# Comparison of Multiagent Learning Algorithms in Ad Hoc Teams

*Stefano V. Albrecht*


Master of Science
Artificial Intelligence
School of Informatics
University of Edinburgh
2011

# Abstract

Multiagent Learning (MAL) is the algorithmic study of learning in a group of two or more agents. If the agents are based on different algorithms, and if there is no form of prior coordination between the agents, then this is called an ad hoc team problem [48]. Following a literature review [2] and a research proposal [1], the work at hand compares the performance of five MAL algorithms in ad hoc teams. These include the Joint Action Learner [12], the Conditional Joint Action Learner [3], Win or Learn Fast with Policy Hill Climbing [6], Modified Regret-Matching [21], and the Nash Q-Learner [24]. The algorithms are evaluated in a range of strategic games, including no-conflict games in which the players agree on what is most preferred, and conflict games in which the players disagree on what is most preferred [40]. In addition, we use an evaluation procedure proposed by Stone et al. [48]. Our performance criteria include the convergence rate, the final expected payoff, social welfare and fairness, and the rates of different solution types. From the results we conclude that *(a)* all algorithms perform well in some sense (i.e., there is no clear winner), and *(b)* the performance of an algorithm ultimately depends on the solution concept that is considered most appropriate for the ad hoc team problem at hand.

# Acknowledgements

I would like to thank Dr. Subramanian Ramamoorthy who agreed to supervise me and thus allowed me to realise my ideas in the form of a thesis. Special thanks go to my brother, Franco Hans Albrecht, and my mother, Heide-Marie Albrecht, without whom I would not be here.

# Declaration

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

(*Stefano V. Albrecht*)

# Table of Contents

# List of Algorithms

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Multiagent Learning is the study of two or more agents attempting to learn something (e.g. to achieve a common goal or to maximise their utility) in a common environment. As a baseline approach, one often considers homogeneous groups of agents in which all agents are identical. A more complicated setting arises when considering heterogeneous groups of agents in which two or more agents are based on different algorithms.[1] However, in both types of groups, agents may still employ some form of prior coordination in order to collaborate efficiently. For example, agents may know in advance how the other agents behave in certain situations, or they may use some kind of a communication protocol. In ad hoc teams, we consider heterogeneous groups of agents without any form of prior coordination. This means that the agents will have to learn how to collaborate efficiently without knowing a priori who the other agents are.

Ad Hoc Team Collaboration is a problem that was stated only recently by Stone et al. [48]. The problem is to design an agent that can learn how to collaborate with an unknown group of agents without any form of prior coordination. It is important to study this problem because today and in future, as we continue to employ more agents in a growing number of areas, it is likely that these agents will have to collaborate with each other at some point (despite the fact that they may not know each other). For example, consider a group of agents such as robots in an assembly line or a team of internet trading agents. From an economic point of view, we would like these agents to be able to continue their work efficiently even if we extend the group by adding newer or different agents of the same type (Stone et al. study such a scenario in [50]). Another example is

[1] In other contexts, such as robotics, the expression *heterogeneous group* may actually refer to a group of agents with different equipment rather than different algorithms [15]. We account for this by the set of actions available to each agent.

the domain of human-robot interaction in which a robot has to collaborate with a human (e.g. a mobile robot giving directions in an airport [61]). This can be cast as an ad hoc team problem if the robot is not given any specific information about the human it has to collaborate with.

Following a literature review [2] and a research proposal [1], the work at hand compares the performance of five different multiagent learning algorithms in ad hoc teams, using a comprehensive set of strategic games. The algorithms include the Joint Action Learner [12], the Conditional Joint Action Learner [3], Win or Learn Fast with Policy Hill Climbing [6], Modified Regret-Matching [21], and the Nash Q-Learner [24]. The games range from no-conflict games in which the players agree on what is most preferred, to conflict games in which the players disagree on what is most preferred [40]. We evaluate the algorithms in ad hoc teams of different sizes. The range of possible team members comprises all investigated algorithms, and a team may contain more than one agent of the same type. We use a variety of performance criteria, including the convergence rate, the final expected payoff, social welfare and fairness, and the rates of several solution types.

All algorithms used in our experiments were primarily evaluated in homogeneous groups of agents. The main contribution of our work is to show how these algorithms perform in ad hoc teams. Furthermore, of all approaches covered by our selection of algorithms, we wish to identify those that appear to be better suited for ad hoc team problems. This knowledge can then be used for the design of new algorithms. We believe that this is a new contribution to the field, since, to our knowledge, there has been no study that investigated the same range of algorithms in similar settings. However, we note that Stone et al. [4] recently presented a paper in which they proposed a new algorithm for an ad hoc agent and evaluated it in teams with fixed-behaviour agents. This is different from our work since we consider teams of learning agents, while Stone et al. considered teams in which one agent learned and the other ones followed fixed behaviours. In their conclusion, Stone et al. stated that one direction for possible future work would be to investigate ad hoc teams of learning agents. Our work may be considered a small contribution towards this direction.

Another motivation behind our work is the simulation framework based on which we conducted our experiments. The framework offers a simulator for arbitrary repeated

games. Furthermore, it contains functions that check if a given solution satisfies certain properties (e.g. if a solution is a Nash equilibrium or Pareto-optimal; see section 2.1), and functions that compute solutions with such properties. All of the algorithms that we investigate are implemented in the framework, and it allows for easy addition of further algorithms. In a recent paper [47], Shoham et al. note that *"it would be useful to have a learning-algorithm repository"*. Our framework marks the beginning of such a repository. We will continue to extend the framework, and it will serve us as a basis for future studies. By making the framework available to the community, we hope that it will establish as a useful tool for research in Multiagent Learning.

The remainder of this report is organised as follows: Chapter 2 provides introductions to related fields of research and chapter 3 describes the algorithms we investigate in our experiments. Chapter 4 discusses the experimental setup and chapter 5 presents the experimental results. Finally, chapter 6 concludes the experiments and chapter 7 discusses possible future work. We provide three appendices: Appendix A briefly describes the simulation framework, appendix B lists the games used in our experiments, and appendix C provides a complete overview of the simulation results.

# Chapter 2

# Background

This chapter provides short introductions to several related fields of research, including Game Theory (section 2.1), Reinforcement Learning (section 2.2), Multiagent Learning (section 2.3), and Ad Hoc Team Collaboration (section 2.4). Note that these are very brief introductions, covering only those aspects which are relevant for our work. However, pointers to further readings are given at the end of each section.

## 2.1 Game Theory

Game Theory (GT) is the study of strategic decision making among multiple decision makers in a formal structure called a *game*. A game may assume numerous forms, each of which possessing different rules and properties. The kind of game we are interested in is called a *strategic form game*.

A strategic form game is a tuple $(N, (A_i)_{i \in N}, (u_i)_{i \in N})$ where $N = \{1, ..., n\}$ is the set of *players*, $A_i$ is the set of *actions* available to player $i$, and $u_i : A \to \mathbb{R}$ is the *payoff function* of player $i$, where $A = A_1 \times ... \times A_n$. The game is played as a single-shot game, meaning that each player $i$ simultaneously chooses an action $a_i \in A_i$ and receives a payoff $u_i(a_1, ..., a_n)$. Each player $i$ chooses its action based on a *strategy* $s_i$ which is a probability distribution over $A_i$, that is, $\forall a \in A_i : s_i(a) \geq 0$ and $\sum_{a \in A_i} s_i(a) = 1$. A strategy $s_i$ is called a *pure* strategy if $s_i(a) = 1$ for some $a \in A_i$. A strategy is called a *mixed* strategy if it is not a pure strategy. We refer to the set of player strategies by the *strategy profile* $S = (s_1, ..., s_n)$. A strategy profile is called a *pure* profile if all strategies contained in the profile are pure strategies. A strategy profile is called a *mixed* profile if it is not a pure profile. Given a strategy profile $S = (s_1, ..., s_n)$, the *expected payoff* of player

$i$ is defined by $U_i(S) = \sum_{a_1,...,a_n} s_1(a_1) * ... * s_n(a_n) * u_i(a_1,...,a_n)$.

There are several solution concepts for a game. The most common concept is that of the *Nash equilibrium*. A strategy profile is a Nash equilibrium (NE) if no player can unilaterally deviate from its strategy to improve its payoff, provided the other players play according to the profile. Formally, a strategy profile $S = (s_1,...,s_n)$ is a Nash equilibrium if $\forall \hat{s_i} : U_i(s_1,...,s_i,...,s_n) \geq U_i(s_i,...,\hat{s_i},...,s_n)$ for all $i \in N$. This corresponds to each player playing a best response to the other players. In 1951, John Nash proved that every strategic form game has at least one (possibly mixed) Nash equilibrium [34]. Another important solution concept is the *Pareto optimality*. A strategy profile $S = (s_i,...,s_n)$ is Pareto-optimal (PO) if there is no other strategy profile $\hat{S}$ such that $\forall i : U_i(\hat{S}) \geq U_i(S)$ and $\exists i : U_i(\hat{S}) > U_i(S)$. Intuitively, this means that there is no other profile in which all players are equally well off and at least one player is better off.

Strategic form games can be conveniently represented using *payoff matrices* (which is why they are sometimes called *matrix games*). Without loss of generality, assume that $A_i = \{1,...,m_i\}$ for each $i \in N$. Then, each player $i$ has a payoff matrix $M^i \in \mathbb{R}^{|A_1| \times ... \times |A_n|}$, and for a given joint action $(a_1,...,a_n)$, it receives the payoff $(M^i)_{a_1,...,a_n} = u_i(a_1,...,a_n)$. For instance, a game with 2 players and 2 actions each can be represented as follows:

| | |
|---|---|
| $(M^1)_{1,1}, (M^2)_{1,1}$ | $(M^1)_{1,2}, (M^2)_{1,2}$ |
| $(M^1)_{2,1}, (M^2)_{2,1}$ | $(M^1)_{2,2}, (M^2)_{2,2}$ |

Player 1 corresponds to the row player while player 2 corresponds to the column player. Each cell $(i, j) \in \{1,2\}^2$ of the table (corresponding to player 1 choosing action $i$ and player 2 choosing action $j$) corresponds to a payoff pair $((M^1)_{i,j}, (M^2)_{i,j})$ where the first payoff goes to player 1 and the second payoff goes to player 2. In general, we call these games $m_1 \times ... \times m_n$ games, based on the number of players and actions. For example, the game shown above is called a $2 \times 2$ game.

There is a wide range of literature on Game Theory. Von Neumann and Morgenstern laid out some of the earliest foundations [56]. Fudenberg and Tirole provide a comprehensive textbook for students [16]. Nisan et al. assembled the work of many authors on the algorithmic foundations of Game Theory [35].

## 2.2 Reinforcement Learning

Reinforcement Learning (RL) is a type of machine learning in which an agent tries to optimise its decision making based on rewards provided by the environment. A learning problem in the RL setting can be posed as a *Markov Decision Process* (MDP). An MDP is a tuple $(S, (A_s)_{s\in S}, \delta, r)$ where $S$ is the set of *states* of the MDP, $A_s$ is the set of *actions* available to the agent in state $s$, $\delta : S \times A \times S \to [0,1]$ is the *state transition function*, and $r : S \times A \times S \to \mathbb{R}$ is the *reward function*, where $A = \cup_{s\in S} A_s$.

An MDP starts at time $t = 0$ in some initial state $s_0 \in S$. In state $s_t$, the agent chooses an action $a_t \in A_{s_t}$, after which the MDP transitions to a state $s' \in S$ with probability $\delta(s_t, a_t, s')$, and the agent receives the reward $r_t = r(s_t, a_t, s')$. This process is repeated using the state $s_{t+1} = s'$. The agent chooses its actions based on a *policy* $\pi : S \times A \to [0,1]$, which is a probability distribution over $A_s$ for each $s \in S$. The goal of the agent is to learn an optimal policy $\pi^*$ that maximises its *expected total reward* $\mathrm{E}[\sum_{t=0}^{\infty} r_t]$. We consider expected total rewards since the state transitions are stochastic. To avoid the problem of infinite total rewards, one typically uses a discounted total reward $\mathrm{E}[\sum_{t=0}^{\infty} \gamma^t r_t]$, where $0 < \gamma < 1$ is the discount factor.

Various algorithms have been developed that solve this problem. A popular way to implement an RL agent is called Q-learning [57]. Q-learning works by maintaining a table $Q$ that contains a value for each state-action pair $(s,a)$. In each iteration of the algorithm, the agent perceives the current state $s$, chooses an action $a$, observes the successor state $s'$, and receives a reward $r$. It then updates the Q-table using the update rule $Q(s,a) = (1-\alpha)Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a')]$, where $0 \leq \alpha \leq 1$ is the learning rate and $0 < \gamma < 1$ is an optional discount factor. Actions are chosen based on a *selection strategy*. A common strategy is called $\varepsilon$-*greedy*. Therein, in each state $s$, the agent chooses the *greedy* action $a^* = \arg\max_a Q(s,a)$ with probability $(1-\varepsilon)$, and a random action with probability $\varepsilon$. This algorithm has been shown to converge to an optimal solution [57], provided that the selection strategy chooses all actions infinitely often, and that the learning rate $\alpha$ satisfies the standard stochastic approximation conditions [25].

Sutton and Barto provide a very approachable textbook on RL [52]. A more recent textbook with emphasis on function approximation is provided by Busoniu et al. [11]. Kaelbling brought together recent papers of a number of researchers [26].

## 2.3 Multiagent Learning

In classical machine learning applications, one typically considers a single entity that tries to learn something (see e.g. [33]). Multiagent Learning (MAL) studies the case in which multiple learning entities, called agents, operate in a common environment. Each agent is intelligent by itself and operates fully autonomously. However, other than in classical machine learning, one is less interested in the performance of single agents, but rather in the performance of the entire group. Thus, the most interesting MAL problems are such that they cannot be solved (or optimised) by single agents, but only through co-operation among multiple agents.

MAL problems can be posed using the concept of *stochastic games* [44]. A stochastic game (which is a stochastic process) is a tuple $(N, S, (A_s^1)_{s\in S}, ..., (A_s^n)_{s\in S}, \delta, r_1, ..., r_n)$ where $N = \{1, ..., n\}$ is the set of agents, $S$ is the set of states of the process, $A_s^i$ is the set of actions available to agent $i$ in state $s$, $\delta : S \times A \times S \to [0,1]$ is the state transition function, and $r_i : S \times A \times S \to \mathbb{R}$ is the reward function of agent $i$, where $A = A^1 \times ... \times A^n$ and $A^i = \cup_{s\in S} A_s^i$. A stochastic game with a single state is called a *repeated* game. If all agents in a stochastic game are of the same type, then the game is called a *self-play* game. We call a stochastic game a *mixed-play* game if it is not a self-play game.

A stochastic game starts in some initial state $s_0 \in S$. In each state $s_t$, all agents simultaneously choose their actions $a_t^1, ..., a_t^n$, after which the process transitions to a successor state $s'$ with probability $\delta(s_t, a_t, s')$ and each agent $i$ receives a payoff $r_i(s_t, a_t, s')$, where $a_t = (a_t^1, ..., a_t^n)$. The process then continues with $s_{t+1} = s'$. Other than in MDPs, the objective may not necessarily be to maximise the total reward of each agent. For example, a possible objective may be to play a Nash equilibrium in each state. One can see that stochastic games combine Markov decision processes with strategic form games: Agents correspond to players, states correspond to different games, and the functions $\delta$ and $r$ are modified to account for joint actions $(a_1, ..., a_n)$ rather than single actions. Thus, while the MDP models *single*-agent *multi*-state problems and the strategic form game models *multi*-agent *single*-state problems, the stochastic game models *multi*-agent *multi*-state problems.

There are a number of algorithms for the MAL domain. Most of these algorithms are designed to either converge to Nash equilibria or to Pareto-optimal solutions. Some

specify more sophisticated solution concepts to which they try to converge. However, most of the algorithms have in common that they were primarily evaluated in self-play games. In this study, we evaluate five algorithms in mixed-play games: the Joint Action Learner [12], the Conditional Joint Action Learner [3], Win or Learn Fast with Policy Hill Climbing [6], Modified Regret-Matching [21], and the Nash Q-Learner [24]. All of these algorithms are discussed in chapter 3.

We recommend two textbooks for further readings: Wooldridge provides a comprehensive introduction to the field [60]. Shoham and Leyton-Brown provide a more advanced textbook with focus on game-theoretic aspects [45]. Recent surveys of the literature in the field can be found in [10], [36], [46], and (less recent) [51].

## 2.4 Ad Hoc Team Collaboration

Ad Hoc Team Collaboration is a branch of Multiagent Learning which is concerned with the ability of a single agent (or *ad hoc* agent) to collaborate with an a priori unknown group of agents. Specifically, we assume that there is no prior coordination between the ad hoc agent and the remaining agents. Possible forms of coordination are, for instance, knowledge about the behaviour of other agents or some kind of a protocol. Given a set of potential (but unknown) team members, a good ad hoc agent will learn how to collaborate efficiently with any constellation of team members. Therefore, other than in traditional MAL, one is not primarily interested in the performance of the entire team, but in the performance of the ad hoc agent as part of the team.

The problem of designing good ad hoc agents was posed only recently by Stone et al. [48]. Therein, they challenge the scientific community:

*"To create an autonomous agent that is able to efficiently and robustly collaborate with previously unknown teammates on tasks to which they are all individually capable of contributing as team members."*

In the same paper, they propose an algorithm to evaluate two ad hoc agents for a given set of potential team members and tasks. We will use this algorithm for some of our own experiments and discuss it in section 5.3. Prior to that, Stone et al. [50] published a paper in which they defined optimal strategies for an ad hoc agent collaborating

with a fixed-behaviour teammate in an environment modelled as a k-armed bandit. In the same year (after they posed the challenge [48]), Stone et al. [49] released the second paper in line with their newly established field of research. They presented an algorithm that would lead a fixed greedy agent towards an optimal joint action in a simple repeated game in which both agents have identical payoff functions.

In their most recent paper [4], Stone et al. propose an ad hoc agent that tries to identify its teammates by observing their behaviour and comparing it with a database of known behaviours. In addition, it learns a new model for the observed behaviour using a tree classifier. The agent combines both the database and the learned model in a Bayesian fashion to anticipate the behaviour of its teammates. It was evaluated in the predator domain in which a group of agents tries to capture another agent. The experimental results showed that the proposed ad hoc agent performed quite well, and in general better than those agents that just mimic their teammates.

Literature on Ad Hoc Team Collaboration is sparse since this is a fairly new field of research. However, besides the work by Stone et al. [50, 48, 49, 4], there are a number of other researchers that investigate similar problems. Specifically, we refer to the work by Knudson and Tumer [28], and Brafman and Tennenholtz [7]. Furthermore, there has been some work on protocol-based team coordination [14, 19, 53]. Wu et al. [61] proposed an algorithm that generates cooperative stage games in each state to learn and predict the behaviour of its teammates.

