University Utrecht

Master thesis

# Multi-agent learning tournaments

**Student:**
J.T. (Tim)
van Daalen B.Eng

**Supervisor:**
dr. G.A.W.
(Gerard) Vreeswijk

Intelligent Systems group
Department of Information and Computing Sciences

Version 1.0
May 29, 2019

# Abstract

In this thesis the methods used in previous multi-agent learning tournaments are compared. The goal of the comparison is to provide insight into why different methods are used and the impact of small, but important, design choices, like normalizing rewards between games to avoid misinterpretation of the results. Additional attention is payed to the fairness of the methods. After the analysis a sample tournament is played to ensure the practical problems are encountered as well. Some of the settings do not have an optimal value, in these cases the options are described and we explain the criteria we used to make a choice.

The resulting methodology is used in a small tournament to show how it can be used. The tournament is run in a modular framework which is published along with this thesis. The framework contains a parameter tuner for the algorithms, something not seen in previous research. To gain insight into N-player games some of the algorithms used in this paper have been slightly modified, which led to a new version of Bully. The tournament is analyzed with a set of statistical analysis techniques and plots which are also published. The metrics give different winners, and to our surprise Markov earned the highest average reward.

# Table of Contents

# 1 Introduction

Most autonomous software agents interact with agents or people on a regular basis, forming multi-agent systems. The self-driving cars of Tesla are a prime example. On the road cars have to adapt to the behavior of other cars as it is impossible to anticipate all the possible scenarios which can occur in these dynamic open environments. This means agents have to learn and adapt to other agents in new environments. The algorithms used for this learning and adaption are studied in multi-agent learning (MAL).

## 1.1 Motivation

Comparing learning algorithms is an important part of MAL. The comparisons help researchers understand why some algorithms perform better than others in certain situations. However, comparing algorithms is not as easy as it sounds. Most algorithms are based on different, incomparable, theoretical properties. Furthermore, the best action depends on the opponents action, whose action is influenced by your action, making empirical analysis the main method of comparison. A bias in the comparison might result in a different winner, which makes fairness an important property.

While studying MAL tournaments, we noticed that researchers use different comparison methodologies. For example Airiau, Sen, and Saha (2005) play games for 100 iterations while Crandall and Goodrich (2011) use 300,000 iterations, both without additional explanation. They also used different games, Airiau, Sen, and Saha (2005) used a subset of all games while Crandall and Goodrich (2011) only use random games. The different methods lead to different winners, raising questions about the adequacy of the used methodologies. So far little research has been done on the comparison methodology, but some important steps have already been made by Nudelman et al. (2004) and Zawadzki, Lipson, and Leyton-Brown (2014).

Nudelman et al. (2004) created Gamut, a suite of game generators to generalize the set of games on which algorithms are compared. A generator consists of a set of rules used to create the game matrix. Whilst they created 35 generators, they use only 22 in their own experiment. Besides calling this subset representative they gave no arguments for this choice, leaving the reader wondering why these 22 are representative for the 35 generators. More recently Zawadzki, Lipson, and Leyton-Brown (2014) asked some important questions about the design choices that should be considered before comparing algorithms. After making the reader aware of the possible problems, they pursued other goals.

## 1.2 Research Goals

We expected that tournaments with the same goal would use similar methodologies, instead different methodologies are used in the literature making us wonder what design choices might influence the fairness of the tournament. To discover this the methodologies used in past tournaments are analyzed and compared. The goal of this comparison is not to find the 'fair' comparison methodology, we hypothesize there is no 'optimal' methodology based on the work of Zawadzki, Lipson, and Leyton-Brown (2014). They showed that the Fictitious play algorithm performs poorly on asymmetric coordination games, like the Dispersion game, which would make a comparison with a fictitious play algorithm on just this game biased and thereby unfair. Based on this example we state the goal (e.g. find the best algorithm on asymmetric coordination games) and the competing algorithms determine what is fair.

It is not our intention to give another comparison, instead we give an overview of the possible comparison methods and explain which is best suited to our goal: determining which algorithm is the best performer from a group of at least five algorithms. Additionally, the choices missing in previous papers are examined. We understand that page limitations prevent researchers from describing all details, despite this we choose to examine them to give a complete overview.

### 1.2.1 Methodology

Too many MAL publications contain tournaments, and it is not possible to study them all. Therefore, a literature search is used to find papers which focus on the comparison methodology or contain a comparison with more than five algorithms. After the search the methodologies were studied to identify the key components: the tournament type, games, algorithms and the results analysis. The key components are analyzed in detail through a combination of literature analysis and small tests to ensures that both the theoretical and practical options and implications are found. The various methods we found are compared on their fairness and we describe their advantages and disadvantages, during this process arguments are given for the method used in our own comparison. If no hard arguments can be made for an option, we explain why and give the criteria we used to choose. As we are mainly interested in the methodology used to find the best algorithm, limited computational power and time is invested in the tournament. We indicate which decisions are made to save time.

## 1.3 Outline

MAL is an extension of game theory, the connection is explained in Chapter 2. The main part of the thesis, the methodological considerations are described in Chapter 3. The methodological consideration of a tournament should not be confused with the research methodology described in the previous section. The methodological considerations are applied to a sample tournament in Chapter 4. Chapter 5 concludes the thesis with the discussion, future works and the conclusion.

# 2 Background

In the first section the aspects of game theory used in this thesis are explained to introduce the notation, for a detailed explanation of game theory we refer to the book of Shoham and Leyton-Brown (2008). Section 2 introduces the game type used in MAL: repeated games.

## 2.1 A Game

A well known normal form game is the Prisoner's Dilemma (PD) game shown in Table 2.1. The game has two players, the row player (player one) and the column player (player two). Both players have two actions ($a \in A$): cooperate (C) and defect (D) and each action pair, strategy (S) results in a reward (R) for both players. If we look at the game from player one's perspective, player one is called the protagonist and player two the opponent. When the protagonist cooperates and the opponent defects: action pair (C,D) the protagonist earns reward 'S' and the opponent earns 'T'. The action played by the protagonist is noted as $\mathrm{a_i}$ with reward $\mathrm{r_i}$, for the opponent $\mathrm{a_{-i}}$ and $\mathrm{r_{-i}}$ are used.

Table 2.1: General form of the Prisoner's Dilemma with rewards $T > R > P > S$

| | *C* | *D* |
|---|---|---|
| *C* | R/R | S/T |
| *D* | T/S | P/P |

### 2.1.1 Game properties

Various concepts have been defined in game theory to find the action that maximizes the expected reward given the knowledge of the game and opponent. Which concept is best depends on both the game and the opponents, the three main concepts used by MAL algorithms are: the Nash Equilibria, Pareto optimality and the Maxmin strategy.

**Nash equilibrium**

Nash et al. (1950) proved that each game has at least one pure or mixed Nash equilibrium and can have multiple equilibria. A Nash equilibrium (NE) is an action pair from which no player wants to deviate, because deviating results in a lower reward. This ensures that a player who plays their part of a NE cannot be exploited.

**Pareto optimal outcomes**

Whereas algorithms that play their part of the NE cannot be exploited, playing the NE does not guarantee it earns the highest possible reward. The NE of the PD is the action pair (D,D) with the lowest possible reward, 'P'. The Pareto optimal (PO) outcomes focus on high rewards, a reward is Pareto optimal when there is no outcome for which one player earns a higher reward without another player earning less. In the PD the action pairs, profiles: (C,C), (C,D) and (D,C) are PO. Assuming algorithms try to maximize their reward, they both have a preference order over the possible outcomes of the game:

- Player 1: (D,C):T, (C,C):R, (D,D):P, (C,D):S.
- Player 2: (C,D):T, (C,C):R, (D,D):P, (D,C):S.

The first preference of both algorithms is to exploit their opponent to earn the highest possible reward, the competitive PO strategy. The second preference action pair is the same for both players, the cooperative PO reward. This distinction cannot be made on all games, an example is Matching Pennies which has only competitive PO outcomes.

**Maxmin strategy**

When the opponents intention is unknown, the protagonist can play safe and assume the opponent plays the action that minimizes his reward. Under this assumption the highest reward the protagonist can secure for itself is the maxmin or security level reward, formally written as:

$$Maxmin(player1) = argmax_{s_i \in S_i} min\{R_i(s_i, s_{-i}) | s_{-i} \in S_{-i}\} \tag{2.1}$$

## 2.2 Repeated Games

In repeated games two or more algorithms play the same game against each other for a finite number of iterations. The number of iterations is usually not given to the algorithms, as far as they know they are playing an infinite amount of iterations. During the game, algorithms can use the information from previous iterations to determine the action for the next iteration. This means algorithms have to consider how their current action influences the opponents' next action. Cooperating might not be smart when the PD is played for one iteration, but will lead to a higher average reward in the repeated game if the opponent also cooperates. The comparisons of multiple algorithms on various games is called a tournament, and a game played by a pair of algorithms is called a comparison.

Some algorithms play the same action in each iteration, this is known as a stationary strategies. An example is ALLC which always cooperates. Other strategies are more competitive and try to learn the opponent's strategy, in order to earn the temptation reward (T). However, to learn, algorithms needs data: the history of play. As both algorithms try to model their opponents strategy, a complex learning dynamic emerges which can be different each time the game is played. For this reason the game is often played multiple times (rounds), in order to evaluate the most common patterns.

