Learning to Share: A Study of Multi-agent Learning in Transportation Systems

By

Edmond Awad

A Thesis Presented to the

Masdar Institute of Science and Technology

in Partial Fulfillment of the Requirements for the Degree of

Master of Science

in

Computing and Information Science

© 2011 Masdar Institute of Science and Technology

All rights reserved

Learning to Share: A Study of Multi-agent Learning in Transportation Systems

By Edmond Awad

A Thesis Presented to the Masdar Institute of Science and Technology in Partial Fulfillment of the Requirements for the Degree of

Master of Computing and Information Science

August 2011

© 2011 Masdar Institute of Science and Technology

All rights reserved

AUTHOR'S DECLARATION

I understand that copyright in my thesis is transferred to Masdar Institute of Science and Technology.

Author _____

RESEARCH SUPERVISORY COMMITTEE MEMBERS

Dr. Jacob Crandall, Chair, _____

Masdar Institute of Science and Technology

Dr. Iyad Rahwan, _____

Masdar Institute of Science and Technology

Dr. Zeyar Aung, _____

Masdar Institute of Science and Technology

Abstract

Intelligent transportation technologies such as car navigation systems can lead to significant energy savings. One task of a navigation system is to help a driver take an efficient path to its destination. If we consider the Personal Rapid Transit (PRT) system in Masdar City, we can see that implementing an effective navigation system can play a pivotal role in improving its performance.

In this study, we consider a non-centralized navigation system that seeks to route self-interested vehicles efficiently taking into consideration the congestion that might happen when most of the vehicles use one link over the others. This case is known in the literature as a congestion game. In a congestion game, each player chooses a resource (a link) out of many resources to use, and the cost of each resource depends on the number of agents who select it.

This research aims to find an algorithm that, when used in every vehicle, can secure each vehicle a fair cost, while maximizing the social welfare of the group. In so doing, we evaluate the performance of several traditional algorithms in a small, but intriguing network. Our results demonstrate the difficulty of even this simple problem. We then propose and evaluate a family of algorithms on a simpler (and related) problem to better understand the kinds of learning biases algorithms should incorporate in this domain.

This research was supported by the Government of Abu Dhabi to help fulfill the vision of the late President Sheikh Zayed Bin Sultan Al Nayhan for sustainable development and empowerment of the UAE and humankind.

Acknowledgments

First and foremost, I would like to thank God for the wisdom and perseverance that he has been bestowed upon me during this research project, and indeed, throughout my life.

I would like to express my deep and sincere gratitude to Dr. Jacob Crandall, my advisor, for his continuous guidance, support and motivation during the course of this research in Masdar Institute of Science and Technology.

I would like to thank Dr. Iyad Rahwan and Dr. Zeyar Aung, members of my research committee, for their valued feedback and helpful comments on my work.

I would like also to thank my brother, my sister, my friends and classmates for providing a loving environment for me. I would like also to thank Masdar Institute and every faculty member who has taught me a course during my study.

Last but not least, I would like to thank my mother and father for their moral support. It is their love and faith in me that made me the person I am today.

Edmond Awad,

Masdar City, July 21, 2011.

Contents

List of Tables

List of Figures

Learning to Share: A Study of Multi-agent Learning in Transportation Systems

by

Edmond Awad

A Thesis Presented to the

Masdar Institute of Science and Technology

in Partial Fulfillment of the Requirements for the Degree of

Master of Science

in

Computing and Information Science

©2011 Masdar Institute of Science and Technology

All rights reserved

AUTHOR'S DECLARATION

I understand that copyright in my thesis is transferred to Masdar Institute of Science and Technology.

ACCEPTANCE DECLARATION

This thesis has been accepted and approved by Masdar Institute of Science and Technology on January 01, 2011.

EXAMINATION COMMITTEE MEMBERS

Jacob Crandall, *Chair, Masdar Institute of Science and Technology*
Iyad Rahwan, *Advisor, Masdar Institute of Science and Technology*
Zeyar Aung, *Masdar Institute of Science and Technology*

Abstract

Intelligent transportation technologies such as car navigation systems can lead to significant energy savings. One task of a navigation system is to help a driver take an efficient path to its destination. If we consider the Personal Rapid Transit (PRT) system in Masdar City, we can see that implementing an effective navigation system can play a pivotal role in improving its performance.

In this study, we consider a non-centralized navigation system that seeks to route self-interested vehicles efficiently taking into consideration the congestion that might happen when most of the vehicles use one link over the others. This case is known in the literature as a congestion game. In a congestion game, each player chooses a resource (a link) out of many resources to use, and the cost of each resource depends on the number of agents who select it.

This research aims to find an algorithm that, when used in every vehicle, can secure each vehicle a fair cost, while maximizing the social welfare of the group. In so doing, we evaluate the performance of several traditional algorithms in a small, but intriguing network. Our results demonstrate the difficulty of even this simple problem. We then propose and evaluate a family of algorithms on a simpler (and related) problem to better understand the kinds of learning biases algorithms should incorporate in this domain.

This research was supported by the Government of Abu Dhabi to help fulfill the vision of the late President Sheikh Zayed Bin Sultan Al Nayhan for sustainable development and empowerment of the UAE and humankind.

Acknowledgments

First and foremost, I would like to thank God for the wisdom and perseverance that he has been bestowed upon me during this research project, and indeed, throughout my life.

I would like to express my deep and sincere gratitude to Dr. Jacob Crandall, my advisor, for his continuous guidance, support and motivation during the course of this research in Masdar Institute of Science and Technology.

I would like to thank Dr. Iyad Rahwan and Dr. Zeyar Aung, members of my research committee, for their valued feedback and helpful comments on my work.

I would like also to thank my brother, my sister, my friends and classmates for providing a loving environment for me. I would like also to thank Masdar Institute and every faculty member who has taught me a course during my study.

Last but not least, I would like to thank my mother and father for their moral support. It is their love and faith in me that made me the person I am today.

Edmond Awad,

Masdar City, July 21, 2011.

Contents

List of Tables

List of Figures

CHAPTER 1

Introduction

1.1 Problem Definition and Motivation

Transportation plays a vital role in supporting sustainable economic growth, and tackling climate change [56]. For example, intelligent transportation technologies for automobiles can save energy by helping the user take more efficient routes to a destination. Furthermore, Personal Rapid Transit (PRT) systems, such as the system used in Masdar City, are expected to consume less energy, have lower noise and have lower local pollution impacts than conventional public transportation systems [14]. These PRT vehicles have much in common with conventional automobiles, as PRT vehicles move from origin to destination with no intermediate stops [59].

One of the most important aspects in applying the PRT system in Masdar City is implementing an effective vehicle navigation system to route each PRT vehicle through an efficient path to reach its destination station. The vehicle navigation system can be also applied to any transportation system, such as GPS navigation

systems currently used all over the world.

There have been a lot of navigation systems that have been accompanied with good routing algorithms in order to help vehicles reaching their destination efficiently. Some of these algorithms considered the case of being in a dynamic world. In a dynamic world, the time consumed on a specific path is not a fixed value. This happens generally for different reasons like traffic changes on each road, the possibility of blocked roads or lanes, and the changeable flow of vehicles to and from the road. These changeable factors make the world dynamic.

One of the biggest problems with routing in dynamic environments is to accurately estimate the time to be consumed on a possible path. Dynamic routing algorithms generally use the given links' costs to calculate and suggest an efficient path. Many algorithms were developed in this direction and their main focus was to do updates and recalculations in competitive time complexities [12, 23, 46, 47, 68]. These algorithms did not consider how to better estimate the links costs or how to anticipate the behavior of other vehicles running in the same world.

Assume we have a set of vehicles. Each of which is trying to choose one of two paths. The two paths have the same end point, and all the vehicles need to reach this point. If all the vehicles use the same navigation systems, all of the vehicles might be directed to use the more efficient path. This will cause this path to be congested, thus becoming a bad path to use. Worse, if the same set of vehicles face the same problem again on the same node, they may all choose to go for the other path this time due to their bad experience using the more efficient path. This may cause a case of infinite oscillation between the two paths for this group of vehicles. While the real-world problem is more complex, this example illustrates the challenge algorithms must learn to overcome.

This study considers a navigation system in two types of networks. One of them is a small, but intriguing, network with very few nodes that are connected by

only a few links. The other is an even less complex network with two nodes connected by two links. This system considers routing each vehicle efficiently taking into consideration the congestion that might happen when most of the vehicles use one link over the others. This case is known in the literature as a congestion game. In a congestion game, each player chooses a resource out of many resources to use, and the cost of each resource depends on the number of agents who select it.

Environments that have the previously explained characteristics are called multi-agent environments. In these environments, although each agent (or vehicle) selects actions to satisfy its own interests, its actions may lead it to reach a balance with others that is efficient for the whole community. Multi-agent environments borrow some representations, techniques and solution concepts from the field of game theory.

In our study, we consider the case where the vehicles have no communication with each others or with a centralized system. Each vehicle has a navigation system that uses the proposed algorithm. We assume that we monopolize the navigation system market, and thus all vehicles would use the proposed algorithm and there is no chance for a different navigation system in the area. The proposed algorithm learns to secure the vehicle with a fair cost. However, fairness in this case (where all vehicles use the same algorithm) implies that a vehicle shares the costs equally with others and as to minimize the overall cost of the group.

1.2 Thesis Statement - Objectives

This study aims to find a learning algorithm that can learn to route vehicles in multi-agent environments. This algorithm, when used by each vehicle should achieve the following two objectives as quickly as possible:

1. Fairness – minimize the variance in vehicles' costs among the population.

2. Maximize the social welfare – maximize the sum of the vehicles' utilities (minimize their costs).

In this thesis, we evaluate a family of learning algorithms in two simple congestion games. In doing so, we seek to identify effective learning biases for these games.

1.3 Relevance to the Masdar Initiative

Transportation systems, are critical to the sustainable development of Masdar City, Abu Dhabi, and the rest of the UAE. This research aims to make these systems more effective and efficient by helping to improve vehicle routing.

1.4 Thesis Organization

The organization of this study is as follows. In Chapter 2, we provide a background and a literature review for some relevance topics including navigation systems, shortest path problem, reinforcement learning, game theory and multi-agent reinforcement learning. The model of the environments and the specifications of the simulators are explained in Chapter 3. In Chapter 4, we show the performance of conventional algorithms in a small, but intriguing network. However, finding an efficient algorithm in this type of networks is not an easy task for different reasons. Chapter 5 shows the performance of different variations of algorithms in a simpler network. Furthermore, an algorithm that learns to share the costs equally and efficiently with others is proposed. We conclude this study with Chapter 6, where we present final conclusions and discuss future work.

CHAPTER 2

Background and Literature Review

In this chapter, we provide a background on different topics and concepts that are relevant to this study. A literature review is provided for topics that are strongly pertinent to our study. In doing so, we first discuss navigation systems since this study aims to propose a routing algorithm to make navigation systems more efficient. We provide then an overview about the Shortest Path Problem (SPP). SPP is a well studied problem in navigation systems. Our study looks for an algorithm that can help the vehicles to reach their goal through a time-efficient path, which has the same concept of shortest path problem.

Next, a background on Reinforcement learning (RL) is given. RL is a set of techniques that are common in learning from environment. RL has been a good source for algorithms and techniques to be used in one-agent and multi-agent environments. After that, a general overview including the critical tools and solution concepts provided by game theory for multi-agent environments is presented. Finally, a literature review is provided for the different RL techniques that have been

used in multi-agent environments.

2.1 Vehicle Navigation Systems

A vehicle navigation system generally uses GPS to acquire position data to locate the user on a road in the database. This navigation device includes a receiving unit configured to receive information, a storage unit configured to store the received information, and a guidance unit which routes the vehicle using this information. Using the road database, directions can be given to other locations through other roads which are also in its database.

According to Leonard and Durrant-Whyte [52], the navigation problem can be addressed as answering the questions: "Where am I?", "Where do I want to go?" and "How do I get there?". In this study, we are concerned with the third question. Although this question has been studied extensively in the literature [15, 51], we are unaware of an effective algorithm for self-interested independent agents in environments where the cost (time consumption) of each link depends on the number of vehicles using this link, and when agents are unable to communicate with each other.

2.2 Shortest Path Problem

The shortest path problem (SPP) is the problem of finding a path between two nodes such that the sum of the weights of its component links is minimized. One SPP example is finding the quickest way to get from one location to another on a map. In this case, the nodes represent locations or decision points, and the links represent segments of road connecting these decision points. Each link is weighted by the time needed to travel that link.

The problem of finding the shortest path between two nodes is well-studied.

Shortest path algorithms have been a subject of extensive research, resulting in a number of algorithms for various conditions and constraints [25, 30, 39, 40].

2.2.1 Common Shortest Path Problems

Shortest-path algorithms can be used to solve many real-world problems, such as vehicle routing in transportation [63], traffic routing in communication networks [29], pickup and delivery systems [76], website page searching for Internet information retrieval systems [69], and path planning in robotic systems [24].

2.2.2 Variations of SPPs According to Source-destination

There are four variations of SPPs [19]:

- Single-Pair Shortest Path problem (SPSP): Find a shortest path from u to v for given vertices u and v.
- Single-Source Shortest Path problem (SSSP): Find a shortest path from a given source vertex u to any destination vertex v. This problem is discovered by Dijkstra [25]. Many studies tried to solve it [36, 37].
- Single-Destination Shortest Path problem (SDSP): Find a shortest path to a given destination vertex t from each vertex v. By reversing the direction of each edge in the graph, we can reduce this problem to a single-source problem.
- All-Pairs Shortest Path problem (APSP): Find a shortest path from u to v for every pair of vertices u and v. Although this problem can be solved by running a single source algorithm once from each vertex, it can usually be solved faster. This variant was discovered by Floyd [34]. Many studies tried to solve it [26, 46, 23].

2.2.3 Categories of SPPs According to the Properties of the Edges Costs

SPPs can fall into two categories depending on the properties of the edges costs [65]:

1. Static SP: Link costs are assumed to be static and deterministic. Many efficient algorithms have been developed by Bellman [5], Dijkstra [25] and Dreyfus [27] to solve this type of problems.

2. Dynamic SP: Link costs are dynamic. Dynamic SPs fall into two classes: Time-Dependent Shortest Path (TDSP) and Dynamic Shortest Path (DSP). Many studies were proposed for this class of problems [12, 23, 46, 47, 68].

2.2.4 Relevance to this Research

In this study, we focus on the Dynamic Single-Pair Shortest Path problem (DSPSP). The cost of each link depends on the number of vehicles using it. The number of vehicles on each link is not fixed since the vehicles change their decisions over time. This makes the problem dynamic. Since each vehicle in our simulations seeks to move between the same start and end nodes, our problems are single pair SP (SPSP).

Most of the previously mentioned studies focused on speeding up the computation of an efficient path when the edge costs are updated. Furthermore, most of them have assumed the knowledge of edges costs and thus ignored the fact that this data might not be available in reality. The estimation of an edge cost should consider different factors including traffic (the number of vehicles using the edge), distance and capacity.

2.3 Reinforcement Learning

Reinforcement learning (RL) [81, 44] is a sub-area of machine learning concerned with learning from experience how an agent should take actions in an environment to maximize its rewards. The agent must learn not only the immediate reward of taking an action, but also the future states and, through that, all subsequent rewards.

RL methods have been used to train neural networks [83], to control dynamic channel assignment in communications networks [62], and to construct fuzzy logic rule bases for fuzzy control systems [7]. Applications of reinforcement learning in multi-agent systems include soccer [4], pursuit games [82, 22], and coordination games [17]. RL generally works in environments that are formulated as Markov decision process (MDP).

2.3.1 Markov Decision Processes (MDPs)

An MDP [53] is a model for decision making in a dynamic, uncertain world. In a Markov model, there is a set of states and a set of actions possible in each state. There is also a transition function which defines the probability of moving from a state to another state when taking a specific action, and finally a reward function that specifies the immediate reward for taking some action when being in some state.

Markov decision processes are classified as stochastic processes as they have a transition function that makes them nondeterministic in general. The agent starts in some state, takes one of the possible actions, and accordingly receives some rewards. The agent then moves probabilistically to another state depending on the transition function. This process is repeated until the agent reaches a goal state or a state from which it has no possible actions (an absorbing state).

Formally speaking, an MDP is a tuple (S, A, p, r).

- S is a set of states,
- A is a set of Actions,
- The transition function $p : S \times A \times S \rightarrow [0,1]$ specifies the probability distribution of moving from one state when taking some action to another state, and
- The reward function r: $S \times A \rightarrow \mathbb{R}$ returns the reward for each state-action pair.

The immediate rewards can be aggregated in limit average reward or future discounted reward.

Every MDP is assumed to satisfy (or possess) the Markov property. Markov property refers to the memory-less property of a stochastic process. A stochastic process has the Markov property if the effect of an action taken in a state depends only on that state and not on previous history.

MDPs are useful in studying a wide range of problems that are solved by reinforcement learning. RL needs not to know the MDP to solve the problem. If we had a finite space of states and actions, the MDP is called finite MDP which is very important to reinforcement learning. When the transition function has for each state-action pair a value of 1 for one of the states and zero for others, MDP is said to be deterministic, otherwise it is called stochastic MDP.

In an MDP, an optimal policy is defined to be the one that maximizes the expected sum of discounted reward and is undominated. Undominated means there is no other policy that can achieve better rewards from the same state. Every MDP has at least one optimal policy, and at least one of the optimal policies is stationary and deterministic [53].

2.3.2 Main Elements of Reinforcement Learning

There are four main sub-elements which can be recognized in a reinforcement learning algorithm (that works in MDP): a policy which defines the learning agent's way of behaving at a given time, a reward function which indicates the intrinsic desirability of the state, a value function which specifies what is good in the long run, and optionally, a model of the environment which mimics the behavior of the environment. There are two ways used in RL algorithms:

- Model-free: Learn a controller without learning a model.
- Model-based: Learn a model, and use it to derive a controller.

There is a matter of some debate in the reinforcement-learning community about which approach is better. A number of algorithms have been proposed using both approaches.

Most RL algorithms generally have a function $Q : S \times A \rightarrow \mathbb{R}$ that maps every state-action pair to a value (usually called the Q-value). The Q-value defines the quality of being in a state $s \in S$ and taking an action $a \in A$. In each time step, the agent is in state s_t, it chooses an action a_t according to its policy, observes the reward $r(s_t, a_t)$, and moves to a state s_{t+1}. The Q-values are updated for state-action pair (s_t, a_t) as follows:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r(s_t, a_t) + \gamma V(s_{t+1}) - Q(s_t, a_t)] \tag{2.1}$$

Where γ is the discount factor, and $V(s_{t+1})$ is the quality of state s_{t+1} (usually called the V-value).

2.3.3 Exploration vs. Exploitation

One of the challenges that arises in RL is the trade-off between exploration and exploitation. To obtain high rewards, a reinforcement learning agent must prefer actions that it has tried in the past and found to produce high rewards. On the other hand, to discover such actions it has to try actions that it has not sampled efficiently in the past. One common exploration method is ε-greedy, wherein the agent exploits its current Q-estimates most of the times, but acts randomely with probability ε.

