1999). Learning Bayesian network structure from massive datasets: The "sparse candidate" algorithm. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, UAI '99, pages 206–215. Morgan Kaufmann.

Fudenberg, D. and Levine, D. K. (1995). Consistency and cautious fictitious play. *Journal of Economic Dynamics and Control*, 19(5-7):1065–1089.

Gilboa, I. and Zemel, E. (1988). Nash and correlated equilibria: Some complexity considerations. Discussion Papers 777, Northwestern University, Center for Mathematical Studies in Economics and Management Science.

Gmytrasiewicz, P. J. and Doshi, P. (2005). A framework for sequential planning in multi-agent settings. *Journal Artificial Intelligence Research*, 24(1):49–79.

Greenwald, A. and Hall, K. (2003). Correlated-Q learning. In *AAAI Spring Symposium*, pages 242–249. AAAI Press.

Grefenstette, J., Ramsey, C. L., and Schultz, A. C. (1990). Learning sequential decision rules using simulation models and competition. *Machine Learning*, 5(4):355–381.

Hara, A. and Nagao, T. (1999). Emergence of cooperative behavior using adg; automatically defined groups. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '99, pages 1039–1046. Morgan Kaufmann.

Haynes, T. and Sen, S. (1996a). Cooperation of the fittest. Technical Report UTULSA-MCS-96-09, The University of Tulsa.

Haynes, T. and Sen, S. (1996b). Evolving behavioral strategies in predators and prey. In *Adaptation and Learning in Multiagent Systems*, pages 113–126. Springer-Verlag.

Haynes, T. and Sen, S. (1997a). Crossover operators for evolving a team. In *Genetic*

*Programming 1997: Proceedings of the Second Annual Conference*, pages 162–167. Morgan Kaufmann.

Haynes, T., Sen, S., Schoenefeld, D., and Wainwright, R. (1995a). Evolving a team. In *Working Notes for the AAAI Symposium on Genetic Programming*, pages 23–30. MIT, Cambridge, MA, USA.

Haynes, T., Sen, S., Schoenefeld, D., and Wainwright, R. (1995b). Evolving multiagent coordination strategies with genetic programming. Technical report, The University of Tulsa.

Haynes, T. D. and Sen, S. (1997b). Co-adaptation in a team. *International Journal of Computational Intelligence and Organizations*, 1:1–4.

Heckerman, D. (1995). A tutorial on learning with Bayesian networks. Technical report, Microsoft Research, Redmond, Washington.

Heckerman, D., Geiger, D., and Chickering, D. M. (1995). Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, 20(3):197–243.

Holland, J. H. (1985). Properties of the bucket brigade. In *Proceedings of the First International Conference on Genetic Algorithms*, pages 1–7. L. Erlbaum Associates Inc.

Holland, J. H. and Miller, J. H. (1991). Artificial adaptive agents in economic theory. *American Economic Review*, 81(2):365–71.

Hu, J. and Wellman, M. P. (1998). Multiagent reinforcement learning: Theoretical framework and an algorithm. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 242–250. Morgan Kaufmann.

Hu, J. and Wellman, M. P. (2003). Nash Q-learning for general-sum stochastic games. *Maching Learning Research*, 4:1039–1069.

Jan't Hoen, P. and Tuyls, K. (2004). Analyzing multi-agent reinforcement learning using evolutionary dynamics. In *Machine Learning: ECML 2004*, volume 3201 of *Lecture Notes in Computer Science*, pages 168–179. Springer.

Ji, J., Liu, C., Yan, J., and Zhong, N. (2004). Bayesian networks structure learning and its application to personalized recommendation in a B2C portal. In *Proceedings of the 2004 IEEE/WIC/ACM International Conference on Web Intelligence*, pages 179–184. IEEE Computer Society.

Ji, J., Yan, J., Liu, C., and Zhong, N. (2005). An improved Bayesian networks learning algorithm based on independence test and MDL scoring. In *Proceedings of the International Conference on Active Media Technology*, pages 315–320.

Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285.

Kapetanakis, S. and Kudenko, D. (2004). Reinforcement learning of coordination in heterogeneous cooperative multi-agent systems. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '04, pages 1258–1259, Washington, DC, USA. IEEE Computer Society.

Kearns, M., Littman, M., and Singh, S. (2001). Graphical models for game theory.