# 3 Methodological considerations

Empirical comparison is an important part of MAL research. After defining and proving theoretical properties of a new algorithm, experiments are commonly used to test if they have the desired effect. Other fields compare algorithms by using them to solve the same problem and comparing the results. In the case of MAL, the question is whether the new properties lead to the best possible outcome on the given scenarios. This is tested through a competition against other algorithms, also known as a tournament. Whilst new computers enable researchers to run bigger tournaments, they can still only test a small portion of the possible scenarios. The results of these tournaments can be used by statistics to make claims about the entire population, but in MAL it is unclear to what extent the results are comparable. The main goal is vague: Find the algorithm that earns the highest reward. As a result different games and settings are used in each MAL tournament. Some researchers make even bigger changes, Zawadzki, Lipson, and Leyton-Brown (2014) decided to boostrap the results of one round instead of playing multiple rounds. Boostrapping samples the results with replacement to test the reproducibility. Bouzy and Métivier (2010) used a different type of tournament, which raises the question if these results are comparable?

Different settings, algorithms, and games are used in each tournament to answer a different question, meaning the results are not one-on-one comparable with previous work. As a result the observations and patterns found in our comparison, for example that games based on the same theoretical properties lead to similar rewards, might not be found in follow-up experiments. Especially since the papers we read did not report all details, therefore we focus on the reasons to choose settings and the possible consequences.

In this chapter the past tournaments are analyzed in four sections, first the tournaments and global settings are compared. In Section 2 the games are analyzed, followed by the algorithms in Section 3 and result analysis methods in Section 4. We argue for standardization where possible. For some choices there is no 'optimal' outcome, in this case the advantages and disadvantages of the various methods are highlighted and we give the criteria we based our decisions on.

## 3.1 Tournaments

Tournaments are organized to find the best algorithms for a given situation. The design choices of the tournament are discussed in this section, starting with the tournament type. In Subsection 2 the burn-in time is analyzed, these are the iterations algorithms can use to learn before the rewards are recorded. In Subsection 3 a special type of tournament is described: evolutionary tournaments. And in the last subsection we argue for the global parameters used in our tournament.

### 3.1.1 Type of tournament

Round robin tournaments were used in all the studied experiments, listed in Table 3.2. In a round robin tournament all algorithms play against each other, with four algorithms six comparisons are played, ten including self-play as shown in Table 3.1. For example, in an elimination tournament with four algorithms only three comparisons are played: Al1 vs Al2, Al3 vs Al4 and one between the winners of the first round, visualized in in Figure 3.1.

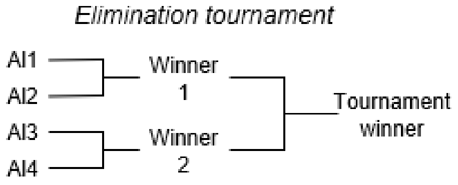


Figure 3.1: Comparisons in an elimination tournament

Table 3.1: Comparisons in a round robin tournament.

| *Round robin tournament* | | | | |
|---|---|---|---|---|
| | *Al1* | *Al2* | *Al3* | *Al4* |
| *Al1* | vs | vs | vs | vs |
| *Al2* | - | vs | vs | vs |
| *Al3* | - | - | vs | vs |
| *Al4* | - | - | - | vs |

With the additional comparisons the round robin tournament avoids the seeding bias of the elimination tournament, assuming Al3 beats Al4, Al1 and Al2 beat Al3 and Al4 beats Al1 and Al2. If Al3 and Al4 are paired together Al1 or Al2 wins while Al4 wins for the other pairings. Because all algorithms play against each other in a round robin tournament there is no seeding, making it a fairer comparison method. The linear relation between the number of algorithms and the computation time is also no problem for modern day computer clusters. Despite this it is interesting to check the alternatives. In the following subsection the demands for a MAL tournament are discussed and other tournament types, including the one introduced by Bouzy and Métivier (2010) are analyzed.

#### 3.1.1.1 Self-play

Human players cannot play against themselves, as a result the main diagonal of the round robin is empty in human tournaments. Instances of algorithms can play against themselves, but why would they? Whilst MAL algorithms do not compete against copies of themselves to learn (yet) as AlphaGo did for example, it is likely that the same algorithm will be implemented in all products of a manufacturer. As a result the algorithms will encounter copies of themselves along with other algorithms in practice, for this reason self-play plays an important role in evolutionary tournaments. Zawadzki, Lipson, and Leyton-Brown (2014) found a significant relationship between self-play and below average rewards for their algorithms, showing that there is still some work left in this area.

Self-play is not used in game and sport tournaments as a result there is no description of how it can be added to existing tournaments types (Byl 2013). A simple solution would be adding it in the seeding process.

#### 3.1.1.2 Seeding

As soon as players are not playing against all their opponents, some form of seeding is needed to determine who plays against who. In human tournaments the seeds are often based on previous performance or chosen randomly. This works because the rules are the same in most sport tournaments, the difference is the opponent. Between MAL tournament the rules (called parameters) differ, making the previous results unusable. A random choice sounds fair, but can lead to a different winner each time the tournament is played, which is undesirable. A third option is multiple preliminary tournaments as done in soccer, shifting the problem to determining who plays who in the preliminaries.

The Swiss-style tournaments, credited to J. Muller who first used the tournament structure, compromises between the seeding and number of rounds (Just, Burg, et al. 2003). In a Swiss tournament the algorithms are seeded in the first round, after this round no algorithm is removed, instead the rewards of the first round are used to create pairings for the second round. The pairings of the following rounds are created in the same way. This ensures opponents of equal strength are paired, creating the pairings was complicated at the time of the first MAL tournaments, but is no problem for modern computers with new pairing algorithms.

The Swiss-style tournament has one more constraint: algorithms are only allowed to play each other once. Eventually no combinations can be made anymore, when this happens a complete round robin tournament has been played. Because no algorithm is removed the influence of the initial pairing is mainly determined by the number of rounds the tournament is played, which might give the same result as the round robin tournament using less comparisons.

#### 3.1.1.3 Rank based elimination

Bouzy and Métivier (2010) introduced a new tournament type: multiple round robin tournaments with elimination. After each tournament the accumulated rewards of the players are used to rank the players, the worst player is removed based on one of the following criteria:

- If the difference between the two worst performing algorithms becomes bigger than a threshold, which decreases with the root of the number of played tournaments.
- If the ranking has not changed during a given number of tournaments, this number increases every time a player is removed.

After an algorithm is removed, the rewards its opponents earned when they played against it are removed from their accumulated rewards. The elimination tournament gives better results for the authors criteria than the round robin tournament: ”An experiment with the elimination mechanism provides the final ranking with better correctness guarantees on the top of the ranking than an experiment without it.” (Bouzy and Métivier 2010). A normal round robin tournament gives the best contestant of the group, which can be different if the tournament is rerun without one of the lower scoring algorithms. The elimination mechanism gives better guarantees that the tournament winner stays the same because removing an algorithm should only affect the ranking of the algorithms beneath it.

The downside of the elimination method is the additional computation time required to play multiple round robin tournaments, as many as required to eliminate all algorithms except the winner. It is unclear if the thresholds can be lowered to reduce the computation time as no arguments are given for the values. Where we can think of practical situations in which the worst performers are eliminated, for example the stock market, we expect algorithms will be replaced by newer variants instead of eliminated in most situations. There will also be situations in which algorithms are eliminated, but for our purpose this information is not worth the computational costs.

#### 3.1.1.4 Performance measure

In most tournaments the outcome is converted to points: two points for a win, one point for a draw and zero points for a loss. This is done to find the performer that won the most comparisons. In MAL tournaments the algorithm that earned the highest average reward wins instead of the algorithm that won most comparisons. A few big wins can be enough to win a tournament, even if the rest of the games are drawn or lost. Which method is preferred depends on the underlying goal i.e. finding the algorithm that won most of the time or finding the algorithm with the highest average reward.

Publishing the resulting round robin tables allows the reader to do both a Win-draw-loss analysis and an average reward analysis even if the author is not interested in them. Furthermore, it also allows a similar analysis to the rank based elimination: removal of algorithms based on average rewards. Making the result more generic than when only the results of the chosen method are published.

#### 3.1.1.5 Used tournament

Most tournament types are developed for sports, for which different design consideration are important compared to algorithms: people might try to cheat, learn from past tournaments and analyze opponents past tournaments. In human tournaments playing fields and rest time between comparisons are also important factors, these do not play a role in algorithm comparisons. Instead self-play and seeding complicate the design. Self-play can be used as seed by letting the algorithms play themselves to determine an initial seeding. Where this could solve both problems, it might introduce a bias as self-play is only one of the situations we are interested in.

Another main consideration is the number of algorithms, which was kept relatively low in previous tournament, between three and fifteen algorithms. The number of algorithms should not be kept low because of the computation time of the round robin tournament. The Swiss tournament is a promising alternative for bigger tournaments, it has been used in tournaments with hundreds of chess players. But for limited sized tournaments, like ours the round robin tournament is still the best option, avoiding the seeding bias entirely.

### 3.1.2 Burn-in time

The burn-in time is the number of iterations algorithms play against each other before the comparison officially starts, these iterations are not recorded and therefore do not influence the outcome of the tournament. The burn-in period gives algorithms time to learn which actions result in the highest rewards, more complicated algorithms usually need more learning iterations. As a result, the burn-in time varies significantly in the studied papers shown in Table 3.2. Next to the last iterations, Crandall (2014) recorded the rewards the algorithms earn in the first 1,000 iterations to show the learning speed of their algorithm. They also compare the rewards earned in the first to the rewards earned in the last 1,000 iterations in Figure 10 of their paper showing that other algorithms, like M-Qubed either need different parameters or more than 1,000 iterations to achieve high rewards.