2.3.4 Limitations of Reinforcement Learning

There are some limitations to the use of RL. One of the difficulties faced by application designers is that RL methods tend to learn very slowly. This can lead to poor performance in dynamic environments [18]. Yen and Hickey [90] proposed a solution for RL adaptation in dynamic environment by using forgetting mechanism through favoring exploration over exploitation. Subsequently, Kamei and Ishikawa [45] proposed another solution in which they introduced sensory information to accelerate learning in a transient stage.

Another limitation is that the number of state values that must be maintained may increase to unmanageable sizes [6]. Yen and Hickey [90] proposed that rather than storing a value for each individual state in the environment, a value function can be used to store the value of each of a set of features.

2.3.5 Q-learning

Q-learning [87, 88] is a model-free, off-policy control algorithm. It is concerned with learning action-value function Q. Since it is off-policy, it approximates the optimal action-value function Q* independently from the policy being followed. However, the policy still have an effect in that it determines which state-action pairs

are visited and updated. Q-learning is guaranteed to converge to the optimal policy in stationary environments with a finite number of states, assuming that each state-action pair is visited infinitely often, and that the learning rate decays appropriately over time [88]. Each time an agent is in a state s_t, it takes an action a_t, observes the reward $r(s_t, a_t)$, and moves to a state s_{t+1}. The Q-values are updated for state-action pair (s_t, a_t) as follows:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r(s_t, a_t) + \gamma.max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \tag{2.2}$$

Where γ is the discount factor, and $max_a Q(s_{t+1}, a)$ is the maximum value for Q in state s_{t+1}.

2.3.6 Relevance to this Research

Reinforcement learning is a popular method for solving problems similar to the ones we consider in this thesis. Q-learning is one of the most used RL algorithms. It has been shown to learn effectively in several domains [17, 82]. Many studies have made changes to Q-learning to make it suitable for multi-agent problems. These variations include PHC [11], Minimax-Q [53], Nash-Q [43], Hyper-Q [84], and M-Qubed [21].

2.4 Game Theory

Game theory is the mathematical study of strategic interactions, in which the success of any individual in choosing actions depends on the actions of others. Game theory assumes that each player forms rational beliefs about what other players will do and then chooses a reaction to those strategies that maximizes its own payoffs. The field of game theory was initially created by Von Neumann and Morgenstern [85]. Their work contained the method for finding mutually consistent solutions

for simple games along with other useful notions. Since then, many concepts have been developed in game theory and it has been widely applied to problems in economics, biology, psychology, and computer science.

2.4.1 Normal-Form vs. Extensive-Form Games

Games analyzed with game theory can have different types of representation. The most two common forms of games are normal-form games, which are also known as matrix games, and extensive-form games.

Normal-form games can be represented as a matrix of payoffs, in which every cell represents the payoffs for each player given the actions taken by all players. Matrix games can be either simultaneous-move games, where players take their actions at the same time, or turn-taking games, where each player takes its action in turn. Table 2.1 shows payoff matrices in Normal-form representation.

In contrast, extensive-form games do not assume that players act simultaneously. This form represents the game as a tree. Each node resembles a possible state of the game. Each extensive-form game can be converted (or reduced) to normal-form.

2.4.2 Classes of Games

Both normal and extensive-form games can differ in their competitive nature. For example, in *common-payoff* games, the payoffs are the same for all players (for each action profile). In contrast, in *constant-sum games* , the summation of all the players' payoffs is constant for all action profiles. *Zero-sum* games are a special case of *constant-sum games* in which the summation of the payoffs is zero (your win is my loss) although the name *zero-sum* is used in the literature usually to refer to *constant-sum* games. The most interesting class of games are *conflicting-interests* games. In this class of games, the agents have somewhat opposing inter-

(a) Conflicting-interests game.

	C	D
C	3, 3	0, 5
D	5, 0	1, 1

(b) Common-payoff game.

	C	D
C	3, 3	0, 0
D	5, 5	1, 1

(c) Zero-sum game.

	C	D
C	3, -3	0, 0
D	5, -5	1, -1

Table 2.1: Examples for different classes of games in normal-form representation. In each cell, the first number is the payoff for the row player, and the second number is the payoff for the column player. Note that payoffs for the row player are given the same in the three examples. Hence, a player can not guess the class of the game from knowing only his own payoffs.

ests, but all players can benefit from making certain compromises. This class of games can be seen as a case between the first two classes of games. Table 2.1 shows examples for the three different classes of games in normal-form representation.

2.4.3 Strategy Profiles

In game theory, a strategy is a complete description of a particular way to play a game, no matter what the other players do and no matter how long the game lasts. A strategy must prescribe actions so thoroughly that you never have to make a decision in following it [67]. One kind of strategy is to select a single action and play it. Such a strategy is called a pure strategy. Strategies that are not pure are called mixed strategies. In a mixed strategy, the player selects an action from a probability distribution over all the possible actions. A pure strategy is a special case of mixed strategy in which the probability is 1 for one action and zero for the others.

A strategy profile is the set of strategies chosen by all the players, each strategy is chosen by one player. Pure strategy profile is the set of strategies that each of which is a pure strategy. Mixed strategy profile is the set of strategies that at least

one of which is a mixed strategy.

2.4.4 Solution Concepts

There are many solution concepts that have been developed to help anticipate how a game will be played.

Maximin and Minimax

Maximin and minimax (also called maxmin and minmax) strategies are important concepts in zero-sum games, and they are used interchangeably in these types of games. Maximin is to play as to maximize your minimum payoffs. The maximin strategy for player i against player $-i$ is $arg\ max_{s_i}\ min_{s_{-i}}\ u_i(s_i, s_{-i})$ and the maximin value is $max_{s_i}\ min_{s_{-i}}\ u_i(s_i, s_{-i})$. Minimax is to play as to minimize the opponent's maximum payoffs. The minimax strategy for player i against player $-i$ is $arg\ min_{s_i}\ max_{s_{-i}}\ u_{-i}(s_i, s_{-i})$ and the minimax value is $min_{s_i}\ max_{s_{-i}}\ u_{-i}(s_i, s_{-i})$.

In zero-sum games, maximin and minimax strategies are the same, and they resemble the best thing you can do, under the assumption that your opponent will play so as to minimize your payoff (i.e. to maximize his own payoffs).

Pareto Optimality and Strategic Dominance

Although both concepts are related to dominance, they refer to different aspects in game theory. Pareto optimality is concerned with strategy profiles (section 2.4.3), while strategic dominance is concerned with the actions possible for a player.

In a game G with a finite set of players N, a strategy profile s Pareto dominates strategy profile s' if for all players $i \in N, u_i(s) \geq u_i(s')$ and at least for one player $j \in N$ there is $u_j(s) > u_j(s')$. A strategy profile s is Pareto optimal (or Pareto dominant) if there is no other strategy profile s' that Pareto dominates s. A strategy profile s is Pareto dominated if there is at least one another strategy profile s' that

Pareto dominates s.

In other words, every Pareto optimal solution cannot be substituted with another solution (another strategy profile) without decreasing the payoffs of at least one player, while a Pareto dominated solution can be substituted with at least one another solution in which the payoffs for at least one player are increased and for all the other players are not decreased.

In a two-player game G, where each player has a finite set of actions (namely A_i and A_{-i}), an action $a \in A_i$ strategically dominates action $a' \in A_i$ if for all actions $b \in A_{-i}, u_i(a,b) \geq u_i(a',b)$ and at least for one action $c \in A_{-i}$ there is $u_i(a,c) > u_i(a',c)$. An action $a \in A_i$ is strategically dominant if it strategically dominates all the other actions $a' \in A_{-i}$.

Nash Equilibria and Best Response

The best response strategy is to play the best action given you know the action played by other agents. The best response of player i, who has a set of strategies S_i (mixed and pure) against a strategy profile s_{-i} is a strategy $s_i^* \in S_i$ such that $u_i(s_i^*, s_{-i}) \geq u_i(s_i, s_{-i})$ for all strategies $s_i \in S_i$.

The Nash equilibrium concept is, perhaps, the most influential solution concept in game theory. This concept was introduced by Nash [60]. A Nash equilibrium is a stable strategy profile in which no agent would want to change his strategy if he knew what strategies the other agents were following. A strategy profile (section 2.4.3) $s = (s_1, ..., s_n)$ is a Nash equilibrium if, for all the agents i, s_i is a best response to s_{-i}, where s_{-i} resembles the strategies of other players.

Nash Bargaining Solution

It is known that Nash equilibria are not guaranteed to optimize the social reward [28]. The most famous example is the "Prisoner's Dilemma" (Table 2.1(a)).

Hence, other solution concepts might be more useful than Nash equilibrium. One of these concepts is also proposed by Nash [61], it is named the Nash bargaining solution (NBS). The NBS is one of the proposed solutions to the bargaining problems. A two-agent bargaining problem considers finding an agreement over some alternative options between two persons. Nash defined four axioms that should be satisfied by a bargaining solution [61, 64]. He proved that there is only one unique NBS that satisfies these axioms, and it takes the following form:

$$f(S,d) = arg\ max\ (s_1 - d_1)(s_2 - d_2) \tag{2.3}$$

Where s_1 and s_2 are the utilities for the two players when both of them play the NBS, and d_1 and d_2 are the utilities for the two players when both of them play the maximin strategy. The maximization is subject to the constraints $s_j \geq d_j$, where $j = 1,2$.

2.4.5 Congestion Games

Congestion games resemble a restricted class of games that are useful in solving real-world problems and that also have interesting theoretical properties. The effect of an individual's action on other's payoffs is restricted in congestion games [80]. Intuitively, in a congestion game, there is a set of resources. All the agents can choose from these resources. The cost of each resource depends on the number of agents who selected it.

Formally speaking, a congestion game is a tuple (N,R,A,c), where:

- N is a set of n players,
- R is a set of r resources (links),
- $A = A_1 \times A_2 \times ... \times A_n$ where $A_i \subseteq 2^R \setminus \emptyset$ is the set of actions available to player i, and

- $c = (c_1, ..., c_r)$, where $c_k : \mathbb{N} \rightarrow \mathbb{R}$ is a cost function for resource $k \in R$.

The players' utility functions are defined in terms of the cost function c_k. The function that counts the number of players who took any action involving resource (link) r is defined as $\sharp : R \times A \rightarrow \mathbb{N}$. Given a pure strategy profile $a = (a_i, a_{-i})$, the utility function for each player is:

$$u_i(a) = -\sum_{r \in R | r \in a_i} c_r(\sharp(r, a)) \tag{2.4}$$

Congestion games have the property of anonymity. Players care only for the number of other players who are using the same resource and do not care about which players are using it.

The concept of a congestion game was first introduced by Rosenthal [70]. He proved that the sequence of improvement steps by different players reach a Nash equilibrium after a finite number of steps. The finite number of improvement steps implies the existence of pure Nash equilibrium in congestion games. A pure strategy profile in congestion games is a selection of a single path (strategy) by each player.

Potential Games

Congestion games are a special case of potential games. Potential games are a class of games in which the incentive of all the players to change their strategy can be expressed in one global function, the potential function. Every finite potential game has a pure-strategy Nash equilibrium. Potential games were introduced by Monderer and Shapley [58]. Rosenthal proved that any congestion game is a potential game, and Monderer and Shapley proved the converse: for any potential game, there is a congestion game with the same potential function.

Network Congestion Games

In computer science, perhaps the most studied congestion games are network routing problems [49, 73, 71, 32] and the load balancing problem [48, 3]. The problems we consider is more likely to be considered as a network congestion game.

In a network congestion game, each player has a source node and a destination node in a directed graph. Every player seeks for a minimum delay path connecting its source with its destination. The delay of an edge depends on the number of players using that edge. It has been shown that in the symmetric variant of the game, where all players have the same source and the same destination, one can compute a Nash equilibrium with the help of a min-cost flow algorithm [32].

Network congestion games can be symmetric or asymmetric games. In symmetric games, all the players have the same source and the same destination, while in asymmetric games players may have different sources and/or destinations. Fabrikant, Papadimitriou and Talwar [32] gave a polynomial time (global) algorithm to find pure Nash equilibria for the case of symmetric network congestion games.

Network congestion games can be categorized by the nature of the traffic. When the game is played by an uncountably infinite (or a very big) number of players, where the effect of each agent on the congestion is very small, the game is assumed to be non-atomic. While atomic congestion games consider the effect of each agent on the congestion, non-atomic games study the effect of a group of agents on the congestion.

Price of Anarchy (PoA) and Price of Stability (PoS)

In the literature, the price of anarchy (PoA) [49, 66] and the price of stability (PoS) [66, 1] have become recently common measures of the quality of equilibria achieved by distributed agents. The PoA measures the cost of the lack of coordination by comparing the worst Nash equilibrium to the social optimal cost, while

PoS aims to measures the minimum penalty required to ensure a stable equilibrium outcome by comparing the best Nash equilibrium to the social optimal cost. When the game has unique Nash equilibrium, the PoA and the PoS are equal. When the PoA is close to 1, one can conclude that if all agents converge to NE, then they are performing as well as possible.

Roughgarden and Tardos [73] showed that if the latency of each edge is a linear function of its congestion, then the upper bound of the PoA is $4/3$. Busch and Magdon-Ismail [13] showed that for every routing game there is a pure Nash equilibrium that is equal to the optimal coordinated cost ($PoS = 1$).

Objective Function

In order to determine the efficiency of a strategy, the social cost is considered as the objective function that needs to be minimized. In network routing problems, a natural choice for social cost is the maximum delay incurred by a packet. In heavily congested networks, the maximum delay of a packet is governed by the maximum congestion over all edges in the network. This choice of social cost is referred to as the maximum social cost [49, 16, 20, 72]. However, still it is common in the literature to use the sum of the players costs (instead of the maximum) [49, 72, 35, 57], where the cost is proportional to the sum of the congestions on the edges along a player's path.

Epstein, Feldman and Mansour [31] have proposed efficient graph topologies that guarantee that the cost of any Nash equilibrium is the social optimum cost (i.e. PoA =1). They proposed the efficient graph topologies for two classes of atomic network: network congestion games and bottleneck routing games for both of the symmetric and asymmetric cases.

2.4.6 Repeated Matrix Games

In repeated matrix games (also called *repeated games*), a given matrix game is played repeatedly by the same set of players. This matrix game is called the stage game. In a repeated game, the set of actions available for each player is the same in all time periods, and regardless of the action history of that player. Additionally, the payoffs to the players depend only on the actions played and not on the time period.

Formally speaking, a repeated matrix game is a tuple (g, N, A, R), where:

- g is a matrix game (the stage game) that is played T times (where T can be a finite or infinite number),
- N is a finite set of n players,
- $A = A_1 \times A_2 \times ... \times A_n$ where A_i is a finite set of actions available to player i, and
- $R = r_1, r_2, ..., r_n$ where $r_i : A \rightarrow \mathbb{R}$ is a real-valued payoff function for player i.

Repeated games provide a general framework in which self-interested agents consider their reputation i.e. the impact of their current actions on the future behavior of other players. In some contexts, this can encourage agents to tend for cooperation in a long-term relationships. Hence, the maximization of their own utilities becomes in accord with the social welfare maximization.

Early studies regarding repeated games include Luce and Riffa [55] who provided the basic ideas behind he folk theorem, and Aumann [2] who showed the theoretical differences between finitely and infinitely repeated games.

Finitely Repeated Games

In finitely repeated games, the time period is fixed and known. However, when the time period is fixed but not known, the game is thought of as infinitely repeated one. Finitely repeated games can be represented in extensive form (although the stage game is in normal form). The payoff of each agent (attached to terminal nodes) in this case would be calculated as the summation of the payoffs in the stage game.

Using the backward induction and considering the finitely repeated "Prisoner's Dilemma" as an example, it can be argued (in a similar way to chain-store paradox [77]) that in the last round, the dominant strategy is to defect no matter what has happened earlier. The reason is that there will be no other rounds for retaliation after that. Hence, in the second-to-last round it is also a dominant strategy to defect. Using induction, it can be said that the only equilibrium in this case is to always defect. However, this argument is vulnerable to both practical and theoretical criticisms. Real life situations showed that people have some tendency for cooperation even in finitely repeated games.

Infinitely Repeated Games

In infinitely repeated games, the time period is infinite. However, the time period might be finite but assumed to be unknown. In these games, when the extensive-form representation is used, the payoff for each player cannot be attached to the terminal nodes (even as the sum of payoffs) since it is infinite (or unknown). Thus other methods are used, such as the average reward or the future discounted reward. The average reward for an agent is the average payoff throughout the games played.

$$\lim_{k\to\infty} \frac{\sum_{j=1}^{k} r_i^{(j)}}{k} \tag{2.5}$$

Where $r_i^{(j)}$ is the reward received by player i at time (iteration) j, and k is the number of iterations played.

The future discounted reward for an agent in a stage is the sum of his payoff in the current stage, plus the future rewards discounted by a constant factor.

$$\sum_{j=1}^{\infty} \beta^j . r_i^{(j)} \tag{2.6}$$

Where β^j is the discount factor at time j ($0 < \beta^j < 1$).

The Folk Theorem

Informally speaking, the folk theorem states that in a repeated game, a strategy profile is a Nash equilibrium (rNE) if it can guarantee, for each player, an average payoff that is at least as good as the maximin value for that player. Likewise, every strategy profile that can guarantee for each player the average payoff of the maximin strategy is a Nash equilibrium (rNE) strategy profile, provided the possibility of playing this strategy profile.

Consider the following n-player game $G = (N, A, u)$ in which $r = (r_1, r_2, ..., r_n)$ is some payoff profile, and v_i is the maximin value for player i. Then, we can say r is an enforceable payoff profile, if for each player i, the payoff $r_i \geq v_i$. We can say also that r is a feasible profile, if for each player i, $r_i = \sum_{a \in A} \alpha_a . u_i(a)$, where $\sum_{a \in A} \alpha_a = 1$.

Formally speaking, the folk theorem states that in any n-player matrix game G and for any payoff profile $r = (r_1, r_2, ..., r_n)$:

1. If r is the payoff profile of any Nash equilibrium of the infinitely repeated game G with average rewards, then r is enforceable.

2. If r is both feasible and enforceable, then r is the payoff profile for some Nash equilibrium of the infinitely repeated game G with average rewards.

2.4.7 Stochastic Games

A stochastic game, also called a Markov game, is a set of matrix games, each of which resembles a one stage game. In each iteration, the agents play one of these games. Playing a stage game depends probabilistically on the previous game played and on the actions taken by all the agents.

Formally speaking, a stochastic game is a tuple (Q,N,A,P,R), where:

- Q is a finite set of games,
- N is a finite set of n players,
- $A = A_1 \times A_2 \times ... \times A_n$ where A_i is a finite set of actions available to player i,
- $P : Q \times A \times Q \rightarrow [0,1]$ is the transition probability function from a stage game to another game when taking an action profile a, and
- $R = r_1, r_2, ..., r_n$ where $r_i : Q \times A \rightarrow \mathbb{R}$ is a real-valued payoff function for player i.

In this sense, we can say that stochastic games are a generalization for both Markov decision processes (MDPs) and repeated matrix games. A repeated matrix game is a stochastic game with one stage game. An MDP is a stochastic game with one agent, hence the name Markov game. On the other hand, in a stochastic game, if all the players except one play a fixed policy, then the problem for this one player becomes an MDP.