In *Proceedings of the Seventeenth Conference Annual Conference on Uncertainty in Artificial Intelligence*, pages 253–260. Morgan Kaufmann.

Lewontin, R. (1974). *The Genetic Basis of Evolutionary Change.* Columbia University Press.

Leyton-Brown, K. and Tennenholtz, M. (2003). Local-effect games. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, IJCAI'03, pages 772–777. Morgan Kaufmann Publishers Inc.

Lichbach, M. I. (1996). *The cooperator's dilemma.* University of Michigan Press, Ann Arbor.

Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 157–163. Morgan Kaufmann.

Littman, M. L. (2001). Friend-or-foe Q-learning in general-sum games. In *ICML '01: Proceedings of the Eighteenth International Conference on Machine Learning*, pages 322–328. Morgan Kaufmann.

Littman, M. L. and Szepesvari, C. (1996). A generalized reinforcement-learning model: Convergence and applications. In *Proceedings of the Thirteenth International Conference on Machine Learning*, pages 310–318. Morgan Kaufmann.

Liu, G., Feng, W., Wang, H., Liu, L., and Zhou, C. (2009). Reconstruction of gene regulatory networks based on two-stage Bayesian Network structure learning algorithm. *Journal of Bionic Engineering*, 6(1):86 – 92.

Luke, S. and Spector, L. (1996). Evolving teamwork and coordination with genetic programming. In *Proceedings of the First Annual Conference on Genetic Programming*, GECCO '96, pages 150–156, Cambridge, MA, USA. MIT Press.

Mannor, S. and Shimkin, N. (2001). Adaptive strategies and regret minimization in arbitrarily varying Markov environments. In *Proceedings of Fourteenth Annual Conference on Computational Learning Theory*, pages 128–142. Springer-Verlag.

Mannor, S. and Shimkin, N. (2003). The empirical Bayes envelope and regret minimization in competitive Markov decision processes. *Mathematics of Operations Research*, 28(2):327–345.

Mataric, M. J. (1994). Interaction and intelligent behavior. Technical report, Cambridge, MA, USA.

Michod, R. E. (1981). Positive heuristics in evolutionary biology. *The British Journal for the Philosophy of Science*, 32(1):1–36.

Mitchell, M. and Forrest, S. (1994). Genetic algorithms and artificial life. *Artificial Life*, 1(3):267–289.

Modi, P. J., Shen, W., Tambe, M., and Yokoo, M. (2005). ADOPT: Asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence Journal*, pages 149–180.

Moody, J., Liu, Y., Saffell, M., and Youn, K. (2004). Stochastic direct reinforcement: Application to simple games with recurrence. In *Proceedings of Artificial Multiagent Learning 2004 AAAI Fall Symposium.*

Nair, R., Tambe, M., Yokoo, M., Pynadath, D., and Sella, S. M. (2003). Taming decentralized POMDPs: Towards efficient policy computation for multiagent settings. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, IJCAI '03, pages 705–711. Morgan Kaufmann Publishers Inc.

Nash, J. F. (1950). Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences of the United States of America*, 36(1):48–49.

Nash, J. F. (1951). Non-cooperative games. *The Annals of Mathematics*, 54(2):286–295.

Nudelman, E., Wortman, J., Shoham, Y., and Leyton-Brown, K. (2004). Run the gamut: A comprehensive approach to evaluating game-theoretic algorithms. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 880–887.

Osborne, M. J. and Rubinstein, A. (1994). *A Course in Game Theory.* MIT Press Books. The MIT Press.

Panait, L. and Luke, S. (2005). Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 11(3):387–434.

Panait, L., Wieg, R. P., and Luke, S. (2004a). A sensitivity analysis of a cooperative coevolutionary algorithm biased for optimization. In *Genetic and Evolutionary Computation Conference*, pages 573–584. Springer.

Panait, L., Wieg, R. P., and Luke, S. (2004b). A visual demonstration of convergence properties of cooperative coevolution. In *Parallel Problem Solving from Nature*, PPSN-2004, pages 892–901. Springer.

Panait, L., Wiegand, R. P., and Luke, S. (2003). Improving coevolutionary search for optimal multiagent behaviors. In *IJCAI'03: Proceedings of the 18th international joint conference on Artificial intelligence*, pages 653–658. Morgan Kaufmann.

Partridge, L. and Harvey, P. H. (1985). Evolutionary biology: Costs of reproduction. *Nature*, 316(6023):20.