The number of burn-in iterations can follow from a constraint of the environment in which case algorithms that need more learning iterations are not suited for the environment. Ishowo-Oloko et al. (2014) found 100 iterations is the maximum before humans start to lose focus, therefore they limited the number of iterations to 100 in a tournament between humans and MAL algorithms. If the number of burn-in iterations is not given but chosen by the designers it is important that all algorithms get enough iterations to learn. Some algorithms have parameters that can be tuned to the given number of iterations, others need a given amount of iterations to model the opponents strategy. The experiments of the authors can give an indication of how many iterations their algorithm needs. Zawadzki, Lipson, and Leyton-Brown (2014) use a statistical test to asses convergence, which also gives an indication if the algorithms played enough learning iterations. They double the accumulated reward earned half-way the recorded iterations and compare this to the accumulated reward after the last recorded iteration. If the two vary less than a given threshold the algorithms played enough learning iterations. Some algorithms like Random will not converge, but if enough iterations are recorded the accumulated rewards of algorithms who play semi-random or a cycle of action does converge, provided that the opponent is not playing random.

### 3.1.3 Evolutionary dynamics

Traditional game theory gives a snapshot of who beats who and who wins a tournament using a handful of players. In real environments there will be more players interacting with each other through one of the available algorithms. The learning dynamics that occur in these groups of players (populations) are studied in evolutionary game theory. Evolutionary game theory is a combination of game theory and evolutionary models from biology (Weibull 1997). The concept works as follows: First a normal round robin tournament is played between the algorithms. After this a group is formed for each algorithm and the members of the population are divided over the groups according to a predefined distribution. All members can view the results of the tournament and the current distribution. Each time-step the members use this information to choose in which group they want to be, updating the distribution. This process is continued until the members stop switching, the distribution converges to a fixed value or the distribution becomes cyclic. If the proportion of an algorithm is zero in the initial distribution no group is created for it.

Evolutionary dynamics are implemented in two ways which both require the results of a full round robin tournament to model the interactions between all algorithms. The first, the evolutionary algorithm (EA) approach is used by Airiau, Saha, and Sen (2004). Each time-step the EA creates pairs of players using a given selection rule, usually based on the rewards of the tournament. The paired players create two offspring and are removed from the population. Several rules are used to create the offspring, it can inherit from the parents, called crossover or it can choose a random group, mutate. There are various combinations of selection, mutation and crossover rules possible giving slightly different dynamics (Goldberg and Deb 1991). If the last player of a group is removed the group is also removed and cannot come back.

The second method, the replicator dynamic abstracts away from the evolutionary process, using a set of equations to update the proportions (P) (Taylor and Jonker 1978). The first step, EQ 3.1 consists of calculating the fitness values (F) for all algorithms (A). The fitness values are summed to calculate the total fitness using EQ 3.2. Both the fitness values and the total fitness are used in EQ 3.3 to calculate the proportions for the next time-step. In contrary to EA's the proportions never reduce to zero in the replicator dynamic, unless they are set to zero at the start.

$$\forall A : F_A = \sum_{a \in A} R_{A,a} * P_a \tag{3.1}$$

$$F_{tot} = \sum_{a \in A} F_a \tag{3.2}$$

$$\forall A : P_a(t+1) = \frac{P_a * F_a}{F_{tot}} \tag{3.3}$$

The replicator dynamic can be calculated faster and is easier to implement than the EA approach. The downside of the replicator is the rationality assumption on which it is based, which might not hold in practice. The advantages of the EA approach are the flexibility and that it can also be used when rationality cannot be assumed.

Evolutionary dynamics can also be used to model what happens when new (or eliminated) algorithms are introduced after the population converged. For instance, if algorithm Al1 loses from Al2 and a given population has converged to Al1 introducing a handful of Al2 players can lead to the extinction off Al1. An evolutionary stable strategy (ESS) is immune to these invasions (Gintis 2009). The ESS are a subset of the NE, a NE is a best reply against other algorithms. The ESS is also a best reply to the algorithm itself, which makes it a promising property for analysis and concept for new algorithms.

We see the evolutionary dynamics as an important second test for algorithms, to get a better understanding of the dynamics that occur in practice. But first a normal tournament should be run to test if the algorithm can beat its opponents. If a round robin tournament is used, the results can also be used for the evolutionary tournament. In the other cases, a small-scale round robin tournament can be played to get all the rewards needed for the evolutionary tournament. Of the two implementations we prefer the replicator dynamics, both because it is an abstraction of the EA approach and because of its lower computation time.

### 3.1.4 Setup

As shown in Table 3.2 different settings are used in each tournament, whilst they are important there are no 'optimal' values as described in Section 3.1.2. If the algorithms stopped learning the number of recorded iterations should ideally be just enough to record two full action cycles to allow convergence testing. It is difficult to determine when this happens and since recording more iterations does not hurt, it only leads to an increase in simulation times, recording more iterations than we expect to need is a common solution.

The same can be said for the number of rounds, more rounds give a higher guarantee that the results are reproducible. Zawadzki, Lipson, and Leyton-Brown (2014) use an alternative method to test the reproducibility of the results: bootstrapping, the idea is to take samples from the results with replacement, creating variations of the original results. Bootstrapping is commonly used to get higher confidence levels when running the experiment multiple times would take too long, as was the case in the authors experiment. They reported the experiment took seven days to run. However, using only one round can give misleading results. Chernick (2011) describes a few situations in which bootstrapping fails, one of them is stationary stochastic processes in which early observations have a large influence on observations taken later in time. This is the case in MAL tournaments, in which a different (random) initial choice can influence the outcome of the round. The actions played during the recorded iterations depend on what happened during the learning process, one algorithm making a different random choice can completely change the outcome of the comparison. Furthermore, most comparisons converge to one or a handful of action combinations while other actions might only be played a few of times or not at all. Therefore we doubt if running the experiment only once is a good idea.

Table 3.2: Number of rounds, iterations played and recorded of the studied comparisons.

| *Paper* | *Rounds* | *Iterations* | *Recorded* |
|---|---|---|---|
| *(Airiau and Sen 2003)* *(Airiau, Saha, and Sen 2004)* *(Airiau, Sen, and Saha 2005)* | 100 | 100 | Last 50 |
| *(Vu, Powers, and Shoham 2006)* | ? | 200,000 | Last 20,000 |
| *(Bouzy and Métivier 2010)* | ? | 1,000,000 | ? |
| *(Bouzy, Métivier, and Pellier 2018)* | ? | 100,000 - 1,000,000 | ? |
| *(Crandall and Goodrich 2011)* | 1,000 | 300,000 | All |
| *(Crandall, Ahmed, and Goodrich 2011)* | 50 | 300,000 | All |
| *(Crandall 2014)* | 50 | 50,000 | First 1,000, Last 1,000 |
| *(Zawadzki, Lipson, and Leyton-Brown 2014)* | 1 | 100,000 | Last 10,000 |

## 3.2 Games

Games represent the scenarios for which the best algorithm is sought through a tournament. It is unlikely there will ever be an algorithm that beats all opponents in all scenarios (Crandall and Goodrich 2011). Algorithms that are made for a specific game are likely to win in it, while algorithms that score good on a wide variety of games are unlikely to win in all of them. In this section the game selection of previous tournaments is analyzed and the games for our tournament are chosen.

### 3.2.1 What games to test on?

The studied tournaments only use normal form games. Some other tournaments used extensive form games in which the players choose actions sequentially, making the game easier for the second player by informing him of the first players action. We expect normal form games are preferred because these have been studied extensively in game theory as described in the following sections.

In the studied literature three types of game selection are used: a subset of all games, random games and games based on real social dilemmas called constructed games. In this section we investigate why these games were chosen.

#### 3.2.1.1 A subset of all 2×2 games

The papers that use a subset of all games focus on the normal form 2×2 games, in particular the 78 games with strictly ordinal rewards. In strictly ordinal reward games the rewards are translated to preferences, a higher reward is represented by a bigger preference. Whilst it is likely the algorithm will have a preference for an action in a real environment (Jose B. Cruz and Simaan 2002), we could not find a reason why these 78 games are seen as representative for all 726 distinct $2 \times 2$ games (Kilgour and Fraser 1988).

Airiau and Sen (2003) use a subset of the 78 ordinal games, the 57 competitive games, in their tournament. They also mention that most games favor the column player, to compensate for this the algorithms play each game both as column and row player against the same opponent. Only a small set, the symmetrical games, give similar payoffs to the row and column player and therefore only have to played once. The PD is a prime example, unfortunately not all strategically distinct situations can be made with the symmetrical games.

The 78 strictly ordinal games can be played as column and row resulting in 156-12 (symmetrical games) = 144 different games. The taxonomy of these games is visualized using colors to show the different types of games (Bruns, n.d.). Bruns' periodic table classifies six strategically different types of games, showing many of the 78 games have similar properties.

#### 3.2.1.2 Constructed games

Certain games like the Prisoner's Dilemma, used by Axelrod and Hamilton (1981), are interesting because they are based on real social dilemmas. There are lists of constructed games, with a motivation why they are interesting ("List of games in game theory" n.d.). Nudelman et al. (2004) did an extensive literature search to find all interesting, constructed games. They used common game properties: 'is the game symmetric?', The number of NE and other properties to create a taxonomy of the constructed games. Based on the taxonomy they created 35 groups for which they made game generators. The taxonomy can be used to select a generator with specific properties from Gamut, which is used in some tournaments (Powers and Shoham 2005b),(Vu, Powers, and Shoham 2006) and (Zawadzki, Lipson, and Leyton-Brown 2014). Others like Crandall and Goodrich (2011) have not heard of it or at least do not mention Gamut in their game selection.

