Experimental evaluation of multi-agent reinforcement learning in real-world scale-free networks

Rashid Al Hashimi

Master of Science in Information Technology
Faculty of Informatics

The British University in Dubai

2010

Abstract

Multi-agent reinforcement learning is a common method for optimizing agents' local decision in a distributed and scalable manner. However, the study and analysis of the state-of-the-art multi-agent reinforcement learning (MARL) algorithms have been limited to small problems involving few number of learning agents.

The purpose of this project is to conduct an extensive evaluation and comparison of MARL algorithms when used in networks that exhibit the scale-free property. The Internet and the social network of collaboration in science are only few examples of real-world networks that exhibit this property. Toward this goal, we developed a simulator that facilitates studying combinations of MARL algorithms, strategic games and networks with control propagation via tokens. These tokens are considered an opportunity for agents to play. Tokens also initiate a factor of randomness in the environment given its probability distribution over agents. Preliminary experimental results showed a significant reaction to the increase of tokens when agents play battle of the sexes in Neural network; the increase in token transfer probability yields a higher reward and a faster conversion.

Acknowledgment

I would like to sincerely thank Dr. Sherief Abdalla for his continuous help and advice throughout my thesis. His outstanding knowledge in the field of reinforcement learning has added a great value to my knowledge-base and understanding of this area. I would like to thank my friend and long-time roommate Muhammed Masood for allowing me to run the simulator on his servers. My sincere thanks also to Dubai Police for their encouragement to pursue higher education.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Rashid Al Hashimi*)

Contents

List of Figures

List of Tables

Chapter 1

Overview

1.1 Introduction

Multi-Agent reinforcement learning has been the focus of many recent scientific studies [8], [7]. The reinforcement learning problem in most of these studies concentrates on considerably small number of learning agents. As the number of learning agents increases and agents are organized in a network, understanding the dynamics of learning becomes increasingly complex.

The primary focus of this research paper is the creation of a multi-agent simulator that can be used to study the dynamics of learning agents. In specific, the simulator allows studying different combinations of learning algorithms, games and networks.

The concepts presented in this research paper are not revolutionary by themselves. The innovation lies in the idea of combining multiple study variables (strategic game, learning algorithm and network structure) that can possibly affect the performance of the agents and the distribution of rewards. We can pair strategic games with one of the learning algorithms, select a network and then let the simulation run for a given number of iterations to examine the results.

Our proposed simulator is designed to facilitate simple change of configuration parameters through a single centralized file. In addition, the simulator captures result in structured format that allows for simple and quick analysis. Hence, the simulator provides us with a powerful tool that remains abstract but at the same time very much extendable.

A key feature we introduce into the simulator is the idea of using tokens. These tokens initiate a factor of randomness in the environment. A token is considered an opportunity for agents to play the game. The goal is to recognize new dependencies between agents that go beyond the simple one-hop connection. For example, it may become possible to observe the impact of an agents interaction with a well connected powerful neighbor (and perhaps derive some benefit from its association with such a neighbor). These tokens are assigned based on pre-determined probability distribution of agents -a factor that is modifiable from the simulator's configuration file.

1.2 Design Consideration

The simulator is developed in Java environment since it is widely used and convenient for others to run and understand its source code. The simulator's source code will be posted online to encourage others to enhance and utilize its functionality. The simulator uses the GML format. The GML format is a standardized and widely-used file format for specifying networks (social and physical). Configuration of the simulator is consolidated in one file for easy control of different study factor of the simulation process. The output result captured data is in plain text comma separated format to ease export and import operation and to facilitate the use of other data analysis applications.

1.3 Contribution

The contribution of the thesis are in the following two areas:

1. I created a new multi-agent simulator to facilitate studying combinations of MARL algorithms, strategic games and networks with control propagation via tokens.
2. I presented preliminary analysis of Q-learning algorithm in scale-free networks and different strategic games.

Chapter 2

Background

In this chapter, we describe some of the necessary background for the dissertation. We also summarize relevant research studies that have been conducted in the past.

2.1 Reinforcement Learning

Reinforcement learning (**RL**) allows an agent to maximize numeric reward base on continuous interaction with the environment. We characterize the agent in this study as learning agent -"the learner" that is simply not told what action to choose rather the agent is expected to discover and learn which action is the best. The elements of Reinforcement learning comprise of a policy, a reward function, a value function and an environment model [15].

A policy defines the agent's behavior in a given state, mapping states to actions to be performed. A policy can be described simply as a search algorithm and lookup tables. Occasionally it can be complex involving extensive search queries and computational process. A reward function maps state-action pair to numerical reward that tells the agent how desirable the state is. A value function defines the long-term reward the agent can expect to accumulate from the current state; given the policy the agent adapts. An environment model defines the behavior of the environment in a sense that it might predict what the next state and reward will be, in relation the current state and action. The concept of reinforcement learning is a key feature we plan to apply on our proposed multi-agent simulator. We intend to capture the effect of agents interactions and the learning process for each agent.

2.1.1 Learning Algorithm

Temporal Difference (**TD**) learning is a combination of Monte Carlo method and Dynamic Programming (**DP**) method [15]. **TD** is similar to Monte Carlo approach in the sense that it learns directly from a row of experience without the environment model. **TD** also updates estimates partly using other existing learned estimates without waiting for the final reward. This approach is similar to **DP** and is also called "bootstrapping".

There are two methods of Temporal Difference learning, On-Policy and Off-Policy learning. On-Policy learning focuses on learning the value of the policy used to make decisions. The value function (of On-Policy learning) is updated using the result of the executed action. Off-Policy learning focuses on learning the behavior of the policy [16]. The value function (of Off-Policy learning) can be updated using hypothetical actions that have not been tried.

2.1.2 Q-Learning

One of the challenges facing the learning agent in RL problem is the separation between exploitation and exploration. When the agent picks the best estimated value for an action, the agent performs what is called a "greedy" selection. Agents that select "greedy" actions are exploiting their current knowledge of estimated action [15]. However, agents are also expected to explore other actions (none-maximum) to improve their policy. An agent may find others (none-maximum) estimated actions to return higher reward in the future using exploration. In this research paper, we address exploitation/exploration dilemma by using the e-greedy method. The agent picks the best action (greedy) with probability "e" otherwise explores a random action [11].

One of the most famous and important learning algorithms in reinforcement learning is Q-Learning. Q-Learning is described as a breakthrough in the field of RL [15]. Q-Learning is an off-policy Temporal Dereference control algorithm whereby it can tell the agent how good an action is given a specific state. The algorithm is defined as follows [11], [16]:

- Let $Q_{\pi}(s, a)$ be the value of performing action **a** is state **s** using policy **p**.
- Let α be the earning rate where $0 \leq \alpha \leq 1$.

1. Initialize Q(s,a) to small random values
2. Observe the current state, **s**
3. Pick an action using policy e-greedy
4. Perform the action, observe next state s', and reward **r**
5. $Q(s, a) \leftarrow (1 - a)Q(s, a) + a(r + max_{a'}Q(s', a'))$
6. repeat the process until state **s** is terminal

2.2 Game Theory

Game theory is the language of applied mathematics that is used in a wide variety of areas such as social science, economy, political science, computer science and biology [3]. Game theory is addressed in the form of a strategy game where individuals are presented with strategic situations. For the purpose of this research paper, individuals in a game are considered agents. The agents success in strategic situations is dependent on the agents actions. Our intention is to capture agents behavior when playing games. There are three main forms of games in Game Theory. The first is the *Normal and extensive form* which represents non-cooperative games. Then there is the *Characteristic function form* which is used to describe cooperative games [18]. Last but not the least is the *Partition function form*, which is similar to *Characteristic function* however it does not ignore the fact that coalitions can also depend on the way players are distributed. For the purposes of our study, we focus on *Normal form. Normal form* defines games in payoff matrix structure which represents players choices and the reward for their actions The game consist of two players, each represented by one position, row and column.

2.2.1 Prisoner's Dilemma

One of the classical strategy games is "Prisoner's Dilemma". In this game players (prisoners) can decide to either cooperate or defect. One such hypothetical situations is that of two suspects who get arrested by the police for committing a crime. The lack of evidence forces the police to put each suspect in a separate holding cell. Table 2.1 shows what sentence each suspect will receive for their choices (cooperate or defect), each value represents jail-time in years. The

	Suspect-B cooperate	Suspect-B defect
Suspect-A cooperate	4,4	0,5
Suspect-A defect	5,0	1,1

Table 2.1: prisoner's dilemma reward matrix

dominant strategy for both "rational" suspects is to defect (safest bet for a suspect regardless of the other suspect choice) [4].