## 2.5 Interchangeable terms and expressions

We close this chapter by noting that some of the terms and expressions discussed in the previous sections are interchangeable, both formally and informally. Specifically, in our work, we consider the terms *player* / *agent*, *payoff* / *reward*, and *strategy* / *policy*, respectively, to be interchangeable. By extension, the expressions *expected payoff* / *expected reward* and *strategy profile* / *policy profile* have the same meaning. We usually prefer the terms *player*, *payoff*, and *strategy*, but use the terms *agent*, *reward*, and *policy* when this is deemed necessary.

# Chapter 3

# Algorithms

This chapter discusses the algorithms that we investigate in our experiments. For each algorithm, we provide basic background information and pseudo-code. Note that our pseudo-code is restricted to stateless stochastic games (i.e., repeated games) since this is the only type of game we consider in our experiments. However, extending them to stochastic games with multiple states is straightforward with one exception:

Whereas in stateless stochastic games it suffices to update Q-tables using the formula $Q(a) \leftarrow (1-\alpha)Q(a) + \alpha r$ (where $a$ is a joint action, $r$ is the reward, and $\alpha$ is the learning rate), for stochastic games with multiple states we have to use a formula that accounts for different future rewards, dependent on the current state and action, e.g. $Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a')]$ (where $s$ is the current state, $s'$ is the successor state, and $\gamma$ is a discount factor).[1] This is since we intend to learn a policy that is optimal with respect to future rewards, which themselves are dependent on the future states. Of course, this applies only to those algorithms that use Q-tables.

The following discussions will use the basic concepts and notation introduced in chapter 2. Specifically, recall that the set of agents is denoted by $N = \{1,...,n\}$, that each agent $i \in N$ has access to a set of actions $A_i = \{1,...,m_i\}$, and that the space of joint actions is denoted by $A = A_1 \times ... \times A_n$. Furthermore, given a joint action $(a_1,...,a_n) \in A$, we denote the *counter profile* to player $i$ by $a_{-i} = (a_1,...,a_{i-1},a_{i+1},...,a_n)$, and the space of such counter profiles by $A_{-i} = \times_{j \neq i} A_j$. Finally, for notational convenience, we assume that the algorithm to be discussed is assigned the number $i$, and that all other agents are assigned a number $j \in N - \{i\}$.

[1] There is, in fact, a variety of different update rules for Q-tables; see [52].

## 3.1 Joint Action Learner

The Joint Action Learner (JAL) [12, 54] is an extension of the single-agent Q-learning algorithm (see section 2.2) to the multi-agent setting. The first modification regards the Q-table, which is now based on the set of joint actions $A$, rather than just on the set of actions $A_i$ for a single agent. In addition, the algorithm observes the frequencies with which each agent selects its actions. These frequencies are used to estimate the agents' strategies, which in turn can be used to estimate the expected payoff $E(a_i)$ each action $a_i \in A_i$ yields (this is called *fictitious play* [9]). Claus and Boutilier [12] propose a soft-max selection strategy which chooses action $a_i$ with probability $p(a_i) = \frac{\exp(E(a_i)/\tau)}{\sum_{a \in A_i} \exp(E(a)/\tau)}$, where $\tau$ is a decay factor that gradually decreases over time. For high values of $\tau$ this corresponds to a uniform randomisation among all actions, while for low values the strategy favours actions with high expected payoffs. As $\tau \to 0$, the strategy will be entirely greedy. For simplicity, we use an $\varepsilon$-greedy strategy. Uther and Veloso [54] provide a description of the algorithm for multi-state stochastic games.

**Algorithm 1** Joint Action Learner

**Initialise:**

$Q(a_1, ..., a_n) \leftarrow 0$ for all joint actions $(a_1, ..., a_n) \in A$

$f_j(a_j) \leftarrow 0$ for all $j \in N - \{i\}$; $t \leftarrow 0$

**loop**

**if** $t > 0$ **then**

Compute expected payoff for each action $a_i \in A_i$:

$E(a_i) \leftarrow \sum_{a_{-i} \in A_{-i}} \left[ \prod_{j \neq i} \frac{f_j(a_j)}{t} \right] Q(a_i, a_{-i})$

Choose action $a_i \in A_i$ (e.g. $\varepsilon$-greedy):

- with probability $\varepsilon$, choose $a_i$ randomly from $A_i$

- with probability $(1 - \varepsilon)$, choose $a_i = \arg\max_a E(a)$

**else**

Choose random action $a_i \in A_i$

**end if**

Observe joint action $(a_1, ..., a_n)$ and reward $r$

Update Q-table: $Q(a_1, ..., a_n) \leftarrow (1 - \alpha) Q(a_1, ..., a_n) + \alpha r$

Update frequencies: $f_j(a_j) \leftarrow f_j(a_j) + 1$ for all $j \neq i$

$t \leftarrow t + 1$

**end loop**

## 3.2 Conditional Joint Action Learner

The Conditional Joint Action Learner (CJAL), proposed by Banerjee and Sen [3], can be considered a modification of JAL. Like JAL, it uses a Q-table based on the space of joint actions. However, rather than to compute marginal probabilities $p_j(a_j)$ of agent $j$ choosing action $a_j$, it computes conditional probabilities $p_j(a_j|a_i)$ of agent $j$ choosing action $a_j$, given that agent $i$ chooses action $a_i$. To do so, it observes the frequencies of entire joint actions (as opposed to single actions). The conditional probabilities are then used to estimate the expected payoffs $E(a_i)$ for each action $a_i \in A_i$.

The algorithm was evaluated [3] in self-play on the set of all structurally distinct strictly ordinal $2 \times 2$ conflict games (section B.2 provides a listing of these games). The same experiments were conducted for JAL and WOLF-PHC (section 3.3). The results showed that CJAL outperformed both algorithms in several criteria, including social welfare and social fairness (section 4.2). An interesting detail is that CJAL had a higher rate of Pareto-optimal solutions (86%) than JAL (81%) and WOLF-PHC (75%). Note that in 75% of the games the Nash equilibria are also Pareto-optimal.

**Algorithm 2** Conditional Joint Action Learner

**Initialise:**

$Q(a_1,...,a_n) \leftarrow 0$ and $f(a_1,...,a_n) \leftarrow 0$ for all joint actions $(a_1,...,a_n) \in A$

**loop**

Compute expected payoff for each action $a_i \in A_i$:

**if** $f(a_i) > 0$, where $f(a_i) = \sum_{a_{-i} \in A_{-i}} f(a_i, a_{-i})$ **then**

$E(a_i) \leftarrow \sum_{a_{-i} \in A_{-i}} \frac{f(a_i,a_{-i})}{f(a_i)} Q(a_i, a_{-i})$

**else**

$E(a_i) = 0$

**end if**

Choose action $a_i \in A_i$ (e.g. ε-greedy):

- with probability ε, choose $a_i$ randomly from $A_i$
- with probability $(1-\varepsilon)$, choose $a_i = \arg\max_a E(a)$

Observe joint action $(a_1,...,a_n)$ and reward $r$

Update Q-table: $Q(a_1,...,a_n) \leftarrow (1-\alpha)Q(a_1,...,a_n) + \alpha r$

Update frequencies: $f(a_1,...,a_n) \leftarrow f(a_1,...,a_n) + 1$

**end loop**

## 3.3 Win or Learn Fast with Policy Hill Climbing

Win or Learn Fast with Policy Hill Climbing (WOLF-PHC) is a multiagent algorithm proposed by Bowling and Veloso [6]. It is based on hill climbing (see, e.g., [41]) in the space of mixed strategies. An important feature of the algorithm is that it uses a variable learning rate. If the algorithm believes itself to be winning, it uses a slower learning rate (denoted by $\delta_w$) to cautiously account for any changes its opponents make. On the other hand, if the algorithm finds itself losing, it uses a faster learning rate (denoted by $\delta_l$) to quickly change its strategy. Bowling and Veloso proved [6] that the combination of policy hill climbing with a variable learning rate would always converge to a Nash equilibrium in the class of $2 \times 2$ self-play games. However, using a variety of examples, they could also show its applicability to a wider range of games.

**Algorithm 3** Win or Learn Fast with Policy Hill Climbing

**Initialise:**

$Q(a) \leftarrow 0$ and $\pi(a) \leftarrow \frac{1}{m_i}$ for all $a \in A_i$

$\hat{\pi} \leftarrow \pi$; $t \leftarrow 0$

**loop**

Choose action $a_i \in A_i$ (e.g. $\varepsilon$-greedy):

- with probability $\varepsilon$, choose $a_i$ randomly from $A_i$
- otherwise, choose $a_i \in A_i$ with probability $\pi(a_i)$

Observe reward $r$

Update Q-table: $Q(a_i) \leftarrow (1-\alpha)Q(a_i) + \alpha r$

**if** $t > 0$ **then**

Update $\hat{\pi}$ for all $a \in A_i$:

$\hat{\pi}(a) \leftarrow \hat{\pi}(a) + \frac{1}{t}\left(\pi(a) - \hat{\pi}(a)\right)$.

Update $\pi$ for all $a \in A_i$:

$\pi(a) \leftarrow \pi(a) + \Delta_a$, where

$$\Delta_a = \begin{cases} -\delta_a, & a \neq \arg\max_{a'} Q(a') \\ \sum_{a' \neq a} \delta_{a'}, & else \end{cases},$$

$$\delta_a = \min\left(\pi(a), \frac{\delta}{m_i - 1}\right), \delta = \begin{cases} \delta_w, & \sum_{a'} \pi(a')Q(a') > \sum_{a'} \hat{\pi}(a')Q(a') \\ \delta_l, & else \end{cases}$$

**end if**

$t \leftarrow t + 1$

**end loop**

A crucial part of the algorithm is how to determine whether it is winning or losing. In their proof, the authors define the algorithm to be winning if its current expected payoff is greater than the expected payoff it would receive if it played an equilibrium strategy. However, computing an equilibrium strategy requires knowledge about the payoff functions of all agents. Therefore, to make the algorithm more practical, they approximate the equilibrium strategy using an average policy $\hat{\pi}$. The average policy at any point in time $t$ is simply the average over all policies used prior to $t$. Note that WOLF-PHC requires less information than JAL and CJAL. Other than the latter two, it does neither require knowledge about the actions taken by the other agents, nor the payoffs received by them.

## 3.4 Modified Regret-Matching

Modified Regret-Matching (ModReg) [21] is a modification of Regret-Matching [20], both proposed by Hart and Mas-Colell. The authors consider two kinds of regret: the *conditional regret* of an action $\hat{a}_i$ at time $t$ with respect to another action $a_i$ is the regret for not having chosen $\hat{a}_i$ whenever $a_i$ was chosen, up until $t$. Formally, this corresponds to $CR_t(\hat{a}_i, a_i) = \frac{1}{t}\sum_{\tau \leq t: a_i^\tau = a_i} u_i(\hat{a}_i, a_{-i}^\tau) - \frac{1}{t}\sum_{\tau \leq t: a_i^\tau = a_i} u_i(a^\tau)$.[2] The *Hannan regret* of an action $\hat{a}_i$ at time $t$ is the regret for not having played $\hat{a}_i$ at all times until $t$. This, in turn, corresponds to $HR_t(\hat{a}_i) = \frac{1}{t}\sum_{\tau=1}^{t} u_i(\hat{a}_i, a_{-i}^\tau) - \frac{1}{t}\sum_{\tau=1}^{t} u_i(a^\tau)$. By $a^\tau$ we denote the joint action played at time $\tau$. In both definitions, $u_i(a^\tau)$ in the right summand can be replaced by $r_i^\tau$, which is the reward received by agent $i$ at time $\tau$. The left summands can be approximated by the formulas $\frac{1}{t}\sum_{\tau \leq t: a_i^\tau = \hat{a}_i} \frac{\pi_i^\tau(a_i)}{\pi_i^\tau(\hat{a}_i)} r_i^\tau$ and $\frac{1}{t}\sum_{\tau \leq t: a_i^\tau = \hat{a}_i} \frac{1}{\pi_i^\tau(\hat{a}_i)} r_i^\tau$, respectively, where $\pi_i^\tau$ denotes the policy of agent $i$ at time $\tau$. As one can see, the modified regrets do not anymore require the payoff function $u_i$.

Each regret type can be used to update the current policy $\pi_i^t$. Let $a_i^t$ denote the action taken by agent $i$ at time $t$. The update rule for the conditional regret is defined as

$$\pi_i^{t+1}(a) = \begin{cases} \left(1 - \frac{\delta}{t^\gamma}\right) \min\left\{\frac{1}{\mu}\left[CR_t(a, a_i^t)\right]_+, \frac{1}{m_i - 1}\right\} + \frac{\delta}{t^\gamma}\frac{1}{m_i}, & a \neq a_i^t \\ 1 - \sum_{a' \in A_i: a' \neq a} \pi_i^{t+1}(a'), & a = a_i^t \end{cases}$$

where $[x]_+ = \max\{x, 0\}$. It suffices to choose $\mu > 2B_i(m_i - 1)$, where $B_i$ denotes an upper bound on $|u_i(a_1, ..., a_n)|$ for all $(a_1, ..., a_n) \in A$. The parameters $\delta$ and $\gamma$, with

[2] Recall that $u_i$ denotes the payoff function of player $i$.

$0 < \delta < 1$ and $0 < \gamma < \frac{1}{4}$, specify the minimum probability for each action at each time step. The update rule for the Hannan regret is defined as

$$\pi_i^{t+1}(a) = \left(1 - \frac{\delta}{t^\gamma}\right) \frac{[HR_t(a)]_+}{\sum_{a' \in A_i} [HR_t(a')]_+} + \frac{\delta}{t^\gamma} \frac{1}{m_i}$$

for all actions $a \in A_i$, where $0 < \delta < 1$ and $0 < \gamma < \frac{1}{2}$.

The first update rule (CR-rule) is guaranteed to converge to the set of correlated equilibria, a superset of the set of Nash equilibria [21]. The second update rule (HR-rule) is *Hannan-consistent*, meaning that the Hannan regret for each action will be lower than or equal to zero as $t$ goes to infinity. However, both properties were shown for self-play in repeated games (i.e. stateless stochastic games) only. Finally, note that both update rules require only knowledge about the received rewards at each time step. Therefore, both ModReg and WOLF-PHC require less information than JAL and CJAL.

We tested both update rules in self-play on a range of $2 \times 2$ and $3 \times 3$ games to decide which one to use in our experiments. Both rules produced similar results. However, the HR-rule converged more quickly in most games. As a result, we decided to use the HR-rule in our experiments. Note also that it has fewer parameters than the CR-rule. Algorithm 4 shows the pseudo-code for ModReg using the HR-rule.

**Algorithm 4** Modified Regret-Matching (HR-rule)

**Initialise:**

$\pi^1(a) \leftarrow \frac{1}{m_i}$ for all $a \in A_i$

$t \leftarrow 1$

**loop**

Choose action $a_i^t \in A_i$ with probability $\pi^t(a_i^t)$

Observe reward $r^t$

Compute regret for all actions $a \in A_i$:

$HR_t(a) \leftarrow \frac{1}{t} \sum_{\tau \leq t : a_i^\tau = a} \frac{1}{\pi^\tau(a)} r^\tau - \frac{1}{t} \sum_{\tau=1}^{t} r^\tau$

Update policy $\pi$ for all actions $a \in A_i$:

$\pi^{t+1}(a) \leftarrow (1 - \delta_t) \frac{[HR_t(a)]_+}{\sum_{a' \in A_i} [HR_t(a')]_+} + \delta_t \frac{1}{m_i}$, where $[x]_+ = max\{x, 0\}$

$t \leftarrow t + 1$

**end loop**

## 3.5 Nash Q-Learner

The Nash Q-Learner (NashQ) was first proposed by Hu and Wellman [22]. Similar to JAL and CJAL, it is an extension of standard Q-learning in which the Q-table is based on the set of joint actions. However, other than JAL and CJAL, it maintains a table $Q_j$ for every agent $j \in N$. Given a state $s$, the Q-tables $(Q_1, ..., Q_n)$ define a strategic form game for which the algorithm computes a (mixed) Nash equilibrium $(s_1, ..., s_n)$. It then uses the equilibrium strategy $s_i$ to choose its action (with suitable exploration, e.g. ε-greedy). An interesting detail of the algorithm is the way in which it updates the Q-tables. For multi-state stochastic games, after observing the joint action $(a_1, ..., a_n)$, the joint reward $(r_1..., r_n)$, and the successor state $s'$, it updates each $Q_j$ using the formula

$$Q_j(s, a_1, ..., a_n) = (1-\alpha)Q_j(s, a_1, ..., a_n) + \alpha\left[r_j + \gamma Nash_j(s', Q_1, ..., Q_n)\right]$$

where $0 < \gamma < 1$ is a discount factor, and $Nash_j(s', Q_1, ..., Q_n)$ is the expected payoff to agent $j$ when all agents play a Nash equilibrium in the strategic form game defined by $(Q_1, ..., Q_n)$ in state $s'$. The update rule accounts for future rewards in future states, assuming that all agents play the same Nash equilibrium in each state.

**Algorithm 5** Nash Q-Learner

**Initialise:**

$Q_j(a_1, ..., a_n) \leftarrow 0$ for all $j \in N$ and $(a_1, ..., a_n) \in A$

**loop**

Compute a NE $(s_1, ..., s_n)$ for the strategic form game defined by $(Q_1, ..., Q_n)$

Choose action $a_i \in A_i$ (e.g. ε-greedy):

- with probability ε, choose $a_i$ randomly from $A_i$
- otherwise, choose $a_i \in A_i$ with probability $s_i(a_i)$

Observe joint action $(a_1, ..., a_n)$ and joint reward $(r_1, ..., r_n)$

Update Q-tables for all agents $j \in N$:

$Q_j(a_1, ..., a_n) \leftarrow (1-\alpha)Q_j(a_1, ..., a_n) + \alpha r_j$

**end loop**

Hu and Wellman proved that NashQ (in self-play) converges to a Nash equilibrium under very restrictive conditions [22]. However, after testing the algorithm in a series of experiments, they showed that it worked reasonably well even if the convergence conditions were not satisfied [23]. In another paper [24], Hu and Wellman discussed possible

extensions for NashQ that could improve its convergence properties. The same paper contains discussions on further experiments showing that NashQ is applicable even if the convergence conditions are not met.

Note that NashQ, of all algorithms considered in our study, requires the greatest amount of information. The algorithm assumes that, in each time step, it can observe the actions taken by all agents and their received rewards. This assumption is very strong since it implies that the agent knows the payoff functions of all agents, or alternatively, that it has a means to observe their rewards (e.g. through communication). Given that the agent will act in ad hoc teams, this assumption is extremely unrealistic. Nonetheless, we still include NashQ in our experiments to see how well the concept of computing Nash equilibria during the learning process works in ad hoc teams.