Stochastic games were first introduced by Shapley [79]. Filar and Vrieze [33] provided a rigorous introduction to the topic integrating single-agent and two-person stochastic games. A stochastic game starts in one of the stage games $q \in Q$. Each agent i takes an action $a_i \in A_i$, then it receives the payoff r_i depending on the action profile $a = (a_1, ..., a_n)$ and on the stage game q. In the next iteration, the agents are given another (or possibly the same) stage game q'. The stage game q'

is chosen depending on the transition probability function P, the stage game q, and the joint action $a = (a_1, ..., a_n)$.

2.4.8 Relevance to this Research

In our study, we aim to route vehicles in multi-agent environments. In these environments, each agent should consider the strategies (the set of actions) of other agents when selecting its actions. Game theory provides many useful concepts, methods, and techniques for multi-agent environments, and it also provides good representation for problems in these environments.

In this study, we consider two types of networks. One network is a small, but intriguing, network and the other is a simpler network. The former can be represented as a stochastic game that happens to be a congestion game (different congestion games are played stochastically), while in the latter, the problem can be represented as a congestion game that is infinitely repeated (repeated congestion game) for two agents with two actions for each, where the same game is played repeatedly.

In the first problem, we assume that the stage games only differ in the payoff values. The stage games are general-sum games. In analyzing this problem, we evaluate the performance of different algorithms. These algorithms can be reinforcement learning-algorithms, or shortest path algorithms (assumed to be in dynamic environments instead of strategic environments). Our main focus is not only to reach equilibrium but also to secure a fair cost for each agent in addition to maximizing the overall performance of the group of agents.

The second problem we consider is a repeated congestion game. This game consists of a single stage game that is played infinitely by the same two agents. This stage game resembles a congestion game that consists of two resources (links). The main focus is learning to share the payoffs with the other agents efficiently. This

can be achieved by maximizing the summation of the utilities of the agents and minimizing the variance in the agents payoffs.

2.5 Multi-agent Reinforcement Learning

With the use of game theory, reinforcement learning was extended in many studies to multi-agent settings [89, 78]. Markov games became the common model of multi-agent RL. As a consequence, applying Q-learning in a simple approach to each agent in multi-agent environment does not guarantee the convergence of Q-learning to the optimal policy anymore, since the environment is not stationary in this case. The reason is that the other agents are also adapting.

2.5.1 Independent Actions vs. Joint Actions

There were different suggestions for extending RL (mainly Q-learning) to multi-agent environments. Using joint actions instead of independent actions was one of these suggestions.

Q-learning and other some RL algorithms use in general the Bellman equation (reproduced in 2.7) to update their Q-values. In this equation, there is a Q-value for each pair (s, a_i), where s is the current state and a_i is the action taken by agent i in state s. This notation is called independent actions, since the action of the agents are assumed to be independent and thus the action of associates are not considered.

$$Q(s,a_i) \leftarrow Q(s,a_i) + \alpha[r(s,a_i,a_{-i}) + \gamma.V(s') - Q(s,a_i)] \tag{2.7}$$

Where $Q(s,a_i)$ is the Q-value of state s when agent i takes an action a_i, $r(s,a_i,a_{-i})$ is the reward received when the actions a_i and a_{-i} are played by agents in state s, γ is the discount factor, and $V(s')$ is the V-value for next state s'.

However, applying Q-learning with the joint actions (Equation 2.8) instead

of independent actions has been demonstrated in some situations [53, 17, 42] to outperform Q-learners who considers independent actions. Using the joint action means considering the action of the agent along with the actions of other agents.

$$Q(s,a_i,a_{-i}) \leftarrow Q(s,a_i,a_{-i}) + \alpha[r(s,a_i,a_{-i}) + \gamma.V(s') - Q(s,a_i,a_{-i})] \quad (2.8)$$

2.5.2 WoLF-PHC

Bowling and Veloso [11] presented a simple extension of Q-learning. They presented policy hill climbing (PHC) algorithm that is similar to Q-learning but can learn mixed policies. However, this algorithm performs poorly in Markov games. They presented a modification to it by adding the principle of Win or Learn Fast (WoLF) [11, 10] producing a new algorithm, named WoLF-PHC. This algorithm is said to learn the best response if possible and have some guarantee of convergence since it meets two specific properties. These two properties are rationality and convergence. Rationality means if the other players' policies converge to stationary policies then the learning algorithm will converge to a best response policy, while convergence means that the algorithm should converge to a stationary policy. If all algorithms have these two properties, they can converge to Nash equilibrium.

2.5.3 Minimax-Q

Littman [53] proposed an algorithm (named minimax-Q) that extends MDPs to zero-sum Markov games. In single-agent environments, an agent can maximize its reward by taking the action with the highest Q value. This strategy is optimal since the Q function becomes an accurate summary for the future rewards. In zero-sum games, if all agents play their best response, then the best an agent can do is to to play minimax. In this case, they converge to the unique Nash equilibrium.

However, the value of a state $s \in S$ in Markov game is

$$V(S) = max_{\pi \in PD(A)}\, min_{o \in O} \sum_{a \in A} Q(s,a,o)\pi_a \tag{2.9}$$

where π_a is the probability distribution of taking action a, o is the action of the opponent, and $Q(s,a,o)$ is the quality of taking action a against action o in state s, which is given as:

$$Q(s,a,o) = R(s,a,o) + \gamma \sum_{s'} T(s,a,o,s').V(S') \tag{2.10}$$

where $R(s,a,o)$ is the immediate reward observed when taking actions a by the agent and o by the opponent in state s, γ is the discount factor, $T(s,a,o,s')$ is the probability distribution of moving from state s to s' when agents take actions a and o, and $V(S')$ is the value calculated in Equation 2.9 for state S'.

2.5.4 Nash-Q

Hu and Wellman [43, 41] generalized Littman's algorithm to a general-sum Markov games (Nash-Q algorithm). They proposed using Nash equilibrium instead of minimax. They showed that when there is a unique Nash equilibrium, their algorithm consistently converges to it. In the case of multiple Nash equilibria, the algorithm sometimes fails to converge to any NE. Bowling [9] clarified the convergence of this algorithm.

2.5.5 OAL

Wang and Sandholm [86] presented optimal adaptive learning (OAL) extending MDP to team Markov games (full coordination games). They proved that this algorithm converges to an optimal Nash equilibrium with probability one in any team Markov game. They showed also that OAL's parameters are easy to meet the

convergence conditions. The algorithm is applicable even when the agents know neither the payoff structure nor the transition probabilities.

2.5.6 Friend-or-Foe

Littman [54] provided another extension for the general-sum Markov games. His algorithm, named Friend or Foe Q-learning (FFQ), converges to NE in constant-sum and common-interest games. Unlike Nash-Q, FFQ does not need to learn the estimates of Q functions for opposing players. However, FFQ does not address the problem of finding equilibrium in games of conflicting interests.

2.5.7 M-Qubed

Crandall and Goodrich [21] proposed an algorithm, named M-Qubed (Max or Minimax Q-learning), that performs well in different general-sum matrix games. In zero-sum games, M-Qubed may try to exploit the associate but in the worst cases it plays minimax (it is secure), while it learns to play the NBS in most common and conflict-interest games in self-play. In games of conflicting interests, M-Qubed often learns to compromise with the associate, so it performs well in self-play and against algorithms which are ready to compromise. When the associate tries to exploit it, M-Qubed plays minimax strategy against it.

M-Qubed uses SARSA reinforcement learning [74] instead of Q-learning and it encodes and balances three learning biases. The first of these biases is the best response learning bias, which encourages it to play a best response with respect to its current Q-estimates. If it begins to be exploited, M-Qubed learns to encode a cautious learning bias to limit its losses by playing its minimax strategy. Finally, it encodes an optimistic learning bias to encourage effective exploration in the early stages of the game.

Although M-Qubed has been shown to outperform most of the state-of-the-

art algorithms in different general-sum matrix games, it has a number of important limitations. For example, M-Qubed is designed for games with a single stage game (repeated matrix game). It also requires knowledge of the actions taken by each agent.

2.5.8 Relevance to this Research

In the first problem, we evaluate the performance of Q-learning in Markov games where the payoff matrix and the actions available for other players are not observable. In the second problem, the performance of M-Qubed and variations of M-Qubed (without the minimax strategy) are evaluated and compared. These variations differ only in their state representations and/or the immediate reward used in updating the Q-values. The actions taken by both agents in the last two iterations (we name them (t-2) and (t-1) iterations) are completely or partially used in these algorithms. In this second problem, we assume the knowledge of one's own payoff matrix and the observation of the opponent's previous actions.

CHAPTER 3

The Model

In this chapter, we provide a description for the studied environments, including the graphs representing the problems, the state representations, and the sequence of events in the simulators. As we have mentioned previously, we consider two different networks forming two different problems. The first network is a small, but intriguing, network that is considered in n-agent environment (Problem I). The second network is a simple (and a small) one that is considered in two-agent environment (Problem II).

Our goal in these two problems is to find a learning algorithm that when used by each vehicle should achieve the following two objectives as quickly as possible:

1. Fairness – minimize the variance in agents' utilities among the population:

$$Min\ (max_{i \in N}\ u_i - min_{i \in N}\ u_i) \tag{3.1}$$

where N is the set of agents, u_i is the utility received by agent i.

2. Maximize the social welfare – maximize the sum of the agents' utilities (minimize their costs):

$$Max \sum_{i \in N} u_i \tag{3.2}$$

The social objective function in "Problem I" is the average cost per agent per trip during the last 5000 time steps (also called iterations):

$$\frac{1}{5000} \sum_{t-4999}^{t} (\frac{1}{n} \sum_{i \in N} u_i) \tag{3.3}$$

where t is the current time step, and n is the number of agents.

The social objective function in "Problem II" is the probability (number of times out of 100 times) of both agents sharing the costs efficiently equally with each other.

$$\frac{1}{100} \sum_{k=1}^{100} \delta(\left|u_i - u_j\right| \leq 0.05) \times \delta(\left|u_i + u_j - max_{u_i,u_j \in U}(u_i + u_j)\right| \leq 0.05) \tag{3.4}$$

where u_i and u_j are the utilities for agents i and j respectively, U is the set of all possible utilities, and $\delta(x) = 1$ if $x = true$ and 0 otherwise.

3.1 Problem I

3.1.1 The Network

In this study, we consider a small directed graph with four nodes and seven directed links. The graph is depicted in Figure 3.1. As for the trips, we consider the symmetric case of a single source (A) and a single destination (D) for all the vehicles. In order to repeat the trip for each vehicle again, the vehicle will need to go through a link from D to A. For simplicity, the cost of this link does not depend on the congestion ($f_7(X_7) = 0$).

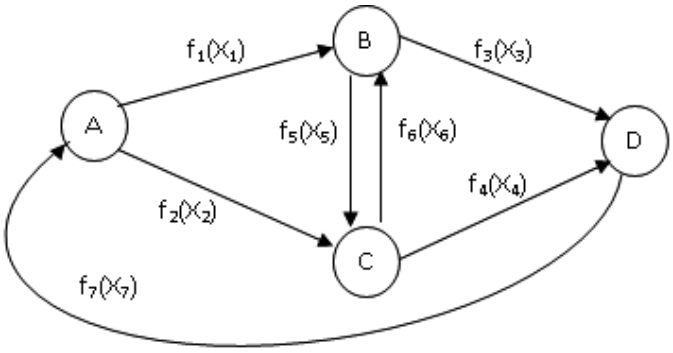


Figure 3.1: Graph network. All trips start at A and end at D. X_i represents the amount of traffic on link i.

3.1.2 The Cost Function

The cost function ($f_i(X_i)$) for using link i is a function of the number of vehicles (X_i) currently using link i. In our model, we assume that two of the links use the same function ($f_5(X) = f_6(X)$ for all X). One of the cost functions is also the one used for the link $D \rightarrow A$ which is equal to a constant value ($f_7(X_7) = 0$). The other functions use the cost function given in [38]:

$$f_i(X_i) = \frac{Distance_i}{V_i * (1 - \frac{X_i}{Capacity_i})} \tag{3.5}$$

where V_i is the maximum possible velocity on link i, $Distance_i$ denotes the length of link i, and $Capacity_i$ denotes the capacity of link i.

In our model, we assume that $Distance_i$, $Capacity_i$ and V_i are fixed for each link during the simulation time. Hence, the costs of the links are dynamic only because the traffic on these links are dynamic. The cost of each link takes its minimum value when the vehicles run with their maximum velocity on that link. This only happens when X_i is zero. There is no vehicle that can run at this velocity since with only one vehicle the velocity would slightly decrease.

Of course, there are cases where the traffic on a link (X_i) exceeds the capacity

$Link$	$Distance_i$	$Capacity_i$	V_i
1 (A-B)	800	50	80
2 (A-C)	1000	54	60
3 (B-D)	1600	68	100
4 (C-D)	1900	76	100
5 (B-C)	500	40	120
6 (C-B)	500	40	120
7 (D-A)	0	∞	-

Table 3.1: The properties of each link.

of that link ($Capacity_i$). According to the previous equation, the cost of that link becomes negative. To avoid this, we assumed that there is a minimum velocity. The minimum velocity is assigned for the vehicles on a link when the number of vehicles (the traffic) exceeds 90% of the capacity. The minimum velocity on each link is 10% of the maximum velocity of that link. Hence, our previous equation becomes:

$$f_i(X_i) = -\frac{Distance_i}{V_i * max(0.1, (1 - \frac{X_i}{Capacity_i}))} \tag{3.6}$$

The cost function $f_i(X_i)$ is non-linear when $X_i \in [0,\ 0.9 * Capacity_i]$, and is a constant function when $X_i \in [0.9 * Capacity_i,\ +\infty[$. We have multiplied the previous equation with (-1), and so the objective function needs to be maximized in this case (instead of being minimized).

The overall capacity of the network (without the last link which has an infinite capacity) is 328 vehicles. In our simulations, we consider 300 vehicles (also called agents).

The properties of each link i including $Distance_i$, $Capacity_i$, and V_i are shown in Table 3.1. We can realize that links 5 and 6 have the same properties (they are assumed to use the same cost function). Properties of link 7 have no practical meaning. Link 7 can have infinite number of vehicles ($Capacity_7 = \infty$) and link 7 is assumed to have no cost ($Distance_7 = 0$).

3.1.3 Description of the World

Initially, each vehicle can be located in any of the four nodes. Every vehicle receives a report about the number of vehicles available on each node, in each time step (also called iteration). In each iteration, all the vehicles move to a new node. Each vehicle then observes its own cost during the last time step, and the new node it is currently at.

The cost observed by each vehicle is used for two purposes. It is used as a reward by learning algorithms that use the Markov model in learning. It is also accumulated over all the simulation for each vehicle in order to be used later to calculate the average cost per trip per vehicle. This average cost is used as the social objective function (the average cost per agent per trip in the last 5000 iterations) that needs to be maximized, and which we use to evaluate the performance of different algorithms. Each simulation was run for $400,000$ iterations.

3.1.4 Representation of the States and Actions

State representations are critical to the success of a learning algorithm. Reinforcement learning algorithms generally work in environments that are formulated as Markov decision process (MDP). Recall that in a MDP, there is a set of states and a set of actions possible in each state. The transition function defines the probability of moving from a state to another state when taking a specific action.

States

The state is a tuple (v, n_A, n_B, n_C, n_D):

- $\forall v \in \{A, B, C, D\}$, and
- n_v is the number of vehicles on node v ($N = \sum_{v \in \{A,B,C,D\}} n_v$).

where N is the overall number of vehicles in the system (300).

node v	*Available Actions*
A	B, C
B	C, D
C	B, D
D	A

Table 3.2: The actions available to an agent from each node.

With this state representation, and since there are 300 agents in this simulation, the number of agents on each node (n_v) can be any value in $[0, 300]$. This means that the number of states will be very huge $(301^4.4 \approx 3.2e10)$. This would make it very hard for each state to be visited even once in our simulations. Thus, we discretized the number of states (equal width discretization or EWD with $width = 30$). In this discretization, every 30 agents are considered as one group (having 1 or 30 agents on the same node is considered the same, while 31-60 is considered as another state). With this, we had $40,000$ states.

Actions

The list of available actions for each player depends on the current state. More specifically, it depends on the node v . Table 3.2 specifies the actions available to an agent from each node.

3.2 Problem II

Solving the previous problem is shown to be a very hard task for many reasons. First, it considers a very big number of agents (300 agents). However, the number of agents is not the only factor. A second reason is the tremendous number of states even though the states were discretized. Additionally, discretization means losing some information also, which means that the agents cannot fully observe the actions of the opponents.

For all of these reasons, we decided to study a simpler problem, which removes many of the previous challenges. In this problem, agents cannot know the actions taken by other agents while playing (taking their actions), but afterwards, they do.

On the other hand, Another challenge is added to this problem. The social objective function is more restricted, and it is the probability (number of times out of 100 times) of sharing costs efficiently equally where the variance in agents' costs and the sum of agents' costs are minimized.

The average cost per agent per trip is considered as less restricted since it does not consider the fairness. Fairness means that both agents receive similar efficient costs. It only cares for maximizing the average cost. In the case of two agents using two links (where one link is better than the other), the average cost would be maximized if each link is used by one agent at a time. This does not consider the case where the better link is always used by the same agent.

3.2.1 The Network

The simple graph consists of two nodes and three directed links. The graph is depicted in Figure 3.2. As for the trips, every trip starts at A and ends at B. Initially, both agents are located at A. There are only two options to choose (link 1 and link 2) to complete a trip. The third Link (link 3) have the same role of the seventh link in the previous problem. It only means starting a new trip.

3.2.2 The Cost Function

As in the previous problem, the cost function $f_i(X_i)$ is used to calculate the time delay for link i as a function of the traffic (X_i) on that link. Similarly to link 7 in the previous problem, link 3 has the cost of zero ($f_3(X_3) = 0$). Table 3.3 shows the cost function $f_i(X_i)$ for two values of X_i (there are only two agents).

In our study, we assume that the cost of a link i when $X_i = 1$ is always better

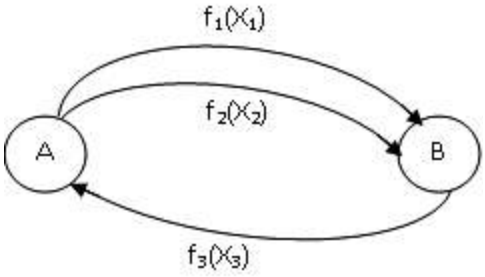


Figure 3.2: Graph network. Trips start at A and end at B. X_i resembles the traffic on link i.

	$f_1(X_1)$	$f_2(X_2)$
$X_i = 1$	d	c
$X_i = 2$	a	b

Table 3.3: The cost function $f_i(X_i)$, where $f_i(1) > f_i(2)$ and $f_1(Xi) > f_2(X_i)$ for $X_i = 1,2$ .Thus, $d > c > a > b$.

than when $X_i = 2$ ($f_i(1) > f_i(2)$). We also assume that link 1 is better than link 2 ($f_1(Xi) > f_2(X_i)$ for $X_i = 1,2$). Thus in Table 3.3 we have $d > c > a > b$. However, we study a more general case in which $d > c \geq a \geq b$ ($f_1(1) > f_2(1)$ and $f_1(2) \geq f_2(2)$).