Szolnoki presented an interesting case-study that demonstrated how to gain an effective pay off for the prisoners dilemma game [2]. The study shows how the interaction of seemingly selfish agents in a scale-free network eventually will lead to cooperative behavior. Even though agents strive to maximize their personal reward. Strategies for maximizing personal benefits flow from larger to smaller agents. A normalization parameter was introduced into the heterogeneous network to encourage cooperation. The result showed a decrease of cooperative behavior to a point where it reaches zero. The decrease however is completely overturned when fully normalized payoff limit is reached. The study also compared strategies adopted by agents with their degree and location/position in the network.

Stephen and *Boyd* argued that continuous version of iterated prisoner's dilemma game is better for many real-world situations [9]. In the discrete version of iterated prisoner's dilemma, agents are engaged in the game with two options, cooperate or defect. The continuous version suggests that agents can choose any level of cooperation, from zero to one. Stephen showed that agents adopting generous strategies leads to their achieving the maximum possible payoff. However, the generous strategy works as advertised to an extend but later the strategy experiences a break point. The authors explained that the breaking point is influenced by un-cooperative strategies and therefore they suggested a solution to destabilize non-cooperative equilibrium.

The two studies presented above are different from the aim of our research paper in the following respect. *Szolnoki* case-study did not specify any use of learning algorithm. *Stephen* and *Boyd* on the other hand did not specify the use of any networks. We attempt to allow agents in different scale-free network structure to continuously play prisoners dilemma and observe the behavior of cooperating/defecting using different learning algorithms. Hence in a manner of speakingm we are combining both these theories into a single experiment in an effort to build a complete picture and factor in the effects of all possible variables.

2.2.2 Battle of the Sexes & Chicken-Hawk

Other problems we also explore in this research paper are "Battle of the sexes" and "Chicken-hawk". Battle of the sexes is usually explained as a couples' desire to attend an event(Opera or Boxing match). The husband prefers to watch the boxing match and the wife prefers to watch Opera. However, the couples wish to be together rather than ending up in separate events [10]. Table 2.2 represents reward couples receive for their choices. The two pure strategy Nash equilibria exist in (Opera, Opera) and (Boxing, Boxing). If the husband knew for certain that his wife is going to the Opera, then the husband joins his wife. The opposite is true for the wife, hence couples have no incentive to deviate from their partners plan. A mixed strategy exists when couples differ and each goes to their preferred event. The mixed strategy for each couple is inefficient in cases where a partner is better off going to his/her less favorite event than going to a combined event.

	Husband -Opera	Husband- Boxing
Wife -Opera	3,2	0,0
Wife -Boxing	0,0	2,3

Table 2.2: battle of the sexes reward matrix

The "chicken-hawk" game also known as the game of "chicken" is expressed in a form of two drivers. The two drivers are traveling on a single lane road traveling in opposite directions. If both drivers stay on course -“Dare”, they will crash (both receive no reward) Table 2.3. If one driver moves out of the way-“Chicken”; the other driver will receive the most possible reward [13]. The two pure strategy Nash equilibria are (Dare, Dare) and (Chicken, Chicken). The mixed strategy exists when both drivers “Dare” with probability of 1/3 [17]. The game of chicken differs from prisoners dilemma in the sense that mutual defection is the worst outcome. In prisoners dilemma, the worst scenario results when one suspect cooperates while the other suspects defects.

	Dare	Chicken
Dare	0,0	7,2
Chicken	2,7	6,6

Table 2.3: chicken reward matrix

2.3 Networks

There are number of networks that are significant and suitable to use with the simulator (network must be of GML format). Networks such as small-world, random, social and scale-free have a significant characteristics that can exhibit real-world problems [19]. The case study we present focuses on scale-free networks.

2.3.1 Scale-Free Networks

In 1998, physicist *Alber-Laszio Barabasi* along with his colleagues at the University of Notre Dame attempted to map the World Wide Web [5]. The physicists expected a notion of random connectivity of the Web. Instead, they discovered a feature pattern within the connectivity of the World Wide Web. The map revealed that a few highly connected WebPages were holding the World Wide Web together. More investigation showed that the distribution between WebPages follow the power law. Hence, Scale-Free network defines network whose degree distribution follow the power law.

The discovery of Scale-Free networks was not limited to the World Wide Web. Scale-Free networks can be leveraged to explain behaviors from business process models to complex organic phenomenon. A Variety of complex systems share an important characteristic of Scale-Free networks. The existence of heavily connected nodes (or hubs) that have an overbearing effect on the way the network functions. It is important to note that the overall ratio of the existence of these hubs remains constant when the network scales up. Hence the term “scale-free”.

Sandip and *Airiau* presented an interesting study of social learning through continuous interactions between agents [14]. The study investigated a bottom-up process for the emergence of social norms. The proposed learning frame work allowed agents to repeatedly interact with other agents and eventually evolve a useful social norm. The study also investigated the effects of population size and different learning algorithms. In one example, the study showed that the higher the population is, the longer it takes for the population to converge using WoLF-PHE learning algorithm. *Sandip* and *Airiau* did not specify the population structure or whether it demonstrates a scale-free property. In this research paper, we would like to use different scale-free networks to observe the performance of heavily connected nodes (agents) when playing a strategic game. In addition, we intended to investigate the impact of agents on their neighbors as well as determining whether the position of a heavily connected agent proves beneficial to itself or its surrounding agents.

2.4 Previous Simulators

Several simulators were developed to study networked systems. The *NS Network Simulator* is developed in C ++ and OTcl by UC Berkeley [6]. The simulator simulates a variety of IP networks using different protocols and traffic behavior. In addition, it facilitates different management techniques for router queues and routing algorithm. The simulator is discrete and event driven, its main focus is the physical network infrastructure and hence it is mainly used in communication network research. It worth to mention that *NS simulator* is also available in Java but not as complete as its C++ version. The developers of the Java version whom are independent of Berkeley hope to make it more accessible to programmers who are not familiar with object TCl. The simulator is complex and concentrates on network communications that are not suitable for MARL analysis. The *Multi-Agent Coordination Simulator* is one of the state-of-the-art project developed by the Laboratory of Artificial Intelligence and Computer Science at the University of Porto (LIACC) [12]. The simulator is used in areas of developing methodologies for team coordination, such as robotic soccer, search and rescue and cyber-mouse. However, the simulation does not support networks where we intend to evaluate network properties with MARL algorithm. *Abdalla* and *Lesser* presented an interesting multi-agent simulator that uses reinforcement learning [1]. A key feature they investigated is the change of the underlying network during which the agent is expected to optimize the learning process. The simulator uses MARL and supports networks but does not support strategy games. Our proposed simulator intends to use large scale network and support different games.

Chapter 3

The Simulator Architecture and Flow

As part of this study we have developed a Token-Base Agent Network Simulator (**TANS**). It is a new multi-agent simulator that supports different network structures, multi-agent learning algorithms and different strategic games. In this chapter, we examine the main components of the simulator and present their functionality. Figure 3.1 shows software block diagram.

3.1 Architecture

TANS consists of four classes, *ConfGame*, *AGame*, *Agent* and *Knowledgeboard* and uses three input files, *main.sim* (which holds main configuration parameters), *network.gml* (which contains information about network) and *game.gm* (which contains information about game played). The output result of the simulation is written into folder specified in *main.sim*; the user can choose to obtain different presentation of the simulation data by setting each available result option to *on* or *off*. Refer to the block diagram of the simulator.