An important aspect of NashQ is the computation of Nash equilibria for arbitrary strategic form games. It turns out that this is an extremely difficult problem that has been the focus of considerable research [37]. There are only a few algorithms that solve the problem for certain kinds of games. The Lemke-Howson algorithm [29] computes Nash equilibria for the class of 2-player games. For the class of n-player games, one may use Simplicial Subdivision [55], or the Govindan-Wilson algorithm [17] which was extended and efficiently implemented by Blum et al. [5]. In our implementation of NashQ, we use a relatively new algorithm proposed by Porter et al. [39]. The algorithm uses a heuristic to quickly find Nash equilibria for n-player games. It has been shown to outperform the previously mentioned algorithms in a number of experiments [39]. However, note that each of these algorithms has an exponential worst-case time complexity.

## 3.6 Justification for selection

We justify our selection of algorithms by the fact that they cover a variety of approaches. JAL and CJAL follow the *opponent modelling* [54] approach, meaning that they try to learn a model of the other agents' behaviours such that their actions can be anticipated. JAL learns the marginal probabilities of its opponents' actions while CJAL learns the conditional probabilities of its opponents' actions given its own actions. WOLF-PHC does not model its opponents. Instead, it uses a hill climbing method in the space of mixed strategies to find optimal strategies. Yet another approach is followed by RegMat. This algorithm tries to minimise its regret for not having taken any other actions. Finally,

NashQ plays a Nash equilibrium strategy in each state, based on the observed rewards of all agents. By doing so, it tries to persuade the other agents to play a Nash equilibrium strategy as well. We believe that this is an interesting selection of algorithms. By studying their behaviour in ad hoc teams, we hope to improve our understanding of a good ad hoc agent.

There are more algorithms that could have been included in our experiments. For example, the Minimax Q-Learner (called *Minimax-Q*) by Littman [30], of which NashQ is an extension, has been shown to converge in self-play in 2-player zero-sum[3] games [32]. Furthermore, the Friend-or-Foe Q-Learner (called *FFQ*), also by Littman [31], which in turn is an extension of NashQ, has been shown to converge in self-play in general-sum games under less restrictive conditions than NashQ. We chose not to include these algorithms in our experiments because their assumptions are too strong for ad hoc team scenarios. Like NashQ, both algorithms assume that they can observe the payoffs of all agents. In addition, Minimax-Q is restricted to zero-sum games (that is, it assumes that the payoffs of the other agents are correlated with its own payoffs), and FFQ assumes that it knows a priori whether an agent is a "friend" or a "foe".

Another interesting algorithm was proposed by Sen et al. [43]. The algorithm is based on JAL and introduces a novel revelation feature. In each state of the game, it decides whether or not to reveal its action to the other agents. These agents would then choose their own actions with full knowledge about the action of the revealing agent. Sen et al. showed that this algorithm converged more often to Pareto-optimal solutions than JAL. We did not include this algorithm in our experiments because its revelation feature renders it unsuitable for ad hoc team problems. Revealing an action to another agent requires a means to communicate this piece of information. In addition, it assumes that the other agent can interpret this information, which then would just correspond to a communication protocol. However, in ad hoc teams we cannot assume that the agents can communicate with each other, and we can certainly not assume that they share any protocols. Note that without the revelation feature the algorithm reduces to JAL.

We note that we considered including the algorithm AWESOME in our experiments. This algorithm was proposed by Conitzer and Sandholm [13]. It was shown to converge

[3] A game is called a *zero-sum* game if, for any strategy profile, the expected payoffs of all players sum up to zero. A game which is not a zero-sum game is called a *general-sum* game.

to a Nash equilibrium in self-play, and to learn optimal strategies against agents that become stationary (that is, they converge to a fixed behaviour). Note, however, that both properties were only shown for repeated games. The idea behind this algorithm is to learn a best response to the other agents if they appear to be stationary, and to play a precomputed equilibrium strategy if the other agents seem to adapt their strategies. We decided not to include AWESOME in our experiments because it requires knowledge about the payoff functions of all agents in the game.[4] This is a strong assumption, and it becomes unrealistic in the context of ad hoc team scenarios.

[4]Conitzer and Sandholm note that the algorithm could learn the payoff functions in an online fashion if they are not known in advance. After the algorithm has gathered sufficient information about the payoff functions of all agents, it would begin to behave as intended. However, this would require a "boot-up" phase which we do not include in our experiments.

# Chapter 4

# Experimental setup

When reporting on an experiment, it is important to provide a complete specification of the setting in which the experiment was conducted. This is important for two reasons: First, it ensures that the experiment is presented in the right context. Second, it gives the reader the information required to reproduce the experiment. Therefore, this chapter contains discussions on the general setup of our experiments. Section 4.1 describes the games on which we tested the algorithms. Section 4.2 provides definitions of our performance criteria. Finally, in section 4.3 we specify the parameter settings and the selection strategies for each of the algorithms.

## 4.1 Games

Our experiments consist of three parts. Each part tests the algorithms on a different set of games. Recall that our experiments are restricted to repeated games. This means that each game can be specified by the payoff matrices of the strategic form game that is being repeated, and by the number of repetitions.[1] Each of our games is played with 100,000 repetitions. We will discuss this decision in section 4.3.

Before we discuss any details of the games used in our experiments, we will have to provide some definitions. First, recall that the expression "$m_1 \times ... \times m_n$ game" denotes a game with $n$ players in which each player $i \in \{1,...,n\}$ has access to $m_i$ actions, given by $A_i = \{1,...,m_i\}$. An *ordinal* game is a game in which each player ranks the possible outcomes $(a_1,...,a_n) \in A_1 \times ... \times A_n$ according to its preferences. So, in a $m_1 \times ... \times m_n$ game, where there are $k = m_1 * ... * m_n$ possible outcomes, each player ranks each out-

[1] Note that a repeated game may have infinitely many repetitions.

come $(a_1, ..., a_n)$ by a number between 1 (least preferred) and $k$ (most preferred). The ranks are used as payoffs, that is, $u_i(a_1, ..., a_n) \in \{1, ..., k\}$ for each player $i$ and outcome $(a_1, ..., a_n)$. Now, a *strictly ordinal* game is an ordinal game in which no player assigns the same rank to any two outcomes. That is, each outcome has a unique rank.

A (strictly) ordinal game is called a *no-conflict* game if all players have the same set of most preferred outcomes. The game is called a *conflict* game if it is not a no-conflict game. Finally, consider a set $G$ of (strictly) ordinal games. The games in $G$ are said to be structurally distinct if no game in $G$ can be reproduced by a transformation of any other game in $G$. The set of possible transformations include interchanging the rows, columns, and players, and any combinations of these.

The first part of our experiments tests the algorithms on the set of all 21 structurally distinct strictly ordinal $2 \times 2$ no-conflict games. The second part tests the algorithms on the set of all 57 structurally distinct strictly ordinal $2 \times 2$ conflict games. A complete listing of all of these games can be found in appendix B. Finally, the third part of our experiments tests the algorithms in 500 randomly generated strictly ordinal $2 \times 2 \times 2$ games, each of which may be a conflict game or a no-conflict game.

We now justify our design decisions: We chose to use ordinal games since these are an abstraction of all non-ordinal games (i.e. games with "real" payoffs). The specific magnitudes of the payoffs should not matter for an algorithm as long as they specify the preferences of the algorithm. For example, if two outcomes have the payoffs 50 and 20 for a certain player, then this player has the same preferences if the outcomes have the payoffs 8 and 6: the first outcome is most preferred. Another reason to choose ordinal games is its normalisation effect. We do not have to consider the specific payoffs as we are only interested in whether the agent managed to arrive at the best or second best (etc.) outcome. This way we can conveniently compare the results of multiple games without considering the specific payoffs. Furthermore, we chose to use strictly ordinal games to keep the number of games down to a reasonable amount when considering all structurally distinct games. Also, in our opinion, strictly ordinal games are more interesting than ordinal games since in the former the players have clear preferences between all outcomes. In the latter, two or more outcomes may have the same preferences, in which case the decision problem is simplified.

We use $2 \times 2$ games due to their simplicity and to obtain reasonable computation times, and, furthermore, because these are used in most papers (e.g. [12, 3, 6, 22, 27]). We also use $2 \times 2 \times 2$ games to see how well the algorithms scale to greater numbers of players. We consider the complete range of structurally distinct strictly ordinal $2 \times 2$ games to include all possible realisations in terms of non-ordinal $2 \times 2$ games. Finally, we chose to divide this range into no-conflict and conflict games since these represent two different levels of difficulty. In strictly ordinal no-conflict games, it is relatively easy to arrive at a solution that is best for all players, since all players have the same most preferred outcome. However, in strictly ordinal conflict games, there is no such outcome, and so the players will have to arrange some form of a compromise.

## 4.2 Performance criteria

This section provides definitions of the performance criteria used in our experiments. Specifically, these include the convergence rate, the final expected payoff, social welfare and fairness, and the rate of different solution types. The solution types we consider are the Nash equilibrium, Pareto-optimal solutions, and, in addition, welfare-optimal and fairness-optimal solutions. We will define the latter two in section 4.2.4.

The following definitions apply to *plays* of repeated games. A play $P_\Gamma$ of a repeated game $\Gamma$ is a tuple $((\pi^t)_{t=1,...,t_f}, (a^t)_{t=1,...,t_f}, (r^t)_{t=1,...,t_f})$, where $t \in \{1,...,t_f\}$ denotes the time, $t_f$ denotes the final time of the play, $\pi^t = (\pi_1^t, ..., \pi_n^t)$ denotes the mixed strategy profile at time $t$, $a^t = (a_1^t, ..., a_n^t)$ denotes the joint action at time $t$, and $r^t = (r_1^t, ..., r_n^t)$ denotes the joint reward at time $t$. Recall that $N = \{1,...,n\}$ denotes the set of players, and $A_i = \{1,...,m_i\}$ the set of actions available to player $i$.

### 4.2.1 Convergence rate

The convergence rate of an agent is defined as the percentage of plays in which the agent converged. Let $P_\Gamma$ be a play of some repeated game $\Gamma$. We say that agent $i \in N$ converged in $P_\Gamma$ if its strategies $\pi_i^t$ stay within a tolerance bound of $\pm 5\%$ in the endmost 20% of the play. Formally, agent $i$ converged in play $P_\Gamma$ if for all $t \in \{t_s, ..., t_f\}$ and $j \in \{1,...,m_i\}$, where $t_s = 0.8 t_f$, we have $|\pi_i^{t_s}(j) - \pi_i^t(j)| \leq 0.05$. We believe that a tolerance of $\pm 5\%$ is acceptable for our purposes. For $t_f = 100{,}000$, this results in 20,000 repetitions in which the strategy must not deviate more than $\pm 5\%$.

### 4.2.2 Final expected payoff

The final expected payoff of an agent is an approximation of the agent's expected payoff after having learned for $t_f$ repetitions. The approximation is based on the endmost 20% of the play. Let $P_\Gamma$ be a play of some repeated game $\Gamma$. The final expected payoff of agent $i \in N$ in play $P_\Gamma$ is formally defined as $\bar{r}_i = \frac{1}{t_f - t_s + 1} \sum_{t=t_s}^{t_f} r_i^t$, where $t_s = 0.8 t_f$. For $t_f = 100{,}000$, this corresponds to the average of the endmost 20,000 rewards of agent $i$.

If agent $i$ converged in $P_\Gamma$, then its final expected payoff will be close to the expected payoff obtained by any strategy $\pi_i^t$ with $t \geq t_s$. However, if the agent did not converge, then we assume that it will continue to play the same strategies it played during the endmost 20% of the play. Therefore, the final expected payoff can be computed as the average of the endmost 20% of its rewards. This works particularly well if the agent does not play mixed strategies, as is the case for JAL and CJAL (see chapter 3). For example, using JAL, one often observes that it changes its pure strategy in regular intervals so as to approximate a mixed strategy (e.g. as part of a Nash equilibrium). Here, taking the average of the endmost payoffs will approximate the expected payoff quite well. A series of tests showed that the endmost 20% of rewards were sufficient in most cases to approximate the agent's expected payoff.

### 4.2.3 Social welfare and fairness

Given a mixed profile $\pi = (\pi_1, ..., \pi_n)$, the social welfare is defined as the sum of the expected payoffs of all agents. Formally, this corresponds to $\sum_{i=1}^{n} U_i(\pi)$ where $U_i(\pi)$ is the expected payoff of agent $i$ under the strategy profile $\pi$ (see section 2.1). The social fairness is defined as the product of the expected payoffs of all agents. This corresponds to $\prod_{i=1}^{n} U_i(\pi)$. Note that the social fairness, when fixing the social welfare, reaches its maximum if all agents have exactly the same expected payoff, corresponding to the general notion of fairness.

Let $P_\Gamma$ be a play of some repeated game $\Gamma$. We define the social welfare and fairness of play $P_\Gamma$, respectively, as the sum and product of the final expected payoffs $\bar{r}_i$ of all agents $i \in N$. Formally, this corresponds to $\sum_{i=1}^{n} \bar{r}_i$ for the social welfare, and $\prod_{i=1}^{n} \bar{r}_i$ for the social fairness.

### 4.2.4 Rate of different solution types

We measure the rates of four different solution types. These include the Nash equilibrium (NE) rate, the Pareto optimality (PO) rate, the welfare optimality (WO) rate, and the fairness optimality (FO) rate. We will discuss each of these in the following sections.

The definitions in the following sections rely on the notion of *averaged final profiles*. Let $P_\Gamma$ be a play for some repeated game $\Gamma$. The averaged final profile (or AFP) of $P_\Gamma$ is the average of all profiles in the endmost 20% of the play. Formally, we denote the AFP by $\bar{\pi} = (\bar{\pi}_1, ..., \bar{\pi}_n)$ where $\bar{\pi}_i = \frac{1}{t_f - t_s + 1} \sum_{t=t_s}^{t_f} \pi_i^t$ and $t_s = 0.8\,t_f$. We use the endmost 20% of the play to approximate mixed strategies for agents that play pure strategies. Recall that the expected payoff of player $i$ under a strategy profile $\pi$ is denoted by $U_i(\pi)$.

#### 4.2.4.1 NE rate

The NE rate is defined as the percentage of plays in which the AFP constitutes a Nash equilibrium. Recall that a strategy profile is a Nash equilibrium if no player can unilaterally deviate from its strategy to improve its payoff (that is, provided that the other players stick to their strategies). Given a play $P_\Gamma$ of some repeated game $\Gamma$, we determine if the play resulted in a Nash equilibrium by solving the following linear program for each player $i \in N$:

$$\begin{aligned} \text{Maximise:} \quad & U_i(\bar{\pi}_1, ..., \pi_i, ..., \bar{\pi}_n) \\ \text{Subject to:} \quad & \forall j \in A_i : \pi_i(j) \geq 0 \\ & \textstyle\sum_{j \in A_i} \pi_i(j) = 1 \end{aligned}$$

We denote the optimised profile for player $i$ by $\pi^i$. If, for any $i \in N$, the expected payoff under the optimised profile, $U_i(\pi^i)$, exceeds the expected payoff under the AFP, $U_i(\bar{\pi})$, by more than 5%, then we conclude that the play $P_\Gamma$ did not result in a Nash equilibrium. Formally, we define $P_\Gamma$ to result in a Nash equilibrium if $\forall i \in N : \frac{U_i(\pi^i)}{U_i(\bar{\pi})} \leq 1.05$.

#### 4.2.4.2 PO rate

The PO rate is defined as the percentage of plays in which the AFP is Pareto-optimal. Recall that a strategy profile is Pareto-optimal if there is no other profile in which all players are equally well off and at least one player is better off. Similar to [3], we determine if a profile is Pareto-optimal by measuring its distance to the *Pareto front*.

Consider a strategic form game $\Gamma = (N, (A_i)_{i \in N}, (u_i)_{i \in N})$ as defined in section 2.1. The space of possible expected (joint) payoffs of $\Gamma$ is a convex polytope in $\mathbb{R}^n$ that has one dimension for each player $i \in N$. It is defined as the convex hull of all joint payoffs $(u_1(a), ..., u_n(a))$ for all joint actions $a \in A_1 \times ... \times A_n$. Each point in this payoff polytope corresponds to a tuple of expected payoffs $(U_1(\pi), ..., U_n(\pi))$ for some profile $\pi = (\pi_1, ..., \pi_n)$. The *Pareto front* of a payoff polytope $\Phi$ is defined as the set of Pareto-optimal faces of $\Phi$. A face $\phi$ of $\Phi$ is Pareto-optimal if the corresponding strategy profiles of all points on $\phi$ are Pareto-optimal. Now, given a play $P_\Gamma$ of $\Gamma$, we say that it results in a Pareto-optimal solution if the minimal orthogonal distance of its AFP, $\bar{\pi}$, when projected onto the payoff polytope $\Phi$ of $\Gamma$, to the Pareto front of $\Gamma$ is not greater than 0.1.[2] Our tests showed that this value works well for ordinal games.

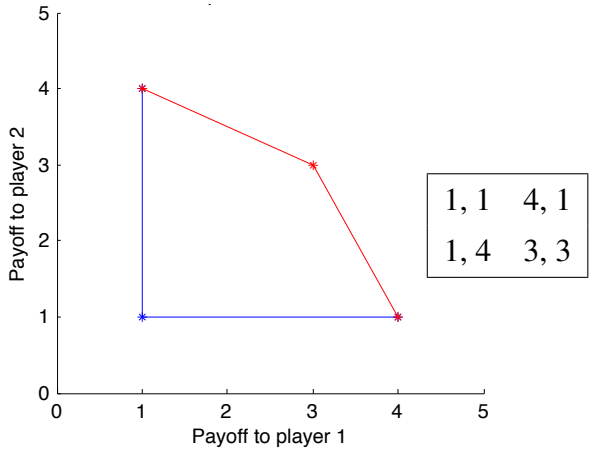



Figure 4.1: Example of a Pareto front.

Figure 4.1 shows the payoff polytope of the $2 \times 2$ game given in the table above. The asterisks in the plot mark the payoff pairs $(u_1(a_1, a_2), u_2(a_1, a_2))$ for each $a_1, a_2 \in \{1, 2\}$. The edges correspond to the faces of the polytope. Together they define the convex hull of all payoff pairs. The red edges show the Pareto-optimal faces of the polytope. Together they define the Pareto front of the payoff polytope. This means that every strategy profile whose expected payoff is on the Pareto front[3] constitutes a Pareto-optimal profile.

[2] Another way to check if a strategy profile is Pareto-optimal would be to compare its expected payoffs to those of the strategy profile corresponding to the nearest Pareto-optimal payoff point. If, for example, the expected payoff of each player is at least 95% of the expected payoff in the nearest point, then the strategy profile may be considered Pareto-optimal. We use our approach because it subsumes all of this by checking the distance, and because it allows for a greater deviation of one player if the others have lower deviations.

[3] Or, as in our case, close to the Pareto front.