Pearl, J. (1986). A constraint-propagation approach to probabilistic reasoning. In Kanal, L. N. and Lemmer, J. F., editors, *Uncertainty in Artificial Intelligence*, pages 357–369. North-Holland.

Pearson, K. (1900). On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling. *Philosophical Magazine Series*, 50(302):157–175.

Pelikan, M. (2008). Probabilistic model-building genetic algorithms. In *Proceedings of the 2008 Genetic and Evolutionary Computation Conference*, GECCO '08, pages 2389–2416. ACM.

Peshkin, L., Kim, K.-E., Meuleau, N., and Kaelbling, L. P. (2000). Learning to cooperate via policy search. In *Sixteenth Conference on Uncertainty in Artificial Intelligence*, UAI '00, pages 489–496. Morgan Kaufmann.

Potter, M. A. and De Jong, K. A. (2000). Cooperative coevolution: An architecture for evolving coadapted subcomponents. *Evolutionary Computation*, 8(1):1–29.

Potter, M. A., Meeden, L., and Schultz, A. C. (2001). Heterogeneity in the coevolved behaviors of mobile robots: the emergence of specialists. In *Proceedings of the*

*17th International Joint Conference on Artificial Intelligence*, volume 2, pages 1337–1343. Morgan Kaufmann.

Puppala, N., Sen, S., and Gordin, M. (1998). Shared memory based cooperative coevolution. In *Proceedings of the 1998 IEEE World Congress on Computational Intelligence*, pages 570–574, Anchorage, Alaska, USA. IEEE Press.

Qiang, L., Xiao, T. Y., and Qiao, G. X. (2002). An improved Bayesian Networks learning algorithm. *Journal of Computer Research and Development*, 39(10):1221–1226.

Quinn, M. (2001). A comparison of approaches to the evolution of homogeneous multi-robot teams. In *Proceedings of the 2001 Congress on Evolutionary Computation*, volume 1, pages 128–135. IEEE Press.

Quinn, M., Smith, L., Mayley, G., and Husbands, P. (2003). Evolving teamwork and role-allocation with real robots. In *Proceedings of the eighth International Conference on Artificial Life*, ICAL '03, pages 302–311, Cambridge, MA, USA. MIT Press.

Robinson, J. (1951). An iterative method of solving a game. *The Annals of Mathematics*, 54(2):296–301.

Rosenthal, R. W. (1973). A class of games possessing pure-strategy Nash equilibria. *International Journal of Game Theory*, 2(1):65–67.

Rossi, F., editor (2013). *Interactive POMDP Lite: Towards Practical Planning to Predict and Exploit Intentions for Interacting with Self-Interested Agents.* IJCAI/AAAI.

Rubinstein, A. (2005). Discussion of "behavioral economics". *Advances in Economics and Econometrics Theory and Applications, Ninth World Congress.*

Salustowicz, R. P., Wiering, M. A., and Schmidhuber, J. (1998). Learning team strategies: Soccer case studies. *Machine Learning*, 33(2-3):263–282.

Sandholm, T. (2003). Making markets and democracy work: A story of incentives and computing. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1649–1671.

Schuster, P. and Sigmund, K. (1983). Replicator dynamics. *Journal of Theoretical Biology*, 100(3):533 – 538.

Shapley, L. S. (1953). Stochastic Games. *Proceedings of the National Academy of Sciences of the United States of America*, 39(10):1095–1100.

Shoham, Y. and Leyton-Brown, K. (2009). *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations.* Cambridge University Press.

Shoham, Y., Powers, R., and Grenager, T. (2007). If multi-agent learning is the answer, what is the question? *Artificial Intelligence*, 171(7):365–377.

Singh, S. P., Kearns, M. J., and Mansour, Y. (2000). Nash convergence of gradient dynamics in general-sum games. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, UAI '00, pages 541–548. Morgan Kaufmann.

Stone, P. and Veloso, M. (2000). Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, 8(3):345–383.

Strogatz, S. H. (2001). Exploring complex networks. *Nature*, 410:268–276.

Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44.

Sutton, R. S. (1989). Implementation details of the TD($\lambda$) procedure for the case of vector predictions and backpropagation. Technical Report TN87-509.1, GTE laboratories.

Sutton, R. S. and Barto, A. G. (1998). *Introduction to Reinforcement Learning.* MIT Press, Cambridge, MA, USA.