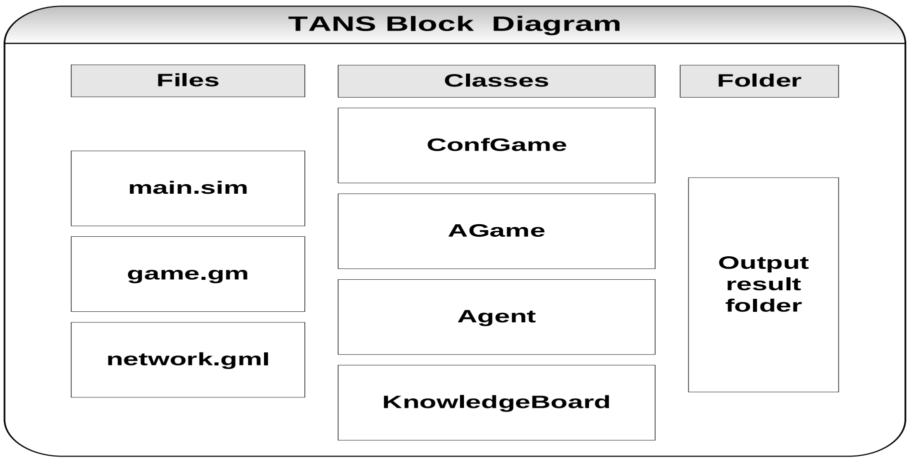

Figure 3.1: Software Block Diagram

3.1.1 Game Configuration

The *ConfGame* class initializes the simulator based on the *main.sim* file. This class creates a network of agents and loads a game board. After the successful upload of the game configuration, the simulation starts. Table 3.1 show list of properties for the simulator.

Term	Description
game	name of the file that consist game matrix
network	name of the file that consist network structure in GML format
output	output folder path for results
gprobability	token generation probability (0.0 - 1.0)
tprobability	transfer token to neighbor probability (0.0 - 1.0)
strategy	strategy on agent or strategy on neighbor (a or n)
greedy	percentage of time agent makes a greedy choice (0.0 - 1)
alfa	Alfa variable used for learning algorithm (0.0 - 0.9)
Iterate	number of iteration for the simulation (real number)
summary	summary output result at every variable iteration (real number)
gameid	seed number for random object when created (real number)
edge_ action	print edge interaction information (on or off)
gml_ network	print interaction result in GML structure (on or off)
game_ summary	print game summary result (on or off)
agent_ summary	print agent summary result (on or off)
agent_ interaction	print agent interaction summary result (on or off)

Table 3.1: List of properties available for simulator

3.2 Simulation Process

The simulator begins by generating tokens and distributing them across agents in network with a fixed probability distribution over agents specified in *main.sim* file, labeled as "gprobability". A token represents the agent's right to pay the game, hence the simulator only selects agents that have at least one token.

The selected agent chooses the best neighbor available to play with. The decision process is explained in section 3.4. Both agents then choose their best action and their reward is given accordingly. The greedy decision to choose the best neighbor/action is controlled by "greediness rate" specified in main.sim file, labeled as "greedy". The selected agent continues to play until it has no more tokens. The simulator then chooses another agent and the former process continues until there are no eligible agents (agents with at least one token) remain in the network. The simulator regenerates tokens after all tokens with agents are consumed and repeats the simulation process again. The *regenerate-tokens* process is considered a new time step and continues to be active until the number of pre-specified iterations, labeled as "iterate" in *main.sim* file is reached. Algorithm 3.1 explains the simulation process above.

The interaction of agents affects the distribution of tokens in the network. An agent that has no token assigned to it may acquire a token from its neighbor. A fixed probability labeled as "tprobability" specified in *main.sim* file is used to determine whether a token gets transferred to a neighbor. Transferred tokens however do not give an agent the right to play within the current time step, this temporary status is retained until the next time step where all transfer tokens are considered real tokens. The distribution of tokens is deliberately unequal, and it is meant to create an imbalance in the network to simulate a dynamic environment.

The concept of token exist in many real world domain, including *packet routing* (token is a packet), *distributed task allocation* (token is a task request) and *rumor spreader* (token is a rumor). We intend to analyze relationship between token transfer and the overall performance of strategic games in different scale-free networks.

Algorithm 3.1: Start Game

```
for number of iterations the user specified do
  $tokenExist \leftarrow true$
  generateToken()
  while $tokenExist$ do
    $gameArray \leftarrow myGame.listIterator()$
    $refAgent \leftarrow null$
    $tokenExist \leftarrow false$
    while $gameArray.hasNext()$ do
      $refAgent \leftarrow gameArray.next()$
      while $refAgent.getNumOfToken()>0$ do
        $refAgent \leftarrow playGame(refAgent)$
        $tokenExist \leftarrow true$
      end while
    end while
    if $tokenExist \leftarrow true$ then
      $createNewTimeStep()$
    end if
  end while
end for
```

3.3 Game

The simulator accommodates both symmetric and asymmetric reward matrix structures. The agent selected to play is considered a row player, and its chosen neighbor is considered a column player. The game consists of a single reward matrix that is used throughout the simulation. Table 3.2 shows an example of reward matrix

	0	1
0	1,9	0,0
1	0,0	9,1

	0	1	2	3
0	2,6	1,7	8,3	4,7
1	1,2	0,6	0,3	0,4
2	8,6	9,6	9,9	10,3

Table 3.2: Symmetric & Asymmetric matrix

3.4 Agent & Knowledgeboard

The "Agent" stores two types of information, *general* and *specific*. *General information* comprises of historic data updated after each game the agent plays -including number of games, total reward received and number of times each action is performed. *General information* is stored in the Agent class.

Specific information is data recorded when the agent interacts with other agents, interaction data is stored in the *knowledgeboard* class. Every agent maintains an array of *knowledgeboard* objects, each representing one of its neighbors and additional players. For example, Agent-A has neighbors B, C, D, and Agent-B has no neighbors. Agent-A creates three *knowledgeboard* object instances for each neighbor during the initialization process where as Agent-B will have none. Agent-B will create the *knowledgeboard* record of Agent-A if and only if Agent-A selects Agent-B to play with. As a result, all agents will have an equal opportunity to learn and yield maximum rewards.

Each agent can play either as a "row player" (if the agent has a token and is selected to play) or as a "column player" (if the agent is chosen by one of its neighbor to play). Therefore, the knowledge board contains two array of values, one for playing as a "row player" and the

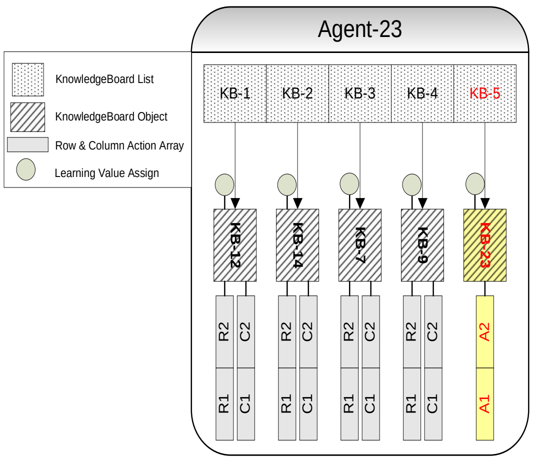

Figure 3.2: Object view of Agent-23 that contains a list of five knowledgeboard object, each representing a neighbor. Agent-23 records interaction information about each neighbor in its corresponding object.

other for playing as a "column player". These arrays hold learning values for each action that agents may select when playing with the specific neighbor.

Figure 3.2 illustrates the *knowledgeboard* of an agent in **TANS**. Agent-23 has a *knowledgeboard* list of five items. Each points to a *knowledgeboard* object which contains two set of arrays; a row array and a column array. The *knowledgeboard* represents players with whom Agent-23 interacted. Agent-23 has played with neighbor-12, neighbor-14, neighbor-7 and neighbor-9. Each array position of this object corresponds to the action performed by Agent-23. The values assigned to these positions are learning values. Row arrays represent variables for which Agent-23 played as a row player (Agent-23 had a token) and column arrays represent when variables for which Agent-23 is chosen by a neighbor. Suppose that Agent-23 has a token and was selected to play. Agent-23 chooses Agent-14 as best neighbor to play with and selects action 2 while Agent-14 selects action 1. Learning value for action chosen by Agent-23 is 0.4, hence Agent-23 updates the row array of the knowledgeboard-14, action 2 value to 0.4. Simultaneously, Agent-14 updates column array of knowledgeboard-23, action 1 value to its corresponding action learning value of 0.6. Figure 3.3 shows visual view of learned values.

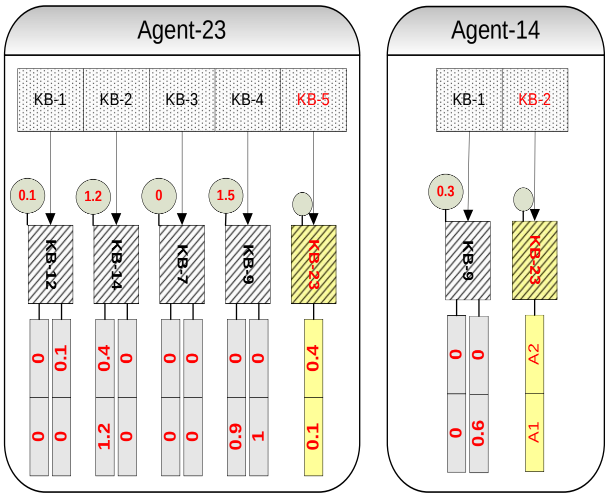

Figure 3.3: Object view of Agent-23 and Agent-14 after interaction. Values in objects are learning values from playing the game.