#### 4.2.4.3 WO/FO rate

The WO rate is defined as the percentage of plays in which the AFP is welfare-optimal. Similarly, the FO rate is defined as the percentage of plays in which the AFP is fairness-optimal. Given a play $P_\Gamma$ of a repeated game $\Gamma$, we say that it results in a welfare-optimal solution if the welfare of its AFP is not more than 5% lower than the maximum welfare of $\Gamma$. Similarly, we say that it results in a fairness-optimal solution if the fairness of its AFP is not more than 5% lower than the maximum fairness of $\Gamma$. The maximum welfare and fairness of a game, respectively, are the highest possible welfare and fairness achievable by any strategy profile.

Given a game $\Gamma$, we compute its maximum welfare by solving the following non-linear optimisation problem:

$$\begin{aligned} \text{Maximise:} \quad & \textstyle\sum_{i \in N} U_i(\pi) \\ \text{Subject to:} \quad & \forall i \in N \, \forall j \in A_j : \pi_i(j) \geq 0 \\ & \forall i \in N : \textstyle\sum_{j \in A_i} \pi(j) = 1 \end{aligned}$$

Similarly, we compute its maximum fairness by solving the following non-linear optimisation problem:

$$\begin{aligned} \text{Maximise:} \quad & \textstyle\prod_{i \in N} U_i(\pi) \\ \text{Subject to:} \quad & \forall i \in N \, \forall j \in A_j : \pi_i(j) \geq 0 \\ & \forall i \in N : \textstyle\sum_{j \in A_i} \pi(j) = 1 \end{aligned}$$

We denote the optimised profile of the first problem by $\pi^w$ and the one of the second problem by $\pi^f$. Then, for a given play $P_\Gamma$, we say that it resulted in a welfare-optimal solution if $\frac{W(\pi^w)}{W(\bar{\pi})} \leq 1.05$, where $W(\pi) = \sum_{i \in N} U_i(\pi)$. Similarly, we say that it resulted in a fairness-optimal solution if $\frac{F(\pi^w)}{F(\bar{\pi})} \leq 1.05$, where $F(\pi) = \prod_{i \in N} U_i(\pi)$.

We close this section by stating the following two corollaries:

**Corollary:** Let $P_\Gamma$ be a play of a repeated game $\Gamma$. If $P_\Gamma$ results in a welfare-optimal solution, then $P_\Gamma$ must also result in a Pareto-optimal solution.

**Proof:** Assume $P_\Gamma$ results in a welfare-optimal solution but not in a Pareto-optimal solution. Let the AFP of $P_\Gamma$ be denoted by $\bar{\pi}$. Then, there must be a profile $\pi$ such that

$\forall i \in N : U_i(\pi) \geq U_i(\bar{\pi})$ and $\exists i \in N : U_i(\pi) > U_i(\bar{\pi})$. However, in this case, $P_\Gamma$ cannot result in a welfare-optimal solution since $\sum_{i \in N} U_i(\pi) > \sum_{i \in N} U_i(\bar{\pi})$. Therefore, $P_\Gamma$ must also result in a Pareto-optimal solution.

□

**Corollary:** Let $P_\Gamma$ be a play of a repeated game $\Gamma$. If $P_\Gamma$ results in a fairness-optimal solution, then $P_\Gamma$ must also result in a Pareto-optimal solution.

**Proof:** The proof for this corollary is perfectly analogous to the previous one.

□

Therefore, in our experiments, we first check if a play results in a welfare-optimal or fairness-optimal solution. If it does, then we conclude that it must also result in a Pareto-optimal solution. Note that the corollaries are strictly theoretical, meaning that there is no tolerance of 5%. Thus, in theory, the AFT of a play must exactly achieve the maximum welfare/fairness, respectively, to be welfare/fairness-optimal. We include the tolerance in our experiments to allow for slight deviations due to the learning process.

## 4.3 Parameter settings and selection strategies

We now specify the parameters settings and the selection strategies of the algorithms.

### 4.3.1 Learning rate $\alpha$

One parameter that is shared by most algorithms is the learning rate $\alpha$. This parameter is used in the update rule for the Q-table and determines the rate at which the algorithm adapts to the underlying payoff distribution. This is especially important if stochastic payoffs are used. Here, one may want to prevent the algorithm from learning statistical outliers, in which case a lower learning rate should be used [25]. However, our setting is simplified in that we use a deterministic (or constant) payoff distribution. Therefore, we could use a learning rate of $\alpha = 1$ such that the algorithm adapts immediately to the observed payoffs. Nonetheless, we chose to set the learning rate to $\alpha = 0.1$ (for all algorithms) for the following reasons: First, initial tests showed that the behaviours of the algorithms were more stable for lower learning rates. Second, a lower learning rate is

useful when visualising the learning process of the algorithms. However, as we will discuss later, the specific value of $\alpha$ is indeed irrelevant for our experiments.

We point out that we use a constant learning rate. However, algorithms based on Q-tables require a gradual decay in $\alpha$ in order to converge (both single-agent [57] and multi-agent [22]). We use a constant learning rate because we want the algorithms to be able to learn irrespective of the time that has passed by. In general, it may be the case that a certain situation is encountered rather late in the learning process. If a gradual decay is used on $\alpha$, then the algorithms may not anymore be able to learn the value of the newly encountered situation.[4] On the other hand, with a constant learning rate we allow for constant learning, but sacrifice the guarantee for convergence. A similar reasoning can be found in [54] and [27].

### 4.3.2 Learning rates $\delta_w$, $\delta_l$, $\delta$, and $\gamma$

Both WOLF-PHC (section 3.3) and RegMat (section 3.4) require the specification of two additional learning rates. In WOLF-PHC, we need to specify the parameter $\delta_w$, which corresponds to the learning rate in case the algorithm is winning, and the parameter $\delta_l$ (with $\delta_l > \delta_w$), which corresponds to the learning rate in case the algorithm is losing. The algorithm requires a gradual decay in these parameters in order to converge to an optimal solution [6]. A series of initial tests showed that the combination of $\delta_w = \frac{1}{1000+t}$ and $\delta_l = 2\,\delta_w$ worked well in most cases. Therefore, we will use this setting throughout our experiments. In RegMat, we need to specify the learning rate $\delta_t = \frac{\delta}{t^\gamma}$, where $0 < \delta < 1$ and $0 < \gamma < \frac{1}{2}$. Again, initial tests showed that a combination of $\delta = 0.1$ and $\gamma = 0.2$ worked well in most cases. Thus, we use this setting throughout our experiments.

### 4.3.3 Exploration rate $\varepsilon$

Recall that the selection strategy of an algorithm specifies the way in which the algorithm chooses its actions (see section 2.2). A common requirement on a selection strategy is that it should select all actions infinitely often as $t$ goes to infinity [52]. This is to ensure that the algorithm will completely explore its environment in the long run, which, in turn, is important if one wishes to find optimal action sequences. We chose to

[4] Yet another problem arises when considering stochastic games with multiple states. Here, it may be the case that the other agents change their behaviours in later stages of the game. If the algorithm uses a decaying learning rate, then it may not be able to account for these changes anymore.

use an $\varepsilon$-greedy selection strategy [52] for all algorithms except RegMat.[5] The parameter $\varepsilon$ (called the *exploration rate*) assumes a value between 0 and 1. In each state, the algorithm chooses a random action with probability $\varepsilon$, and the *greedy* action (i.e., the action that is currently believed to be best) with probability $(1-\varepsilon)$. One can see that this strategy, as $t \to \infty$, will cause the algorithm to choose each of its actions infinitely often in each state. We set the exploration rate to $\varepsilon = 0.05$ for all algorithms.

Note that we use a constant exploration rate. However, a common strategy is to start with a high exploration rate in the beginning and then to gradually decrease the rate such that the algorithm will eventually become completely greedy (that is, it will always choose its greedy actions) [52]. This ensures that the algorithm will first explore its environment, and then, as the time goes by, use its experience to make more informed decisions. A popular strategy that achieves this type of behaviour is the softmax selection strategy which we discussed in section 2.2. We chose to use a constant exploration rate because we are in fact not interested in the specific way the algorithm explores its environment. If we filter out the *exploration noise* (noise as in suboptimal decisions), then we obtain the algorithm's actual strategy, which is what we are interested in. We will now discuss this difference in more detail.

From a game-theoretic standpoint, there are two different views on the strategy of an algorithm. We call these views the *internal* strategy, denoted by $\pi^I$, and the *external* strategy, denoted by $\pi^E$. The internal strategy of an algorithm corresponds to the strategy that is being learned by the algorithm. For example, in WOLF-PHC, the internal strategy corresponds to the strategy $\pi$ (see section 3.3). On the other hand, the external strategy of an algorithm corresponds to the strategy that is being played by the algorithm. That is, for WOLF-PHC, the external strategy corresponds to $\pi^E(j) = (1-\varepsilon)\pi(j) + \frac{\varepsilon}{|A_i|}$, for all actions $j \in A_i$, where $\varepsilon$ denotes the exploration rate. In our experiments, we always use the internal strategy $\pi^I$ since we are interested in what the agent has actually learned, not in what the agent is playing from an external view point. Note that if external strategies are used to compute the metrics discussed in section 4.2, then these metrics, despite the tolerance values, may not reflect what the agent has actually learned. This is because the exploration noise produces a skewed view on the agent's internal findings.

[5] In fact, RegMat uses $\varepsilon$-greedy as well. Here, the parameters $\delta$ and $\gamma$ do the job (see section 3.4).

### 4.3.4 Summary of parameters

Our parameter settings are summarised in table 4.1.

| | $\alpha$ | $\varepsilon$ | $\delta_w$ | $\delta_l$ | $\delta$ | $\gamma$ |
|---|---|---|---|---|---|---|
| JAL | 0.1 | 0.05 | - | - | - | - |
| CJAL | 0.1 | 0.05 | - | - | - | - |
| WOLF-PHC | 0.1 | 0.05 | $\frac{1}{1000+t}$ | $2\delta_w$ | - | - |
| RegMat | 0.1 | - | - | - | 0.1 | 0.2 |
| NashQ | 0.1 | 0.05 | - | - | - | - |

Table 4.1: Summary of parameter settings.

### 4.3.5 Relevance of parameters

We end this chapter by noting that the specific parameter settings and selection strategies are indeed irrelevant for our experiments, provided that they satisfy certain conditions. This is because their main purpose is to regulate the rate at which the algorithms adapt and explore. However, in the long run, different parameter settings and selection strategies will not lead to fundamental improvements (such as an enhanced capability to learn Nash equilibria) because the algorithms will have explored their environment completely (or at least that part of the environment that is relevant for the algorithms). Therefore, their behaviour will be dominated by their experience, not by the parameter values. This is why the convergence and optimality proofs, if existent[6], do not regard the specific values of the parameters, but instead require them to satisfy certain conditions (such as a gradual decay over time [25]). In addition, these proofs are based on infinite plays ($t_f \to \infty$) which guarantees that the algorithms will have sufficient time to explore their environment, regardless of the specific parameter setting.

Therefore, we compared our parameter settings to a number of other settings, but did not seek to optimise them entirely. Instead, we intended to use identical or similar parameter settings and selection strategies for all algorithms in order to simplify the analysis of our results. Finally, we use 100,000 repetitions in each play to compensate for eventual sub-optimalities in our parameter settings and selection strategies.

[6] See, for example, [57, 22, 21, 6]

# Chapter 5

# Experimental results

This chapter presents and analyses the results of our experiments. As pointed out in section 4.1, our experiments are divided into three parts. The results of the first part are discussed in section 5.1, those of the second part in section 5.2, and those of the third part in section 5.3. Appendix C provides a complete overview of all simulation results.

It is important to note that, in general, the performance of an algorithm may depend on the player position it takes on. For example, if the players are in a Nash equilibrium, then the expected payoffs of the players may not necessarily be the same. To account for this, we repeated each simulation once for every permutation of the agent order. We call this process a *sweep*. Furthermore, in the following sections, whenever we refer to statistical significance, we used a paired t-test with a significance level of 5%. We use the notation "Alg1 / Alg2" if the performances of the algorithms Alg1 and Alg2 are statistically equivalent (i.e., the difference is statistically insignificant).

## 5.1 No-conflict games

The first part of our experiments evaluated the algorithms in the set of all structurally distinct strictly ordinal $2 \times 2$ no-conflict games. Section 4.1 of the previous chapter and appendix B provide definitions for each of these terms. Specifically, we refer to section B.1 for a full listing of the games. We evaluated all combinations of the algorithms discussed in chapter 3 in each of the games. In total, we evaluated each pair of algorithms in 25 sweeps (50 simulations), where each simulation consisted of 100,000 repetitions of the played game. The results were then averaged over all simulations.

We note that the class of strictly ordinal no-conflict games is one of the simplest game classes. In a strictly ordinal no-conflict game, there is one outcome that is most preferred by all players. This outcome provides the highest possible payoffs to all players, which makes it the only Pareto-optimal solution of the game. In addition, it is also the only welfare-optimal and fairness-optimal solution of the game (recall from section 4.2.4.3 that there cannot be more welfare-optimal or fairness-optimal solutions as otherwise these would be Pareto-optimal solutions as well). Therefore, the sets of Pareto-optimal, welfare-optimal, and fairness-optimal solutions are identical in the class of strictly ordinal no-conflict games, and they consist only of the one outcome that is most preferred by all players. Based on these observations, we expect the algorithms to perform well in any constellation of agents.

| Agent | Conv. | Fexp. | Welfare | Fairness | NE | PO | WO | FO |
|---|---|---|---|---|---|---|---|---|
| JAL | 1 | 3.9866 | 7.9720 | 15.9063 | 1 | 0.9920 | 0.9920 | 0.9920 |
| CJAL | 1 | 3.9831 | 7.9663 | 15.8874 | 1 | 0.9897 | 0.9897 | 0.9897 |
| WOLF | 0.9996 | 3.9449 | 7.8908 | 15.6426 | 1 | 0.9638 | 0.9638 | 0.9638 |
| RegMat | 0.9990 | 3.9107 | 7.8170 | 15.3906 | 0.9954 | 0.9457 | 0.9457 | 0.9457 |
| NashQ | 0.9987 | 3.9840 | 7.9733 | 15.9144 | 0.9954 | 0.9939 | 0.9939 | 0.9939 |

Table 5.1: Simulation results for no-conflict games.

The performance metrics for every algorithm (averaged over all teams) are shown in table 5.1[1]. For completeness, table C.1 in appendix C shows the results for each team. The maximum payoff any player can achieve in any game is 4, and the maximum social welfare and fairness, respectively, are 8 and 16. As expected, all algorithms performed quite well. However, there are several observations to be made. First of all, we note that NashQ, both in self-play and in mixed-play, performed extremely well. A closer look at the strategy trajectories reveals that NashQ persuades the other agent to play a NE strategy by playing a NE strategy itself, regardless of whether it could achieve a higher payoff by using another strategy. However, note that NashQ requires more information than any other algorithm (this was discussed in section 3.5).

Figure 5.1 shows an example of the strategy trajectories of WOLF-PHC and NashQ in game 20. Both algorithms play non-optimal strategies for the first 12,000 repetitions until NashQ has gathered enough information to change its strategy to the optimal NE

[1] We abbreviate WOLF-PHC by WOLF to save space.

strategy (always playing action 1). Shortly after that, WOLF-PHC begins to gradually adapt its strategy until the optimal NE is completed (both always playing action 1).

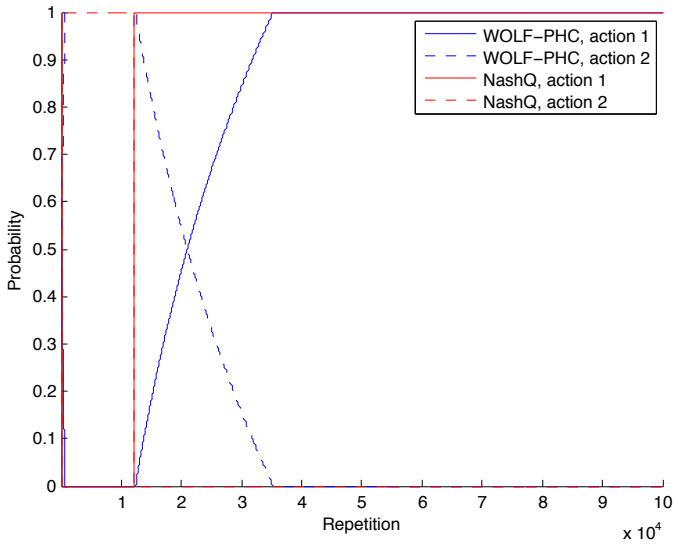



Figure 5.1: Strategy trajectories of WOLF-PHC and NashQ.

The next observation is that those algorithms that model their opponents (i.e. JAL and CJAL) perform generally better than those that do not (i.e. WOLF-PHC and RegMat, leaving NashQ aside[2]). Both JAL and CJAL have better results than WOLF-PHC and RegMat throughout all constellations of agents. Note also that JAL and CJAL have almost identical performances (all significance tests were negative). Most strategy trajectories of JAL and CJAL were such that they converged to the optimal outcome within a few hundred repetitions. The results indicate that opponent modelling techniques may be well suited for ad hoc team scenarios because they learn the strategies of the other agents irrespective of the algorithms on which they are based on. This is in line with e.g. Stone et al. [4] and Wu et al. [61], whose algorithms are also based on opponent modelling techniques.

Policy hill climbing (WOLF-PHC) and no-regret approaches (RegMat) seem to work less well in some situations because they rely more on the other agents being based on the same approach. We can see that the results of both algorithms are generally worse than those of the other algorithms. This applies especially to teams that consist of these algorithms only (all significance tests were positive). A closer look at

[2]In our opinion, NashQ is incompatible with the ad hoc team domain. We discussed this in section 3.5.

the results for each game showed that WOLF-PHC and RegMat had problems to converge to the optimal outcome in games 16 to 21. While the other games have one pure NE only (which is also the optimal outcome), these games have two pure NE and one mixed NE. That is, while in the other games converging to the only NE implies converging to the optimal outcome, this is not the case for games 16 to 21.[3] Here, the agents may converge to one of three possible NE, but only one of these corresponds to the optimal outcome. If the algorithms arrive at a suboptimal NE, there is no guarantee that they will ever converge to the optimal NE.

Figure 5.2 shows an example of the strategy trajectories of WOLF-PHC and RegMat in game 19. While WOLF-PHC quickly converged to the suboptimal NE strategy $(0,1)$, RegMat tried to converge to the optimal NE strategy $(1,0)$, but then failed to do so because WOLF-PHC did not change its strategy anymore, which forced RegMat to play its best response $(0,1)$ (i.e., they converged to the suboptimal NE in which both constantly play action 2). This may be an indicator that those algorithms that do not model their opponents (or at least WOLF-PHC and RegMat, again leaving NashQ aside) are less effective when they have to coordinate their strategies in order to arrive at a globally optimal solution.