3.2.3 Description of the World

Only two agents are trying to go from A to B. Before every iteration, the two agents are assumed to be at A. Each one chooses a link simultaneously, the cost is observed, data is updated and the two agents go back to A. In game theory terms, this problem can be represented as a repeated matrix game. The same game is played over and over between two agents. It resembles a special case of the previous problem (there is only one game that is played). Using game theory representation for normal form games (matrix game), this problem can be represented as in Table 3.4.

As we can see, this game is classified as a game of conflicting interests. If we assumed that the two links are equal in their efficiency then $a = b$ and $c = d$. We

	L1	L2
L1	a, a	d, c
L2	c, d	b, b

Table 3.4: Payoff matrix for Problem II.

are concerned with the case where one link is more efficient than the other. With the assumption that link 1 is better than link 2 and with normalized payoff we can say $1 = d > c \geq a \geq b = 0$.

In our experiments, we use different values for 'c' and 'a' that meet the previous condition in order to generate different versions of the game. The values of $b = 0$ and $d = 1$ are constant since we are normalizing the payoffs.

3.2.4 Representation of the States and Actions

There are two available actions for each agent which are: choosing to use link 1 and choosing to use link 2. As for the states, generally algorithms that use the history of joint actions as the current state (also called recurrent state) helps the learning algorithm to reason about how its current actions affect its future payoffs. There are many algorithms that have used the recurrent states [75, 8, 21].

In our study, we evaluate the performance of different algorithms in the considered game. These algorithms use different states representations. All of these algorithms use information from the past (history) to encode their current state. The actions taken by both agents in the last two iterations (which we refer to as iterations $t-2$ and $t-1$, respectively) are completely or partially used in these algorithms. We show in this study that the state representation has a significant effect on the behavior of algorithm and its tendency toward maximizing the summation of agents' utilities and minimizing the variance in agents' utilities.

CHAPTER 4

Results – Problem I

In this chapter, we present results for Problem I. Experiments were done using two algorithms: Q-learning, the most common reinforcement learning algorithm and Dijkstra's algorithm, the most common search algorithm. We compare the performance of the two algorithms with the socially optimal solution (also called the baseline solution). Appendix A includes the parameters values used in our simulations for both algorithms.

4.1 Baseline Solution

To evaluate the performance of different algorithms in this problem, we needed to find a baseline solution that we can compare other solutions with. This solution should be optimal or near optimal to be a useful comparison.

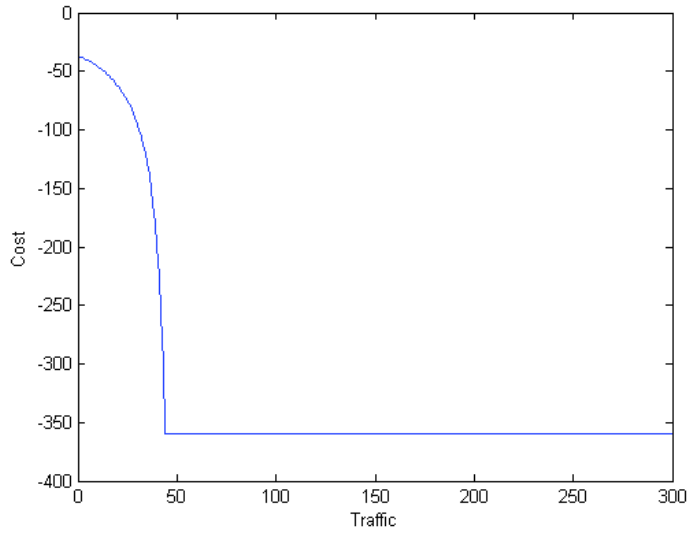


Figure 4.1: Cost function for link A-B.

4.1.1 Assumptions and Conditions

To compute the baseline solution, we used the following assumptions and conditions:

- We assumed that each link will experience the same traffic for the entire simulation. Ideally, the decision of each player should converge to a pure or a mixed policy. This means that the number of vehicles using each link should converge so that the same number of vehicles on each link will be repeated every iteration.

  Recall that our cost function which depends on the traffic, is a concave function in the domain $[0,\ 0.9 * capacity]$:

$$f_i(X_i) = -\frac{Distance_i}{V_i * max(0.1, (1 - \frac{X_i}{Capacity_i}))} \tag{4.1}$$

  The cost function for link A-B (time versus traffic) is shown for the values of traffic in [0, 45] in Figure 4.1. Here 45 is 90% of 50 which is the capacity

of this link. All the other links have similar plots. As we can see for this domain, the cost is a concave function. Hence Jensen's inequality [50] holds for this function:

$$E[f(x)] \leq f(E[x]) \tag{4.2}$$

Assuming that there is an optimal traffic value X_i^* for each link i, and assuming that the average traffic on a link i is $\bar{X}_i(v)$, where v is the variance, then the previous equation means that the cost value $f_i(\bar{X}_i^*(0)) \geq f_i(\bar{X}_i^*(v))$ for all $v \in \mathbb{N}$. In other words, converging to X_i^* as a mean is better done using zero variance ($\bar{X}_i^*(0)$). However, the assumption of fixed traffic per link forces some other conditions.

- The number of agents on the two crossing-links (B-C and C-B) alternates. Assuming the number of vehicles on A-B in iteration n is X_{A-B}^n, then $X_{B-C}^n = X_{C-B}^{n+1} = X_{B-C}^{n+2}$.

- There is no agent moving through B-C and C-B successively, since this path will be less efficient than another path (e.g. A-C-B-C-D is less efficient than A-B-D). Although there might be some better solutions in which there are some agents who act as the scapegoats and fluctuate between the crossing-links in order to provide better costs for others, but since in our study we focus on the self-interested agents that demand at least a fair cost, we ignore such solutions.

- Following from the last two assumptions, we can conclude that $X_{A-B}^n + X_{A-C}^n = X_{B-D}^{n+1} + X_{C-D}^{n+1} = X_{D-A}^{n+2}$. Combining this with the first assumption, we can find that $X_{A-B}^n + X_{A-C}^n = X_{B-D}^n + X_{C-D}^n = X_{D-A}^n$ (The summation of numbers of vehicles on links A-B and A-C in iteration n is equal to the summation of numbers vehicles on B-D and C-D and is equal to the number of

Link	*Number of vehicles on each link* in the baseline solution	*Link's Capacity*	*0.9* Capacity* (traffic threshold)
A-B	40	50	45
A-C	40	54	39
B-D	40	68	61
C-D	40	76	68
B-C	30	40	36
C-B	30	40	36
D-A	80	∞	∞

Table 4.1: The number of vehicles on each link in the baseline solution. Last column shows for each link the traffic threshold beyond which the cost is constant.

vehicles on D-A in the same iteration).

4.1.2 Finding the Baseline Solution

Taking all of these assumptions and conditions into consideration and with the condition of having an integer number of vehicles on the link, we made an exhaustive search for the optimal solution. The solution has an average cost of -396 per vehicle per trip. The fixed number of vehicles on each link is shown in the Table 4.1.

The exhaustive search done to find this solution depends on 3 parameters:

- The number of vehicles on link D-A, we call it N.
- The number of vehicles on link A-B, we call it x_1.
- The number of vehicles on link B-D, we call it x_3.

Table 4.2 shows how these 3 parameters are enough to define the values of the traffic on all the links. As you can see from Tables 4.1 and 4.2, we got the values ($x_1 = 40$, $x_3 = 40$, $N = 80$) for the optimal solution, given the previously specified link attributes.

Link	*Traffic as a function of x_1, x_3, and N*
A-B	x_1
A-C	$N - x_1$
B-D	x_3
C-D	$N - x_3$
B-C	$0.5 * [300 - 3 * N + x_1 - x_3]$
C-B	$0.5 * [300 - 3 * N - x_1 + x_3]$
D-A	N

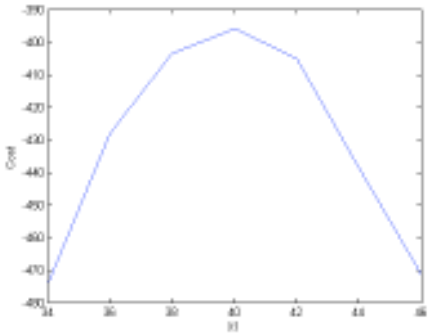
Table 4.2: The seven links' traffics as functions of the three parameters x_1, x_3, N.

Figures 4.2(a), 4.2(b), and 4.2(c) show the effect of the variation of each of the 3 parameters on cost when fixing the other 2 parameters. Although the search for these parameters was exhaustive, there are many values that are not feasible solutions. These values are not plotted.

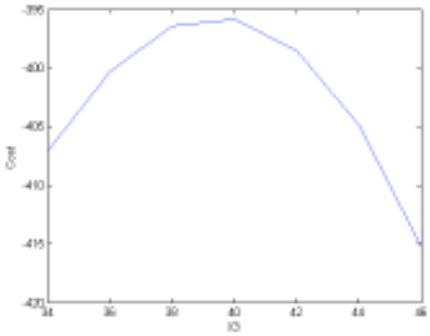
4.2 Q-learning

Q-learning is guaranteed to converge to the optimal policy in stationary environments with a finite number of states, assuming that each state-action pair is visited infinitely often, and that the learning rate decays appropriately over time [88]. However, in multi-agent environments, where the actions of other agents affect the environment, the algorithm is not guaranteed to converge when other agents also learn. This is because it is not a stationary environment (the state-transitions distributions are not fixed) in such circumstances. Instead, it is a strategic environment, where the next state is affected by the actions taken by other agents.

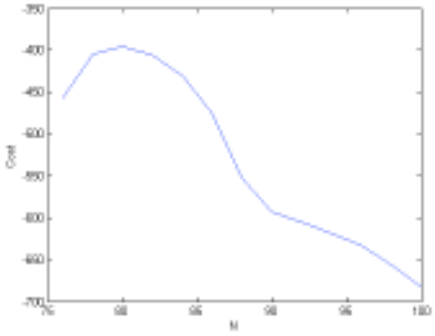
To show the convergence of Q-Learning in a stationary environment, we used a single agent who employs Q-learning. We show that the Q-learner converges in a stationary environment to the optimal solution (-269). Figure 4.3 shows the performance of one Q-learner (exploration rate $\varepsilon = 0$ after 10,000 iterations) in a static environment for 200,000 iterations. Links' costs are fixed and assumed to



(a) Cost as a function of x_1 given that $x_3 = 40$, $N = 80$.



(b) Cost as a function of x_3 given that $x_1 = 40$, $N = 80$.



(c) Cost as a function of N given that $x_1 = 40$, $x_3 = 40$.

Figure 4.2: Cost values given different values of x_1, x_3, and N. Two values are fixed in each graph.

be equal to the case where 37.5 agents use each link and 75 use link (D-A). The average cost is calculated in each iteration $t > 5000$ for the last 5000 iterations

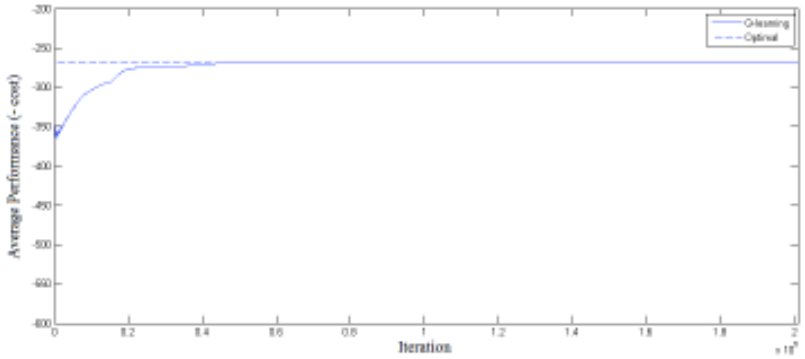


Figure 4.3: The performance of a single Q-learner in a static environment (exploration rate $\varepsilon = 0$ after 10,000 iterations). For each iteration $t > 5000$, the average cost (y-axis) is taken for the last 5000 iterations (from $t - 4999$ to t).

(from $t - 4999$ to t).

After showing the performance of one agent in a stationary environment, we wanted to evaluate the performance of a group of Q-learners. Figure 4.4 shows for groups with sizes (1, 5, 10, 20, 50, 100 and 200 Q-learners), while the other agents act randomly. The lowest line in Figure 4.4 (the one with '+' signals) resembles the average cost of the other random agents, which was approximately the same in all simulations. The highest line is for one Q-learner agent, and the next one is for five Q-learners and so on. This shows that the average performance of a Q-learner decreases inversely with the number of Q-learners. Table 4.3 shows the average cost per Q-learner for populations with various numbers of Q-learners.

With 200 Q-learning agents, the performance of a Q-learner is very close to the random agents. One can say that Q-learners are not learning since their behavior is very close to the random ones. Figure 4.5 shows the performance of 300 Q-learners in a full strategic environment compared to the performance of 300 random agents. The figure shows that the performance of Q-learners is improving over time. This means that Q-learners are learning but they do not converge to an optimal solution. Actually they converge to a very bad solution, which is slightly better than the

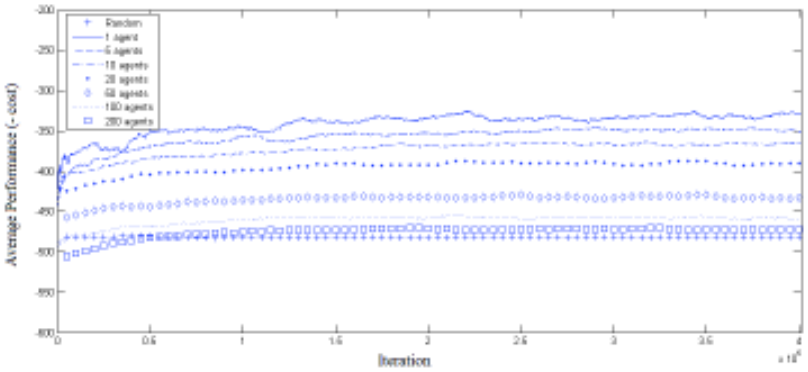


Figure 4.4: The performance of different numbers of Q-learners compared to the other random agents.

Number of Q-learners	*Average Cost*
1 agent	-330
5 agents	-350
10 agents	-365
20 agents	-390
50 agents	-430
100 agents	-460
200 agents	-470
random agents	-482

Table 4.3: The performance of different numbers of Q-learners.

random behavior. The reason might be that none of the Q-learners are converging to a fixed policy, although the overall mean value of the group does converge.

In order to check the instability of the policies of the Q-learners, we show in Table 4.4 the mean and the standard deviation for the number of agents on each link during the last 1000 iterations. The mean values for all links (except for the last one) are similar to each other. CV values are also similar for all the links, except again for the last one. It is clear that these values have big variance. The number of agents per link is oscillating even after 400,000 iterations.

The instability in the number of agents per link probably comes from the instability of the policies of these agents. In other words, it may come from the

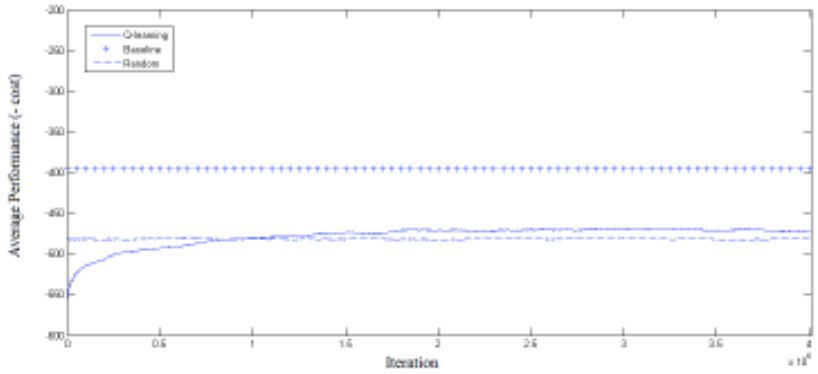


Figure 4.5: The performance of Q-learning compared to random and baseline solutions. Results are from population of 300 Q-learners.

Link	*Baseline Traffic*	*Mean* (μ)	*Standard Deviation* (σ)	*Coefficient of Variance* ($\frac{\sigma}{\lvert\mu\rvert} \times 100\%$)
A-B	40	40.29	6.59	16%
A-C	40	36.74	5.47	15%
B-D	40	40.09	6.27	16%
C-D	40	36.93	6.08	16%
B-C	30	34.56	5.28	15%
C-B	30	34.38	5.08	15%
D-A	80	77	8.64	11%

Table 4.4: The mean, standard deviation and the coefficient of variance for traffic on each link. Results are from a population of 300 Q-learners. The baseline traffic is shown for comparison.

instability of the Q-values of the Q-learners. Figure 4.6 shows one Q-value for one agent.

We can see that some Q-values are still fluctuating even after 400,000 iterations. The Q-value shown in Figure 4.6 is oscillating between -2000 and -2300, which yields $CV \approx \frac{|2300-2000|}{0.5\times(2300+2000)} \approx 14\%$.

Although Q-learning can help an agent to converge to an optimal policy in a stationary environment, and although it can help the agents to learn and improve their behavior in a strategic environment, it does not converge to the optimal policy

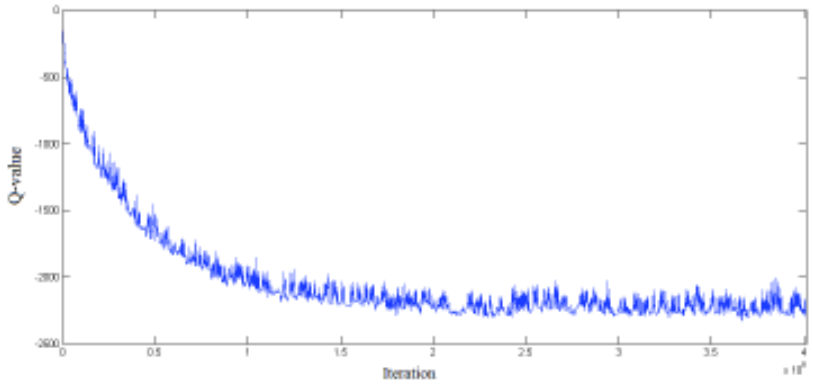


Figure 4.6: One Q-value for a Q-learning agent. Even after 400,000 iterations, the Q-values keep oscillating.

in this environment. In this domain, it fails to learn a cooperative and efficient solution (such as the NBS). Instead, it learns (as a group) to play a solution that is only slightly better than the performance of random agents.