#### 3.2.1.3 Random Games

Bouzy, Métivier, and Pellier (2018) used random games: "One experiment consists in drawing N matrix games at random with return values in the interval [-9,+9]". Despite referencing the work of Nudelman et al. (2004), they did not give an argument why they decided to use random games. And where Crandall and Goodrich (2011) use twelve constructed games they argue that using the constructed games might lead to overfitting, therefore they decided to use random games in a following tournament: "While random games tend to produce less variation in average payoffs than do selected games, we use random games as a guard against overfitting the selected games." (Crandall 2014). Most tournaments we studied indeed focus on the selected, constructed games (e.g. the Prisoner's Dilemma) and this can lead to overfitting. However, we think using only random games is not the solution. The constructed games are based on real economical and sociological situations which do not occur in random games picked from a uniform distribution (Nudelman et al. 2004). There are also fewer strategically different situations in random games than in constructed games.

Using a variety of games is not only important to prevent overfitting but it also ensures fairness. If an algorithm performs poorly on one game, e.g. the Prisoner's Dilemma and only this game is used it would be labeled 'bad', while it might earn high rewards on other games. A similar scenario occurs when only random games are used, using more strategically similar games e.g. the full set of 78 strict ordinal payoff games does not add variety either. The best solution we see is Gamut, using multiple (all) generators to ensure diversity between games while using a combination of constructed and random games. The games are also checked for symmetry, only the asymmetric games are played as both column and row player.

### 3.2.2 Why $2 \times 2$ games?

Most research focuses on two player two action games with a handful of three action games like the Shapley game and Rock Paper Scissors. One reason for this is the PPAD hard task of finding the Nash Equilibria, each game has at least one NE but to find it a directed graph problem has to be solved (Daskalakis, Goldberg, and Papadimitriou 2009). The complexity of a PPAD problem is comparable to that of NP-hard problems, the difference is that it is known there is a solution for PPAD problems (Papadimitriou 1994). Furthermore, the complexity of the graph increases with the game size without the guarantee that all equilibria will be found (McKelvey, McLennan, and Turocy 2006). The results of small games are also easier to analyze: "Although it would also be interesting to study performance in larger games, we decided to focus on a simpler setting in which it would be easier to understand the results of our experiments" (Nudelman et al. 2004). Intuitively more players and more actions give more possible strategies making it harder to analyze the results, despite this the number of actions is increased in recent works:

- Crandall, Ahmed, and Goodrich (2011) used constructed and random two player games with two to five actions.
- Zawadzki, Lipson, and Leyton-Brown (2014) used two player games with five action sizes: $2 \times 2$, $4 \times 4$, $6 \times 6$, $8 \times 8$ and $10 \times 10$.

We expect the number of actions is increased instead of the number of players because it is computationally cheaper and most result analysis methods are created for two player games, the result analysis of 3/N-player games is discussed in Section 4, result analysis. With more players more combinations of opponents can be made, as shown in Table 3.3. The possible combinations for different numbers of algorithms (NAL) and numbers of players (NPL) are calculated with Eq. 3.4 and Eq. 3.5. In asymmetric games every player can be played by all instances of the algorithms. Symmetric games do not have to be played both as column and row, resulting in less combinations of players.

$$\text{Matches on an assymetric game} = \text{NAL}^{\text{NPL}} \tag{3.4}$$

$$\text{Matches on a symmetric game} = \binom{\text{NAL} + \text{NPL} - 1}{\text{NPL}} = \frac{(\text{NAL} + \text{NPL} - 1)!}{\text{NPL}! \times ((\text{NAL} + \text{NPL} - 1) - \text{NPL})!} \tag{3.5}$$

Table 3.3: Top, algorithm combination on asymmetric games. Bottom, algorithm combination on symmetric games.

| | | *Algorithms* | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | *2* | *3* | *4* | *5* | *6* | *7* | *8* | *9* | *10* |
| *Players* | *2* | 4 | 9 | 16 | 25 | 36 | 49 | 64 | 81 | 100 |
| | *3* | - | 27 | 64 | 125 | 216 | 343 | 512 | 729 | 1,000 |
| | *4* | - | - | 256 | 625 | 1,296 | 2,401 | 4,096 | 6,561 | 10,000 |

| | | *Algorithms* | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | *2* | *3* | *4* | *5* | *6* | *7* | *8* | *9* | *10* |
| *Players* | *2* | 3 | 6 | 10 | 15 | 21 | 28 | 36 | 45 | 55 |
| | *3* | - | 10 | 20 | 35 | 56 | 84 | 120 | 165 | 220 |
| | *4* | - | - | 35 | 70 | 126 | 210 | 330 | 495 | 715 |

Each combination of algorithms plays all games. With seven algorithms there are 343 possible combinations of opponents on three player games, versus 49 on a two player game. That is significant increase in computation time. Adding actions has a smaller impact on the computation time for most algorithms, others like Hyper-Q are too inefficient to play games with more than two or three actions (Crandall and Goodrich 2011).

Whilst adding more actions has a smaller impact on the computation time than adding players, it should not be the only reason to choose this variation. Increasing the number of players or changing other parameters might lead to more interesting learning situations. Stochastic games are an example, in these games the parameters are changed when specific action pairs are played (Shapley 1953). Zawadzki, Lipson, and Leyton-Brown (2014) compared the results of games with various action sizes and found "There is no general relationship between game size and reward: on some generators there is a strong positive correlation and on other generators there is a strong negative correlation." They also note that the results depends heavily on the opponents: "Algorithm performance depended substantially on which opponent was played", because of this we expect different results when the number of players is increased.

### 3.2.3 Game selection

In Figure 5 of their paper Nudelman et al. (2004) show that different algorithms perform better on different generators of Gamut, but decided to use thirteen instead of all 35 generators in their own experiment. The other cited papers in the constructed games section also use subsets of the generators without giving a reason why these generators were chosen.

To understand why researchers decided not to use all generators, we created games for all generators. Four generators: Congestion Game, Greedy Game, Guess Two Thirds Ave and the Simple Inspection Game did not return normal form games. To support these games the framework and some algorithms have to be modified, for this reason they are excluded from the comparison. During the test we noticed the implemented version of the Hawk and Dove game gives the same reward matrix as the Prisoner's Dilemma therefore it is removed, the Prisoners Dilemma and Chicken are also excluded because they create the same games as their N-player variants. We also found six random generators out of 28 too many of one type, therefore we removed two random chosen generators to get a better distribution of the game types: the Random Graphical Game and Two by two game. The 26 remaining generators are used in our tournament, the excluded generators are marked bold in Table A.4. Fifteen of the remaining generators create symmetric games, and the following groups can be distinguished within the 26 generators:

- Four random generators, which despite being random all sample from a normal distribution giving similar games.
- Four common interests generators, in these games both players benefit from playing the same action. The Coordination game shown in Table 4.3 is an example.
- Five conflicting interests generators, the Prisoner's Dilemma shown in Table 4.3 is an example. Both players prefer the reward when they confess while the opponent defect.
- Three constant sum generators, in a constant sum game the combined reward of the players is the same for each action pair.

## 3.3 Algorithms

Tournaments are organized to find the best algorithms for a scenario, making the algorithm selection and parameter tuning important design considerations. In this section the algorithms and parameter tuning methods used in previous comparisons are analyzed and a new parameter tuning method is described.

### 3.3.1 What algorithms to use?

Only a few open tournaments have been organized in MAL, mainly in the early days of MAL research (Axelrod and Hamilton 1981). The number of algorithms grew quickly, making all-against-all tournaments unfeasible. This raised the question: if not all then which algorithms should be used in the comparison? Using well-known algorithms is a common solution as researchers build on past work of colleagues (Airiau and Sen 2003), (Bouzy and Métivier 2010). Next to popularity, the performance in past tournaments and the complexity are important factors. New algorithms become more and more complicated, combining simpler algorithms (Crandall and Goodrich 2011) or using meta learners (Bouzy, Métivier, and Pellier 2018). Whilst more complicated algorithms generally perform better than simple algorithms, especially when a variety of games is used, they need more computation time and resources. Some algorithms learn from previous iterations, while others like Tit-for-Tat only need one iteration.

Bouzy and Métivier (2010) decide not to implement recent algorithms (from 2003-2007), instead they used older variants mainly because of the complexity of new algorithms. Combined with lack of pseudo code and short descriptions due to page limitation re-implementing modern algorithms, like M-qubed is a serious challenge (Crandall and Goodrich 2011). Zawadzki, Lipson, and Leyton-Brown (2014) also noticed the re-implementation problem and ask researchers to put their algorithms online for others to use, which will save a lot of (re-implementation) time and ensure correctness. Next to the performance and re-implement-ability there is an other important selection criteria discussed in the following subsection.

#### 3.3.1.1 Categories

Using variants of the same algorithm or algorithms that are based on the same theoretical properties only shows how well the algorithm performs against variants of itself (an extensive self-play test). Copies of itself are only a small part of the algorithms it will encounter in a real environment, to ensure the thoroughness of the tournament different types of algorithms should be used.

The theoretical properties of the algorithms can be used to categorize them. Crandall and Goodrich (2011) distinguish five categories: belief-based, reinforcement learning, gradient-ascent, no-regret and a satisficing learning algorithm. Zawadzki, Lipson, and Leyton-Brown (2014) group their algorithms in three categories and noticed the algorithms in the groups performed similarly. Extending this work results in a taxonomy of algorithms as made for the $2 \times 2$ games, to our knowledge this is only done for a set of the minimal information algorithms (Crandall, Ahmed, and Goodrich 2011).

Since we are not interested in the performance of a new learning algorithm we decided to use simple algorithms used in past comparisons to save time, just like Bouzy and Métivier (2010). Algorithms based on different concepts are preferred to avoid an extensive self-test. It is an arbitrary choice, but we decided to use ten algorithms in our comparison, of which we choose a maximum of two that are based on the same theoretical properties. This is comparable with recent works, Zawadzki, Lipson, and Leyton-Brown (2014) used eleven algorithms and Crandall (2014) used fifteen algorithms.