3.4.1 Strategy Level

There are two strategy options for agents in **TANS**, strategy *on agent* and strategy *on neighbor*. When strategy is set *on neighbor*, an agent finds the best neighbor to play with by examining all array learning values in the *knowledgeboard* list. The *knowledgeboard* object that returns maximum value is considered the best neighbor, and maximum value array position is considered best action number. On the other hand, strategy *on agent* suggests that agents use the sum of learning values attached to each *knowledgeboard* object for best neighbor decision. However, the choice for best action will be in accordance with the agent local decision array in the *knowledgeboard* object belonging to the agent itself. Therefore agents learn one strategy for all neighbors regardless of which neighbor the agents are playing with.

When strategy level is set *on agent* -labeled as “strategy” in *main.sim* file, another learning value is assigned to each *knowledgeboard* object. This value (circle shape object in Figure 3.2) is the sum of all learning values of the *knowledgeboard* regardless of action number. In addition each agent creates a knowledgeboard object by itself (colored in yellow, refer to Figure 3.2) that contains only one array. Refer to Figure 3.3 and the example explained from the previous section (interaction between Agent-23 and Agent-14). Agent-23 updates learning value assigned to the knowledgeboard-14 to 1.2 and updates the array that is under its knowledgeboard (knowledgeboard-23), action 2 to 0.4. Hence, when strategy is set for an agent, it uses one array to determine which action is the best action for all its neighbors regardless of agents role as a “row player” or “column player”.

From Figure 3.3, suppose that the strategy level is set *on agent*. Agent-23 has a token and is selected to play. The best neighbor for Agent-23 is neighbor-14 since the maximum value for all learning arrays (considering row array only) is 1.2 which resides in knowledgeboard-14. The best action for Agent-23 is action number 1, the array position of the maximum value 1.2. When strategy is set *on neighbor*, Agent-23 examines learning values of each *knowledgeboard* (circle shape object in Figure 3.3), hence the best neighbor is neighbor-9 with maximum value 1.5. For best action, Agent-23 only examines its own knowledgeboard-23, the maximum learning value is 0.4, hence the best action is action 2.

3.5 Result Data

TANS provides the capability to output five different structured data as a result of running the simulator. The user can choose to select which result data to obtain by setting print values to “on” or “off”. In Table 3.1, the last five items correspond to output options. In addition to summary data about agent interaction and overall performance, **TANS** facilitates output data per user specific interval during simulation. Table 3.3 shows example of GML formatted network result created after the simulator terminates. See Appendix A for all output structure.

File Name		networkout.gml	
Summary		show result of simulation in gml format	
parent	Item	Format	Description
..	graph	object	graph structure of network in gml format
graph	node	object	object represent agent
graph	edge	object	object represent edge
node	id	Integer	Agent ID
node	label	String	Agent Label
node	play	Integer	Number of games Agent ID played
node	actionR0	Double	Percentage of action row 1 Agent ID chose
node	actionR1	Double	Percentage of action row 2 Agent ID chose
node	actionC0	Double	Percentage of action column 1 Agent ID chose
node	actionC1	Double	Percentage of action column 2 Agent ID chose
node	reward	Integer	Total reward Agent ID received
edge	source	Integer	Source Agent of the edge
edge	target	Integer	Target Agent of the edge
edge	value	Integer	Number of games source agent played using this edge
edge	reward	Integer	Total reward Agent ID received using this edge

Table 3.3: Network output values in GML format

Chapter 4

Experiments & Results

In this chapter, we show number of experiments conducted using **TANS** and present some preliminary analysis that should be subject to further study.

4.1 Experimental Setup

There are several factors that can affect the overall performance result of the simulation, these factors are:

- Learning algorithm used
- The games played
- The network structure
- Token transfer probabilities (transfer and generation)
- The exploration rate
- The learning rate α
- The Strategy level on (agent or neighbor)

The parameters used in the experiments presented in this chapter are the following:

1. Two Network Structure: *Neural* and *NetScience*
2. One Learning Algorithm: *Q-Learning*
3. Three Games: *Prisoner's dilemma*, *Battle of the sexes* and *chicken-hawk*. Table 4.1
4. The exploration rate: is set to%10

prisoner's dilemma		battle of the sexes		chicken-hawk	
4,4	0,5	3,2	0,0	0,0	7,2
5,0	1,1	0,0	2,3	2,7	6,6

Table 4.1: Games used in experiment

4.2 The effect of learning rate and strategy level

The following experiment examines the effect of learning rate ($0 \leq \alpha \leq 0.9$) on average reward received when playing games. Figure 4.1 plots the average reward received at pulse 100,000 as a function of learning rate with strategy level set *on neighbor*. Agents playing *battle of the sexes* and *chicken-hawk* return a higher reward with learning rate > 0. Agents playing *prisoners dilemma* yield the maximum reward with no learning. The maximum reward received when playing *battle of the sexes* is at learning rate 0.1 and 0.4. The maximum reward received when playing *chicken-hawk* is at learning rate 0.2 and 0.3 where then the average reward decreases as the learning rate increase. Figure 4.2 plots the same experiment but with strategy level set on agent. The effect of learning rate on average reward received appears to be closely similar with the two different strategy levels. However, agents playing *battle of the sexes* yield a significantly higher average reward when agents adopt strategy level *on neighbor*.

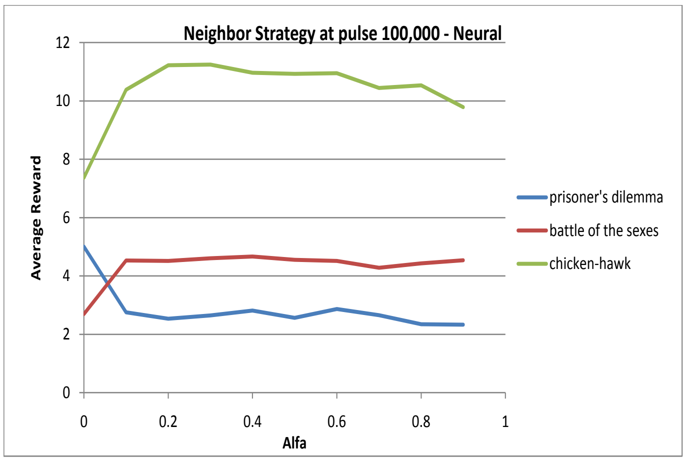

Figure 4.1: Change in average reward over learning rate, neighbor strategy

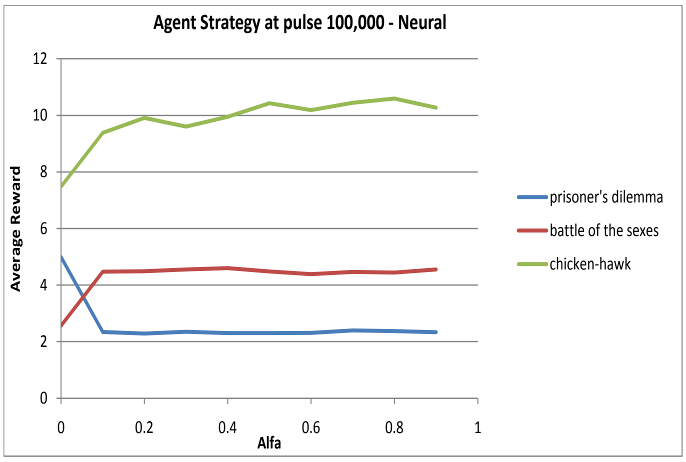

Figure 4.2: Change in average reward over learning rate, agent strategy

4.3 The effect of token transfer probability and strategy level

The following experiment examines the effect of token transfer probability ($0 \leq token \leq 0.9$) on average reward received when playing games. Figure 4.3 plots the average reward received at pulse 100,000 as a function token transfer probability with strategy level set *on neighbor*. We notice a slight decrease in average reward as token transfer probability increases when playing *chicken-hawk*. On the other hand, agents playing *prisoners dilemma* received slightly higher reward as token transfer probability increase. Figure 4.4 plots the same experiment but with strategy level set *on neighbor*. Figure 4.5 plots the average reward at pulse 100,000 as function of token transfer probability with both strategy levels. The difference in strategy level is significant when playing *battle of the sexes*; agents that adopt strategy level *on neighbor* yield higher average reward. This interesting observation suggests an important effect of strategy level on *battle of the sexes* game.