4.3 Dijkstra's Algorithm

Dijkstra's algorithm is guaranteed to calculate the optimal solution in a static graph where the costs of the links are fixed for the whole duration of the simulation. In this simulation, similar to Q-learning, we assume the agent does not know the costs beforehand. In the first 50 iterations, we implemented the algorithm so that it acts randomly. After that, it starts to calculate the shortest path depending on its estimation of the links' costs. Learning the links's costs continues for all the time of the simulation. We estimate the links cost as follows after each time step:

$$f(i) = f(i) * (1 - \alpha) + C_i * \alpha \tag{4.3}$$

Where $f(i)$ is the agent's current estimation for link i, C_i is the new cost experienced on link i in the last time, and α is the learning rate. Link cost estimations

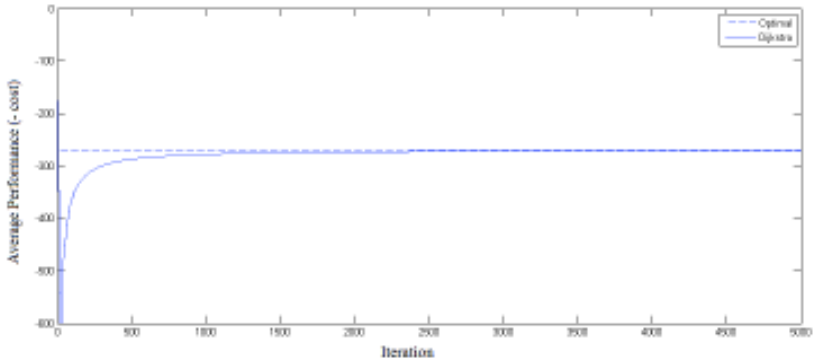


Figure 4.7: The performance of a single Dijkstra's agent in a static environment.

are used to calculate the shortest path using Dijkstra's algorithm [25].

In the case of a static graph, costs can be learned easily and hence the algorithm converges to the optimal solution (-269). Figure 4.7 shows the performance of one Dijkstra agent in a static environment for the first 5000 iterations. Links' costs are fixed and assumed to be equal to the case where 37.5 agents use each link and 75 use link (D-A).

However, when the costs are not fixed, Dijkstra's algorithm has difficulty estimating the costs and hence calculating the shortest path. Figure 4.8 shows for groups with sizes (1, 5, 10, 20, 50, 100 and 200 Dijkstra's agents), while the other agents act randomly. Table 4.5 shows the average cost per Dijkstra agent for populations with various numbers of Dijkstra agents.

As we can see from the plot, the performance of 50, 100, and 200 agents is even worse than the random behavior. This comes from the oscillation of agents in choosing links. When a group of agents choose a particular link, they all get the same cost which they use to update the link cost using Equation 4.3. They then act accordingly. This causes them to all have the same policy. However, the performance of other random agents play some role in changing the experience of these agents to mitigate the problem somewhat for populations with many random

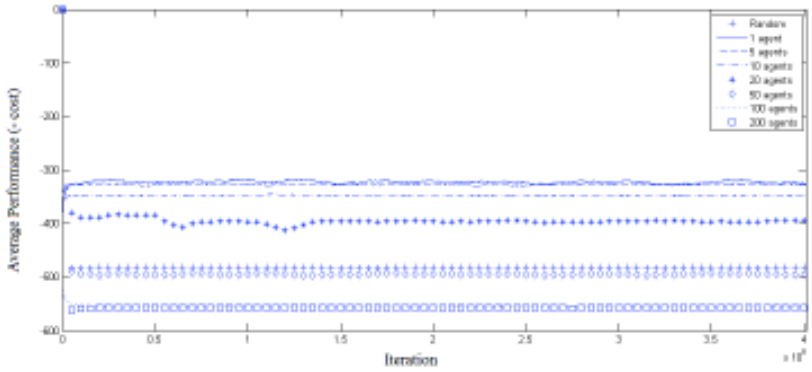


Figure 4.8: The performance of different number of Dijkstra agents compared to the other random agents.

Number of Dijkstra agents	*Average Cost*
1 agent	-326
5 agents	-327
10 agents	-347
20 agents	-393
50 agents	-495
100 agents	-555
200 agents	-557
random agents	-482

Table 4.5: The performance of different numbers of Dijkstra agents.

agents.

In the case of having no random agents. The performance of 300 Dijkstra agents in the environment is significantly worse than the random behavior. Figure 4.9 shows the performance of Dijkstra agents along with the random behavior and the baseline solution.

4.4 Using a Smaller Number of Agents

As we can see in Figure 4.10, the performance of both Q-learning and Dijkstra's algorithm are substantially less than the baseline solution for a population of 300

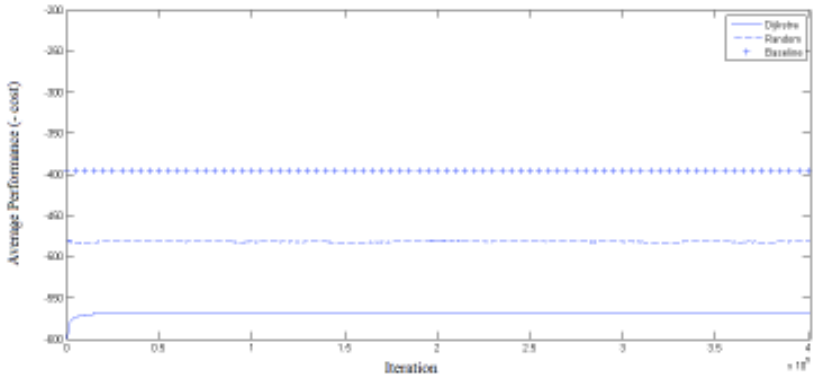


Figure 4.9: The performance of Dijkstra's algorithm compared to the random and baseline solutions. Results are from a population of 300 Dijkstra agents.

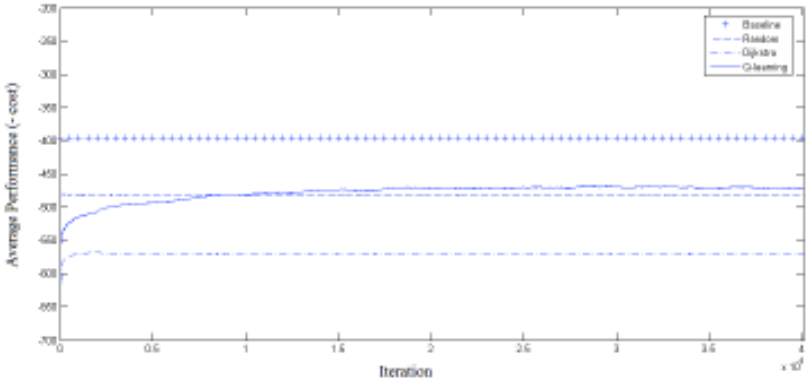


Figure 4.10: The performance of Q-learning and Dijkstra's algorithms compared to the random and baseline solutions with a population of 300 agents.

agents. This happens for different reasons. First, it assumes only part of the information is available for the algorithm. Another challenge is that it is in 300-agent world (big number of agents). Additionally, it has a large state space.

To determine the effect of population size on the performance of the algorithms, we minimized the number of agents from 300 to 14 agents. These agents were used on the same graph, with some modifications on links' properties, hence links' costs (refer to Table 4.6 for the new properties). The number 14 was chosen because it was easy to find a baseline solution with this number. A baseline solution for the

Link	$Distance_i$	$Capacity_i$	V_i
1 (A-B)	800	3	80
2 (A-C)	1000	3	60
3 (B-D)	1600	4	100
4 (C-D)	1900	4	100
5 (B-C)	500	2	120
6 (C-B)	500	2	120
7 (D-A)	0	∞	-

Table 4.6: The new properties of each link (14 agents).

Link	*Number of vehicles on each link* in the baseline solution	*Link's Capacity*	*0.9* Capacity* (traffic threshold)
A-B	2	3	> 2
A-C	2	3	> 2
B-D	2	4	> 3
C-D	2	4	> 3
B-C	1	2	> 1
C-B	1	2	> 1
D-A	4	∞	∞

Table 4.7: Baseline solution for a population of 14 agents. Last column shows for each link the traffic threshold beyond which the cost is constant.

new problem was computed (-287) as before. This solution has the number of agents on each link as shown in Table 4.7.

The performance of Q-learning and Dijkstra's algorithm in this new scenario is shown in Figure 4.11 along with the performance of random and the baseline solution. We can see again that neither Q-learning nor Dijkstra are close to the baseline solution. An interesting difference is that Dijkstra's performance is much better in this case. It outperforms Q-learning and random. This is due in large part to Q-learning's continued exploration. When exploration is eliminated after 200,000 iterations, the performance of Q-learning becomes very close to Dijkstra's performance (Figure 4.12).

In order to find the reasons behind the improved performance of Dijkstra's

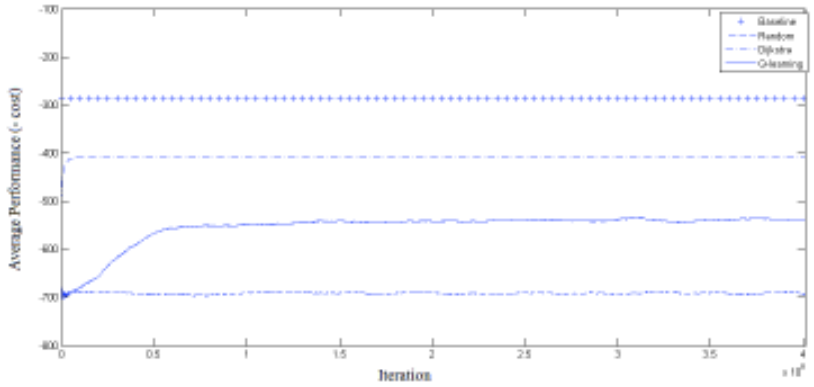


Figure 4.11: The performance of Q-learning and Dijkstra's algorithm compared to random and baseline solutions with a population of 14 agents.

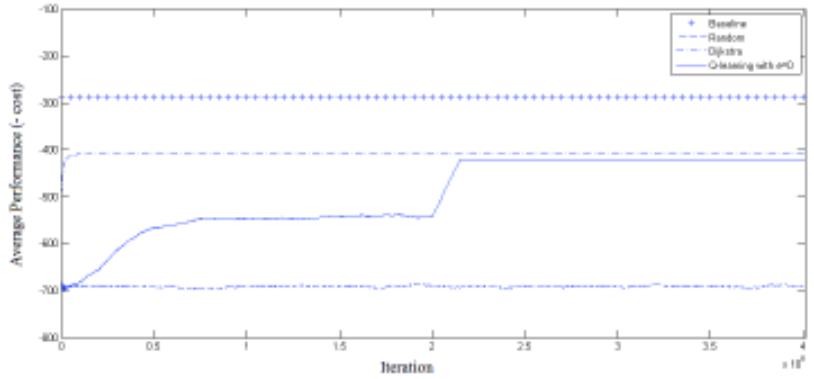


Figure 4.12: The performance of Q-learning ($\varepsilon = 0$ after 200,000 iterations) and Dijkstra's algorithm compared to random and baseline solutions with a population of 14 agents.

algorithm with 14 agents compared to the random and Q-learning behaviors, we show the mean and variance of the links in the last 1000 iterations. From Table 4.8, we can see that the mean and the variance for number of agents using each link are very small which means a small range of instability. This small range makes the estimation for links' costs easier for Dijkstra's algorithm and hence the improved performance.

Link	*Mean* (μ)	*Standard Deviation* (σ)
A-B	2.58	0.64
A-C	2	0
B-D	2.33	0.47
C-D	2.25	0.43
B-C	0.25	0.43
C-B	0	0
D-A	4.58	0.64

Table 4.8: The mean and standard deviation for traffic on links. Results are from a population of 14 Dijkstra agents.

4.5 Summary

In this chapter, we have shown the performance of Q-learning and Dijkstra's algorithm in an intriguing network. The performance has been evaluated with respect to the average cost per agent per trip in the last 5000 iterations. We compared the algorithms to a socially optimal solution (baseline solution).

Although both algorithms are guaranteed to find optimal solutions in stationary environments where only one agent is involved, they have been shown in this chapter to be far from the social optimal solution. However, Q-learning has been shown in other studies to perform well in some multi-agent situations [82, 78, 17], but the failure of performing well in this problem can be for different reasons. Mainly, this problem is better represented as a stochastic game where different matrix games are played. This would make it hard for algorithms who showed some success in some repeated matrix games to perform well in this problem. There might be also other reasons such as the big number of agents, the tremendous number of states, and the incomplete information assumption.

Further experiments showed that minimizing the number of agents (from 300 to 14) does not make any difference in the case of Q-learning. On the other hand, the performance of Dijkstra's algorithm improves since the estimation of link costs

became easier. However, the performance of both algorithms is still far from the social optimal even with less agents.

Although the aim of our study is to both maximize the summation of agents utilities (or average utilities) and to minimize the variance in agents utilities (fairness). We have not looked at the second part of the goal (fairness) since the first part could not be achieved.

CHAPTER 5

Experiments and Results – Problem II

In the previous chapter, we have shown the difficulty of achieving one part of our goal in this study (maximizing the summation of the agents' utilities), which happened for different reasons. However, we decided to moved to a simpler problem in which achieving the maximization of average utilities is an easy task for a conventional algorithm like Q-learning. The new challenge in this problem is to achieve fairness while retaining the average maximization. By fairness, we mean minimizing the variance in the agents payoffs.

In this chapter, we present results and discussion for the algorithms considered for solving problem II. In this problem, two agents must select one of two links to travel on. The cost of each link depends on the number of agents that use it. One of these links is better than the other. The main question of this problem is to find an algorithm that can, when used by both players (self-play), learn to share the payoffs with itself in a repeated matrix game, and to maximize the social welfare. We first describe the algorithms used, and the games played in our study. We then

present and discuss the results.

5.1 Experiments

Experiments in this chapter show how the different representations of states and the usage of current or previous payoffs for updating Q-values can affect the level of sharing in a specific type of games. All of the algorithms considered in this study use information from the past (history) to encode their current state. The actions taken by both agents in the last two iterations (which we refer to as iterations $t-2$ and $t-1$, respectively) are completely or partially used in these algorithms to encode state. Let $\vec{\mathbf{a}}_\mathbf{t} = (a_t^i, a_t^{-i})$ be the joint action played at time t, where a_i^t is the action taken by player i at time t, and a_t^{-i} is the action taken by the other player at time t.

State representation is not the only difference among the algorithms we evaluate. We also vary the reward used to update the algorithms' Q-values. We use two different types of rewards: One type uses the current observed cost to update the Q-values. Updating the Q-values is done using the standard Q-updates:

$$Q(s_t, a_t^i) \leftarrow Q(s_t, a_t^i) + \alpha[r(s_t, a_t^i) + \gamma V(s_{t+1}) - Q(s_t, a_t^i)] \tag{5.1}$$

where $Q(s_t, a_t^i)$ is the quality (Q-value) of taking action a_t^i in state s_t, $V(s_{t+1})$ is the quality (V-value) of next state s_{t+1}, γ is the discount factor, and $r(s_t, a_t^i)$ is the observed reward after taking action a_t^i in state s_t.

The other version uses a slightly different equation which is 5.2.

$$Q(s_t, a_t^i) \leftarrow Q(s_t, a_t^i) + \alpha[r(s_{t-1}, a_{t-1}^i) + \gamma V(s_{t+1}) - Q(s_t, a_t^i)] \tag{5.2}$$

The only difference between Equations 5.1 and 5.2 is that Equation 5.2 uses $r(s_{t-1}, a_{t-1}^i)$

instead of $r(s_t, a_t^i)$.

5.1.1 Algorithms

All of the algorithms considered in this study are implemented similarly to M-Qubed [21] (but without using the minimax strategy), except Q-learning and Random algorithms. M-Qubed uses the previous w joint actions to represent its current state. It was shown that a length of history $w = 1$ (only one joint actions. i.e. $\vec{\mathbf{a}}_\mathbf{t}$) is enough to perform well in most two-agent matrix games [21]. However, using a longer history (such as $w = 2$) can improve performance in some games but it makes learning slower, which means worse performance in other games.

We consider variations of M-Qubed (without minimax) that each represents state differently. Table 5.1 shows the algorithms in this study. Each member of this family has a different state representation. Names are used to code this representation. Let b_{t-1}^i represents a boolean value that equals to one if a_{t-1}^i is used by the algorithm to encode its state, and zero otherwise. Hence, the name of each algorithm takes the following form: ($'Q'$, b_{t-2}^i, b_{t-2}^{-i}, b_{t-1}^i, b_{t-1}^{-i}). For example, an algorithm that uses the tuple $(a_{t-2}^i, a_{t-1}^{-i})$ to encode its state is called $Q1001$.

There is another member in this family which uses less orthodox state representation, hence it is called *Maverick* . Let $a_t^i \in \{0,1\}$. Then *Maverick* represents its state using the tuple $(2 - a_{t-2}^i - a_{t-2}^{-i}, a_{t-1}^i)$ (where $2 - a_{t-2}^i - a_{t-2}^{-i}$ represents the number of agents chose action 0 at time $t-2$).

Each member of this family (including *Maverick*) has two versions. One version that updates its Q-values using Equation 5.1. We use the suffix C (e.g. $Q1010C$). The second version updates its Q-values using Equation 5.2. We use the suffix P (e.g. $Q1010P$).

All the family members use the same parameters and same implementation. Note that Q0000P and Q0000C represent stateless Q-learning. Note also that

Algorithm	a^i_{t-2}	a^{-i}_{t-2}	a^i_{t-1}	a^{-i}_{t-1}	$2-a^i_{t-2}-a^{-i}_{t-2}$
Q0000	0	0	0	0	0
Q0011	0	0	1	1	0
Q0101	0	1	0	1	0
Q0110	0	1	1	0	0
Q1001	1	0	0	1	0
Q1010	1	0	1	0	0
Q1100	1	1	0	0	0
Q0111	0	1	1	1	0
Q1011	1	0	1	1	0
Q1110	1	1	1	0	0
Q1111	1	1	1	1	0
Maverick	0	0	1	0	1

Table 5.1: State representations used in the study. '0' denotes that the algorithm does not use the attribute to represent state, '1' denotes that it does.

Q0011C and Q1111C resemble M-Qubed (without minimax), with $w = 1$ and $w = 2$, respectively. Appendix A includes the parameters values for Q-learning and the family of algorithms. In addition to these 24 algorithms, we also consider the performance of Random for the sake of comparison.

M-Qubed has shown excellent performance in different games against different state-of-the-art algorithms [21]. However, there were three games in which M-Qubed's performance in evolutionary tournament was not among the top, comparing with other algorithms. These three games were 'Offset game', 'Battle of the sexes', and 'Coordination Game' (Table 5.2). These games are similar to the class of games we consider with slight variations.

To play well in these games in self-play and against other successful algorithms, an algorithm needs to coordinate with others in order to reach the Nash bargaining solution, which is reached when the agents alternate between two action profiles that are on the major or the minor diagonal. This alternation causes the algorithms to share (divide equally) the payoffs between them.

(a) Offset Game.

	C	D
C	0, 0	0, 1
D	1, 0	0, 0

(b) Coordination Game.

	C	D
C	1, 0.5	0, 0
D	0, 0	0.5, 1

(c) Battle of the Sexes.

	C	D
C	0, 0	0.67, 1
D	1, 0.67	0.33, 0.33

Table 5.2: Games in which the Nash bargaining solution is played when both agents alternate between two action profiles that are on the major or the minor diagonal.

	link 0	*link 1*
link 0	a, a	d, c
link 1	c, d	b, b

Table 5.3: The type of considered games, where $1 = d > c \geq a \geq b = 0$.