Suzuki, J. (1993). A construction of Bayesian networks from databases based on an mdl principle. In *Proceedings of the Ninth International Conference on Uncertainty in Artificial Intelligence*, pages 266–273. Morgan Kaufmann.

Taylor, M. E., Jain, M., Jin, Y., Yooko, M., and Tambe, M. (2010). When should there be a "me" in "team"? Distributed multi-agent optimization under uncertainty. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems.* IFAAMAS.

Travers, J. and Milgram, S. (1969). An experimental study of the small world problem. *Sociometry*, 32(4):425–443.

Tuyls, K., Verbeeck, K., and Lenaerts, T. (2003). A selection-mutation model for Q-learning in multi-agent systems. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, AAMAS '03, pages 693–700, New York, NY, USA. ACM.

Uther, W. T. B. and Veloso, M. M. (2003). Adversarial reinforcement learning. Technical Report CMU-CS-03-107, Carnegie Mellon University.

Velagapudi, P., Varakantham, P., Sycara, K., and Scerri, P. (2011). Distributed model shaping for scaling to decentralized POMDPs with hundreds of agents. In *The Tenth International Conference on Autonomous Agents and Multiagent Systems*, pages 955–962. International Foundation for Autonomous Agents and Multiagent Systems.

Vidal, J. and Durfee, E. (1998). The moving target function problem in multi-agent learning. In *Proceedings of the third International Conference on Multi Agent Systems*, ICMAS '98, page 317, Washington, DC, USA. IEEE Computer Society.

Vidal, J. M. and Durfee, E. H. (2003). Predicting the expected behavior of agents that learn about agents: The CLRI framework. *Autonomous Agents and Multi-Agent Systems*, 6(1):77–107.

von Neumann, J. and Morgenstern, O. (1944). *Theory of Games and Economic Behavior.* Princeton University Press.

Vrieze, O. (1987). *Stochastic games with finite state and action spaces.* CWI tracts.

Wang, X. and Sandholm, T. (2002). Reinforcement learning to play an optimal Nash equilibrium in team markov games. In *Advances in Neural Information Processing Systems*, volume 15, pages 1571–1578. MIT Press.

Watkins, C. (1989). *Learning from Delayed Rewards.* PhD thesis, Cambridge University, England.

Watkins, C. J. C. H. and Dayan, P. (1992). Technical note: Q-learning. *Machine Learning*, 8(3-4):279–292.

Watts, D. J. (1999). Networks, dynamics, and the small-world phenomenon. *American Journal od Sociology*, 105(2):493–527.

Watts, D. J. and Strogatz, S. H. (1998). Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442.

Wiegand, R. P. (2004). *An analysis of cooperative coevolutionary algorithms.* PhD thesis, George Mason University, Fairfax, VA, USA.

Wiering, M., Salustowicz, R., and Schmidhuber, J. (1999). Reinforcement learning soccer teams with incomplete world models. *Autonomous Robots*, 7(1):77–88.

Wilensky, U. (1999). *NetLogo itself.* Center for Connected Learning and Computer-Based Modeling, Northwestern University., Evanston, IL.

Wong, M. L., Lam, W., and Leung, K. S. (1999). Using evolutionary programming and minimum description length principle for data mining of Bayesian Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(2):174–178.

Wooldridge, M. (2008). *An Introduction to MultiAgent Systems.* Wiley.

Yan, J. (2003). Bayesian Network structure learning. Technical report, Beijing University of Technology.

Yan, J. (2007). Bayesian Networks in gene selection. Master thesis, Beijing University of Technology.

Yan, J., Lv, S., and Zhong, N. (2007). Artificial life modeling in corporate strategy. *Journal of Guangxi Normal University*, 25(4).

Yan, L. J. and Cercone, N. (2010a). Bayesian Network modeling for evolutionary genetic structures. *Computers & Mathematics with Applications*, 59:2541–2551.

Yan, L. J. and Cercone, N. (2010b). Thoughts on multiagent learning: From a reinforcement learning perspective. Technical Report CSE-2010-07, Department of Computer Science and Engineering, York University, 4700 Keele St., Toronto.

Yan, L. J. and Cercone, N. (2011). Hierarchical adaptive cooperation for emergency response. In *First IEEE Canada Women in Engineering National Conference*, IEEE WIENC, Toronto, Canada.