### 3.3.2 Parameter tuning

The parameters can make or break an algorithm, finding the optimal values is a challenge on its own. There are a few systematic ways to tune hyper parameters used in machine learning ("Hyperparameter optimization" n.d.):

- Grid search, usually there are too many values to try them all. Instead, values are tried based on past experience and intuition.
- Random search, trying random values in the hope of finding an optimal outcome.
- Baysian optimization and evolutionary optimization, while these give good results with the right hyper parameters, they have their own parameters that need tuning.
- Gradient-based learners, only find local optima and need a step size parameter.

Experience has taught us that smart parameter optimizers like evolutionary optimization only find good results when the parameters are set right and finding them is a study on its own, for this reason most researchers still use a grid or random search (Bergstra and Bengio 2012). Zawadzki, Lipson, and Leyton-Brown (2014) are one of the few who comment on the parameter optimization process: "Some initial experiments showed that the settings of the algorithm used in the paper performed very poorly, and so we used some hand-picked parameter settings that were more aggressive and seemed to perform better.". While we understand the lack of a better method, it feels arbitrary to hand pick parameters. In the following sections we introduce a new approach to find the optimal values. The parameters used in previous research are compared to the values of the new method to verify the new approach.

#### 3.3.2.1 The optimizer

Malherbe and Vayatis (2017) proved both mathematically and empirically that their optimizer, LIPO is better than random search in a number of non-trivial situations. The authors also compared LIPO to other algorithms like Bayesian optimization and showed it is competitive.

King (2017) showed that while LIPO is a good global optimizer it struggles to find the optimum (i.e. the top of the peak). To solve this King combined LIPO with a local searcher, BOBYQA (Powell 2009) and Powell's trust region method (TR) (Powell 2012) creating MaxLIPO + TR. King (2009) shows that MaxLIPO + TR outperforms the original LIPO and other optimizers. Furthermore, the function it tries to optimize may contain discontinuities, many local optima, and behave stochastically. It is also designed to use as few function evaluations as possible, making it ideal for our tuning challenge. The evaluation function and the parameter tuning of the algorithms are explained in the next chapter.

## 3.4 Result analysis

The previous sections described the tournament setup and design consideration, the analysis of the results is described in this section. A tournament creates too much data to analyze it without additional processing, some form of summarizing is required. There are various methods available which are categorized in three groups:

- The Grand table shows the summarized results of the round robin tournament, on which different analysis techniques can be applied. These are described in the first subsection.
- Performance measures are different criteria which can be used to determine the winner. A method used to visualize the results is shown and an alternative metric: regret, is introduced.
- Statistical analysis is used to test if the outcome occurred due to chance or is repeatable, and can be used to compare the results of algorithms.

All analysis and images in this section use the results of an example tournament in which four algorithms: Best response, Bully, Ngreedy, and TFT play two rounds on both a Random and a Prisoner's Dilemma game for 1.000 iterations, of which the last 100 are recorded.

The rewards of the game are different for each created instance, a reward of 50 might be the highest possible reward on one game whilst it is the lowest in another. Averaging or directly comparing these results gives misleading information. To make the results comparable they are normalized, using feature scaling, to bring the rewards of each game to the $0-1$ range. The scaler uses both the maximum reward the player could have earned in the game ($Rew_{max}$) and the minimal reward ($Rew_{min}$):

$$Rew_{norm} = \frac{Rew - Rew_{min}}{Rew_{max} - Rew_{min}} \tag{3.6}$$

### 3.4.1 Grand table

The results of a round robin tournament are visualized in a grand table. The table is created by first averaging the rewards earned during the recorded iterations of all rounds. The averages are then normalized and averaged over the games for each combination of algorithms. An example of the resulting grand table is shown in Table 3.4. The first reward in each cell of the table is the reward of the row algorithm, called the Partner, and the second reward is from the column algorithm, called the Actor. In representations with one value, the value represents the reward the Actor earned against the given partner.

Table 3.4: Example Grand table, the pure NE are underlined and the PO action pairs are shown bold

| | | *Actor* | | | |
|---|---|---|---|---|---|
| | | *Best response* | *Bully* | *Ngreedy* | *TFT* |
| *Partner* | *Best response* | 0.47/0.43 | **0.79/0.71** | **0.75/0.84** | 0.63/0.57 |
| | *Bully* | **0.79/0.71** | 0.54/0.59 | **0.75/0.84** | 0.47/0.43 |
| | *Ngreedy* | **0.84/0.58** | **0.79/0.71** | 0.67/0.68 | 0.59/0.53 |
| | *TFT* | 0.63/0.57 | 0.34/0.34 | 0.59/0.53 | 0.76/0.68 |

A grand table can be viewed as a game, in which each algorithm represents an action. A three player version of Table 3.4 has four times as many rows, one for each algorithm combination of player one and three. The grand table can not only be visualized like a game, it can also be analyzed like one. In following subsections the NE, PO analysis, and the implementation of the replicator dynamic are described.

#### 3.4.1.1 Game concept analysis

The Nash Equilibria of the grand table are the algorithm pairs in which both players play a best response to the opponent. Our four algorithm comparison translates to a four action game, finding all of the NE of this game is a costly operation. A game is also likely to have multiple mixed/pure NE. Whilst we only report pure NE, the framework also finds the mixed NE. The NE analysis can also be extended to include self-play, an ESS analysis, which is left for further work.

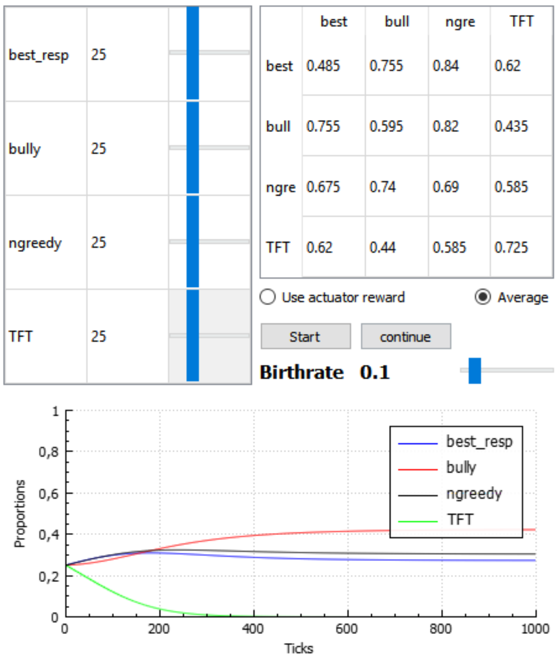
The Pareto optimal outcomes do not show if the algorithm played a PO action in all games, only that the reward of the algorithm combination was not pareto dominated by other combinations. In the Grand table shown in Table 3.4, Ngreedy played Pareto optimally against all algorithms and even a few Pareto optimal NE. Both the NE and PO analysis are used by meta learners: algorithms that compute which algorithm gives the highest expected reward against the given opponents. Modern meta learners use less computationally expensive methods to guarantee they play a not Pareto dominated NE (Powers and Shoham 2005a).

#### 3.4.1.2 Replicator dynamic

The replicator dynamic described in Section 3.1.3 comes with a few design choices. The first is the conversion from the grand table to fitness values. Removing the Partner's reward is a common solution. Whilst this works for symmetrical games, which are commonly used in evolutionary game theory, it introduces a bias on asymmetrical games. Instead, the reward the algorithms earned playing as row and column player against the same opponent are averaged to create a symmetrical fitness value. The replicator dynamic is extended with an adjustable birthrate ($\beta$) which allows us to change the speed of the dynamic, the updated formula is shown in EQ. 3.7. A lower birthrate results in a faster dynamic, we chose a value of 0.1.

$$\forall A : P_a(t+1) = \frac{P_a * (1 + \beta * F_a)}{1 + \beta * F_{tot}} \tag{3.7}$$

In some tournaments noise was added to simulate random errors made in real environments (Wu and Axelrod 1995). Noise is introduced into the preliminary round robin tournament by randomly replacing the players actions with random ones to simulate miscommunication. The downside of this approach is the increased stochasticity, the differences between rounds will become bigger, which is undesired. Therefore, we decided not to add noise. The initial proportions are set the same for all algorithms and we decided not to check if the stable state is an ESS, this can however easily be done by restarting the dynamic from the stable distribution after increasing the portion of one algorithm. If the distribution converges to the same proportions for each algorithm, the stable distribution is an ESS. Figure 3.2 shows the algorithms of the sample tournament converge in less than one thousand steps.



| | |
|---|---|
| best_resp | 25 |
| bully | 25 |
| ngreedy | 25 |
| TFT | 25 |

| | best | bull | ngre | TFT |
|---|---|---|---|---|
| best | 0.485 | 0.755 | 0.84 | 0.62 |
| bull | 0.755 | 0.595 | 0.82 | 0.435 |
| ngre | 0.675 | 0.74 | 0.69 | 0.585 |
| TFT | 0.62 | 0.44 | 0.585 | 0.725 |

Use actuator reward [✓ Average]

Birthrate 0.1

Figure 3.2: Replicator dynamic run on the example grand table show in Table 3.4

### 3.4.2 Performance measures

Two performance measures are used in the studied papers: the average reward and regret. Next to these, an example is given of how the number of wins of the algorithms can be used to determine the winner.

#### 3.4.2.1 Average Reward

Whilst a grand table gives a nice overview it is not easy to interpret. The results can also be visualized though heatplots, in which the rewards are indicated with colors. The downside of heatplots is that only one value can be shown: the Actor's or Partner's reward. Therefore, we decided to average the rewards as done for the replicator dynamic.