5.1.2 Games

Recall that in Problem II, two agents play the repeated matrix game shown in Table 5.3. We tested our various algorithms using different values of a and c subject to the constraints. Table 5.4 shows the different games used in our study. The games are named according to the values of a and c such that $('G', a \times 10, c \times 10)$. For example, $G26$ game $(a = 0.2, c = 0.6)$. We can classify the games in four groups:

1. *G00 game.* In this game, $a = c = 0$. Thus, "link 1" is not traffic dependent as it has a fixed cost (since $c = b = 0$). When both agents use the same link, they both receive a payoff of zero. Additionally, if the opponent always chooses "link 0", the agent is indifferent between choosing "link 0" or "link 1".

2. *Gxx games.* In these games, $c = a \neq 0$ (G22, G44, G66, G88). When the opponent chooses "link 0", the agent is indifferent between choosing "link 0" or "link 1".

3. *G0x games.* In these games, $c \neq a = 0$ (G02, G04, G06, G08). "Link 0" is

Game	*a*	*c*
G00	0	0
G02	0	0.2
G04	0	0.4
G06	0	0.6
G08	0	0.8
G22	0.2	0.2
G24	0.2	0.4
G26	0.2	0.6
G28	0.2	0.8
G44	0.4	0.4
G46	0.4	0.6
G48	0.4	0.8
G66	0.6	0.6
G68	0.6	0.8
G88	0.8	0.8

Table 5.4: The family of games we consider in this chapter.

loosely better than link "1" ($d > c$ and $a = b$).

4. *Gxy games.* Here, $d > c > a > b$ (G24, G26, G28, G46, G48, G68). In these games, the case of having one agent per link is strictly better than having two agents per link ($d > a$ and $c > b$). "Link 0" is strictly better than link "1" ($d > c$ and $a > b$).

5.2 Results and Analysis

In the following sub-sections, we present and analyze the various algorithms in each of the games we have mentioned. All the results are shown for self-play only. This is to measure how well the algorithm learns to share the payoffs (i.e. alternating in choosing the links) against itself. If the algorithm cannot learn to share payoffs with itself, it would be difficult for it to share it with anyone else.

	link 0	*link 1*
link 0	0, 0	1, 0
link 1	0, 1	0, 0

Table 5.5: The payoff matrix of *G*00 game.

5.2.1 *G*00 Game

In the *G*00 game, there are three different action profiles that yield the reward of '0' for each player. It is functionally equivalent to the offset game shown in 5.2(a) and studied in [21]. Both agents have a strategically dominant action (choosing "link 0"). However, if both players use link '0', they both get nothing. There is also a possibility for confusion since the value '0' can be received in both actions for each agent (Table 5.5). Thus, effective algorithms in this game represent state with high amount of information in order to recognize the state of the world effectively.

Figure 5.1 shows the performance of algorithms in terms of the probability (number of times out of 100) that an algorithm learns to share the payoffs efficiently and equally with itself (0.65 means the algorithm learned to share payoffs efficiently and equally with itself 65 times out of 100). As we can see from Figure

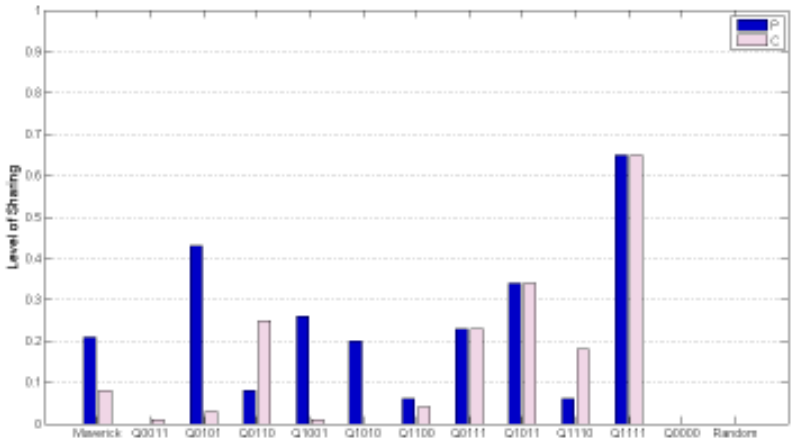


Figure 5.1: The performance of the algorithms in *G*00 game.

Game	*Top Algorithms*
G00	1) Q1111C (0.65) 1) Q1111P (0.65) 3) Q0101P (0.43) 4) Q1011C (0.34) 4) Q1011P (0.34) 6) Q1001P (0.26) 7) Q0110C (0.25) 8) Q0111C (0.23) 8) Q0111P (0.23) 10) MaverickP (0.21)

Table 5.6: Results for *G*00 game.

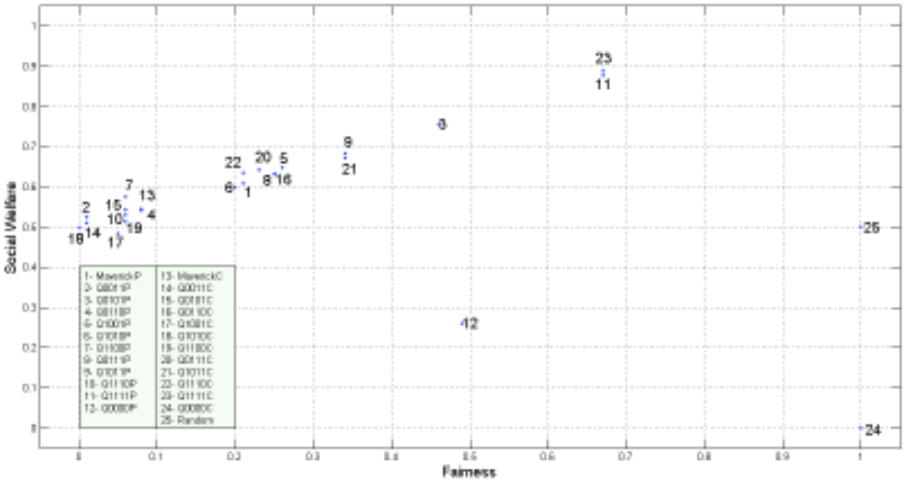


Figure 5.2: The average performance of all algorithms in the *G*00 game. Fairness is the probability of having the same average reward for both agents. Social Welfare is the summation of rewards for both agents divided by the maximum possible summation.

5.1 and Table 5.6, Q1111P and Q1111C are the top algorithms in this game. There is no effect for using previous or current payoffs for updating Q-values with Q1111 and algorithms that encode states with high amount of information (three or more). Other effective algorithms are either the ones that use three pieces of information or the ones that use the previous payoffs update.

Figure 5.1 shows that using the previous payoff (algorithms suffixed with P) for updating Q-values has positive effect on algorithms that use two pieces of information, and no positive effect on algorithms that use three or more pieces of information.

Another observation from Table 5.6 is that the top averages are not high ($<$ 0.65). This happens because this game needs an algorithm that uses much information to converge to the preferred solution, which means a big number of states. Having a big number of states may increase the possibility that one algorithm learns before the other, and it also makes learning slower. When one algorithm learns before the other, it converges to use the better link leaving the other one alternating.

Additionally, in Figure 5.2, we have plotted all algorithms in terms of the number of times both algorithms shared the payoffs in 100 times (Fairness), and in terms of the summation of the payoffs for both agents divided by the maximum possible summation (Social Welfare).

Both agents, when using Q0000C (stateless Q-learning), learn to share the payoffs (which is 0 for each). Random (No.25) gives similar payoffs for both players also but with half the maximum possible summation. Q1111C and Q1111P (No.23 and No.11 respectively) have high social welfare but with fairness less than 0.7 for sharing the payoffs. Random, Q1111C, and Q1111P produce Pareto optimal solutions.

5.2.2 *Gxx* games

This group of games is similar to the previous game. However, games in this group are slightly easier for our algorithms to learn effective solutions in (refer to Table 5.7). As in the previous game, each agent has a strategically dominant action ("link 0"), but an undesirable solution emerges when both agents play the dominant action. Each agent may receive the payoff x for two different action profiles (it is

	link 0	*link 1*
link 0	x, x	1, x
link 1	x, 1	0, 0

Table 5.7: The payoff matrix of *Gxx* games.

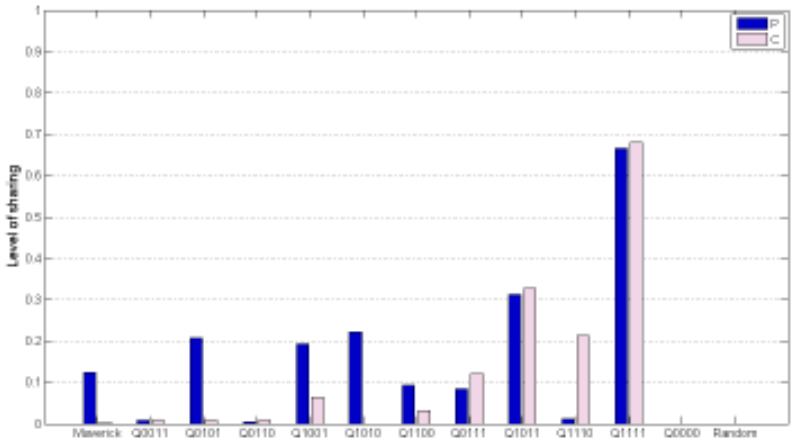


Figure 5.3: The performance of all algorithms averaged over all *Gxx* games.

Game	*Top Algorithms*
G22	1) Q1111P (0.72) 2) Q1111C (0.70) 3) Q1011P (0.45)
G44	1) Q1111P (0.73) 2) Q1111C (0.68) 3) Q1010P (0.45)
G66	1) Q1111C (0.75) 2) Q1111P (0.67) 3) Q1011C (0.40)
G88	1) Q1111C (0.59) 2) Q1111P (0.54) 3) Q1011C (0.22)
Average	1) Q1111C (0.68) 2) Q1111P (0.66) 3) Q1011C (0.33) 4) Q1011P (0.31) 5) Q1010P (0.22) 6) Q1110C (0.21) 7) Q0101P (0.21) 8) Q1001P (0.19) 9) MaverickP (0.12) 10) Q0111C (0.12)

Table 5.8: Results for *Gxx* games.

less confusing than *G*00 game). However, results in these games are similar to the ones in *G*00 game (Figure 5.3 and Table 5.8). If one algorithm learns faster to always choose "link 0", the other one will be indifferent about the actions. Thus, it will keep alternating between the two actions. The top averages are not high (< 0.7) for the same reason explained in *G*00 game. See Figure 5.3 and Table 5.8.

Looking at Figure 5.4, we can see that in this group, Q0000C (No.24) learns to share but with a higher payoff for each agent (> 0.6) compared to the previous

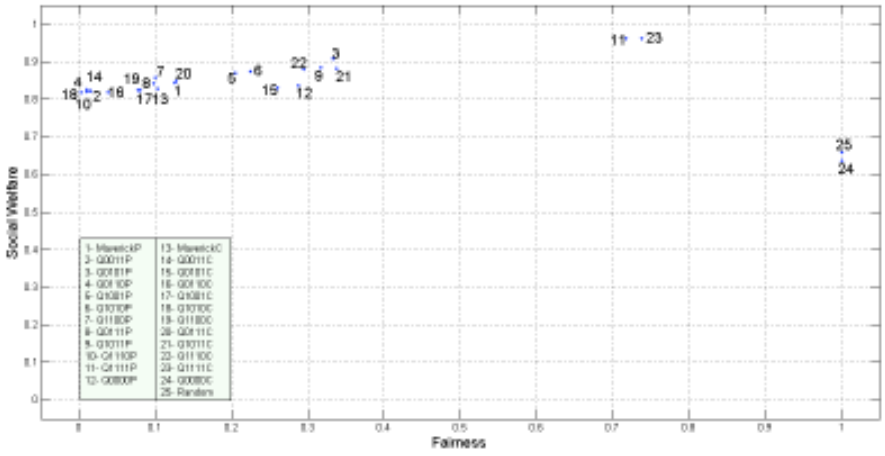


Figure 5.4: The average performance of all algorithms in the *Gxx* games. Fairness is the probability of having the same average reward for both agents. Social Welfare is the summation of rewards for both agents divided by the maximum possible summation.

game (which was zero). Random behavior (No.25) also guarantees sharing with even a higher average compared to the previous game (which was 0.5). On the other hand, Q1111P and Q1111C (No.11 and No.23) have high social welfare for agents reward but with a fairness of is 0.73. Pareto optimal solutions in *Gxx* games are Random and Q1111C.

5.2.3 *G0x* games

In this group of games, there is no strategically dominant action, which helps the agents to avoid quickly converging to a pure strategy. However, each agent may receive the payoff of '0' for two different action profiles which causes some con-

	link 0	*link 1*
link 0	0, 0	1, x
link 1	x, 1	0, 0

Table 5.9: The payoff matrix of *G0x* games.

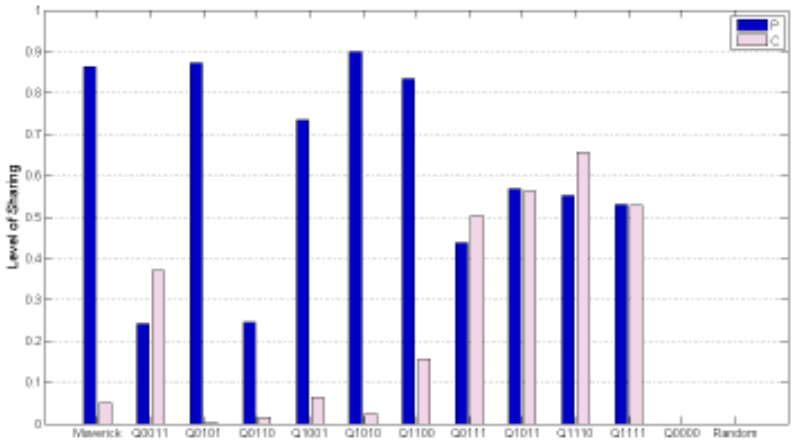


Figure 5.5: The performance of the algorithms averaged over all *G*0*x* games.

Game	*Top Algorithms*
G02	1) Q1100P (0.82) 2) Q1001P (0.77) 3) Q1011P (0.75)
G04	1) Q0101P (1.00) 2) MaverickP (0.99) 2) Q1010P (0.99) 2) Q1100P (0.99)
G06	1) MaverickP (1.00) 2) Q1010P (0.99) 3) Q0101P (0.97) 3) Q1100P (0.97)
G08	1) MaverickP (0.93) 2) Q1010P (0.92) 3) Q1110P (0.63)
Average	1) Q1010P (0.90) 2) Q0101P (0.87) 3) MaverickP (0.86) 4) Q1100P (0.84) 5) Q1001P (0.74) 6) Q1110C (0.66) 7) Q1011P (0.57) 8) Q1011C (0.56) 9) Q1110P (0.55) 10) Q1111P (0.53) 10) Q1111C (0.53)

Table 5.10: Results for *G*0*x* games.

fusion that makes this group even harder than the last group of games for learning (refer to Table 5.9). In these games, when both agents choose the same action, it does not matter which action they choose (they get the same rewards if they both choose “link 0” or both chose “link 1”).

Figure 5.5 and Table 5.10 show the superiority of algorithms with simpler state representation and previous payoff update over the algorithms with more complex

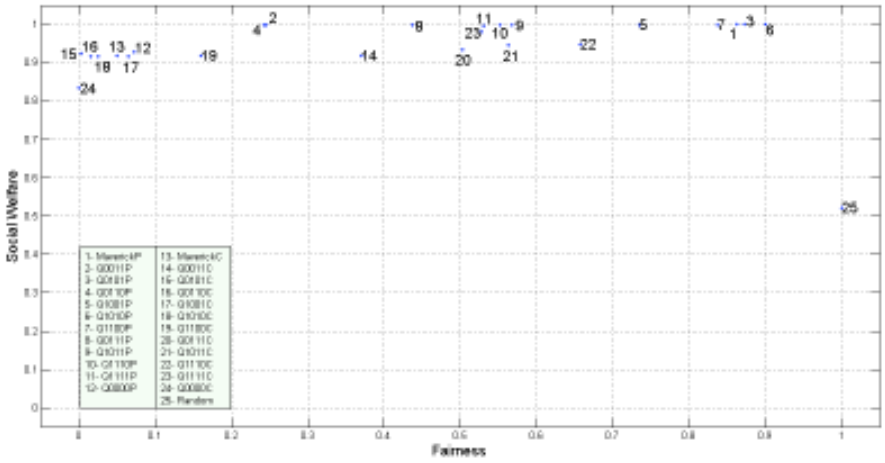


Figure 5.6: The average performance of all algorithms in the *G*0*x* games. Fairness is the probability of having the same average reward for both agents. Social Welfare is the summation of rewards for both agents divided by the maximum possible summation.

state representations.

In Figure 5.6, we can see that Random (No.25) guarantees sharing the payoff with more than half of the maximum possible summation (fairness$>$ 0.5). Both Random and Q1010P (No.6) produce the Pareto optimal outcomes. However, Q0101P, MaverickP, and Q1100P (No.3, No.1, and No.7) are also close to Pareto optimal.

5.2.4 *Gxy* games

In this group of games, agents have no strategically dominant action. Furthermore, each action profile yields a different payoff for the player. Thus, there is no confusion in the received payoffs (refer to Table 5.11). Algorithms that have simple state representation and use previous payoffs to update Q-values are among the top performing algorithms. Gxy games provides a good environment for MaverickP and other similar algorithms such as Q1110 (which has similar amount of information but with a bigger number of states) in order to learn sharing the payoffs.

	link 0	*link 1*
link 0	x, x	1, y
link 1	y, 1	0, 0

Table 5.11: The payoff matrix of *Gxy* games.

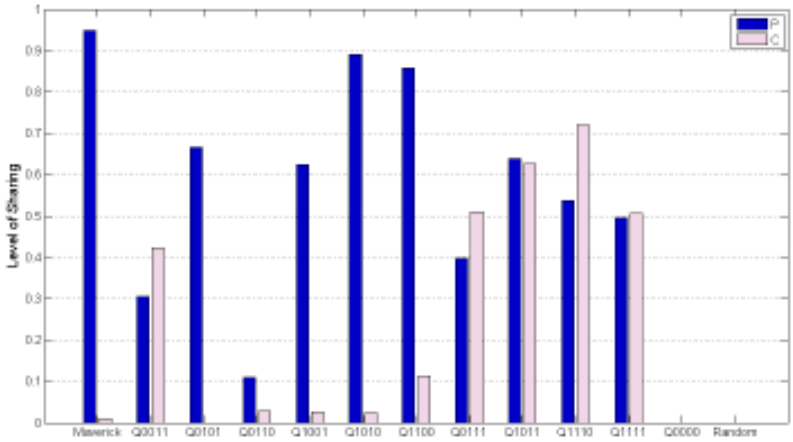


Figure 5.7: The performance of the algorithms averaged over all *Gxy* games.