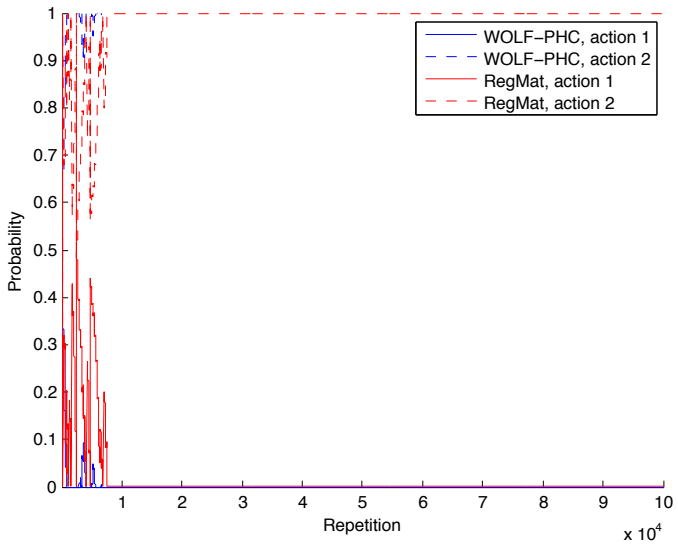



Figure 5.2: Strategy trajectories of WOLF-PHC and RegMat.

[3] Indeed, we found that in games 19 to 21, WOLF-PHC and RegMat were almost unable to converge to the optimal outcome (in self-play and in mixed-play). This might come from the fact that all of these games have a uniform mixed NE. However, as we observed this for self-play as well as mixed-play, we concluded that it was not specific to ad hoc team problems, and so we did not investigate this further.

Finally, we note that JAL and CJAL performed robustly in all teams (see table C.1). That is, their performances were similar in self-play and in mixed-play. In addition, both algorithms significantly improved the performances of WOLF-PHC and RegMat (relative to their self-play performances) when playing with them in a team. This effect was even stronger for NashQ, which produced perfect plays in most teams. We explain this by the fact that NashQ quickly learns to play the optimal NE strategy, regardless of the strategies of the other agents. These, in turn, quickly adapt the same strategy, which is the best they can do. However, when comparing NashQ to the other algorithms, it is important to take into account that it requires much more information in each state. The result from this part of the experiments is that those algorithms that model their opponents seem to perform better and more robustly in that their performance is less affected by the constellation of agents they are playing with.

## 5.2 Conflict games

The second part of our experiments tested the algorithms in the set of all structurally distinct strictly ordinal $2 \times 2$ conflict games. Section 4.1 of chapter 4 and chapter B provide definitions for each of these terms. Section B.2 of appendix B contains a full listing of the games. As before, we evaluated each combination of the algorithms in each of the games, using 25 sweeps and 100,000 repetitions per simulation. The results were averaged over all simulations and are shown in table 5.2. For completeness, table C.3 of appendix C shows the results for each team.

| Agent | Conv. | Fexp. | Welfare | Fairness | NE | PO | WO | FO |
|---|---|---|---|---|---|---|---|---|
| JAL | 0.8901 | 3.0140 | 6.0592 | 8.9997 | 0.8982 | 0.7781 | 0.7021 | 0.6164 |
| CJAL | 0.9456 | 3.0326 | 6.0978 | 9.0900 | 0.8470 | 0.8050 | 0.7184 | 0.6250 |
| WOLF | 0.9430 | 3.0392 | 6.0620 | 9.0517 | 0.9047 | 0.7636 | 0.6992 | 0.6142 |
| RegMat | 0.8673 | 3.0313 | 6.0368 | 8.9610 | 0.8946 | 0.7662 | 0.7000 | 0.6109 |
| NashQ | 0.9990 | 3.0446 | 6.0667 | 9.0755 | 0.8722 | 0.7767 | 0.6946 | 0.6097 |

Table 5.2: Simulation results for conflict games.

The maximum payoff any player can achieve in any of these games is 4. However, the maximum welfare and fairness may vary among the games and can be as high as 7 and 12, respectively. First, we note that NashQ achieved the highest convergence rate, followed by CJAL / WOLF-PHC, then JAL, and then RegMat. The high convergence

rate of NashQ is explained by the fact that it plays a NE strategy in each state, regardless of whether another strategy would provide higher payoffs. Therefore, as soon as NashQ has learned the payoff structure of the game, it will always play the same strategy. The low convergence rate of RegMat can be explained by the fact that it constantly tries to maintain (or restore) the Hannan consistency (see section 3.4), forcing it to frequently change its strategy.

Interestingly, the final expected payoffs and the average welfare and fairness of all algorithms are very similar. Indeed, all of these are statistically equivalent. This is an interesting result since it does not correspond to the solution rates of the algorithms. Here, one can see that WOLF-PHC / JAL / RegMat have the highest NE rates, followed by NashQ and CJAL. On the other hand, CJAL has the highest PO rate, followed by all other algorithms with statistically equivalent PO rates. The same applies to the WO and FO rates, where CJAL again achieved the highest rate, while the other algorithms achieved equivalent rates. Furthermore, note that NashQ has the second lowest NE rate although it plays a NE strategy in every state. Other than in the previous section, playing a NE strategy in every state does not necessarily seem to persuade the other agent to play a NE strategy as well.

The results indicate that CJAL may be better suited for ad hoc team problems than the other algorithms. It achieved the highest PO, WO, and FO rates, and these are significantly higher than those of the other algorithms. Moreover, it has the highest average welfare and fairness (yet these are statistically equivalent to the other algorithms). Note also that, apart from NashQ, it achieved the highest convergence rate. A high convergence rate is useful since it allows the other agents to adapt to a stationary opponent (that is, once it has converged). This is especially useful in ad hoc team scenarios because the agents may not be able to resort to coordination strategies. However, note that CJAL has the lowest NE rate of all algorithms. Thus, whether CJAL is better suited for ad hoc team problems will ultimately depend on the solution concept that is considered most appropriate for this domain.

A closer look at the results revealed that the algorithms had problems in games 49 to 57. These games are different from the other games in that they have no pure Nash equilibria. That is, in order to play a Nash equilibrium in these games, the agents have to coordinate towards a mixed strategy profile such that, on average, every agent is play-

ing a best response to the other agent. We will refer to these games as mixed NE games. Table 5.3 shows the performance metrics for each algorithm in games 49 to 57.

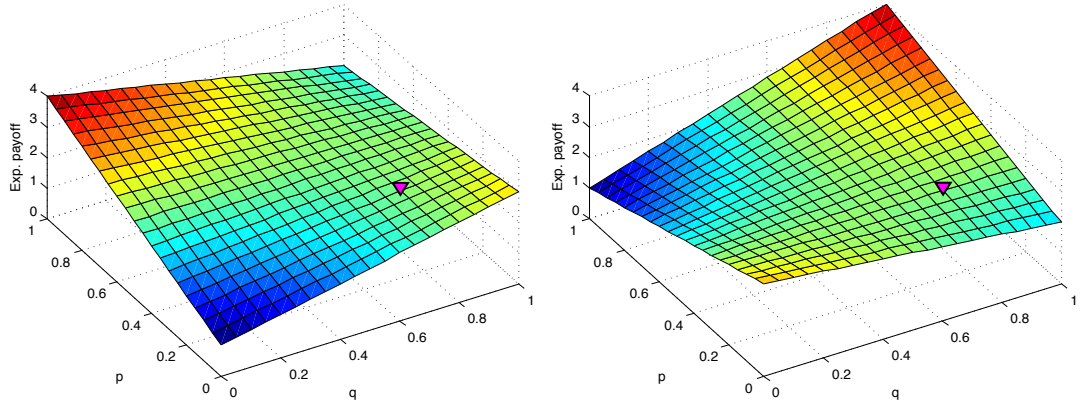
| Agent | Conv. | Fexp. | Welfare | Fairness | NE | PO | WO | FO |
|---|---|---|---|---|---|---|---|---|
| JAL | 0.3037 | 2.5224 | 5.1135 | 6.2755 | 0.3553 | 0.2610 | 0.1480 | 0.1387 |
| CJAL | 0.6563 | 2.6441 | 5.3594 | 6.8513 | 0.0308 | 0.4314 | 0.2514 | 0.1932 |
| WOLF | 0.6391 | 2.5434 | 5.1150 | 6.5297 | 0.3967 | 0.1742 | 0.1338 | 0.1249 |
| RegMat | 0.1656 | 2.5331 | 5.0710 | 6.2559 | 0.3539 | 0.2067 | 0.1527 | 0.1400 |
| NashQ | 1 | 2.6703 | 5.1703 | 6.6757 | 0.2119 | 0.2729 | 0.1361 | 0.1283 |

Table 5.3: Simulation results for mixed NE games.

The results essentially correspond to those in table 5.2 in that they lead to the same rankings of the algorithms. However, as one can see, the gaps between the algorithms are now more significant than before. First, we note that the convergence rates of JAL and RegMat, particularly the latter, are very low. As pointed out earlier, this may be a disadvantage in ad hoc teams since it is easier for the agents to optimise their strategies if more agents become stationary. Next, we note that the NE rate of CJAL is extremely low (in fact, it is almost zero). This corresponds to the previous results, but the difference to the other algorithms is now much more significant. On the other hand, the PO, WO, and FO rates are now more significant as well. Indeed, in almost half of all plays, CJAL managed to coordinate the team towards a Pareto-optimal solution, half of which were even welfare-optimal. This could be a valuable property in ad hoc team problems because efficient collaboration may not necessarily mean that the agents are in a Nash equilibrium, but to arrive at a globally optimal (e.g. welfare-optimal) solution. However, as pointed out earlier, these considerations depend on the solution concept that is deemed most appropriate for the problem at hand.

We demonstrate some of our observations in the following example. Consider figure 5.3. The figure shows the expected payoffs of both players in the conflict game 52. The variable $p$ corresponds to the probability of player 1 to choose action 1, while the variable $q$ corresponds to the probability of player 2 to choose action 1. These variables are sufficient to specify a strategy profile, as there are only two actions to choose from.

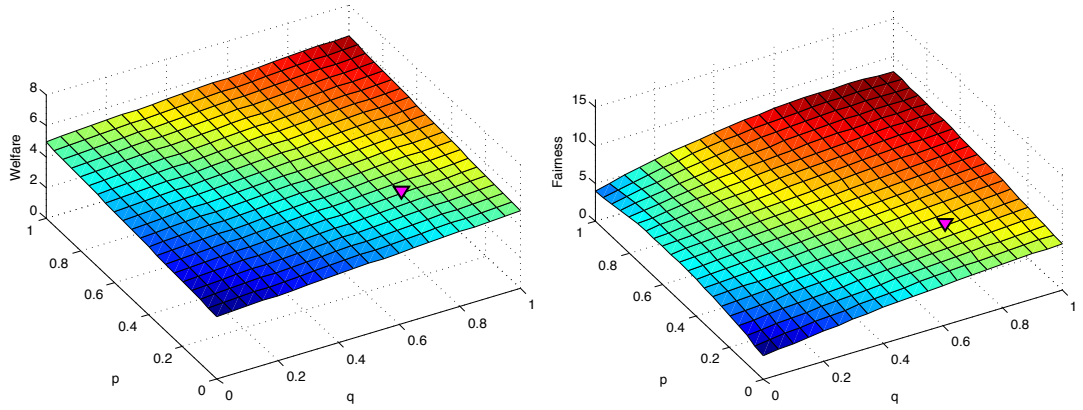
Both players have two local maxima for their expected payoffs. For player 1, the maxima are at $p = 1/q = 0$ and $q = 1/p = 0$, while for player 2 these are at $p = 1/q = 1$




(a) Expected payoffs of player 1 (b) Expected payoffs of player 2

Figure 5.3: Expected payoffs in conflict game 52.

and $p = 0/q = 0$. Since the maxima are all at different points, the agents will have to find some compromise (i.e. a mixed profile). One such compromise is a Nash equilibrium. The magenta triangle marks the only Nash equilibrium of the game, which is given by $p = 0.25/q = 0.75$. The equilibrium yields an expected payoff of 2.5 for both players. We note that this may not the best the agents can do in this case. Although both agents have the same expected payoffs (making the equilibrium perfectly fair), they could change their strategies to achieve both a higher welfare and a higher fairness.



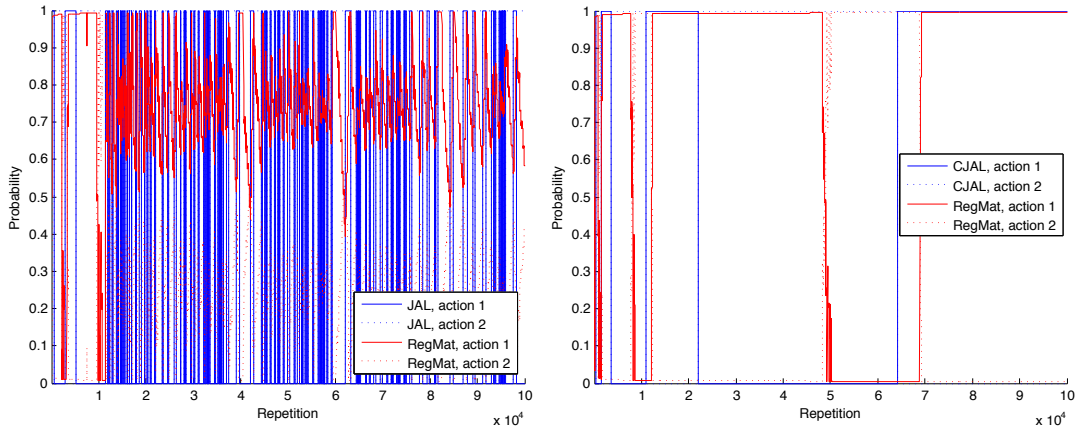

(a) Social welfare (b) Social fairness

Figure 5.4: Social welfare and fairness in conflict game 52.

Figure 5.4 shows the social welfare and fairness of the game. While the welfare has one maximum at $p = 1/q = 1$, the fairness has multiple maxima located around $p = 1/q = 1$. Thus, the profile $p = 1/q = 1$ is both welfare-optimal and fairness-optimal,

yielding a welfare of 6 and a fairness of 8, and the expected payoffs 2 and 4 for player 1 and 2, respectively. If welfare and fairness are of greater importance than the individual expected payoffs, then the agents would be better off by playing the profile $p = 1/q = 1$. However, if the individual payoffs have greater weight, then the agents are best off with the Nash equilibrium of the game. This underlines our earlier statement that such priorities must be set out in advance when assessing the performance of an ad hoc team.

Figure 5.5a shows an example of the strategy trajectories of JAL and RegMat in conflict game 52. Neither algorithm converged during the simulation. However, the final averaged strategies of both agents (averaged over the endmost 20% of the simulation; see section 4.2.4) correspond to the Nash equilibrium of the game. The averaged strategies are $p = 0.24961$ for JAL and $q = 0.7519$ for RegMat. Furthermore, the final expected payoffs are 2.4960 for JAL and 2.4961 for RegMat, and the average welfare and fairness are 4.9921 and 6.1812, respectively.




(a) JAL and RegMat

(b) CJAL and RegMat

Figure 5.5: Strategy trajectories in conflict game 52.

Now, consider figure 5.5b. The figure shows an example of the strategy trajectories of CJAL and RegMat in the same conflict game. This time, both algorithms converged. The averaged strategies are $p = 1$ for CJAL and $q = 0.9949$ for RegMat. Although this does not correspond to the Nash equilibrium of the game, it is in fact Pareto-optimal, welfare-optimal, and fairness-optimal, yielding an averaged welfare of 5.9949 and an averaged fairness of 8.0101. The final expected payoffs are 2.0102 for JAL and 3.9847 for RegMat. Thus, we have demonstrated the behaviour of two ad hoc teams, both of which are optimal in some sense.

## 5.3 Random games

In the third part of our experiments we investigated how well the algorithms scale to ad hoc teams with more than two agents. We used a modified version of the evaluation procedure proposed by Stone et al. [48]. Specifically, we tested each of the algorithms in 500 randomly generated strictly ordinal $2 \times 2 \times 2$ games. The teams were randomly drawn from the set of all algorithms, and a team may contain more than one agent of the same algorithm. The following sections discuss the evaluation procedure and the simulation results.

### 5.3.1 Evaluation procedure

The experiments are based on the evaluation procedure by Stone et al. [48]. Consider a domain $D$ from which tasks $d$ can be sampled. Furthermore, consider a set $A$ of potential team members. The procedure compares the performance of two ad hoc agents, denoted by $a_0$ and $a_1$, within domain $D$ in random teams drawn from $A$. It assumes that there is a function $s(B,d)$ that measures the performance (or *score*) of a team $B \subset A$ for a task $d \in D$. Note that $s$ may be a stochastic function, meaning that one has to consider expected scores $\mathrm{E}[s(B,d)]$. Algorithm 6 shows the pseudo-code of the procedure.

**Algorithm 6** Evaluation procedure (Stone et al. [48])

**Initialise:** score counters $r_0 = 0$ and $r_1 = 0$ for agents $a_0$ and $a_1$, respectively
**loop**
  Sample task $d$ from $D$
  Randomly draw a subset of agents $B$, $|B| \geq 2$, from $A$ such that $\mathrm{E}[s(B,d)] \geq s_{min}$
  Randomly select one agent $b \in B$ to remove from the team to create the team $B^-$
  Increment $r_0$ by $s(\{a_0\} \cup B^-, d)$
  Increment $r_1$ by $s(\{a_1\} \cup B^-, d)$
**end loop**
If $r_0 > r_1$ then we conclude that $a_0$ is a better ad hoc team player than $a_1$ in domain $D$ over the set of potential team members $A$

The set of potential team members consists of agents that have some *competency* in domain $D$. Formally, an agent $a \in A$ has competency in $D$ if there is a subset $B \subset A$ such that $a \in B$ and $\mathrm{E}[s(B,d)] \geq s_{min}$ for all tasks $d \in D$, where $s_{min}$ denotes the minimum expected score a team must have (we assume that a higher score is a better score). This

is useful if one needs to ensure that a team is in fact able to perform the tasks in the considered domain. However, note that there may be an individual agent capable of achieving the tasks on its own. Therefore, the procedure randomly removes a member from $B$ so that a team may not always rely on such individual agents.

We instantiate the procedure as follows: the domain $D$ is the set of all strictly ordinal $2 \times 2 \times 2$ games. The set $A$ of potential team members comprises all algorithms discussed in chapter 3. Each ad hoc team $B$ consists of the agent to be evaluated and two more agents which are randomly drawn from $A$. The score $s(B,d)$, where $d$ denotes a random game, comprises all performance criteria discussed in section 4.2. We do not use a minimum score $s_{min}$, hence there is no need to remove a random member from $B$. Algorithm 7 shows the pseudo-code of our modified evaluation procedure.

**Algorithm 7** Modified evaluation procedure

**Initialise:** Empty vector $M_a$ for each agent $a \in A$
**loop**
  Randomly generate a strictly ordinal $2 \times 2 \times 2$ game $G$
  Randomly generate a team $B$ from $A$ with $|B| = 2$
  **for all** $a \in A$ **do**
    Simulate $G$ with agents $\{a\} \cup B$, where agent $a$ is player 1
    Compute performance metrics for agent $a$ and store them in $M_a$
  **end for**
**end loop**
Return averaged metrics $avg(M_a)$ for each $a \in A$

Each game is simulated once for each agent $a \in A$. Note that the agent $a$ always takes the position of the first player. This ensures that all agents are evaluated under the same conditions.