The results are averaged over both the generators in Figure 3.3, and players in Figure 3.4. Figure 3.4 shows all algorithms scored similarly on the Prisoner's Dilemma and that Ngreedy earned its high rewards in the random game. Figure 3.3 shows against which algorithms Ngreedy earned its high reward. Ngreedy was the only algorihm that earned 'red' rewards, which makes it unsurprising that it earned the highest average reward in Figure 3.5.

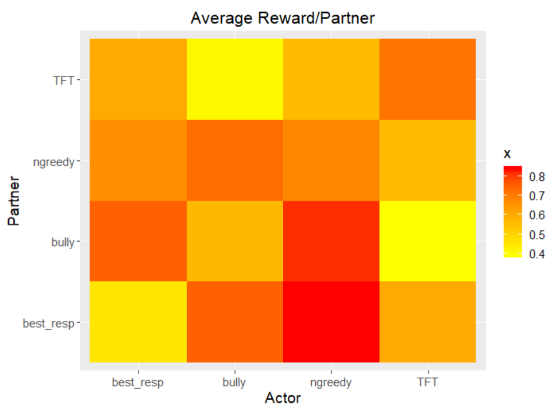


Figure 3.3: Heatplot of the average reward of the Actors against the Partners

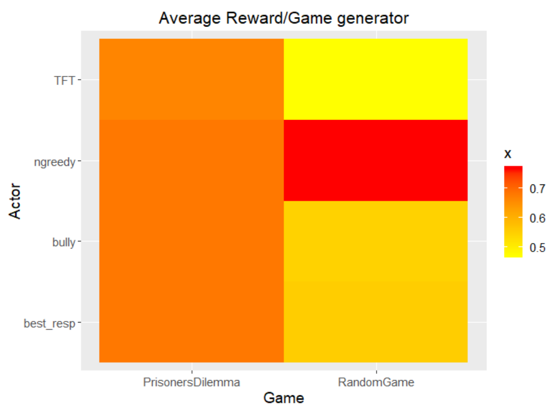


Figure 3.4: Heatplot of the average reward of the Actor's on each gametype

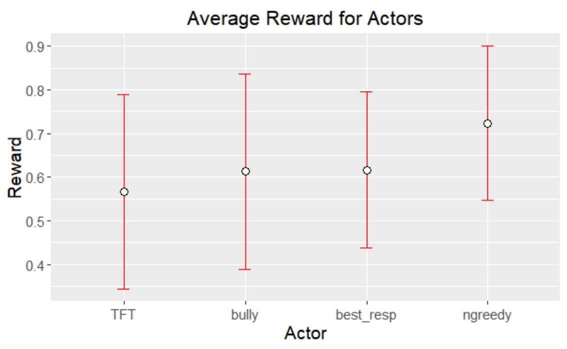


Figure 3.5: Error plot of mean Actor reward with standard deviation

#### 3.4.2.2 Average Regret

Regret is the difference between the reward the algorithm could have earned by playing its best action, given the opponents action, and the reward it earned. The following formula is used to calculate the accumulated regret of an algorithm over all iterations (t):

$$Regr_{acc} = \sum_{t=1}^{T} \operatorname*{Max}_{a \in A} R_t(a, a_{-i}) - R_t(a_i, a_{-i}) \tag{3.8}$$

Regret is used to show that the algorithm played the best possible action, making one big assumption: that the opponent would have played the same if the algorithm played differently. As a result ignoring regret can lead to higher accumulated rewards in some games, whilst it gives insight in other games, like the Prisoner's Dilemma, as shown in Figure 3.7. Most algorithms played the NE but if both players had cooperate instead of trying to exploit, they would have gotten a higher average reward and accumulated more regret.

Regret is normalized with different values than used for the rewards, instead of the minimal reward the lowest amount of regret ($Regr_{min}$) an algorithm can have is zero. The maximal amount of regret ($Regr_{max}$) is the biggest difference between the maximum reward and minimum reward for one of possible opponent actions. The resulting normalization formula is shown in EQ. 3.9. Regret earned against the partners, games, and the accumulated regret are shown in Figure 3.6, 3.7, and 3.8. The scale of the regret plots is inverted as less regret is better.

$$Regr_{norm} = \frac{Regr}{Regr_{max}} \tag{3.9}$$

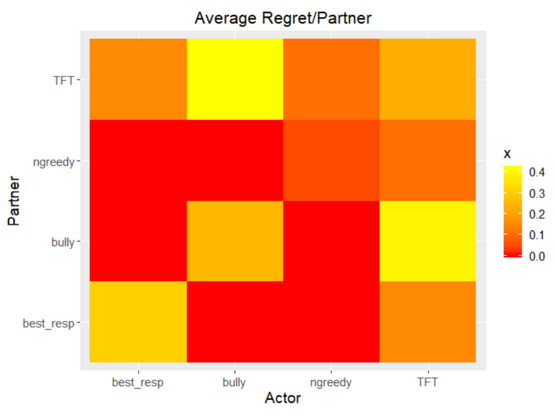


Figure 3.6: Heatplot of the average regret of the Actors against the Partners

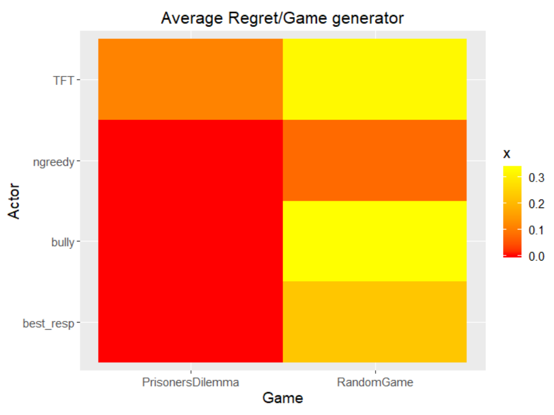


Figure 3.7: Heatplot of the average regret of the Actors on each gametype

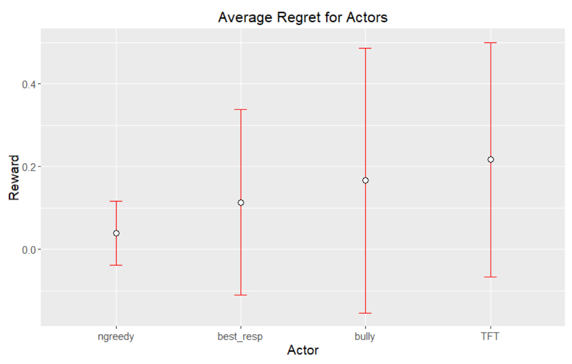


Figure 3.8: Error plot of mean Actor regret with standard deviation

#### 3.4.2.3 Summary

As described in the tournament section we also look at the number of wins, draws and loses of the algorithms. The self-play comparisons are removed and the games in which the algorithms played as partner and actor are analyzed separately to show the influence of the games asymmetry. A win is awarded two points, a draw with one, and a loss with zero points. The resulting scores are shown in Table 3.5. Both the average reward and the number of wins is higher for Ngeedy, showing that it won most comparisons. It is also interesting that TFT scored the highest average reward in self-play, showing the opponents have room for improvement.

Table 3.5: Mean rewards and number of wins earned as Partner and Actor

| | *Best response* | *Bully* | *Ngreedy* | *TFT* |
|---|---|---|---|---|
| *Average reward as Partner* | 0.66 | 0.64 | 0.72 | 0.58 |
| *Score as Partner* | 7 | 7 | 9 | 8 |
| *Average reward as Actor* | 0.57 | 0.59 | 0.72 | 0.55 |
| *Score as Actor* | 3 | 4 | 7 | 3 |
| *Average reward in self-play* | 0.45 | 0.57 | 0.68 | 0.72 |

### 3.4.3 Statistical tests

Only a few of the studied papers use statistical tests, Crandall and Goodrich (2011) make pairwise statistical comparisons without mentioning which test is used. We expect it is a pairwise t-test which one of the authors also uses in a later research (Crandall 2014). Zawadzki, Lipson, and Leyton-Brown (2014) used various statistical tests which are covered in the following subsections. In the remainder of this section the underlying assumptions of the pairwise tests are analyzed.

One assumption of commonly used tests like the ANOVA and t-test is the normality assumption. We expect that both the grand table and the rewards algorithms earned during the recorded iterations are not normally distributed because the algorithms compete for the highest rewards. The rewards of most game instances are also not normally distributed, normalizing these will not change the normality of the results. For example, in the Prisoners Dilemma most algorithm combinations end up playing the NE, the other actions pairs are played less often giving skewed results. The normality of the grand table is tested with the Shapiro Wilkinson test of normality (Shapiro and Wilk 1965). The test results indicate that the grand table is normally distributed (p-value of 0.5), as also shown in Figure 3.9. However, normality tests of the individual games show that the rewards of the Prisoners Dilemma are not normally distributed (p-value of 4.553e-08), while the rewards of the Random game are normal distributed (p-value of 0.4348). In this example the results off the Random game compensated for the results off the Prisoner's Dilemma showing that the normality of the grand table, results in general, depends on both the games that are played and the algorithms. Therefore, normality cannot be assumed and we decided to use tests without a normality assumption.