Game	*Top Algorithms*
G24	1) Q0101P (1.00) 2) Q1010P (0.95) 3) Q1100P (0.92)
G26	1) MaverickP (1.00) 2) Q1010P (0.99) 3) Q1100P (0.99)
G28	1) Q1010P (0.95) 2) MaverickP (0.93) 3) Q0101P (0.66)
G46	1) MaverickP (1.00) 2) Q1010P (0.98) 2) Q1100P (0.98)
G48	1) MaverickP (0.98) 2) Q1010P (0.93) 3) Q1100P (0.85)
G68	1) MaverickP (0.90) 2) Q1110C (0.81) 3) Q1100P (0.78)
Average	1) MaverickP (0.95) 2) Q1010P (0.89) 3) Q1100P (0.86) 4) Q1110C (0.72) 5) Q0101P (0.67) 6) Q1011P (0.64) 7) Q1011C (0.63) 8) Q1001P (0.62) 9) Q1110P (0.54) 10) Q0111C (0.51) 10) Q1111C (0.51)

Table 5.12: Results for *Gxy* games.

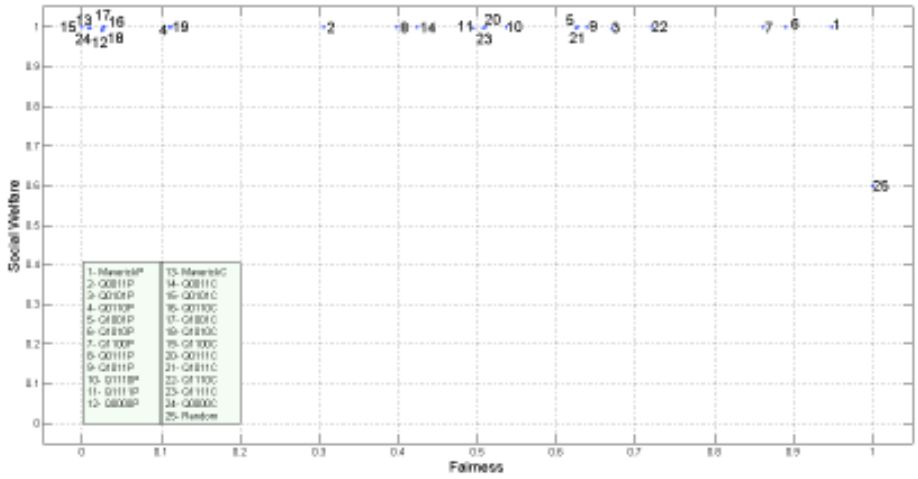


Figure 5.8: The average performance of all algorithms in the *Gxy* games. Fairness is the probability of having the same average reward for both agents. Social Welfare is the summation of rewards for both agents divided by the maximum possible summation.

In Figure 5.8, we can see that Random (No.25) guarantees sharing the payoff with more than half of the maximum possible summation (fairness$>$ 0.5). Both Random and MaverickP (No.1) produce Pareto optimal outcomes. However, Q1010P and Q1100P (No.6 and No.7) are also close to the Pareto optimal.

Performance Analysis

In general, our results show that algorithms that perform well in the *Gxy* and *G*0*x* games have a number of characteristics:

- They typically encode a small number of states. Having fewer states increases the probability of learning at the same rates as ones associates.
- They update Q-values with the previous payoff. Using minimal state makes the algorithm more myopic, which makes it perform worse in this problem. We can see from our results that algorithms that use two attributes to define state and use the current payoffs to update Q-values (suffixed with C) did

not do well in these games. Updating the Q-values with the previous payoffs appears to allow algorithms to prefer actions that lead to better states instead of actions that yield higher immediate rewards.

- They use information from time $t-2$. While using a small number of states and the previous payoff for updating Q-values showed good results in most cases, Q0011P performed poorly compared to the others, even though Q1100P and Q0011P encode a similar type of states. This is likely because Q0011P does not encode information from time $t-2$ to encode states, which makes using the previous payoff (which is associated with time $t-2$) irrelevant.

- They use a specific combination of information. Although Q1001P and Q0110P have all the previous characteristics, they perform poorly in *Gxy* and *G*0*x* games. This is likely due to the combination of information they use.

In order to explain the previous four points, we consider the game *G*26 (refer to Table 5.13). In so doing, we analyze different algorithms and explain the performance of each (refer to Table 5.14). For simplicity, we consider in this analysis only algorithms that use two pieces of information to encode states (including Maverick) along with Q1110, which uses three pieces of information to encode its state.

Table 5.15 shows all the possible values for the history ($w = 2$) of the joint actions (16 values) in terms of $\{a^i_{t-2}, a^{-i}_{t-2}, a^i_{t-1}, a^{-i}_{t-1}\}$. Algorithms use part of this

	link 0	*link 1*
link 0	0.2, 0.2	1, 0.6
link 1	0.6, 1	0, 0

Table 5.13: The payoff matrix of *G*26 game.

Algorithm	P	C
Q1100	0.99	0.05
Q0011	0.34	0.41
Q1010	0.99	0.05
Q0101	0.91	0.00
Q1001	0.74	0.04
Q0110	0.20	0.00
Maverick	1.00	0.02
Q1110	0.74	0.89

Table 5.14: the performance of some algorithms in $G26$ game.

history ($w = 2$) to encode their state (while Q1111 uses all of this history to encode its state). The first column shows the possible values for history ($w = 2$) of joint actions. The following columns show how different algorithms encode this history in their state representation (the first column is also equivalent to how Q1111 encodes this history). For example, Q1001 only uses a^i_{t-2} and a^{-i}_{t-1}, so it encodes the history $\{0,1,1,0\}$ as $\{0,0\}$, while Q1010 encodes it as $\{0,1\}$. Maverick, which has the representation of $\{2-a^i_{t-2}-a^{-i}_{t-2}, a^i_{t-1}\}$ encodes it as $\{1,1\}$. The last two columns show the payoffs received by each agent in $G26$ game averaged over two iterations. The two preferred behaviors (histories) are $\{0,1,1,0\}$ and $\{1,0,0,1\}$. These two behaviors resemble the case of alternation in using the better link.

Table 5.16 shows the possible states for each algorithm with their probabilities (number of occurrences out of 16 times *prob*) and the average payoffs for both agents in these states ($a1$ *avg* and $a2$ *avg*). The preferred states (with their relevant information) were put in bold in all tables.

Looking at the field "prob" in each of the tables in 5.16, we see that algorithms that use two pieces of information have two preferred states out of four states. Hence, each of these two has the probability of 0.25 to be chosen randomly (assuming that algorithms initially act randomly, so they visit states with equal probabilities). Algorithm Q1110 has eight states, two of them are the preferred ones (probability=0.125 for each). From this, we can say that the good states in Q1110 have the probability of 0.125 for each, while in algorithms that encode two pieces of information they have the probability of 0.25 for each to be chosen. This

History	Mav	Q0011	Q0101	Q0110	Q1001	Q1010	Q1100	Q1110	a1	a2
0, 0, 0, 0	2, 0	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0, 0	0.2	0.2
0, 0, 0, 1	2, 0	0, 1	0, 1	0, 0	0, 1	0, 0	0, 0	0, 0, 0	0.6	0.4
0, 0, 1, 0	2, 1	1, 0	0, 0	0, 1	0, 0	0, 1	0, 0	0, 0, 1	0.4	0.6
0, 0, 1, 1	2, 1	1, 1	0, 1	0, 1	0, 1	0, 1	0, 0	0, 0, 1	0.1	0.1
0, 1, 0, 0	1, 0	0, 0	1, 0	1, 0	0, 0	0, 0	0, 1	0, 1, 0	0.6	0.4
0, 1, 0, 1	1, 0	0, 1	1, 1	1, 0	0, 1	0, 0	0, 1	0, 1, 0	1	0.6
0, 1, 1, 0	**1, 1**	**1, 0**	**1, 0**	**1, 1**	**0, 0**	**0, 1**	**0, 1**	**0, 1, 1**	**0.8**	**0.8**
0, 1, 1, 1	1, 1	1, 1	1, 1	1, 1	0, 1	0, 1	0, 1	0, 1, 1	0.5	0.3
1, 0, 0, 0	1, 0	0, 0	0, 0	0, 0	1, 0	1, 0	1, 0	1, 0, 0	0.4	0.6
1, 0, 0, 1	**1, 0**	**0, 1**	**0, 1**	**0, 0**	**1, 1**	**1, 0**	**1, 0**	**1, 0, 0**	**0.8**	**0.8**
1, 0, 1, 0	1, 1	1, 0	0, 0	0, 1	1, 0	1, 1	1, 0	1, 0, 1	0.6	1
1, 0, 1, 1	1, 1	1, 1	0, 1	0, 1	1, 1	1, 1	1, 0	1, 0, 1	0.3	0.5
1, 1, 0, 0	0, 0	0, 0	1, 0	1, 0	1, 0	1, 0	1, 1	1, 1, 0	0.1	0.1
1, 1, 0, 1	0, 0	0, 1	1, 1	1, 0	1, 1	1, 0	1, 1	1, 1, 0	0.5	0.3
1, 1, 1, 0	0, 1	1, 0	1, 0	1, 1	1, 0	1, 1	1, 1	1, 1, 1	0.3	0.5
1, 1, 1, 1	0, 1	1, 1	1, 1	1, 1	1, 1	1, 1	1, 1	1, 1, 1	0	0

Table 5.15: All the possible histories with $w = 2$ as encoded by different algorithms. The first column shows the possible histories, the next columns show how each algorithm encodes state given the history, and the last two columns show the payoffs for both agents averaged over two iterations.

is a natural consequence of the number of states (the bigger number of states you have, the slower you learn). Although Maverick has six states, but each of the two preferred states has the probability of 0.25 to be chosen. Hence, Maverick has increased the number of states but retained the same learning pace.

However, the big number of states reduces shadowing the good states with other bad ones. For example, in Q0110 the states $\{1,0,0,1\}$, $\{0,0,0,0\}$, $\{1,0,0,0\}$, $\{0,0,0,1\}$ are assumed to be the same. Visiting each of these four states is considered by Q0110 as visiting the state $\{0,0\}$. This decreases the average value of the state $\{0,0\}$ which should be representing one of the good states ($\{1,0,0,1\}$). On the other hand, in Q1110 the states $\{1,0,0,1\}$ and $\{1,0,0,0\}$ are assumed to be the same. We can see from the table of Q1110 in 5.16 that the good states (the ones in bold) have comparably higher averages for both agents in most cases.

Although using a bigger number of states reduces shadowing the good states with the bad ones, it is not the only factor that should be considered. Using specific combinations of information can help reducing the shadowing of the good states also. Maverick, Q1100 and Q0011 have good averages for the preferred states compared to other states. Furthermore, these algorithms encode two pieces of information. This makes these algorithms gain the advantage of decreasing the shad-

Q1100 states	prob	a1 avg	a2 avg		Q0011 states	prob	a1 avg	a2 avg
1,0	**0.25**	**0.525**	**0.725**		**1,0**	**0.25**	**0.525**	**0.725**
0,1	**0.25**	**0.725**	**0.525**		**0,1**	**0.25**	**0.725**	**0.525**
0,0	0.25	0.325	0.325		0,0	0.25	0.325	0.325
1,1	0.25	0.225	0.225		1,1	0.25	0.225	0.225
Q1010 states	prob	a1 avg	a2 avg		Q0101 states	prob	a1 avg	a2 avg
1,0	**0.25**	**0.45**	**0.45**		**1,0**	**0.25**	**0.45**	**0.45**
0,1	**0.25**	**0.45**	**0.45**		**0,1**	**0.25**	**0.45**	**0.45**
1,1	0.25	0.3	0.5		1,1	0.25	0.5	0.3
0,0	0.25	0.6	0.4		0,0	0.25	0.4	0.6
Q1001 states	prob	a1 avg	a2 avg		Q0110 states	prob	a1 avg	a2 avg
0,0	**0.25**	**0.5**	**0.5**		**0,0**	**0.25**	**0.5**	**0.5**
1,1	**0.25**	**0.4**	**0.4**		**1,1**	**0.25**	**0.4**	**0.4**
1,0	0.25	0.35	0.55		1,0	0.25	0.55	0.35
0,1	0.25	0.55	0.35		0,1	0.25	0.35	0.55
Maverick states	prob	a1 avg	a2 avg		Q1110 states	prob	a1 avg	a2 avg
1,0	**0.25**	**0.7**	**0.6**		**1,0,0**	**0.125**	**0.6**	**0.7**
1,1	**0.25**	**0.55**	**0.65**		**0,1,1**	**0.125**	**0.65**	**0.55**
2,0	0.13	0.4	0.3		0,1,0	0.125	0.8	0.5
2,1	0.13	0.25	0.35		1,0,1	0.125	0.45	0.75
0,0	0.13	0.3	0.2		1,1,0	0.125	0.3	0.2
0,1	0.13	0.15	0.25		0,0,1	0.125	0.25	0.35
					0,0,0	0.125	0.4	0.3
					1,1,1	0.125	0.15	0.25

Table 5.16: Probabilities and average values for the states in some algorithms.

owing of the good states, while retaining relatively fast learning. The performance of MaverickP and Q1100P in Table 5.14 are in accord with this analysis. However, the performance of Q0011P is not very good even though it encodes a very similar state representation to that of Q1100P. The reason might be that Q1100P and Q0011P use the previous payoff for Q-updates, and this payoff is related to time $t-2$. To Q0011, the previous payoff has no relevant information since Q0011 only

uses information from time $t-1$. This may also justify why Q0011C performs better than Q1100C.

Algorithms Q1010P and Q0101P have better performance than Q0110P and Q1001P in G26 game (refer to Table 5.14), although all four algorithms have similar levels of confusion (i.e. shadowing) between the good states and the bad states (refer to Tables 5.16). The reason here is very hard to capture depending on the previous arguments, but it might be that the good states in Q1010 and Q0101 have the same average value (0.45) which encourages alternation between these states. Note that agents in general would prefer to alternate between $\{1,0,0,1\}$ and $\{0,1,1,0\}$ i.e. Q1010 and Q0101 would prefer to alternate between $\{1,0\}$ and $\{0,1\}$. Furthermore, in Q1010 and Q0101, when agent1 is in state $\{1,0\}$ then agent2 is in state $\{0,1\}$. This alternation in these two algorithms would be easier than the case in Q0110 and Q1001 who have different average values (refer to Table 5.16).

5.2.5 All Games

We have seen in the previous sections that algorithms that use more information (Q1111P and Q1111C) perform better than the others in games of types *G*00 and *Gxx* for which the agent may receive the same payoffs for different action profiles which causes more confusion. On the other hand, algorithms that encode simpler state representations and use the previous payoffs to update its Q-values (suffixed with P) perform generally better (depending on their chosen information) than the others in games of types *G*0*x* and *Gxy*. These types, especially *Gxy* games, have less confusion in their payoffs (different payoffs are received in different action profiles). Hence, algorithms need to have less information.

It is important to see the overall performance of the considered algorithms in general. Showing for all games answers the question of "How much likely an algorithm, that was successful in one type of games, would perform in the other

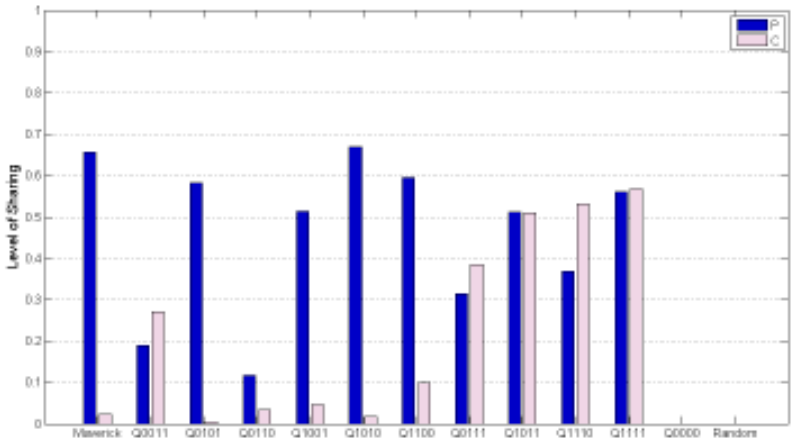


Figure 5.9: The performance of the algorithms in all games.

Top Algorithms
1) Q1010P (0.67)
2) MaverickP (0.66)
3) Q1100P (0.59)
4) Q0101P (0.58)
5) Q1111C (0.57)
6) Q1111P (0.56)
7) Q1110C (0.53)
8) Q1001P (0.51)
9) Q1011P (0.51)
10) Q1011C (0.51)

Table 5.17: Top algorithms in all games.

types".

Figure 5.9 and Table 5.17 show the overall performance of the algorithms in repeated congestion games. The results, as all the previous ones, are shown only in self-play. The preferred behavior is learning to share (i.e. alternation in choosing the links).

We can infer from the results that using less number of states, if chosen carefully, would make the algorithm perform even better than algorithms that use more number of states if the previous payoffs instead of the current payoffs were used to update the Q-values. The reason behind this might be that, algorithms with bigger number of states learn slower and the possibility of one algorithm learn before the other is bigger. On the other hand, algorithms with less number of states learn

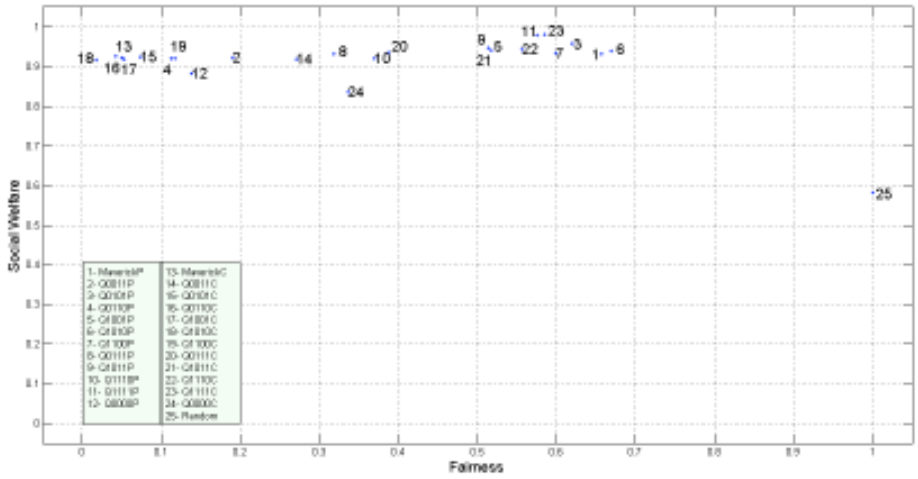


Figure 5.10: The average performance of all algorithms in all games. Fairness is the probability of having same average reward for both agents. Social Welfare is the summation of rewards for both agents divided by the maximum possible summation.

faster, and the possibility of one agent learning faster is less than the other (exploration of states before discovering a good strategy), but using less number of states make the algorithms myopic. Hence, the usage of previous payoff for update makes the algorithm more long-sighted since it encourages the actions that lead to better states instead of actions that yield higher immediate rewards.

From Figure 5.10, we can see that the Random (No.25) guarantees sharing the payoff with more than half of the maximum possible summation (fairness > 0.5). Random, Q0101P, Q1010P, and Q1111C (No.3, No.6, and No.23) produce the Pareto optimal outcomes. However, MaverickP, Q1100P, Q1110C, and Q1111P (No.1, No.7, No.22 and No.11) are also close to the Pareto optimal.