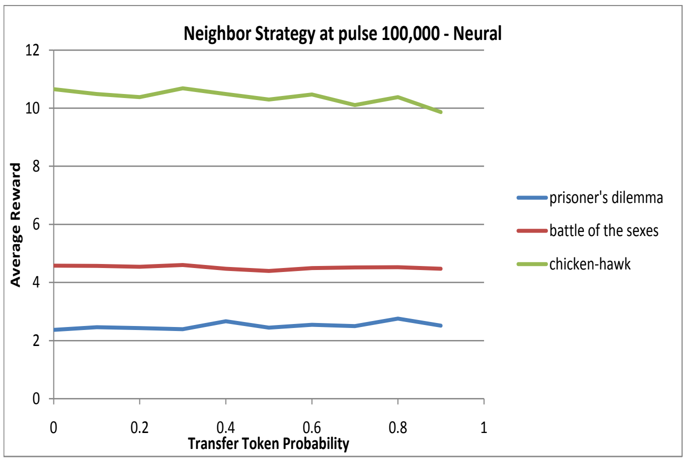

Figure 4.3: Change in average reward over transfer token, neighbor strategy

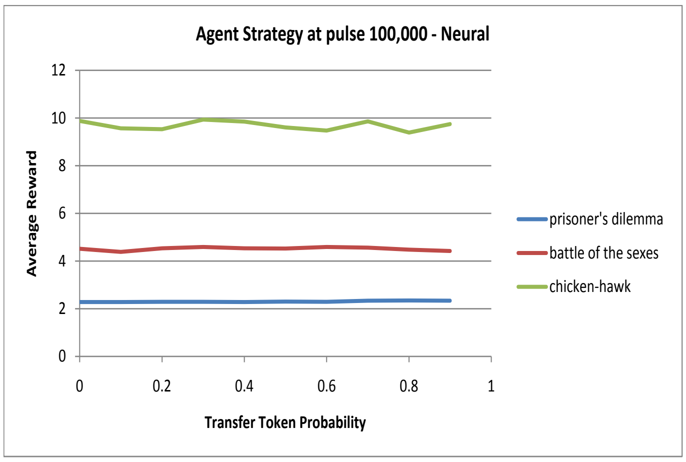

Figure 4.4: Change in average reward over transfer token, agent strategy

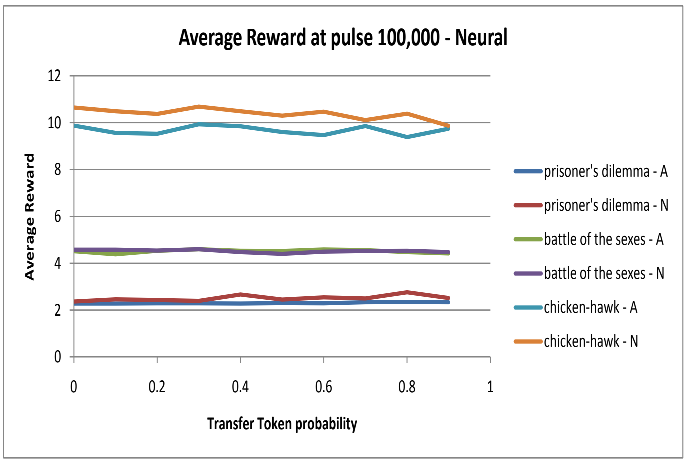

Figure 4.5: Chane in average reward over Agent and neighbor strategy, Neural

4.4 The effect of token transfer and generation probability in neural network

In previous experiments, the average rewards values were plotted at pulse 100,000 as a function of learning rate and token transfer probability. In this section, we examine the effect of token transfer probability over time. Figure 4.6 plots the average reward over 100,000 pulses as function of the pulse with setting the following properties; token generation probability = %7, token transfer probability = %50 and strategy level on agent. The figure shows that agent playing battle of the sexes converges at approximately pulse 12,000 pulses. To save simulation time, the next experiments are run for 25,000 pulses (since the experiment showed a fairly stable average reward after pulse 15,000).

We experiment the effect of token transfer probability using three strategy games in *neural* network. The token generation probability is set to %7, token transfer probabilities are set to %0, %50 and %90, learning rate is set to 0.1 and strategy level is set *on agent*.

Figures 4.7, 4.8 and 4.9 plot games average rewards over 25,000 pulses as function of pulse with the different token transfer probabilities. Figure 4.9 showed a significant change in the average reward when transfer token probability increases. However, after pulse 10,000; the average reward when probability is at %50 is higher. To confirm this finding, we run three more simulations with different seeds (of the random generator) to obtain a clearer view of the effect (change in token transfer probability on average reward). Figure 4.10 plots the result of the three simulations. It is clear that there are certain properties in battle of the sexes or neural network that reacts greatly to the change of token transfer probability.

4.5 The effect of token transfer and generation probability in netscience network

In the following experiments, we examine another network (netscience) and compare it to the same experiment conducted using *neural* network. Figures 4.11, 4.12 and 4.13 plot games average rewards over 25,000 pulses as a function of pulse with different token transfer probabilities. The figures show no significant reaction to the change of transfer token probability. Unlike *battle of the sexes* in *neural* network Figure 4.9, the game here showed no noteworthy change in average reward as a result of increase in transfer token probability.

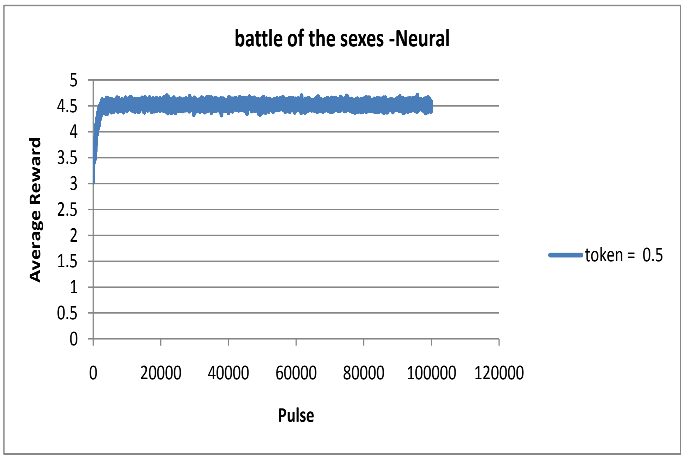

Figure 4.6: Average Reward stabilizes after pulse 25,000

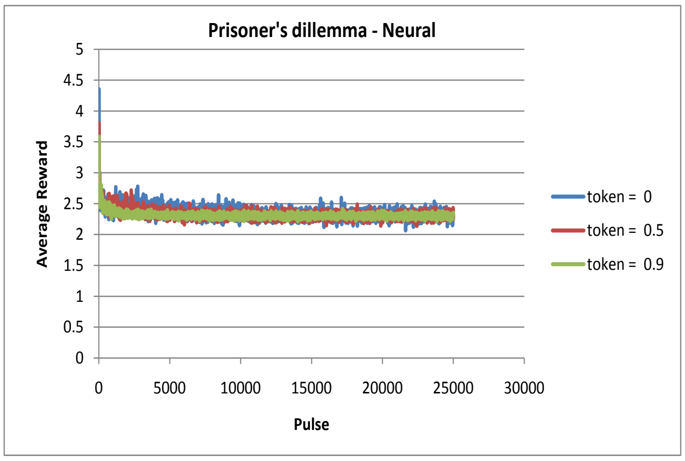

Figure 4.7: Change in average reward over transfer token, Neural

4.6 Constant number of tokens

An increase in transfer token probability is a direct increase in number of games played. We would like to keep the average number tokens constant over different probability, suppose that:
Probability of generating token $= \gamma$
Probability of transferring token $= \beta$
Exact number of tokens generated per agent that are still active is
$= [1 + \beta^2 + \beta^3 + ...] = \sum_{i=0}^{\infty} \beta^i$
$= [\frac{\gamma}{1-\beta}]$
The previous simulation parameters we used, *game probability= 0.07* and *transfer probability= 0.8*
$= [\frac{0.07}{1-0.8}]$
$= 0.35$, i.e on average %35 of agent had a token to play
Consider for directed network, only agents with outgoing edges can play with other agents, therefore agents with no outgoing edges cannot be chosen to play. Thus, the average number of games cannot be constant because tokens have to disappear one reaching a leaf agent.