### 5.3.2 Results

Table 5.4 shows the performance metrics for each algorithm (averaged over all games). The maximum payoff any player can achieve in any game is 8, while the maximum welfare and fairness may vary among the games. If the game happens to be a no-conflict game, then these values amount to 24 and 512, respectively. However, if the game is

a conflict game, then the maximum welfare and fairness may assume any value as high as 23 and 448, respectively. Of all games generated in our experiments, 2% were no-conflict games and 98% were conflict games.

| Agent | Conv. | Fexp. | Welfare | Fairness | NE | PO | WO | FO |
|---|---|---|---|---|---|---|---|---|
| JAL | 0.854 | 5.7964 | 17.2174 | 193.1478 | 0.804 | 0.712 | 0.466 | 0.396 |
| CJAL | 0.922 | 5.7856 | 17.3521 | 196.1594 | 0.760 | 0.742 | 0.486 | 0.418 |
| WOLF | 0.918 | 5.7400 | 17.1956 | 193.0255 | 0.824 | 0.676 | 0.442 | 0.388 |
| RegMat | 0.852 | 5.7290 | 17.2315 | 193.9011 | 0.844 | 0.708 | 0.438 | 0.392 |
| NashQ | 0.980 | 5.7562 | 17.2452 | 193.6470 | 0.790 | 0.734 | 0.452 | 0.388 |

Table 5.4: Simulation results for random games.

First, we note that NashQ has the highest convergence rate. This is because once the algorithm has learned the game structure, it will always play the same NE strategy. The second and third highest convergence rates were achieved by CJAL / WOLF-PHC and JAL / RegMat, respectively. It is interesting to see that CJAL / WOLF-PHC and JAL / RegMat have similar convergence rates, since, in both groups respectively, the first algorithm plays pure strategies while the second algorithm plays mixed strategies. One would expect the former to have a significantly lower convergence rate than the latter because those algorithms that play pure strategies need to change their strategies periodically in order to approximate mixed strategies (e.g. for a mixed Nash equilibrium).

We demonstrate the convergence behaviours of these algorithms in two examples. First, consider figure 5.6. The figure shows an example of the strategy trajectories of JAL, WOLF-PHC and RegMat in a random $2 \times 2 \times 2$ game whose only Nash equilibrium is the mixed profile $\pi_1 = (0,1)$, $\pi_2 = (0.5714, 0.4286)$, $\pi_3 = (0.5, 0.5)$. We can see that JAL immediately adapts the strategy $\pi_1$. This is because action 1 happens to be a strictly dominating action[4] for player 1 and as such it is the best choice. WOLF-PHC gradually adapts the strategy $\pi_2$ within the first 15,000 repetitions of the game. RegMat seems to be adapting strategy $\pi_3$, but it does not converge in the long-term. In fact, it is changing its strategy quite frequently in order to maintain (or restore) its Hannan consistency (see section 3.4).

Next, consider figure 5.7. The figure shows the strategy trajectories of WOLF-PHC,

[4] An action is strictly dominating if it always yields a higher payoff than any other action.

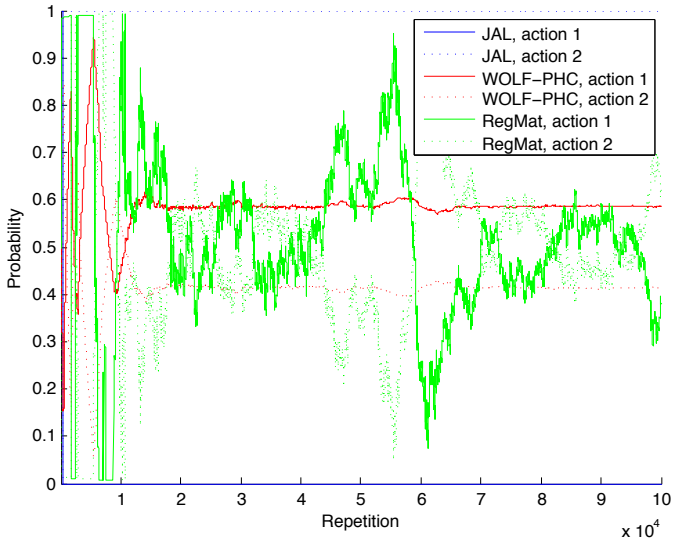



Figure 5.6: Strategy trajectories of JAL, WOLF-PHC, and RegMat.

CJAL, and RegMat in a random $2 \times 2 \times 2$ game. The only Nash equilibrium of this game is the mixed profile $\pi_1 = (1,0)$, $\pi_2 = (0.6667, 0.3333)$, $\pi_3 = (0.875, 0.125)$. This example is somewhat more complicated than the previous one because the CJAL agent has to learn a mixed strategy in order to complete the Nash equilibrium. However, rather than to approximate the strategy $\pi_2$ by changing its strategy periodically, it tries both actions for more than 15,000 repetitions until it finally decides to constantly play action 1. This is consistent with $\pi_2$ insofar as that action 1 has the highest probability. After JAL converged, WOLF-PHC gradually adapted the strategy $\pi_1$. Finally, RegMat changed its strategy to constantly play action 1. Again, this is consistent with the strategy $\pi_3$ because action 1 has the highest probability. Although the algorithms did not converge to a Nash equilibrium, they ended up playing a solution that is at least similar to the Nash equilibrium. However, a good ad hoc agent should be able to learn any strategy in any team. Thus, the behaviour of CJAL may be considered sub-optimal in this example.

The final expected payoffs (FEPs) of all algorithms are statistically equivalent. Note that NashQ achieved the third highest FEP. This is interesting because NashQ plays a NE strategy regardless of whether or not another strategy provides higher payoffs. Moreover, it is remarkable that JAL, CJAL, WOLF-PHC, and RegMat have equivalent FEPs despite the fact that the former two play pure strategies while the latter two play mixed strategies. This could indicate that the impact on the average payoffs due to the constellation of agents in a team may not be as strong as one might otherwise expect.

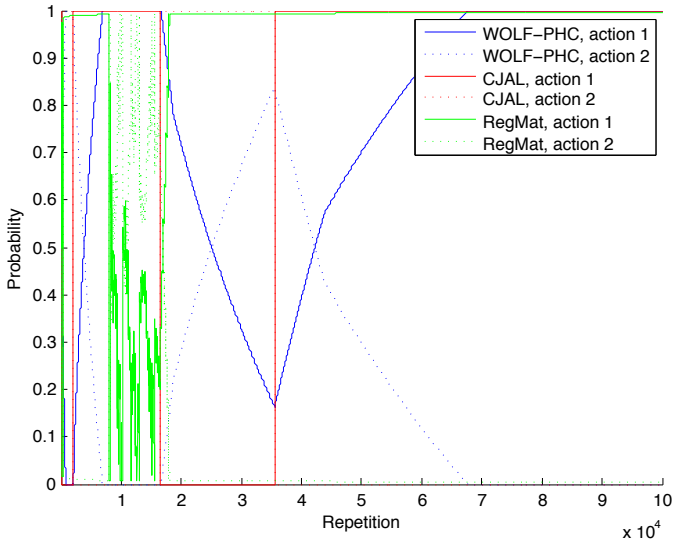



Figure 5.7: Strategy trajectories of WOLF-PHC, CJAL, and RegMat.

The average welfare and fairness confirm our observations of section 5.2. Although the welfares of all algorithms are statistically equivalent, CJAL appears to have achieved a better welfare then any other algorithm. In addition, CJAL has a significantly higher fairness than the other algorithms, whose fairness is statistically equivalent. This corresponds to the WO and FO rates of the algorithms, where CJAL achieved a significantly higher rate than the other algorithms. In fact, in almost half of all simulation, CJAL managed to arrive at a welfare-optimal solution, of which a majority was fairness-optimal as well. As noted earlier, this may be a valuable property for ad hoc team problems. An ad hoc agent that is able to collaborate with an unknown group of agents such that the overall performance of the entire group is optimised (in terms of welfare and fairness) may be better than an agent that attempts to optimise its own payoff only. However, we note again that this essentially depends on the priorities of both the ad hoc agent and the entire group. We will discuss this further in chapter 6.

Finally, consider the different solution rates. The highest NE rate was achieved by RegMat, followed by WOLF-PHC, JAL / NashQ, and then CJAL. Note that the NE rate of CJAL is relatively low when compared to the other ones. However, this is opposed by the PO rates. Here, CJAL / NashQ achieved the highest rate, followed by JAL / RegMat and WOLF-PHC. It is interesting that CJAL and NashQ achieved equivalent PO rates, despite the fact that CJAL was specifically designed to learn PO solutions [6], whereas NashQ was specifically designed to learn NE solutions [24]. Note also that the PO rate of WOLF-PHC is quite low, especially in comparison to its relatively high NE rate.

## 5.4 Overall results

We end this chapter with a discussion on the overall results of the experiments. Figure 5.8 shows the results averaged over all three parts. Note that we cannot take the average of the payoffs since we consider ordinal games. However, for the final expected payoffs, we first normalised the values by dividing through the respective maximum, after which we took the average of all results. Thus, the final expected payoffs in figure 5.8 are to be read as percentages where 0% means that the algorithm always achieved its least preferred outcome, and 100% means that it always achieved its most preferred outcome.

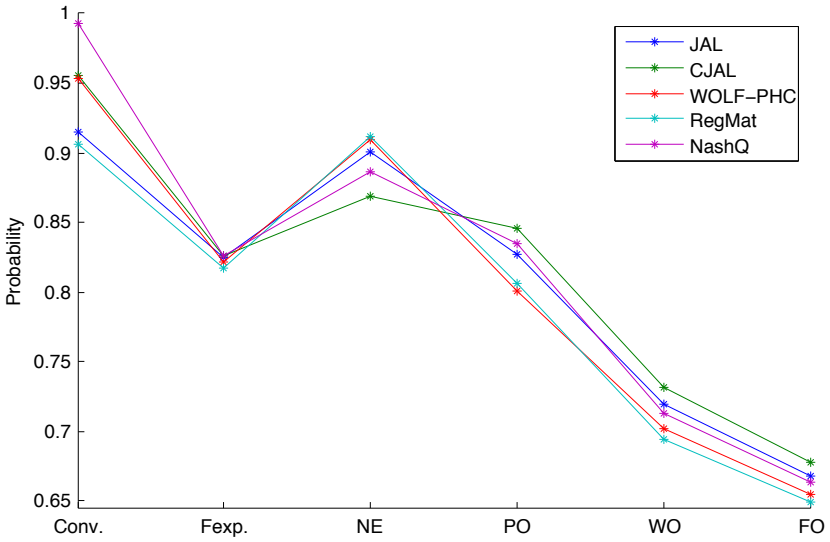



Figure 5.8: Overall results.

Throughout all three parts of our experiments, the NE rates were generally higher than the PO rates, the PO rates were generally higher than the WO rates, and the WO rates were generally higher than the FO rates. This appears to indicate an order on the difficulty of these solution concepts. First, it seems that Nash equilibria are easier to reach than Pareto-optimal solutions. We explain this by the fact that the agents can locally decide on whether the current solution, from their individual viewpoints, may be a Nash equilibrium or not. If they can deviate from their strategies in order to increase their payoffs, then the solution cannot be a Nash equilibrium. However, deciding on whether the current solution is Pareto-optimal is more difficult as this requires some form of coordination. To do so, the agents will have to decide if the current solution is Pareto-dominated by another solution, and this can only be decided if the group acts together. Finding a welfare-optimal solution goes one step further in that the agents

have to find a Pareto-optimal solution that maximises the welfare of the group. Finally, fairness-optimal solutions can be even harder to find since this corresponds to finding a Pareto-optimal solution with the best possible tradeoff between maximised welfare and identical individual payoffs (as only then the fairness will be maximised).

The results do not allow for a clear statement about which algorithm performed best in the experiments. The following summary shows that there is no clear favourite among the algorithms:

- JAL has the second lowest convergence rate with about 91.47%. This comes from the fact that it changes its strategy periodically in order to approximate a mixed strategy. Furthermore, it has the third highest payoff rate with 82.49%. This rate is statistically equivalent to the highest rates. The NE rate of JAL is the third highest with about 90%. Finally, it has the third highest PO rate (82.74%), the second highest WO rate (72%), and the second highest FO rate (66.81%).

- CJAL has the second highest convergence rate with about 95.59%. This is since it attempts to learn Pareto-optimal solutions, which, in our case, are often pure profiles. Moreover, of all algorithms used in our experiments, it has the highest PO rate (84.56%), WO rate (73.14%), and FO rate (67.76%), which consequently leads to the highest payoff rate with about 82.57%. However, this is opposed by the fact that it achieved the lowest NE rate with 86.9%.

- WOLF-PHC has the third highest convergence rate with about 95.35%. This is statistically equivalent to CJAL. Furthermore, it has the second lowest payoff rate with 82.12%. On the other hand, with about 90.96% it has the second highest NE rate. This rate is statistically equivalent to the highest NE rate. Finally, it has the second lowest PO rate (80.12%), WO rate (70.17%), and FO rate (65.53%).

- RegMat achieved the lowest convergence rate with 90.61%. As pointed out earlier, it frequently changes its strategy in order to maintain the Hannan consistency. Moreover, with about 81.72%, it also has the lowest payoff rate. This is a consequence of its frequent changes. Interestingly, these efforts lead to the highest NE rate of all algorithms with about 91.14%. However, they also lead to the lowest PO rate (80.66%), WO rate (69.46%), and FO rate (64.95%).

- NashQ, with about 99.26%, managed to converge in most of the games. As was explained earlier, this is since it will always play the same strategy after it learned

the payoff structure of the game. It is worth noting that it has the second highest payoff rate with about 82.56%, which is almost identical to the highest rate. This is interesting since NashQ chooses its strategies irrespective of the strategies of the other agents. On the other hand, it is surprising that it has the second lowest NE rate (88.59%), despite the fact that it plays a NE strategy in each state. It is equally surprising that it achieved the second highest PO rate (83.49%) and the third highest WO rate (71.35%) and FO rate (66.39%), despite the fact that it was not optimised for these solution types.

We conclude that the assessment of an ad hoc agent ultimately depends on the solution concept that is deemed most appropriate for the ad hoc team problem at hand. In other words, its performance depends on the priorities of the entire team. For example, in the predator domain investigated by Stone et al. [4], it would be most desirable to arrive at a welfare-optimal solution if we define the welfare of the predator group to be the inverse of the average steps needed to capture the prey. Moreover, if we think of the agents as real robots (e.g. as in [50]), then we might want to arrive at a fairness-optimal solution in order to ensure that all robots have identical or similar energy consumptions. On the other hand, in a multiagent marketing application (see e.g. [38]) in which the other agents cannot be trusted (as they may want to deceive us in order to increase their payoffs), we would want to arrive at a Nash equilibrium (or minimax profile in 2-player zero-sum games, see e.g. [16]) such that we can guarantee a minimum average payoff.

Our conclusion may seem somewhat trivial in that it applies to a vast range of problems. Nonetheless, we argue that this issue has been largely neglected (or ignored) by the MAL community. In most cases, the authors almost exclusively focused on the concept of Nash equilibria (or, equivalently, minimax profiles in 2-player zero-sum games). For a selection, see [12, 6, 22, 23, 24, 54, 9, 29, 30, 32, 31, 13]. Some authors focused on the concept of Pareto optimality as an alternative [43, 27, 3], others focused on the concept of correlated equilibria [20, 21, 18], and some did not address this issue at all [50, 49, 4, 61]. However, as can be seen from our results, it is important to consider a wider spectrum of solution concepts in order to fully assess the performance of an algorithm. Indeed, this bears an interesting resemblance to the No Free Lunch theorems of Wolpert and Macready [58, 59]. Therein, roughly speaking, it is argued that the performance of any two algorithms is identical when averaged over all possible problems. That is, whenever an algorithm is superior to another algorithm on a certain set of problems,

this is paid for by inferiority on a different set of problems. Our results show a trade-off relation of this kind. For instance, CJAL has the highest PO rate and the lowest NE rate whereas RegMat has the lowest PO rate and the highest NE rate. Other algorithms range somewhere in the middle, without best or worst performances. This seems to indicate that superiority in one solution type is evened by inferiority in another solution type.

A natural question in consequence of our conclusion is which solution concepts one should consider when assessing the performance of an algorithm. There is, in fact, a variety of solution concepts, and it is not a priori clear which of the concepts are most appropriate. In our work, we considered the concepts of Nash equilibria and Pareto optimality. We further introduced the concepts of welfare and fairness optimality. However, another interesting solution concept is given by the correlated equilibrium, which we will briefly describe in chapter 7. More solution concepts are used throughout the literature. We argue that the choice of appropriate solution concepts depends on various factors. In the domain of multiagent learning, a primary factor is given by the priorities of the entire group (e.g. maximum welfare or fairness) as well as those of the individual agents (e.g. maximum individual payoff). The choice of appropriate solution concepts is further complicated in ad hoc teams, in which we cannot assume that these priorities are familiar to all agents. Therefore, a solution concept for an ad hoc team problem may be such that neither agent has a complete specification of the priorities. This, in turn, implies that those solution concepts which are appropriate for homogeneous or heterogeneous groups of agents may not necessarily be appropriate for ad hoc teams. For instance, Schelling's concept of the focal point [42] may be inapplicable in ad hoc teams since it is based on things such as common norms and conventions, which we cannot assume in ad hoc teams. Another example of a solution concept which may not be appropriate for ad hoc teams is Brams' concept of the non-myopic equilibrium [8]. Therein, the players are in an equilibrium if no player has an incentive to unilaterally deviate from its strategy after anticipating all possible moves and countermoves from the current state. In an ad hoc team, it would be extremely difficult to arrive at such an equilibrium as we can hardly assume that every agent has the ability to anticipate all possible countermoves and their outcomes. Thus, when choosing appropriate solution concepts for ad hoc team problems, one has to take into account that the the range of possible assumptions is very restricted.

# Chapter 6

# Conclusion

This work compared the performance of five multiagent learning algorithms in ad hoc teams. Our experiments were divided into three parts. The first two parts tested the algorithms on all structurally distinct strictly ordinal $2 \times 2$ no-conflict and conflict games, respectively. For the third part, we used an evaluation procedure proposed by Stone et al. [48]. Our performance criteria included the convergence rate, the final expected payoff, social welfare and fairness, and the rates of different solution types.

The motivation behind our work was to see how the algorithms perform in heterogeneous groups of agents without prior coordination (i.e., ad hoc teams). From this, we hoped to identify those algorithms (or, more precisely, their approaches) that were best suited for ad hoc team problems. This knowledge could then be used for the design of new ad hoc agents. Another motivation for our work was the creation of a framework that can be used to simulate arbitrary repeated games.