CHAPTER 6

Conclusion and Future Work

6.1 Conclusion

In this thesis, we have studied the case of routing self-interested vehicles in multi-agent environments where the cost of each link depends on the number of vehicles using it. Our aim has been to find an algorithm that, when applied to every vehicle, can secure this vehicle with a fair of cost. Furthermore, the group of these vehicles should reach an equilibrium in which the summation of the agents' utilities is maximized and the variance in agents' utilities is minimized.

We have shown that solving this problem in even a simple network is a difficult task, even for achieving only one part of the goal (maximizing the summation). However, we have proposed an algorithm (named Maverick) for a simpler problem. In this problem, two agents must select one of two links to travel on. The cost of each link depends on the number of agents using it. One of these links is better than the other. Maverick performed well in some versions of this game. However,

other algorithms we studied performed better in some variations of the game.

We have shown that algorithms that perform well in *G*00 and *Gxx* games, such as Q1011 and Q1111 (for both *P* and *C*) use in general a high amount of information to encode their states, while algorithms that perform well in *G*0*x* and *Gxy* such as MaverickP, Q1100P, Q1010P and Q0101P use less amount of information to encode their state. Additionally, these algorithms use the previous payoffs to update their current Q-values, use information from iteration $(t-2)$, and use a specific combination of states that reduce the shadowing of good states with bad states.

6.2 Future Work

In this study, we have provided some hypotheses to explain the performance of different algorithms. As we are mainly interested in understanding the kinds of learning biases algorithms should incorporate, it would be better to have further simulations as future work to back these hypotheses up and have solid explanations.

One limitation of our results is that they are shown only for self-play. Interesting future work would be to study the performance of the algorithms used in this study against each other and against different state-of-the-art algorithms. In real life, vehicles may use different navigation devices, each employing a different algorithm. It is important to have a robust algorithm that is not exploited by others.

Another limitation is that the family of algorithms we consider in this study have been implemented using similar parameters values to the ones used in M-Qubed. It would be interesting to vary the parameters values and observe their effects on the performance of algorithms.

Other possible extensions that can be added to the problem are to scale up the network graph by adding more links and/or nodes. We can likely add more links between the same two nodes and expect a similar performance from the algorithms

we studied. However, adding more nodes (and hence some more links) would make the problem definitely harder. In fact, extending the graph used in Problem II with one more step (in terms of number of nodes) would probably produce the one studied in Problem I.

Finally, an interesting extension would also be to increase the number of agents on the same graph. Consider for example the case of having three agents who try to use one of the two links. Establishing alternation between the three agents would be a challenging and interesting problem to look at.

APPENDIX A

Parameters Values and Explanations

A.1 Problem I

Parameter	*Value*	*Explanation*
α	0.1	Learning rate
γ	0.95	Discount rate
ε	0.05	Exploration Rate (ε-greedy)

Table A.1: Parameters values and explanations for Q-learning.

Parameter	*Value*	*Explanation*
α	0.1	Learning rate

Table A.2: Parameters values and explanations for Equation 4.3 used to update links estimates in Dijkstra's algorithm.

A.2 Problem II

Parameter	*Value*	*Explanation*
α	0.1	Learning rate
ε	0.1	Exploration Rate (ε-greedy)

Table A.3: Parameters values and explanations for stateless Q-learning (Q0000).

Parameter	*Value*	*Explanation*
α	0.1	Learning rate
γ	0.95	Discount rate
E	1000	Length of the random phase
ε	$\frac{40}{1000+max_s \kappa_t(s)}$	Exploration Rate (ε-greedy)
$Q(s,a)$	$\frac{r_i^{max}}{1-\gamma}$	The initial Q-values for all state-action pairs (s,a)

Table A.4: Parameters values and explanations for the family of algorithms considered in Problem II. $\kappa_t(s)$ is the number of times state s has been visited before time t, the random phase is the first E iterations where the agent chooses its action by random and observes the received rewards to define the maximum one, and r_i^{max} is the highest received reward by agent i during the random phase (E).

APPENDIX B

Abbreviations

FFQ Friend-or-Foe Q-learning

GBS Global Positioning System

MDP Markov Decision Process

NBS Nash Bargaining Solution

NE Nash Equilibrium

OAL Optimal Adaptive Learning

PRT Personal Rapid Transit

RL Reinforcement Learning

rNE repeated Nash Equilibrium

SPP Shortest Path Problem

Bibliography

[1] E. Anshelevich, A. Dasgupta, E. Tardos, and T. Wexler. Near-optimal network design with selfish agents. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 511–520. ACM, 2003. ISBN 1581136749.

[2] R.J. Aumann. Acceptable points in games of perfect information. *Pacific Journal of Mathematics*, 10(2):381–417, 1960.

[3] B. Awerbuch, Y. Azar, E.F. Grove, M.Y. Kao, P. Krishnan, and J.S. Vitter. Load balancing in the Lp norm. In *Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on*, pages 383–391. IEEE, 1995. ISBN 0818671831.

[4] T. Balch. Learning roles: Behavioral diversity in robot teams. *College of Computing Technical Report GIT-CC-97-12, Georgia Institute of Technology, Atlanta, Georgia*, 1997.

[5] R. Bellman. On a routing problem. *NOTES*, 16(1), 1958, google scholar.

[6] R.E. Bellman. *Dynamic programming*. Courier Dover Publications, 2003. ISBN 0486428095.

[7] H.R. Beom and H.S. Cho. A sensor-based navigation for a mobile robot using fuzzy logic and reinforcement learning. *Systems, Man and Cybernetics, IEEE Transactions on*, 25(3):464–477, 1995. ISSN 0018-9472.

[8] B. Bouzy and M. Métivier. Multi-agent learning experiments on repeated matrix games. 2010.

[9] M. Bowling. Convergence problems of general-sum multiagent reinforcement learning. In *Machine Learning-International Workshop Then Conference-*, pages 89–94. Citeseer, 2000.

[10] M. Bowling and M. Veloso. Rational learning of mixed equilibria in stochastic games. In *UAI2000 Workshop entitled Beyond MDPs: Representations and Algorithms*. Citeseer, 2000.

[11] M. Bowling and M. Veloso. Multiagent learning using a variable learning rate. *Artificial Intelligence*, 136(2):215–250, 2002. ISSN 0004-3702.

[12] L.S. Buriol, MGC Resende, and M. Thorup. Speeding up dynamic shortest path algorithms. *INFORMS Journal on Computing, accepted for publication*, 2007.

[13] C. Busch and M. Magdon-Ismail. Atomic routing games on maximum congestion. *Theoretical Computer Science*, 410(36):3337–3347, 2009. ISSN 0304-3975.

[14] J.A. Carnegie, P.S. Hoffman, New Jersey. Dept. of Transportation. Bureau of Research, and NJ Transit. *Viability of Personal Rapid Transit in New Jersey*. New Jersey Dept. of Transportation, 2007.

[15] H.M. Choset. *Principles of robot motion: theory, algorithms, and implementation*. MIT press, 2005. ISBN 0262033275.

[16] G. Christodoulou and E. Koutsoupias. The price of anarchy of finite congestion games. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 67–73. ACM, 2005. ISBN 1581139608.

[17] C. Claus and C. Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *Proceedings of the National Conference on Artificial Intelligence*, pages 746–752. John Wiley & Sons Ltd, 1998.

[18] J.A. Coelho Jr, E.G. Araujo, M. Huber, and R.A. Grupen. Dynamical categories and control policy selection. In *Intelligent Control (ISIC), 1998. Held jointly with IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA), Intelligent Systems and Semiotics (ISAS), Proceedings*, pages 459–464. IEEE. ISBN 0780344235.

[19] T.H. Cormen. *Introduction to algorithms*. The MIT press, 2001. ISBN 0262032937.

[20] J.R. Correa, A.S. Schulz, and N.E. Stier Moses. Computational complexity, fairness, and the price of anarchy of the maximum latency problem. *Integer Programming and Combinatorial Optimization*, pages 59–73, 2004.

[21] J.W. Crandall and M.A. Goodrich. Learning to compete, coordinate, and cooperate in repeated games using reinforcement learning. *Machine Learning*, 82(3):281–314, 2011.

[22] E. De Jong. Non-random exploration bonuses for online reinforcement learning. In *Collected papers from the AAAI-97 workshop on multiagent learning*, 1997.

[23] C. Demetrescu and G. Italiano. Fully dynamic all pairs shortest paths with real edge weights. In *focs*, page 260. Published by the IEEE Computer Society, 2001.

[24] G. Desaulniers and F. Soumis. An efficient algorithm to find a shortest path for a car-like robot. *Robotics and Automation, IEEE Transactions on*, 11(6): 819–828, 1995. ISSN 1042-296X.

[25] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959. ISSN 0029-599X.

[26] F.F. Dragan. Estimating all pairs shortest paths in restricted graph families: a unified approach. *Journal of Algorithms*, 57(1):1–21, 2005. ISSN 0196-6774.

[27] S.E. Dreyfus. An appraisal of some shortest-path algorithms. *Operations Research*, 17(3):395–412, 1969. ISSN 0030-364X.

[28] P. Dubey. Inefficiency of Nash equilibria. *Mathematics of Operations Research*, pages 1–8, 1986. ISSN 0364-765X.

[29] A. Ephremides and S. Verdu. Control and optimization methods in communication network problems. *Automatic Control, IEEE Transactions on*, 34(9): 930–942, 1989. ISSN 0018-9286.

[30] D. Eppstein. Finding the k shortest paths. *SIAM J. Comput.*, 28(2):652–673, 1998.

[31] A. Epstein, M. Feldman, and Y. Mansour. Efficient graph topologies in network routing games. *Games and Economic Behavior*, 66(1):115–125, 2009. ISSN 0899-8256.

[32] A. Fabrikant, C. Papadimitriou, and K. Talwar. The complexity of pure Nash

equilibria. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 604–612. ACM, 2004. ISBN 1581138520.

[33] J.A. Filar and K. Vrieze. *Competitive Markov decision processes*. Springer Verlag, 1997. ISBN 0387948058.

[34] R.W. Floyd. Algorithm 97: Shortest paths* 1. *Communications of the ACM*, 5:345, 1962.

[35] D. Fotakis, S. Kontogiannis, E. Koutsoupias, M. Mavronicolas, and P. Spirakis. The structure and complexity of Nash equilibria for a selfish routing game. *Theoretical Computer Science*, 410(36):3305–3326, 2009. ISSN 0304-3975.

[36] D. Frigioni, A. Marchetti-Spaccamela, and U. Nanni. Fully Dynamic Algorithms for Maintaining Shortest Paths Trees* 1. *Journal of Algorithms*, 34 (2):251–281, 2000. ISSN 0196-6774.

[37] D. Frigioni, A. Marchetti-Spaccamela, and U. Nanni. Fully dynamic shortest paths in digraphs with arbitrary arc weights* 1. *Journal of Algorithms*, 49(1): 86–113, 2003. ISSN 0196-6774.

[38] D.C. Gazis. *Traffic science*. John Wiley & Sons, 1974.

[39] E. Gelenbe, P. Liu, and J. Laine. Genetic algorithms for route discovery. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 36(6):1247–1254, 2006. ISSN 1083-4419.

[40] A.V. Goldberg and C. Harrelson. Computing the shortest path: A search meets graph theory. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 156–165. Society for Industrial and Applied Mathematics, 2005. ISBN 0898715857.

[41] J. Hu and M.P. Wellman. Multiagent reinforcement learning: Theoretical framework and an algorithm. In *Proceedings of the Fifteenth International Conference on Machine Learning*, volume 242, pages 242–250. Citeseer, 1998.

[42] J. Hu and M.P. Wellman. Experimental results on Q-learning for general-sum stochastic games. In *Proceedings of the Seventeenth International Conference on Machine Learning*, page 414. Morgan Kaufmann Publishers Inc., 2000. ISBN 1558607072.

[43] J. Hu and M.P. Wellman. Nash Q-learning for general-sum stochastic games. *The Journal of Machine Learning Research*, 4:1039–1069, 2003. ISSN 1532-4435.

[44] L.P. Kaelbling, M.L. Littman, and A.W. Moore. Reinforcement learning: A survey. *Arxiv preprint cs/9605103*, 1996.

[45] K. Kamei and M. Ishikawa. More effective reinforcement learning by introducing sensory information. In *Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on*, volume 4, pages 3185–3188. IEEE, 2004. ISBN 0780383591.

[46] V. King. Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs. In *Foundations of Computer Science, 1999. 40th Annual Symposium on*, pages 81–89. IEEE, 1999. ISBN 0769504094.

[47] P. Klein, S. Rao, M. Rauch, and S. Subramanian. Faster shortest-path algorithms for planar graphs. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pages 27–37. ACM, 1994. ISBN 0897916638.

[48] J. Kleinberg, E. Tardos, and Y. Rabani. Fairness in routing and load balancing. In *focs*, page 568. Published by the IEEE Computer Society, 1999.

[49] E. Koutsoupias and C. Papadimitriou. Worst-case equilibria. In *Proceedings of the 16th annual conference on Theoretical aspects of computer science*, pages 404–413. Springer-Verlag, 1999. ISBN 354065691X.

[50] M. Kuczma. *An introduction to the theory of functional equations and inequalities: Cauchy's equation and Jensen's inequality*. Birkhauser, 2009. ISBN 3764387483.

[51] S.M. LaValle. *Planning algorithms*. Cambridge Univ Pr, 2006. ISBN 0521862051.

[52] J.J. Leonard and H.F. Durrant-Whyte. *Directed sonar sensing for mobile robot navigation*. Springer, 1992. ISBN 0792392426.

[53] M.L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the eleventh international conference on machine learning*, volume 157, page 163. Citeseer, 1994.

[54] M.L. Littman. Friend-or-foe Q-learning in general-sum games. In *Machine Learning-International Workshop Then Conference-*, pages 322–328, 2001.

[55] R.D. Luce, H. Raiffa, and T. Teichmann. Games and decisions. *Physics Today*, 11:33, 1958.

[56] A. Machell, P. Eng, L. AP, and T.P. Branch. Towards a Sustainable Transportation System. *Retrieved October*, 7, 2010.

[57] M. Mavronicolas and P. Spirakis. The price of selfish routing. *Algorithmica*, 48(1):91–126, 2007. ISSN 0178-4617.

[58] D. Monderer and L.S. Shapley. Potential games. *Games and economic behavior*, 14:124–143, 1996. ISSN 0899-8256.

[59] K. Mueller and S.P. Sgouridis. Simulation-based analysis of personal rapid transit systems: service and energy performance assessment of the Masdar City PRT case. *Journal of Advanced Transportation*. ISSN 2042-3195.

[60] J. Nash. Non-cooperative games. *Annals of mathematics*, pages 286–295, 1951. ISSN 0003-486X.

[61] J.F. Nash Jr. The bargaining problem. *Econometrica: Journal of the Econometric Society*, pages 155–162, 1950. ISSN 0012-9682.

[62] J. Nie and S. Haykin. A dynamic channel assignment policy through Q-learning. *Neural Networks, IEEE Transactions on*, 10(6):1443–1455, 1999. ISSN 1045-9227.

[63] CS Orloff. Routing a fleet of M vehicles to/from a central facility. *Networks*, 4(2):147–162, 1974. ISSN 1097-0037.

[64] M.J. Osborne and A. Rubinstein. *Bargaining and markets*. Academic Press, 1990. ISBN 0125286325.

[65] S. Panahi and MR Delavar. A GIS-based Dynamic Shortest Path Determination in Emergency Vehicles. *World Applied Sciences Journal*, 3(1):88–94, 2008. ISSN 1818-4952.

[66] C. Papadimitriou. Algorithms, games, and the internet. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 749–753. ACM, 2001. ISBN 1581133499.

[67] W. Poundstone and J. Von Neumann. *Prisoner's dilemma*. Anchor Books, 1993. ISBN 038541580X.

[68] G. Ramalingam and T.W. Reps. An incremental algorithm for a generalization of the shortest-path problem. *J. Algorithms*, 21(2):267–305, 1996.

[69] F. Ricca and P. Tonella. Understanding and restructuring Web sites with ReWeb. *Multimedia, IEEE*, 8(2):40–51, 2001. ISSN 1070-986X.

[70] R.W. Rosenthal. A class of games possessing pure-strategy Nash equilibria. *International Journal of Game Theory*, 2(1):65–67, 1973. ISSN 0020-7276.

[71] T. Roughgarden. How unfair is optimal routing? In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 203–204. Society for Industrial and Applied Mathematics, 2002. ISBN 089871513X.

[72] T. Roughgarden. The maximum latency of selfish routing. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 980–981. Society for Industrial and Applied Mathematics, 2004. ISBN 089871558X.

[73] T. Roughgarden and É. Tardos. How bad is selfish routing? *Journal of the ACM (JACM)*, 49(2):236–259, 2002. ISSN 0004-5411.

[74] G.A. Rummery and M. Niranjan. *On-line Q-learning using connectionist systems*. Citeseer, 1994.

[75] T.W. Sandholm and R.H. Crites. Multiagent reinforcement learning in the iterated prisoner's dilemma. *Biosystems*, 37(1-2):147–166, 1996.

[76] M.W.P. Savelsbergh and M. Sol. The general pickup and delivery problem. *Transportation science*, 29(1):17–29, 1995. ISSN 0041-1655.

[77] R. Selten. The chain store paradox. *Theory and decision*, 9(2):127–159, 1978.

[78] S. Sen, M. Sekaran, and J. Hale. Learning to coordinate without sharing information. In *Proceedings of the National Conference on Artificial Intelligence*, pages 426–426. John Wiley & Sons Ltd, 1994.

[79] L.S. Shapley. Stochastic games. *Proceedings of the National Academy of Sciences of the United States of America*, 39(10):1095, 1953.

[80] Y. Shoham and K. Leyton-Brown. *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*. Cambridge Univ Pr, 2009. ISBN 0521899435.

[81] R.S. Sutton and A.G. Barto. *Reinforcement learning: An Introduction*, volume 9. MIT Press, 1998.

[82] M. Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning*, volume 337. Citeseer, 1993.

[83] G. Tesauro. Temporal difference learning and TD-Gammon. *Communications of the ACM*, 38(3):58–68, 1995. ISSN 0001-0782.

[84] G. Tesauro. Extending Q-learning to general adaptive multi-agent systems. *Advances in neural information processing systems*, 16, 2004.

[85] J. Von Neumann, O. Morgenstern, A. Rubinstein, and H.W. Kuhn. *Theory of games and economic behavior*. Princeton Univ Pr, 2007. ISBN 0691130612.

[86] X.F. Wang and T. Sandholm. Reinforcement learning to play an optimal Nash equilibrium in team Markov games. *Advances in neural information processing systems*, pages 1603–1610, 2003. ISSN 1049-5258.

[87] C.J.C.H. Watkins. *Learning from delayed rewards*. PhD thesis, King's College, 1989.

[88] C.J.C.H. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3):279–292, 1992. ISSN 0885-6125.

[89] G. Weifi. Learning to coordinate actions in multi-agent systems. *Readings in agents*, page 481, 1998.

[90] G. Yen and T. Hickey. Reinforcement learning algorithms for robotic navigation in dynamic environments. In *Neural Networks, 2002. IJCNN'02. Proceedings of the 2002 International Joint Conference on*, volume 2, pages 1444–1449. IEEE, 2002. ISBN 0780372786.