Figure 4.14 and 4.15 plot the average reword over 25,000 pulse as a function of pulse using neural directed and undirected network. We examine agents playing *battle of the sexes* while 'trying' to keep number to tokens in the game constant. We observe that directed neural network converges faster than undirected neural network. The properties used in this experiment were the following:

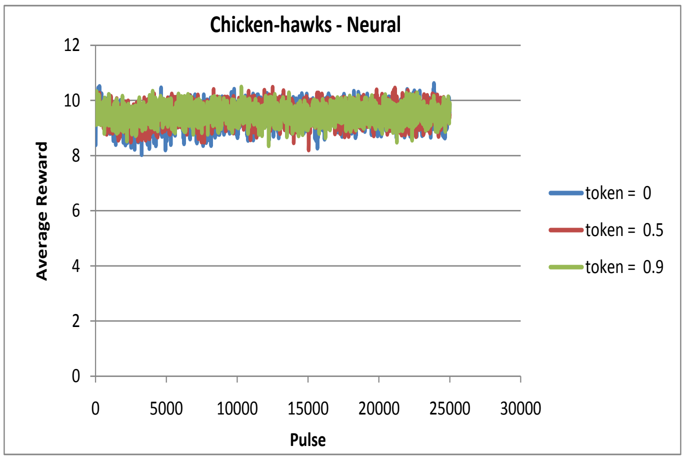

Figure 4.8: Change in average reward over transfer token, Neural

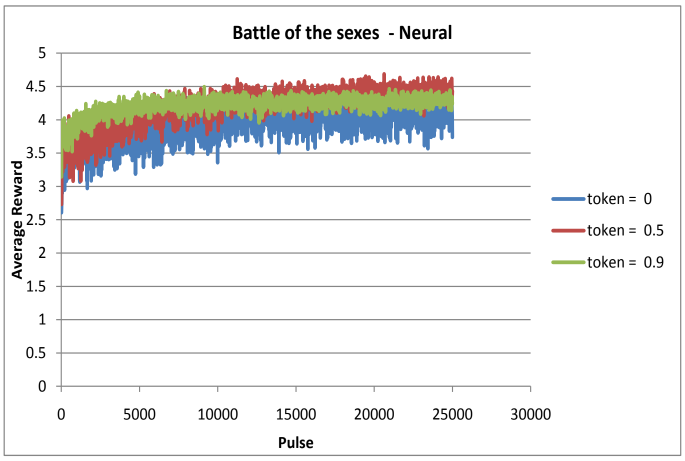

Figure 4.9: Change in average reward over transfer token, Neural

- token generation probability = %35 and transfer token probability = %0
- token generation probability = %17.5 and transfer token probability = %50
- strategy level *on agent*

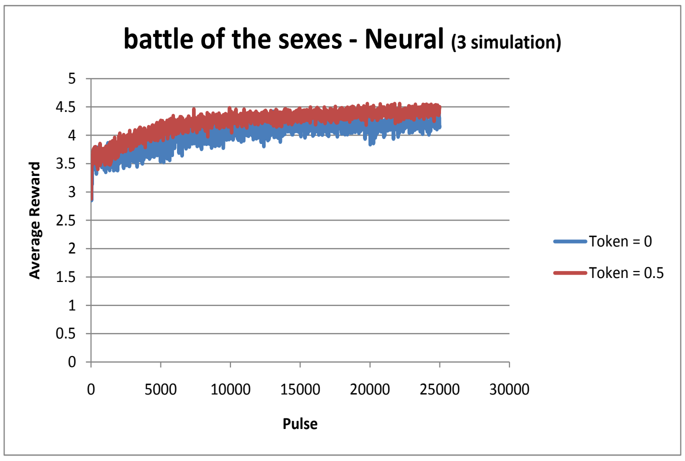

Figure 4.10: Change in average reward over transfer token playing battle of th sexes, Neural

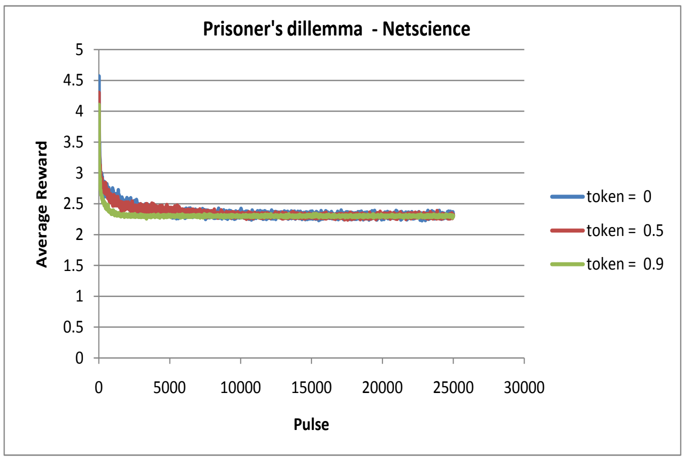

Figure 4.11: Change in average reward over transfer token, Netscience

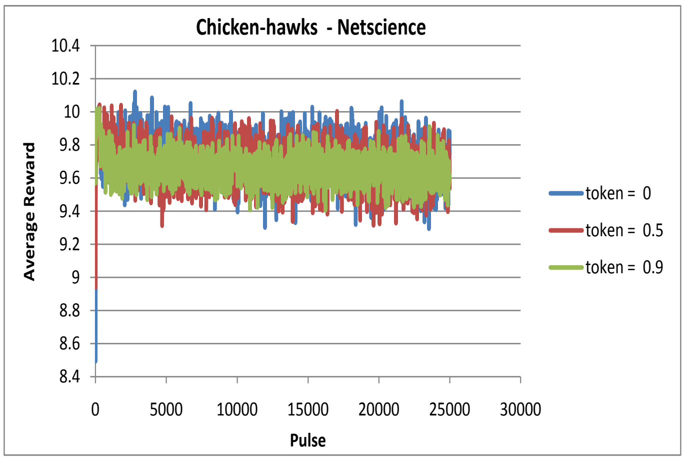

Figure 4.12: Change in average reward over transfer token, Netscience

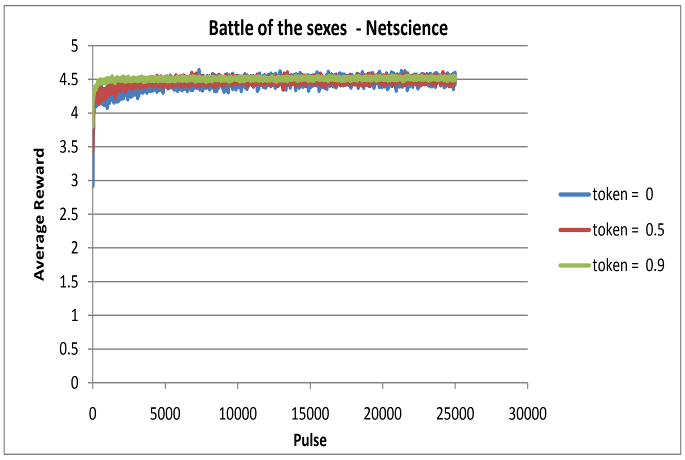

Figure 4.13: Change in average reward over transfer token, Netscience

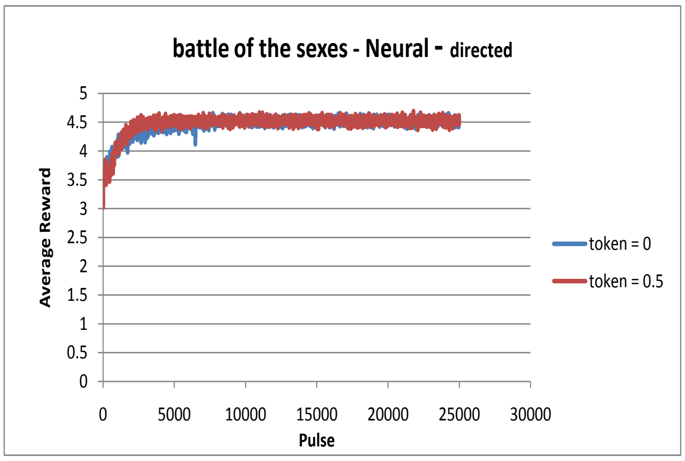

Figure 4.14: Change in average reward over transfer token playing battle of the sexes, Neural -directed