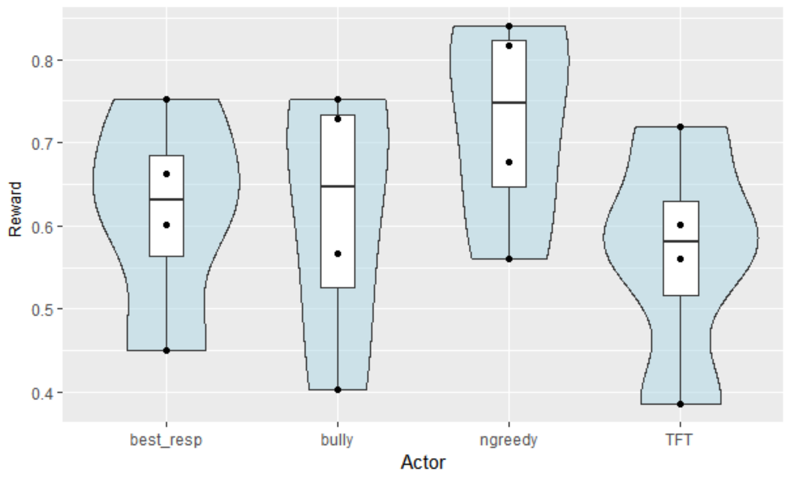


Figure 3.9: Beanplot of rewards the algorithms earned against Best response.

The grand table can be analyzed as an repeated measure experiment in which each algorithm (Actor) plays against the same opponents. Usually a global test is used to test if one of the columns differs significantly from the others to avoid excessive post-hoc testing, in an attempt to find a significant result. In our case the statistical question is: which Actor scored significantly higher. Showing that it significantly outperforms that opponent. The global test does not answer this question, therefore it is debatable whether it should be used. Furthermore, the post-hoc tests that comes with the global test only tell which pairs of Actors differ significantly, not which Actor scored significantly higher. For these reasons we decided not to use the global Friedman test, an alternative for the repeated measures one way ANOVA without normality assumption.

#### 3.4.3.1 Paired test

The Wilcox signed rank test is an alternative for the paired t-test that does not assume the data is normally distributed, as a result it has less statistical power which can give problems on small sample sizes. Because no distribution is assumed there is a fifty percent chance the difference between two paired samples is positive/negative. With the sample size (N) of ten used in our main tournament the chance that all pairs have a positive/negative differences is: $\frac{1}{10^2} = 0.01$. As a result the p-value cannot become smaller than 0.01, the default rejection threshold for the null hypothesis ($\alpha$) is 0.05, so the hypothesis can still rejected. For the example tournament with four algorithms the lowest possible p-value is 0.0625 and the null hypothesis cannot be rejected. Furthermore, the Wilcox signed rank test assumes the data is distributed symmetrically around the median. Figure 3.9 shows this assumption is violated. Voraprateep (2013) showed that the Wilcox signed rank test loses significant power if this assumption is violated, they use a transformation as solution.

The Sign test does not use this assumption, as a result it has less statistical power than the Wilcox signed rank test. The Sign test ignores the magnitude of the difference between the paired samples, it simply marks the paired samples as higher or lower and uses these counts to determine if there is a significant difference.

The Wilcox method with transformation is relatively new, published in 2013 and there are no follow-up papers in which it is compared to the Sign test. We decided to use the Sign test because we are not sure how much statistical power is lost by transforming the data. The last design consideration is the compensation method: Each test has a chance of giving a false positive, the $\alpha$ level of 0.05 compensates for this. When multiple tests use the same data the chance of getting a false positive increases, the Holm-Bonferroni method can be used to control the $\alpha$ value. The downside of a control method is that it lowers the $\alpha$ value of the individual comparisons, combined with the lower statistical power of the Sign test it might be too conservative. Whether a compensation should be used depends on the application, as a result it is a topic of many discussions. Especially our scenario in which different columns are compared in each test to answer different hypothesis. The probability that column B is greater than column A is one minus the probability that column A is greater than B. Comparing the column with itself does not provide information, so each column of our ten algorithm tournament is used in nine comparisons. For this reason we decided to compensate for nine comparisons.

#### 3.4.3.2 Convergence test

Our convergence test is a combination of the convergence and probabilistic domination test used by Zawadzki, Lipson, and Leyton-Brown (2014). Both tests use a plot of the accumulated rewards of the recorded iterations, algorithm A probabilistic dominates algorithm B if its accumulated reward is above the accumulated reward of algorithm B for all recorded iterations. Though probabilistic domination the algorithm shows it did not only earn a higher average reward, but was ahead of its opponent the entire time. The convergence test is used to test if the algorithms have converged to stationary play, this is done by doubling the accumulated reward half-way the recorded iterations and comparing the result to the accumulated reward after all iterations. If the difference is smaller than a given threshold the distribution has converged. Where the rewards of the games are normalized to a given range by Gamut the values differ from instance to instance, this means a fixed threshold as used by the authors has a different relative size on each instance. To account for the different reward distributions the average reward of the game is used as threshold.

Using the recorded iterations the accumulated maxmin reward of each player is calculated as benchmark. The algorithms should never earn a reward below their maxmin reward. Combined with the accumulated payoffs and the rewards of the game the maxmin rewards can be used to tell if the algorithm was exploited or exploiting, giving additional insight in the learning process. An example plot is shown in Figure 3.10. Average regrets plots are created in the same way. An algorithm without regret keeps it accumulated regret at zero, making the horizontal-axis the best possible outcome. Figure 3.11 shows that both Best response and Bully played the best they could given the opponents action.

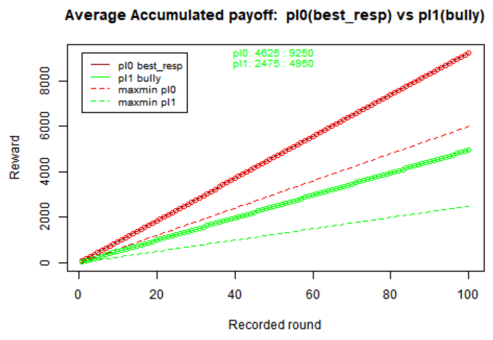


Figure 3.10: Accumulated payoff plot with convergence check, the rewards of both players after half and all the recorded iterations is shown in green if they have converged.

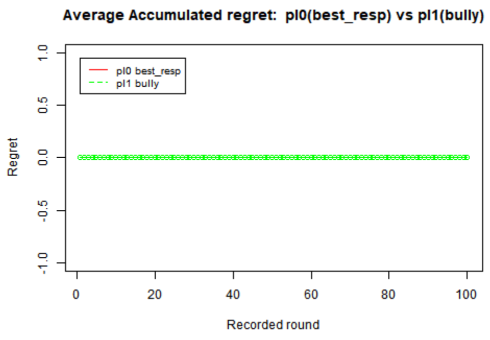


Figure 3.11: Accumulated regret plot

The four algorithms, two games, and two rounds tournament results in 64 graphs, after averaging over the rounds there are still 16 graphs for each game. A bigger tournament will create too many graphs to sort through and information is lost by averaging over the rounds. To make this information easier to analyze without averaging, the values are stored in a list. A similar list is created for the regret plots.

#### 3.4.3.3 Similarity tests

The Kolmogorov Smirnov (KS) test and Spearman's rank correlation test are used by Zawadzki, Lipson, and Leyton-Brown (2014) to compare results. They use the KS test to test if two accumulated reward distributions are the same or just look similar. The KS test is commonly used test for these comparisons because it is non-parametric, does not assume that the data is based on a known distribution. On top of this, the KS test sensitive to both the shape of the curve and the values. It is worth mentioning that due to the different rewards the column and row player get on asymmetric games the distributions usually differ. Comparing the rewards both algorithms earned playing as row player against each other solves this problem. If these are the same, the accumulated rewards earned as column player are likely to be the same as well.

The authors used the Spearman's rank correlation test to test if the number of actions and the average rewards of the algorithms are monotonically related. The test is non-parametric just like the KS test, the difference is that it tests if two variables are monotonically related instead the same. Variables that are monotonic related increase/decrease at the same time, but do not have to increase with the same proportion as required in a linear relation.

Both the KS and Spearman's rank correlation test are used to test specific hypothesis. The KS test to check for similarity and the Spearman's rank correlation test to find relations between variables.

### 3.4.4 What analysis to use?

We showed various analysis methods in this section, the grand table only gives the aggregated rewards losing a lot of information. But the PO, NE analysis, and replicator dynamic give more insight into how algorithm will perform in the real world. However, they say little about who wins the tournament and the grand table takes up more page space than the heatplots, as shown in the next chapter. For these reasons we would not publish the grand table despite the clarity it provides. The metrics are easier to interpret and additional information can be added to the heatmap by placing a symbols in the squares, for example to show the results of a statistical test. The convergence tests use the unaveraged results which makes them ideal for analysis, but there are too many of them. Therefore, we decided to create the table with the results of all rounds and make graphs for the cases which seem interesting. The replicator dynamic is run for 1,000 iterations as done in the example tournament.

We decided to analyze the convergence tests and discuss the interesting findings. The grand table and metrics are published along with the results of the paired tests. The paired test can be applied to the grand table like we showed, but can also be applied to the less aggregated data used for the metrics: the rewards the Actor and Partner earned playing against each other. This method has two advantages over the grand table. First, different data is used in each comparison so there is no need for a correction. Secondly, algorithms are usually compared on multiple games and rounds resulting in more samples than available in the grand table. The similarity tests are not used because we do not have a hypothesis.

#### 3.4.4.1 More than two players

Each player can be seen as a dimension. Where three dimensions can still be captured in a graph, four or more become a problem. The alternative is averaging the results. The results of a two player tournament can be averaged to one dimension: A versus B can be averaged with B versus A as done to create the rewards used by the replicator dynamic. For three players this method no longer works because the grand table becomes rectangular, as illustrated in Table 3.6. Player three becomes the second row player. As a result the heatplots and replicator dynamic become a challenge.

Table 3.6: Example Grand table of a three player game played by two algorithms.

| | | | *Actor* | |
|---|---|---|---|---|
| | | | *one* | *two* |
| *Partners* | *one* | *one* | ./. | ./. |
| | *one* | *two* | ./. | ./. |
| | *two* | *one* | ./. | ./. |
| | *two* | *two* | ./. | ./. |

Vu, Powers, and Shoham (2006) used three player games, they avoided the large number of possible algorithm combinations and results by looking at special cases: three player self-play and two instances of one algorithm against another showing only the averaged results.