Based on the results of our experiments, we first established that there was no clear favourite among the algorithms since all algorithms performed well in some sense. We then concluded that the performance of an algorithm ultimately depends on the solution concept that is considered most appropriate for the problem at hand. Finally, we argued that it is important to consider a broad spectrum of solution concepts in order to fully assess the performance of an algorithm.

# Chapter 7

# Outlook

There are several directions in which our research can be continued. First of all, it would be interesting to see how the algorithms perform in games with increasing numbers of agents and actions. This is interesting since it shows how the algorithms scale. Similarly, it would be interesting to study the algorithms in stochastic games with multiple states. For example, one may consider grid world scenarios in which the environment is divided by a grid and the agents have to navigate through the environment using actions such as *top*, *down*, *left*, and *right*. Hu and Wellman [23] use a grid world to investigate the Nash Q-Learner. Stone et al. [4] evaluate their ad hoc agent in the predator domain, a grid world in which a group of agents has to catch another agent. Grid world problems are useful because both environment and strategies are easy to visualise, and because the visualisation can be used to verify learned strategies.

Our work largely neglects the theoretical foundations of the algorithms. However, we believe that a theoretical investigation could result in helpful insights into the behaviour of the algorithms in ad hoc teams. This, in turn, might aid our notion of which properties a good ad hoc agent should possess. For instance, one may investigate if and how the convergence and optimality properties of the algorithms change in the setting of ad hoc teams. Another interesting question is how the algorithms could be modified in order to improve the welfare and/or fairness in ad hoc teams.

The problem of ad hoc team collaboration imposes a new level of difficulty in that it couples heterogeneous groups of agents with the assumption that there is no form of prior coordination. We believe that it may be useful to study new solution concepts for this type of problem. So far, research has mostly focused on the concepts of Nash equi-

libria and Pareto-optimal solutions (or some combination of these, e.g. [3]). In our work, we have introduced the concepts of welfare optimality and fairness optimality. These can be considered to be mainly of theoretical value since they define difficult subsets of the set of Pareto-optimal solutions. However, another interesting solution concept is the *correlated equilibrium* [20, 21]. Suppose that in each state, all agents are provided with an action recommendation by an independent referee. The referee uses a strategy profile to generate the recommendations. If no agent has an incentive to deviate from its recommendations, then the agents are said to be in a correlated equilibrium. The set of correlated equilibria is a strict superset of the set of Nash equilibria (in some cases these sets coincide completely [18]). Correlated equilibria are interesting because they resemble Nash equilibria, yet they are easy to calculate (in polynomial time using linear programming [18]). If RegMat uses the CR-rule (see section 3.4), then the algorithm is guaranteed to converge to a correlated equilibrium in self-play. Another algorithm that uses this solution concept is the Correlated Q-Learner by Greenwald and Hall [18]. It may be useful to study ad hoc team problems using the concept of correlated equilibria as this could simplify the problem to some extent.

The ultimate goal of our research is to solve the scientific challenge posed by Stone et al. (see section 2.4). That is, our goal is to design an autonomous agent that can efficiently and robustly collaborate with a previously unknown group of agents. As pointed out by Stone et al. [48], this involves the study of three aspects: *(a)* identifying the full range of possible teamwork situations, *(b)* formulating theoretically optimal and empirically efficient algorithms for each teamwork situation, and *(c)* the definition of methods to identify the current teamwork situation. Our future work will focus on these aspects, particularly *(b)* and *(c)*. We will propose and investigate new designs for ad hoc agents, study their performance both empirically and theoretically, and try to find properties a good ad hoc agent should preferably have.

Finally, we note that our simulation framework provides a fertile ground for future work. The next step is to extend the framework such that it supports the simulation of stochastic games with multiple states. Further extensions include the addition of new algorithms and solution concepts. We hope that this marks the beginning of the development of a universal multiagent learning framework.

# Appendix A

# Simulation framework

This appendix provides a brief description of the simulation framework. The framework was implemented in Matlab 7.10 (R2010a) and tested[1] in a number of previous releases, including Matlab 7.9 (R2009b), Matlab 7.8 (R2009a), and Matlab 7.5 (R2007b). The framework is provided to any one who would like to use it for his or her own research. The author can be contacted by e-mail: s.v.albrecht@sms.ed.ac.uk

## A.1 Functions

The current version of the framework is 0.1 (August 2011). It has a total of 40 functions. The following sections describe the most important functions. However, for more details on each of the functions, please see the comments within the functions.

### A.1.1 Simulator

The function `simulate` can be used to simulate an arbitrary repeated game. It takes as input the game to be simulated, a list of agents together with their workspaces, and the number of repetitions. It returns two lists containing the histories of strategy profiles and expected payoffs, and two more lists containing the histories of expected and estimated payoffs for each action. The estimated payoff of an action is the agent's approximated expected payoff for that action.

The function `simulate_all` can be used to simulate a list of repeated games. In addition, it computes the performance metrics discussed in section 4.2 using the function `metrics`

[1] The function `test_fw` can be used to test the basic functionality of the framework.

(see section A.1.3). It takes as input the game to be simulated, a list of agents and their workspaces, the number of repetitions per simulation, and the number of sweeps per game. A sweep contains one simulation for each permutation of the player order. The function returns a list with the performance metrics for each of the games (averaged over all simulations of the game), and another list with the performance metrics averaged over all games.

### A.1.2 Agents

The current version of the framework provides functions for 6 different MAL algorithms. These are `jal` for JAL, `cjal` for CJAL, `wolfphc` for WOLF-PHC, `regmat` for RegMat (HR-rule), `cndregmat` for RegMat (CR-rule), and `nashq` for NashQ. All of these algorithms are described in chapter 3. Each algorithm takes as input the current repetition, its own number, the joint action and joint reward of the previous repetition, the number of players and actions per player, and its workspace. It returns an action, its strategy, its estimated action payoffs, and an updated version of its workspace.

The workspace of an algorithm is a memory structure that is used to store intermediate results. The workspace is also used to specify the parameter setting of the algorithm. Each algorithm has a corresponding function (`jal_w`, `cjal_w`, `wolfphc_w`, `regmat_w`, `cndregmat_w`, `nashq_w`) that can be used to create a new workspace.

### A.1.3 Metrics

The function `metrics` can be used to compute the performance metrics discussed in section 4.2. It takes as input the simulated game and the history of strategy profiles and expected payoffs of the simulation. It returns logical values indicating if the algorithms converged, the final expected payoffs, social welfare and fairness, and logical values indicating the type of the solution (see section 4.2.4).

There are some more functions worth mentioning. For example, the function `check_ne` can be used to check if a given strategy profile is a Nash equilibrium (see section 4.2.4.1). Furthermore, the functions `pfront` and `wf_values` can be used to compute the Pareto front (see section 4.2.4.2) and the maximum welfare and fairness (see section 4.2.4.3) of a given game.

### A.1.4 Games

The framework provides two functions to generate games. The first function, `games_2x2`, takes no input arguments and returns two game lists, the first one containing all structurally distinct strictly ordinal $2 \times 2$ no-conflict games, and the second one containing all structurally distinct strictly ordinal $2 \times 2$ conflict games. Both lists are given in appendix B. The second function, `gamegen`, can be used to randomly generate a strictly ordinal game of any size. It takes as input the number of players and the number of actions for each player, and returns a random strictly ordinal game of the given size.

### A.1.5 Solvers

There are several functions that can be used to compute different solutions for a given game. The function `sample_ne` computes a sample of the set of (mixed) Nash equilibria of the given game. It uses the method described by Porter et al. [39]. For $2 \times 2$ games, one may also use the function `mixed_ne_2x2` to compute a mixed Nash equilibrium. The function `pfront` computes the Pareto front, that is, the set of all (mixed) Pareto-optimal profiles. Finally, the functions `pure_ne`, `pure_po`, `pure_wo`, and `pure_fo` can be used to compute the set of pure Nash equilibria, pure Pareto-optimal profiles, pure welfare-optimal profiles, and pure fairness-optimal profiles, respectively.

### A.1.6 Plotting

The function `plot_prf` can be used to plot a history of strategy profiles. The function takes as input a history of profiles and creates a figure that shows the development of each agent's strategy. Each strategy has its own colour, and each action has its own line style. A legend is added to the figure such that colours and line styles can be associated with strategies and actions, respectively. See figure 5.6 for an example.

Another useful plotting tool is provided by the function `plot_wf`. This function takes as input a $2 \times 2$ game and creates two 3-d figures. In both figures, the x-axis corresponds to the probability of player 1 choosing action 1, and the y-axis corresponds to the probability of player 2 choosing action 1. Thus, a point in the x/y-plane completely specifies a strategy profile for the given game. The first figure shows the social welfare for each profile, and the second figure shows the social fairness for each profile. See figure 5.4 for an example.

Finally, the function `plot_exp` can be used to plot the expected payoffs of both players in a given $2 \times 2$ game. It takes the same inputs as `plot_wf` and produces two figures, one for each player, in a similar way. The strategy profiles are plotted against the respective expected payoffs. See figure 5.3 for an example.

### A.1.7 Miscellaneous

The framework offers a variety of tools that can be used for all kinds of purposes. For example, the function `profiles` generates the set of all pure strategy profiles for the given game. In addition, the function `cprofiles` generates the set of pure counter profiles with respect to a given player in a given game. Further tools include, among others, the function `crd2ind`, which can be used to translate a pure strategy profile into an index for a flattened game matrix, and the function `ind2crd`, which can be used to translate an index into a pure strategy profile.

Another useful tool is provided by the function `evaluate`. This function implements the evaluation procedure discussed in section 5.3. It takes as input a set of potential team members, a game structure (i.e., the number of players and actions for each player), and the number of rounds and repetitions per game. In each round, the procedure generates a random strictly ordinal game of the specified size and a random team drawn from the set of potential team members. The game is simulated using the random team and each potential team member as the first player. Thus, all agents are evaluated under the same conditions. The function returns the averaged performance metrics for each agent.

# Appendix B

# Games

This chapter contains a listing of all structurally distinct strictly ordinal $2 \times 2$ games (based on [40]). The games are distinct in that no game can be reproduced by any transformation of any other game. Possible transformations include interchanging the rows, columns, and players, and any combinations of these. The games are strictly ordinal, meaning that each player ranks the 4 possible outcomes from 1 (least preferred) to 4 (most preferred), and no two outcomes can have the same rank.

A game is represented using the following format:

| $N$ | $(X)$ |
|---|---|
| $\underline{a_{1,1}, b_{1,1}}$ | $a_{1,2}, b_{1,2}$ |
| $a_{2,1}, b_{2,1}$ | $a_{2,2}, b_{2,2}$ |

$N$ is the number of the game and $X$ is the corresponding number of the game in the original listing [40]. The variables $a_{i,j}$ and $b_{i,j}$, where $i, j \in \{1,2\}$, contain the payoffs to player 1 (row player) and player 2 (column player), respectively, if player 1 chooses action $i$ and player 2 chooses action $j$. A payoff pair is underlined if the corresponding actions constitute a pure Nash equilibrium (see section 2.1).

The listing is divided into one listing for all no-conflict games and another listing for all conflict games. In a no-conflict game, the players have the same set of most preferred outcomes. In a conflict game, the players disagree on the most preferred outcomes.

## B.1 No-conflict games

| 1 | *(1)* |
|---|---|
| 4, 4 | 3, 3 |
| 2, 2 | 1, 1 |

| 2 | *(2)* |
|---|---|
| 4, 4 | 3, 3 |
| 1, 2 | 2, 1 |

| 3 | *(3)* |
|---|---|
| 4, 4 | 3, 2 |
| 2, 3 | 1, 1 |

| 4 | *(4)* |
|---|---|
| 4, 4 | 3, 2 |
| 1, 3 | 2, 1 |

| 5 | *(5)* |
|---|---|
| 4, 4 | 3, 1 |
| 1, 3 | 2, 2 |

| 6 | *(6)* |
|---|---|
| 4, 4 | 2, 3 |
| 3, 2 | 1, 1 |

| 7 | *(22)* |
|---|---|
| 4, 4 | 3, 3 |
| 2, 1 | 1, 2 |

| 8 | *(23)*[1] |
|---|---|
| 4, 4 | 3, 3 |
| 1, 1 | 2, 2 |

| 9 | *(24)* |
|---|---|
| 4, 4 | 3, 2 |
| 2, 1 | 1, 3 |

| 10 | *(25)* |
|---|---|
| 4, 4 | 3, 2 |
| 1, 1 | 2, 3 |

| 11 | *(26)* |
|---|---|
| 4, 4 | 2, 3 |
| 3, 1 | 1, 2 |

| 12 | *(27)* |
|---|---|
| 4, 4 | 2, 2 |
| 3, 1 | 1, 3 |

| 13 | *(28)* |
|---|---|
| 4, 4 | 3, 1 |
| 2, 2 | 1, 3 |

| 14 | *(29)* |
|---|---|
| 4, 4 | 3, 1 |
| 1, 2 | 2, 3 |

| 15 | *(30)* |
|---|---|
| 4, 4 | 2, 1 |
| 3, 2 | 1, 3 |

| 16 | *(58)* |
|---|---|
| 4, 4 | 2, 3 |
| 1, 1 | 3, 2 |

| 17 | *(59)* |
|---|---|
| 4, 4 | 2, 2 |
| 1, 1 | 3, 3 |

| 18 | *(60)* |
|---|---|
| 4, 4 | 2, 1 |
| 1, 2 | 3, 3 |

| 19 | *(61)* |
|---|---|
| 4, 4 | 1, 3 |
| 3, 1 | 2, 2 |

| 20 | *(62)* |
|---|---|
| 4, 4 | 1, 2 |
| 3, 1 | 2, 3 |

| 21 | *(63)* |
|---|---|
| 4, 4 | 1, 2 |
| 2, 1 | 3, 3 |

## B.2 Conflict games

| 1 | *(7)* |
|---|---|
| 3, 3 | 4, 2 |
| 2, 4 | 1, 1 |

| 2 | *(8)* |
|---|---|
| 3, 3 | 4, 2 |
| 1, 4 | 2, 1 |

| 3 | *(9)* |
|---|---|
| 3, 3 | 4, 1 |
| 1, 4 | 2, 2 |

| 4 | *(10)* |
|---|---|
| 2, 3 | 4, 2 |
| 1, 4 | 3, 1 |

| 5 | *(11)* |
|---|---|
| 2, 3 | 4, 1 |
| 1, 4 | 3, 2 |

| 6 | *(12)* |
|---|---|
| 2, 2 | 4, 1 |
| 1, 4 | 3, 3 |

| 7 | *(13)* |
|---|---|
| 3, 4 | 4, 2 |
| 2, 3 | 1, 1 |

| 8 | *(14)* |
|---|---|
| 3, 4 | 4, 2 |
| 1, 3 | 2, 1 |

| 9 | *(15)* |
|---|---|
| 3, 4 | 4, 1 |
| 2, 3 | 1, 2 |

| 10 | *(16)* |
|---|---|
| 3, 4 | 4, 1 |
| 1, 3 | 2, 2 |

| 11 | *(17)* |
|---|---|
| 2, 4 | 4, 2 |
| 1, 3 | 3, 1 |

| 12 | *(18)* |
|---|---|
| 2, 4 | 4, 1 |
| 1, 3 | 3, 2 |

| 13 | *(19)* |
|---|---|
| 3, 4 | 4, 3 |
| 1, 2 | 2, 1 |

| 14 | *(20)* |
|---|---|
| 3, 4 | 4, 3 |
| 2, 2 | 1, 1 |

| 15 | *(21)* |
|---|---|
| 2, 4 | 4, 3 |
| 1, 2 | 3, 1 |

[1] Game no. 23 in the original listing [40] has a typo: payoff $a_{2,1}$ must be 1.

| 16 | *(31)* |
|---|---|
| 3, 4 | 2, 2 |
| 1, 3 | 4, 1 |

| 17 | *(32)* |
|---|---|
| 3, 4 | 2, 1 |
| 1, 3 | 4, 2 |

| 18 | *(33)* |
|---|---|
| 3, 4 | 1, 2 |
| 2, 3 | 4, 1 |

| 19 | *(34)* |
|---|---|
| 3, 4 | 1, 1 |
| 2, 3 | 4, 2 |

| 20 | *(35)* |
|---|---|
| 2, 4 | 3, 2 |
| 1, 3 | 4, 1 |

| 21 | *(36)* |
|---|---|
| 2, 4 | 3, 1 |
| 1, 3 | 4, 2 |

| 22 | *(37)* |
|---|---|
| 3, 4 | 2, 3 |
| 1, 2 | 4, 1 |

| 23 | *(38)* |
|---|---|
| 3, 4 | 1, 3 |
| 2, 2 | 4, 1 |

| 24 | *(39)* |
|---|---|
| 2, 4 | 3, 3 |
| 1, 2 | 4, 1 |

| 25 | *(40)* |
|---|---|
| 3, 4 | 4, 1 |
| 2, 2 | 1, 3 |

| 26 | *(41)* |
|---|---|
| 3, 4 | 4, 1 |
| 1, 2 | 2, 3 |

| 27 | *(42)* |
|---|---|
| 3, 3 | 4, 1 |
| 2, 2 | 1, 4 |

| 28 | *(43)* |
|---|---|
| 3, 3 | 4, 1 |
| 1, 2 | 2, 4 |

| 29 | *(44)* |
|---|---|
| 2, 4 | 4, 1 |
| 1, 2 | 3, 3 |

| 30 | *(45)* |
|---|---|
| 3, 2 | 4, 1 |
| 2, 3 | 1, 4 |

| 31 | *(46)* |
|---|---|
| 3, 2 | 4, 1 |
| 1, 3 | 2, 4 |

| 32 | *(47)* |
|---|---|
| 2, 3 | 4, 1 |
| 1, 2 | 3, 4 |

| 33 | *(48)* |
|---|---|
| 2, 2 | 4, 1 |
| 1, 3 | 3, 4 |

| 34 | *(49)* |
|---|---|
| 3, 4 | 4, 3 |
| 2, 1 | 1, 2 |

| 35 | *(50)* |
|---|---|
| 3, 4 | 4, 3 |
| 1, 1 | 2, 2 |

| 36 | *(51)* |
|---|---|
| 3, 4 | 4, 2 |
| 2, 1 | 1, 3 |

| 37 | *(52)* |
|---|---|
| 3, 4 | 4, 2 |
| 1, 1 | 2, 3 |

| 38 | *(53)* |
|---|---|
| 3, 3 | 4, 2 |
| 2, 1 | 1, 4 |

| 39 | *(54)* |
|---|---|
| 3, 3 | 4, 2 |
| 1, 1 | 2, 4 |

| 40 | *(55)* |
|---|---|
| 2, 4 | 4, 3 |
| 1, 1 | 3, 2 |

| 41 | *(56)* |
|---|---|
| 2, 4 | 4, 2 |
| 1, 1 | 3, 3 |

| 42 | *(57)* |
|---|---|
| 2, 3 | 4, 2 |
| 1, 1 | 3, 4 |

| 43 | *(64)* |
|---|---|
| 3, 4 | 2, 1 |
| 1, 2 | 4, 3 |

| 44 | *(65)* |
|---|---|
| 2, 4 | 3, 1 |
| 1, 2 | 4, 3 |

| 45 | *(66)* |
|---|---|
| 3, 3 | 2, 4 |
| 4, 2 | 1, 1 |

| 46 | *(67)* |
|---|---|
| 2, 3 | 3, 4 |
| 4, 2 | 1, 1 |

| 47 | *(68)* |
|---|---|
| 2, 2 | 3, 4 |
| 4, 3 | 1, 1 |

| 48 | *(69)* |
|---|---|
| 2, 2 | 4, 3 |
| 3, 4 | 1, 1 |

| 49 | *(70)* |
|---|---|
| 3, 4 | 2, 1 |
| 4, 2 | 1, 3 |

| 50 | *(71)* |
|---|---|
| 3, 3 | 2, 1 |
| 4, 2 | 1, 4 |

| 51 | *(72)* |
|---|---|
| 3, 2 | 2, 1 |
| 4, 3 | 1, 4 |

| 52 | *(73)* |
|---|---|
| 2, 4 | 4, 1 |
| 3, 2 | 1, 3 |

| 53 | *(74)* |
|---|---|
| 2, 4 | 3, 1 |
| 4, 2 | 1, 3 |

| 54 | *(75)* |
|---|---|
| 2, 3 | 4, 1 |
| 3, 2 | 1, 4 |

| 55 | *(76)* |
|---|---|
| 2, 3 | 3, 1 |
| 4, 2 | 1, 4 |

| 56 | *(77)* |
|---|---|
| 2, 2 | 4, 1 |
| 3, 3 | 1, 4 |

| 57 | *(78)* |
|---|---|
| 2, 2 | 3, 1 |
| 4, 3 | 1, 4 |

# Appendix C

# Simulation results

## C.1 No-conflict games

This section contains the simulation results of section 5.1.