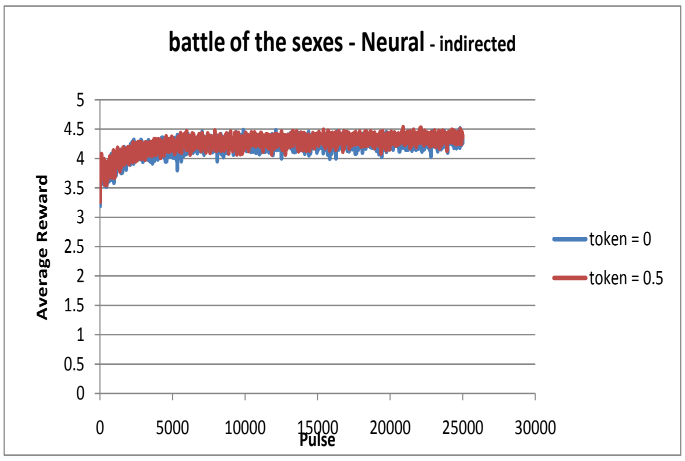

Figure 4.15: Change in average reward over transfer token playing battle of the sexes, Neural -undirected

Chapter 5

Conclusion and Future Work

In this thesis, we developed a new multi-agent network simulator that provides the facility to study a combination of MARL algorithms, strategic games, and networks. We introduced the technique of using tokens in the simulator. Tokens are considered the agent's right to play the game. Tokens are assigned based on a pre-determined probability distribution over agents. The concept of tokens exists in many real-world domains, including distributed task allocation (token is a task request) and packet routing (token is a packet). The token helps us to recognize new dependencies between agents that go beyond the simple one-hop connection. The simulator is also designed to facilitate simple change of configuration parameters through a single centralized file. In addition, the simulator captures all useful interaction data in a structured format that allows for easy and quick analysis.

We conducted a preliminary analysis of Q-learning algorithm in two scale-free networks (neural and netscience) and three different strategic games (battle of the sexes, prisoners dilemma and chicken-hawk). The results showed a significant reaction to the increase of tokens when agents play *battle of the sexes* in *neural* network. The increase in token transfer probability yielded a higher reward and a faster conversion. It is interesting that this observation was only examined when agents played *battle of sexes* in a *neural* network.

Our preliminary analysis focused mainly on the reward agents received. However, there are many other promising areas that the simulator can assist by providing useful data for analysis. For example, it can be leveraged to help evaluate decisions made by agents regarding which neighbors to play with over time.

One theory we can then postulate based on our observations is that well-connected agents will always select poorly-connected agents within the network under the presumption that the later has less experience and hence, provides an opportunity to gain maximum reward. However this interaction can prove to be mutually beneficial; if we conclude that the poor-agents stands to increase its own experience through these interactions regardless of the fact that it will most likely lose in the first few rounds. We hence put forth the idea that the poor-agent in fact can be in a better suited position to win compared to a medium connected agent; by virtue of gaining a transfer token from the highly connected agent. While still un-proven, there is a good chance our simulator can be used to run enough statistical experiments so as to test the validity of these claims. This in fact can serve as a good starting point to continue conducting more analysis.

Appendix A

Result Data Structure

The following tables represent output data after running 'bfTANS, each table is considered a separate file. Except for result 3.3 on page 21, output result format is in plain text where rows are separated by newline and columns by comma.

A.1 Game Summary

Table A.1 presents game summary information per user specific pulse, check property *summary* in Table 3.1 on page 17

File Name		summaryGame.txt	
Summary		Show average reward over period at every time-step user specified	
row N.	Item	Format	Description
1	pulse	Integer	Time step when result obtained from total iteration
2	reward	Integer	Total game reward received
3	game	Integer	Total number of games player
4	average	Double	Averaged reward

Table A.1: Game simulation summary values

A.2 Agent Summary

Table A.2 presents summary information for each agent created after the simulator terminates.

A.3 Agent Interaction

The simulator outputs Table A.3 for all agent in network. Information are written per user specific pulse to monitor agents overall performance during simulation process.

A.4 Edge Action

The simulator outputs Table A.3 for all edges in network. Information are written per user specific pulse to record edges interaction history and *knowledgeboard* information used during

File Name		summaryAgent.txt	
Summary		Show average reward of all Agents in network	
row N.	Item	Format	Description
1	agent	Integer	Agent ID
2	degree-out	Integer	Agent ID degree-out
3	degree-in	Integer	Agent ID degree-in
4	games	Integer	Number of games Agent ID played
5	reward	Integer	Total reward Agent ID received
6	average	Double	Average Reward of Agent ID

Table A.2: Agent simulation summary values

File Name		Agent-agentID.txt	
Summary		Show information about agents and their number with total reward	
row N.	Item	Format	Description
1	pulse	Integer	Time step when result obtained from total iteration
2	agent	Integer	Agent ID
3	neighbor	Integer	Agent ID's neighbor
4	neighbor-LV	Double	Learning value for neighbor Agent ID recorded
5	reward	Integer	Total reward Agent ID received when play neighbor

Table A.3: Agent interaction values during simulation process

simulation process.

File Name		edge-source-destination.txt	
Summary		Show information about edge usages, destination Agent and recorded knowledgeboard	
row N.	Item	Format	Description
1	pulse	Integer	Time step when result obtained from total iteration
2	source	Integer	Source Agent ID of the edge
3	destination	Integer	Destination Agent ID of the edge
4	degree-out	Integer	Source Agent degree-out
5	degree-in	Integer	Source Agent degree-in
6	Games	Integer	Number of games source Agent played
7	source-row player	Integer	Number of games source Agent played as row player
8	source-col. Player	Integer	Number of Games source Agent played as column player
9	source-row Vs. des.	Integer	Number of time source Agent played with destination Agent as row player
10	source-col. Vs. des.	Integer	Number of time source Agent played with destination Agent as column player
KnowledgeBoard about destination Agent source Agent recorded when using edge-S-D			
11	LV-R1	Double	Learning value for action row one
12	N. R1	Integer	Number of times action row one used
13	sum-LV-R1	Double	Sum of all learning value for action row one
14	LV-R2	Double	Learning value for action row two
15	N. R2	Integer	Number of times action row two used
16	sum-LV-R2	Double	Sum of all learning value for action row two
17	LV-R1 > LV-R2	Character	Learning value row action one is greater than row action two (T or F)
18	LV-R2 > LV-R1	Character	Learning value row action two is greater than row action one (T or F)
19	LV-C1	Double	Learning value for action column one
20	N. C1	Integer	Number of times action column one used
21	sum-LV-C1	Double	Sum of all learning value for action column one
22	LV-C2	Double	Learning value for action column two
23	N. C2	Integer	Number of times action column two used
24	sum-LV-C2	Double	Sum of all learning value for action column two
25	LV-C1 > LV-C2	Character	Learning value column action one is greater than column action two (T or F)
26	LV-C2 > LV-C1	Character	Learning value column action two is greater than column action one (T or F)

Table A.4: Edge action values during simulation process

Appendix B

Software Architecture

B.1 Simulation Flow

Figure B.1 illustrates game simulation flow which begins by generating token across network, this process is considered a new time step. The simulator looks for eligible agents and allow them to play until there are no more agents with token. If number of iteration user specified has no reached, the cycle continues with new time step.