| Agent 1 | Agent 2 | Conv. 1 | Conv. 2 | Fexp. 1 | Fexp. 2 | Welfare | Fairness | NE | PO | WO | FO |
|---|---|---|---|---|---|---|---|---|---|---|---|
| JAL | JAL | 1 | 1 | 3.9895 | 3.9895 | 7.9790 | 15.9305 | 1 | 0.9924 | 0.9924 | 0.9924 |
| | CJAL | 1 | 1 | 3.9905 | 3.9905 | 7.9810 | 15.9400 | 1 | 0.9943 | 0.9943 | 0.9943 |
| | WOLF | 1 | 1 | 3.9924 | 3.9933 | 7.9857 | 15.9533 | 1 | 0.9952 | 0.9952 | 0.9952 |
| | RegMat | 1 | 1 | 3.9605 | 3.9536 | 7.9142 | 15.7075 | 1 | 0.9781 | 0.9781 | 0.9781 |
| | NashQ | 1 | 1 | 4 | 4 | 8 | 16 | 1 | 1 | 1 | 1 |
| CJAL | CJAL | 1 | 1 | 3.9895 | 3.9895 | 7.9790 | 15.9333 | 1 | 0.9933 | 0.9933 | 0.9933 |
| | WOLF | 1 | 1 | 3.9752 | 3.9743 | 7.9495 | 15.8343 | 1 | 0.9819 | 0.9819 | 0.9819 |
| | RegMat | 1 | 1 | 3.9605 | 3.9612 | 7.9218 | 15.7293 | 1 | 0.9790 | 0.9790 | 0.9790 |
| | NashQ | 1 | 1 | 4 | 4 | 8 | 16 | 1 | 1 | 1 | 1 |
| WOLF | WOLF | 1 | 1 | 3.8768 | 3.8807 | 7.7576 | 15.2133 | 1 | 0.9171 | 0.9171 | 0.9171 |
| | RegMat | 1 | 1 | 3.8803 | 3.8813 | 7.7616 | 15.2130 | 1 | 0.9248 | 0.9248 | 0.9248 |
| | NashQ | 0.9981 | 1 | 3.9999 | 3.9998 | 7.9998 | 15.9990 | 1 | 1 | 1 | 1 |
| RegMat | RegMat | 1 | 1 | 3.8109 | 3.8099 | 7.6208 | 14.7304 | 1 | 0.8771 | 0.8771 | 0.8771 |
| | NashQ | 0.9952 | 0.9933 | 3.9463 | 3.9203 | 7.8666 | 15.5730 | 0.9771 | 0.9695 | 0.9695 | 0.9695 |
| NashQ | NashQ | 1 | 1 | 4 | 4 | 8 | 16 | 1 | 1 | 1 | 1 |

Table C.1: Simulation results for no-conflict games.

| Agent | Conv. | Fexp. | Welfare | Fairness | NE | PO | WO | FO |
|---|---|---|---|---|---|---|---|---|
| JAL | 1 | 3.9866 | 7.9720 | 15.9063 | 1 | 0.9920 | 0.9920 | 0.9920 |
| CJAL | 1 | 3.9831 | 7.9663 | 15.8874 | 1 | 0.9897 | 0.9897 | 0.9897 |
| WOLF | 0.9996 | 3.9449 | 7.8908 | 15.6426 | 1 | 0.9638 | 0.9638 | 0.9638 |
| RegMat | 0.9990 | 3.9107 | 7.8170 | 15.3906 | 0.9954 | 0.9457 | 0.9457 | 0.9457 |
| NashQ | 0.9987 | 3.9840 | 7.9733 | 15.9144 | 0.9954 | 0.9939 | 0.9939 | 0.9939 |

Table C.2: Simulation results for no-conflict games (total).

## C.2 Conflict games

This section contains the simulation results of section 5.2.

| Agent 1 | Agent 2 | Conv. 1 | Conv. 2 | Fexp. 1 | Fexp. 2 | Welfare | Fairness | NE | PO | WO | FO |
|---|---|---|---|---|---|---|---|---|---|---|---|
| JAL | JAL | 0.8421 | 0.8421 | 3.0219 | 3.0157 | 6.0375 | 8.8914 | 0.9456 | 0.7544 | 0.6937 | 0.6060 |
| | CJAL | 0.9326 | 0.9242 | 3.0414 | 3.0546 | 6.0960 | 9.0437 | 0.8586 | 0.8011 | 0.7112 | 0.6175 |
| | WOLF | 0.8421 | 0.9382 | 2.9897 | 3.0554 | 6.0451 | 9.0088 | 0.9218 | 0.7544 | 0.7018 | 0.6316 |
| | RegMat | 0.8463 | 0.8453 | 2.9906 | 3.0415 | 6.0321 | 8.9337 | 0.9218 | 0.7582 | 0.7021 | 0.6130 |
| | NashQ | 0.9871 | 1 | 3.0263 | 3.0588 | 6.0851 | 9.1208 | 0.8433 | 0.8222 | 0.7018 | 0.6140 |
| CJAL | CJAL | 0.9656 | 0.9596 | 3.0653 | 3.0669 | 6.1322 | 9.1377 | 0.8421 | 0.8214 | 0.7333 | 0.6274 |
| | WOLF | 0.9368 | 0.8660 | 3.0091 | 3.0790 | 6.0881 | 9.1106 | 0.8460 | 0.8000 | 0.7186 | 0.6239 |
| | RegMat | 0.9393 | 0.9302 | 3.0076 | 3.0706 | 6.0782 | 9.0130 | 0.8439 | 0.8063 | 0.7277 | 0.6323 |
| | NashQ | 0.9623 | 1 | 3.0263 | 3.0685 | 6.0948 | 9.1449 | 0.8443 | 0.7961 | 0.7013 | 0.6241 |
| WOLF | WOLF | 1 | 1 | 3.0229 | 3.0217 | 6.0446 | 9.0114 | 0.9553 | 0.7509 | 0.6895 | 0.6048 |
| | RegMat | 0.9109 | 0.8439 | 3.0126 | 3.0231 | 6.0357 | 8.9789 | 0.9372 | 0.7544 | 0.6930 | 0.6053 |
| | NashQ | 1 | 1 | 3.0263 | 3.0701 | 6.0964 | 9.1489 | 0.8635 | 0.7586 | 0.6930 | 0.6053 |
| RegMat | RegMat | 0.8509 | 0.8537 | 3.0161 | 3.0129 | 6.0290 | 8.9387 | 0.9165 | 0.7600 | 0.6979 | 0.6084 |
| | NashQ | 0.8662 | 0.9952 | 3.0052 | 3.0036 | 6.0088 | 8.9409 | 0.8539 | 0.7522 | 0.6794 | 0.5956 |
| NashQ | NashQ | 1 | 1 | 3.0219 | 3.0263 | 6.0482 | 9.0219 | 0.9561 | 0.7544 | 0.6974 | 0.6096 |

Table C.3: Simulation results for conflict games.

| Agent | Conv. | Fexp. | Welfare | Fairness | NE | PO | WO | FO |
|---|---|---|---|---|---|---|---|---|
| JAL | 0.8901 | 3.0140 | 6.0592 | 8.9997 | 0.8982 | 0.7781 | 0.7021 | 0.6164 |
| CJAL | 0.9456 | 3.0326 | 6.0978 | 9.0900 | 0.8470 | 0.8050 | 0.7184 | 0.6250 |
| WOLF | 0.9430 | 3.0392 | 6.0620 | 9.0517 | 0.9047 | 0.7636 | 0.6992 | 0.6142 |
| RegMat | 0.8673 | 3.0313 | 6.0368 | 8.9610 | 0.8946 | 0.7662 | 0.7000 | 0.6109 |
| NashQ | 0.9990 | 3.0446 | 6.0667 | 9.0755 | 0.8722 | 0.7767 | 0.6946 | 0.6097 |

Table C.4: Simulation results for conflict games (total).

## C.3 Random games

This section contains the simulation results of section 5.3.

| Agent | Conv. | Fexp. | Welfare | Fairness | NE | PO | WO | FO |
|---|---|---|---|---|---|---|---|---|
| JAL | 0.854 | 5.7964 | 17.2174 | 193.1478 | 0.804 | 0.712 | 0.466 | 0.396 |
| CJAL | 0.922 | 5.7856 | 17.3521 | 196.1594 | 0.760 | 0.742 | 0.486 | 0.418 |
| WOLF | 0.918 | 5.7400 | 17.1956 | 193.0255 | 0.824 | 0.676 | 0.442 | 0.388 |
| RegMat | 0.852 | 5.7290 | 17.2315 | 193.9011 | 0.844 | 0.708 | 0.438 | 0.392 |
| NashQ | 0.980 | 5.7562 | 17.2452 | 193.6470 | 0.790 | 0.734 | 0.452 | 0.388 |

Table C.5: Simulation results for random games (total).

# Bibliography


[1] S.V. Albrecht. Comparison of mal algorithms in ad hoc team problems. Informatics Research Proposal, School of Informatics, University of Edinburgh, 2011.

[2] S.V. Albrecht. On the interface between game theory and reinforcement learning. Informatics Research Review, School of Informatics, University of Edinburgh, 2011.

[3] D. Banerjee and S. Sen. Reaching pareto-optimality in prisoners dilemma using conditional joint action learning. *Autonomous Agents and Multi-Agent Systems*, 15(1):91–108, 2007.

[4] S. Barrett, P. Stone, and S. Kraus. Empirical evaluation of ad hoc teamwork in the pursuit domain. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems*, May 2011.

[5] B. Blum, C.R. Shelton, and D. Koller. A continuation method for nash equilibria in structured games. *Journal of Artificial Intelligence Research*, 25:457–502, 2006.

[6] M.H. Bowling and M. Veloso. Multiagent learning using a variable learning rate. *Artificial Intelligence*, 136(2):215–250, 2002.

[7] R. Brafman and M. Tennenholtz. On partially controlled multi-agent systems. *Journal of Artificial Intelligence Research*, 4:477–507, 1996.

[8] S.J. Brams. *Theory of Moves*. Cambridge University Press, 1994.

[9] G.W. Brown. Iterative solution of games by fictitious play. In T.C. Koopmans, editor, *Activity Analysis of Production and Allocation*. Wiley, 1951.

[10] L. Busoniu, R. Babuska, and B. De Schutter. A comprehensive survey of multiagent reinforcement learning. *Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 38:156–172, 2008.

[11] L. Busoniu, R. Babuska, B. De Schutter, and D. Ernst. *Reinforcement learning and dynamic programming using function approximators*. CRC Press, 2010.

[12] 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, 1998.

[13] V. Conitzer and T. Sandholm. Awesome: A general multiagent learning algorithm that converges in self-play and learns a best response against stationary opponents. In *Proceedings of the 20th International Conference on Machine Learning*, volume 20, pages 83–90, 2003.

[14] K. Decker and V. Lesser. Designing a family of coordination algorithms. *Proceedings of the First International Conference on Multi-Agent Systems*, pages 73–80, 1995.

[15] M.B. Dias, B. Browning, M.M. Veloso, and A. Stentz. Dynamic heterogeneous robot teams engaged in adversarial tasks. *Robotics Institute, Carnegie Mellon University, Tech. Rep. CMU-RI-TR-05-14*, 2005.

[16] D. Fudenberg and J. Tirole. *Game theory*. MIT Press, 1991.

[17] S. Govindan and R. Wilson. A global newton method to compute nash equilibria. *Journal of Economic Theory*, 110:65–86, 2003.

[18] A. Greenwald and K. Hall. Correlated q-learning. In *Proceedings of the 20th International Conference on Machine Learning*, pages 242–249, 2003.

[19] B. Grosz and S. Kraus. Collaborative plans for complex group actions. *Artificial Intelligence*, 86:269–368, 1996.

[20] S. Hart and A. Mas-Colell. A simple adaptive procedure leading to correlated equilibrium. *Econometrica*, 68(5):1127–1150, 2000.

[21] S. Hart and A. Mas-Colell. A reinforcement procedure leading to correlated equilibrium. *Economic Essays: A Festschrift for Werner Hildenbrand*, pages 181–200, 2001.

[22] 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, page 250, 1998.

[23] J. Hu and M.P. Wellman. Experimental results on q-learning for general-sum stochastic games. In *Proceedings of the 17th International Conference on Machine Learning*, page 414. Morgan Kaufmann Publishers Inc., 2000.

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

[25] T. Jaakkola, M.I. Jordan, and S.P. Singh. On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, 6(6):1185–1201, 1994.

[26] L.P. Kaelbling. *Recent advances in reinforcement learning*. Springer-Verlag, 2010.

[27] S.O. Kimbrough and M. Lu. Simple reinforcement learning agents: Pareto beats nash in an algorithmic game theory study. *Information Systems and E-Business Management*, 3(1):1–19, 2005.

[28] M. Knudson and K. Tumer. Robot coordination with ad-hoc team formation. In *Proceedings of the 9th International Joint Conference on Autonomous Agents and Multiagent Systems*, 2010.

[29] C.E. Lemke and J.T. Howson. Equilibrium points of bimatrix games. *Journal of the Society for Industrial and Applied Mathematics*, 12(2):413–423, 1964.

[30] M.L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the 11th International Conference on Machine Learning*, volume 157, page 163, 1994.

[31] M.L. Littman. Friend-or-foe q-learning in general-sum games. In *Proceedings of the 18th International Conference on Machine Learning*, ICML '01, pages 322–328. Morgan Kaufmann Publishers Inc., 2001.

[32] M.L. Littman and C. Szepesvári. A generalized reinforcement-learning model: Convergence and applications. In *In Proceedings of the 13th International Conference on Machine Learning*, pages 310–318. Morgan Kaufmann, 1996.

[33] T.M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

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

[35] N. Nisan, T. Roughgarden, E. Tardos, and V. Vazirani. *Algorithmic game theory*. Cambridge University Press, 2007.

[36] L. Panait and S. Luke. Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 11:2005, 2005.

[37] C. Papadimitriou. Algorithms, games, and the internet. In *Proceedings of the 33rd Annual ACM Symposium on the Theory of Computing*, 2001.

[38] S. Phelps, K. Cai, P. McBurney, J. Niu, S. Parsons, and E. Sklar. Auctions, evolution, and multi-agent learning. *Adaptive Agents and Multi-Agent Systems III. Adaptation and Multi-Agent Learning*, pages 188–210, 2008.

[39] R. Porter, E. Nudelman, and Y. Shoham. Simple search methods for finding a nash equilibrium. *Games and Economic Behavior*, 63:642–662, 2008.

[40] A. Rapoport and M. Guyer. A taxonomy of $2 \times 2$ games. *General Systems: Yearbook of the Society for General Systems Research*, 11:203–214, 1966.

[41] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2nd edition, 2003.

[42] T.C. Schelling. *The Strategy of Conflict*. Harvard University Press, 1980.

[43] S. Sen, S. Airiau, and R. Mukherjee. Towards a pareto-optimal solution in general-sum games. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 153–160. ACM, 2003.

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

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

[46] Y. Shoham, R. Powers, and T. Grenager. Multi-agent reinforcement learning: a critical survey. *AAAI Fall Symposium on Artificial MultiAgent Learning*, 3:1–13, 2004.

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

[48] P. Stone, G.A. Kaminka, S. Kraus, and J.S. Rosenschein. Ad hoc autonomous agent teams: Collaboration without pre-coordination. In *Proceedings of the 24th Conference on Artificial Intelligence*, July 2010.

[49] P. Stone, G.A. Kaminka, and J.S. Rosenschein. Leading a best-response teammate in an ad hoc team. In *Agent-Mediated Electronic Commerce: Designing Trading Strategies and Mechanisms for Electronic Markets*, pages 132–146, November 2010.

[50] P. Stone and S. Kraus. To teach or not to teach? decision making under uncertainty in ad hoc teams. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems*, May 2010.

[51] Peter Stone and Manuela Veloso. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, 8:345–383, 1997.

[52] R.S. Sutton and A.G. Barto. *Reinforcement learning: An introduction*. Cambridge University Press, 1998.

[53] M. Tambe. Towards flexible teamwork. *Journal of Artificial Intelligence Research*, 7:83–124, 1997.

[54] W. Uther and M. Veloso. Adversarial reinforcement learning. Technical report, Computer Science Department, Carnegie Mellon University, 1997.

[55] G. van der Laan, A. Talman, and L. van der Heyden. Simplicial variable dimension algorithms for solving the nonlinear complementarity problem on a product of unit simplices using a general labelling. *Mathematics of Operations Research*, 12(3):377–397, 1987.

[56] J. von Neumann and O. Morgenstern. *Theory of games and economic behavior*. Princeton University Press, 1944.

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

[58] D.H. Wolpert and W.G. Macready. No free lunch theorems for search. Technical Report SFI-TR-95-02-010, Santa Fe Institute, 1995.

[59] D.H. Wolpert and W.G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

[60] M.J. Wooldridge. *An Introduction to Multiagent Systems*. Wiley, 2nd edition, 2009.

[61] F. Wu, S. Zilberstein, and X. Chen. Online planning for ad hoc autonomous agent teams. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, 2011.