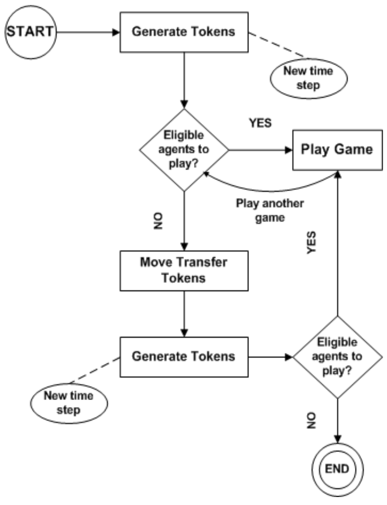

Figure B.1: Game simulation flow

B.2 Functional Model

B.2.1 Game configuration

ConfGame class handles setup configuration of the simulation. It read simulation properties from *main.sim* file the user specifies. It also read in network structure in GML format and creates all necessary nodes and properties to start the simulation. Figure B.2

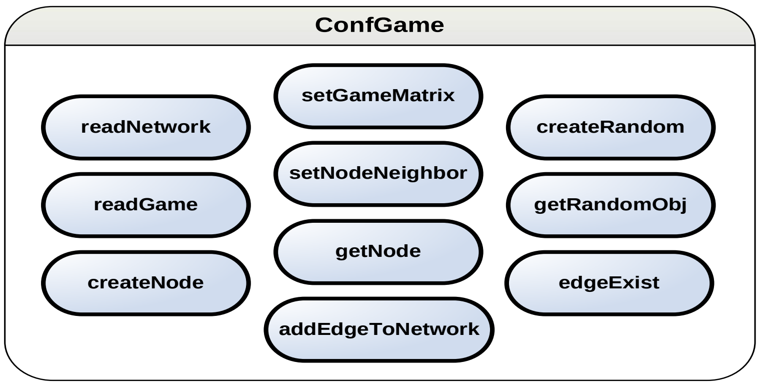

Figure B.2: ConfGame class function

B.2.2 Agent Game

AGame class starts the simulation, generate tokens, select player, ask player to choose action, calculate result and continue cycle until no agent with token exist in the network. Figure B.3

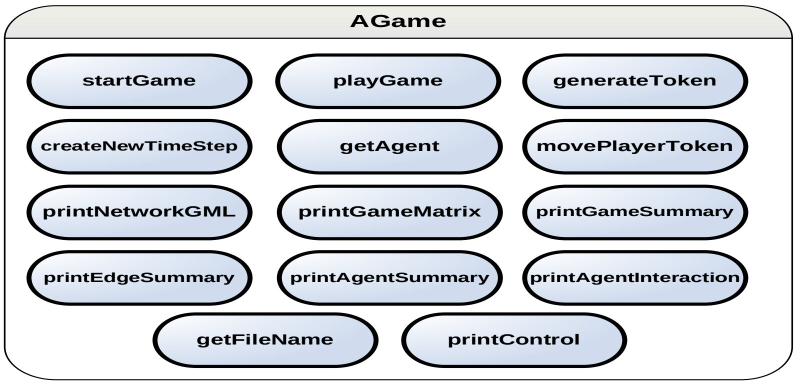

Figure B.3: AGame class function

B.2.3 KnowledgeBoard

Knowledgeboard class contains information about each action the agent played with a corresponding neighbor. The learning value for each action is recorded in an array structure. Class Agent creates this class for each of its neighbor.

B.2.4 Agent

Agent class represent agent in social network, it contains an object of *knowledgeboard* for each neighbor where all interaction information is recorded.

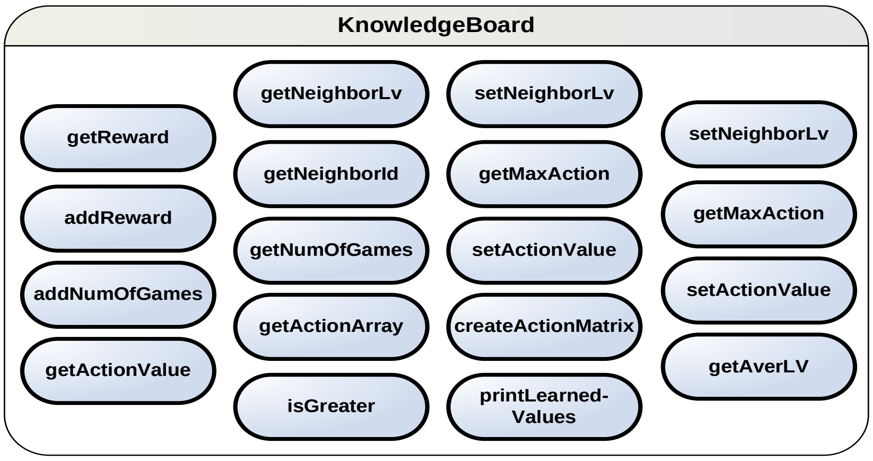

Figure B.4: KnowledgeBoard class function

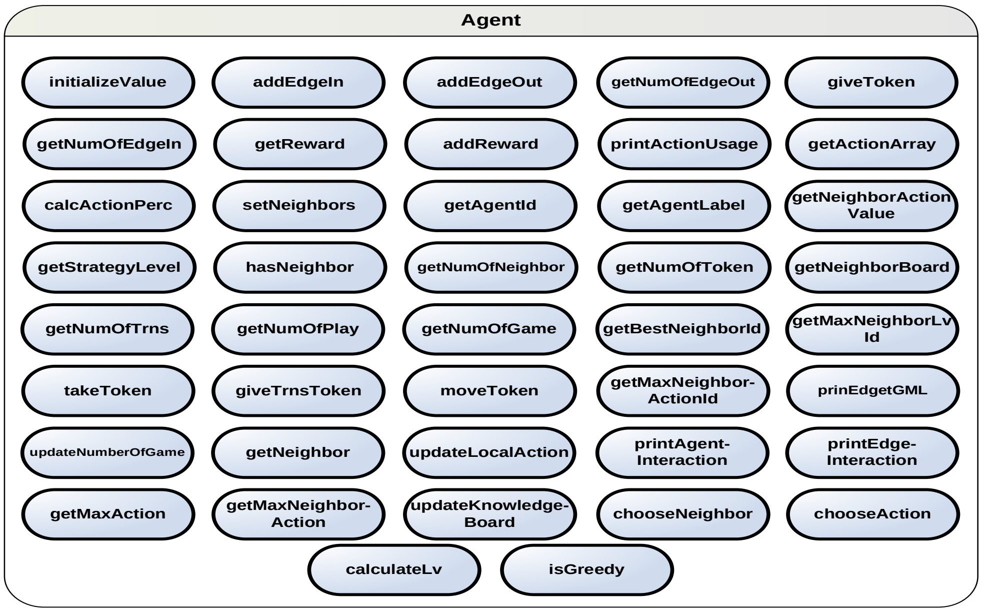

Figure B.5: Agent class function

Bibliography

[1] Sherief Abdallah and Victor Lesser. Multiagent reinforcement learning and self-organization in a network of agents. In *AAMAS '07: Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, pages 1–8, New York, NY, USA, 2007. ACM.

[2] Zsuzsa Danku Attila Szolnoki, Matjaz Perc. Towards effective payoffs in the prisoners dilemma game on scale-free networks. *Physica A: Statistical Mechanics and its Applications*, 2007.

[3] Robert J Aumann. What is game theory trying to accomplish?, 1985.

[4] Robert Axelrod. The evolution of cooperation. *Journal of Economic Behavior and Organization*, 5(3-4):406–409, 1984.

[5] Bonabeau Eric Barabsi, Albert-Lszl. Scale-free networks. *Scientific American*, 2003.

[6] UC Berkeley. The network simulator - ns, 1995.

[7] Patrice Caire and Leendert van der Torre. Temporal dependence networks for the design of convivial multiagent systems. In *AAMAS '09: Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems*, pages 1317–1318, Richland, SC, 2009. International Foundation for Autonomous Agents and Multiagent Systems.

[8] Junling Hu and Michael P. Wellman. Multiagent reinforcement learning in stochastic games. In *Online]. Available: citeseer.ist.psu.edu/hu99multiagent.html*, 1999.

[9] Stephen Le and Robert Boyd. Evolutionary dynamics of the continuous iterated prisoner's dilemma. *Journal of Theoretical Biology*, 245(2):258–267, March 2007.

[10] Martin J. Osborne and Ariel Rubinstein. *A Course in Game Theory*, volume 1 of *MIT Press Books*. The MIT Press, December 1994.

[11] Bryan Pardo. *Machine Learning*, 2005.

[12] António Pereira, Luís Paulo Reis, and Pedro Duarte. A multi-agent system for ecological simulation and optimization. In *EPIA*, pages 473–484, 2009.

[13] William Poundstone. *Prisoner's Dilemma*. Doubleday, 1992.

[14] Sandip Sen and Airiau. Emergence of norms through social learning. In *IJCAI'07: Proceedings of the 20th international joint conference on Artifical intelligence*, pages 1507–1512, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[15] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning*. The MIT Press, 1998.

[16] Gerald Tesauro. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, March 1995.

[17] Wikipedia. Chicken (game), 2009. online; access 20-December-2009.

[18] Wikipedia. Game theory, 2009. online; access 22-December-2009.

[19] Wikipedia. Small-world network, 2010. online; access 09-February-2010.