# 4 Sample tournament

Describing how a tournament can be organized and organizing a tournament are two different things, therefore we also run an example tournament to put our methodology to the test. The following sections describe the platform, tournament settings, game creation, and the algorithms that are used. After the parameter tuning the tournament is run, the results are shown in the last section.

## 4.1 Platform

In this section we describe the tools we found and explain the main components of our framework. The literature and web search resulted in the following tools:

- **Gambit**, a tool to find the Nash equilibria of games (McKelvey, McLennan, and Turocy 2006). Gambit is still being developed further by the community, the latest release is from 15 May 2017.
- **Gamut**, a suite containing 35 game generators which can be used to create games (Nudelman et al. 2004). The latest version of Gamut dates back to 2004, but the games have not changed in the meantime.
- **MALT**, a complete MAL experiment framework with an interface for Gambit and Gamut. The creators, who argue for publishing test environments and algorithms, put a lot of work into making an example of MALT: "We have worked hard to make MALT easily extensible. For example, adding a new algorithm to the MALT GUI is as simple as providing a text file with a list of parameters, and adding an algorithm to the engine requires very little coding beyond the implementation of the algorithm itself." (Zawadzki, Lipson, and Leyton-Brown 2014). The downsides of MALT are that it comes without an installation manual and that it was created for Matlab version R14 SP1, released on October 2004 ("MATLAB release history" n.d.).

After downloading and testing the tools we estimated that modifying the MALT framework would take too much time. Despite this, the implementation off the algorithms provided additional insight beyond the explanation found in research papers. Gambit is used to find the NE as done in MALT and Gamut is used for the game creation, as described in the previous chapter.

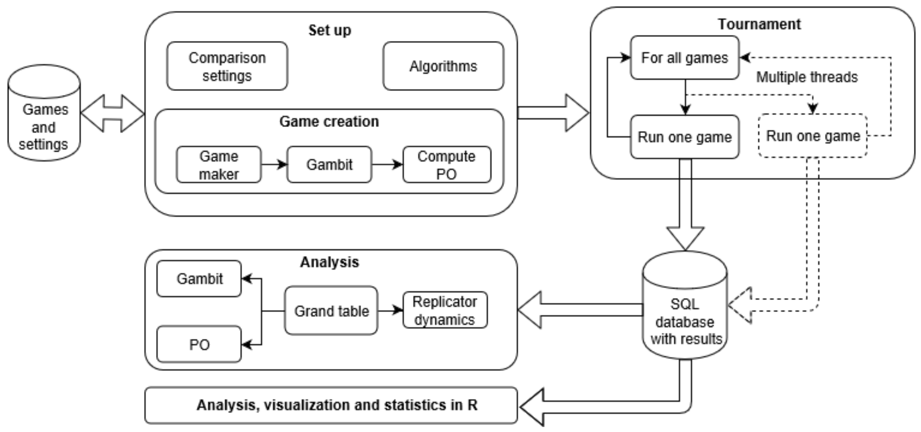


Figure 4.1: Flowchart of of the framework from setup to analysis.

### 4.1.1 Own platform

Just as we don't try to give the optimal methodology, we did not try to write the optimal framework, there was not enough time to do this. We did create a modular framework which can easily be modified and reused. The framework has four main components, connected though a database, as shown in Figure 4.1. The first component handles the setup of the tournament. The second module runs the tournament and stores the results in a SQL database. The third module performs some simple result analysis within the framework and module four does the advanced analysis in R. The code and settings used for our experiments can be found here: *www.github.com/autimator/MAL-framework*.

## 4.2 Tournament settings

The number of rounds, iterations, and recorded iterations have to be chosen for the round robin tournament. The values used in previous tournaments variate from a few to many rounds and iterations as shown in Table 3.2. Since there are no computer clusters available and the main focus is the methodology the values are kept low, we decided to play five rounds on each game. The number of iterations is set to 10,000 of which the last 1,000 are recorded. This should be enough iterations for the relative simple learning algorithms, to be sure we tested if the algorithms converge to stationary play within 9,000 iterations.

Lastly the number of actions and players have to be chosen. As it is unclear how to analyze N-player games, we used only two player games. The number of actions is varied: two instances are created with two actions. For the generators that allow more actions three more instances are created: one with three, one with four and one with five actions. The Rock Paper Scissors and the Shapleys Game can only be played with two players and three actions, therefore one instance is created of both generators.

## 4.3 Game creation

Some generators have additional parameters that can be tuned. Since we are not interested in the performance on specific games, we decided not to tune the generators parameters. Instead, the random values of Gamut are used, in the user documentation the authors warn that the Location Game might give strange matrices with random parameters. One of the players got the maximal reward and the other got a zero reward for all actions in the created instances, for this reason the location game is removed.

Besides the generators specific parameters, the number of actions can be set different for each player and the rewards can be normalized. There are two drawbacks to setting the number of actions different for each player, the first is that some games are only symmetric when the number of actions is the same for each player. The second is that it makes algorithms which use the past actions become more complicated. It is also doubtful if different numbers of actions make the learning environment more complicated than when both algorithms have the same number of actions. Because of these reasons we decided to keep the number of actions the same for all algorithms.

The rewards are stored as integers to save memory, during a comparison the rewards of all recorded iterations are kept in memory. Using floating-points with the same range gives no benefit. And since zero sum games are strategically equivalent to general sum games, there is no reason to use negative payoffs. We decided to normalize all payoffs to the range of 0-100, this gives algorithms that use expected rewards room to work, while avoiding overflows (and zero divisions) for algorithms that use the accumulated rewards.

## 4.4 Algorithms

In this section the algorithms are introduced and tuned. One of our requirements was the ability to test on N-player games, for this reason mostly N-player algorithms are implemented. As a result of misinterpretation we also wrote an N-player variant of a two player algorithm which is compared to the original in the tuning section.

The algorithms are split in three categories which are divided into subgroups groups based on their underlying theoretical properties:

- Stationary algorithms, these algorithms play the same action in each iteration or use a response rule to react to the opponents action:
  - The Nash players play one of the NE of the game.
  - Tit for tat, a simple but surprisingly effective response rule.
  - Benchmark algorithms, these algorithms are used in other more complicated algorithms, but they also participated in previous tournaments.
  - Pareto optimal plays a PO action of the game.
- Learning algorithms, some learning algorithms create a model of the opponents strategy and use it to exploit the opponent, others model the average rewards of the actions:
  - Fictitious play.
  - No-regret.
  - Markov.
  - Reinforcement learners, QL.
  - Satisficing play.
  - Greedy algorithms, the greedy learners are primarily interested in maximizing their own reward: exploiting the opponent. Some of the iterations are used for exploration in an attempt to find actions with a higher reward.
  - Teachers try to force the opponent to play the action they want, they do this persistently creating the impression they are stationary algorithms.
- Random does not try to learn but is also not playing a stationary strategy or response rule, it plays random.

The algorithms are explained in the following subsections.

##### 4.4.0.1 Random

**Random** is a classical algorithm used as benchmark in some papers, it shows how an algorithm that is not trying to win performs (Airiau and Sen 2003), (Zawadzki, Lipson, and Leyton-Brown 2014). Next to this it gives insight in how algorithms respond when they meet a non-rational opponent whose strategy is too complicated to model. A disadvantage of using random is that no algorithm achieves high rewards against it, making all players look worse. However, it is a big assumption to expect all the opponents try to maximize their rewards, play rational, as this does not always lead to the highest rewards. For these reasons we decided to include a random player.

##### 4.4.0.2 Nash players

**Nash and Determined** are two NE players. Nash plays his part of the NE every iteration (Airiau and Sen 2003). Each game has at least one pure or mixed NE, in the case of multiple NE Nash randomly picks one and keeps playing his part of this NE. Determined plays its part of the NE with the highest personal reward. If two or more NE result in the same reward Zawadzki, Lipson, and Leyton-Brown (2014) decided to play the NE with the highest opponent reward. This solution works in their two player N-action games, but with N-players there are situations in which one opponent would be better off in one NE and another in a different NE. Therefore, we implemented a different solution: only accept a new NE if it leads to a higher reward for all opponents, otherwise the first found NE is played. This is similar to a random choice because Gambit returns the NE in random order.

Although both NE-players are stationary they do not play the same action each iteration, in case of a mixed NE they use a probabilistic choice. The computational costs of finding the NE, which becomes significant on bigger games, is ignored. We encountered no problems, despite this finding all NE of big games might take too long. As a solution different algorithms of Gambit can be used to stop after a NE is found or only search for the pure NE. This will probably influence the performance of the MAL algorithms.

#### 4.4.0.3 Tit for tat

**Tit for tat (TFT)** is a classic algorithm that won the first big comparison on the Prisoner's Dilemma (Axelrod and Hamilton 1981). TFT plays the action the opponent played in the previous iteration. In the case of multiple opponents a random opponent is mimicked to stay close to the original algorithm.

#### 4.4.0.4 Benchmark algorithms

Both **Best response and Maxmin** are used in other algorithms, Maxmin is used by Godfather and Best response by Bully and Fictitious play. The algorithms were implemented separately for testing purposes and kept as players (Airiau, Saha, and Sen 2004). Best response is similar to Tit for Tat, instead of playing the action the opponents played in the last round it plays the best response to the opponent(s) action(s). In the first iteration the opponents action is not available, therefore Best response opens with a random action. Maxmin on the contrary performs mainly as a reference like random, it assumes the opponents tries to minimize its reward and responds by playing the action with the highest minimum reward (EQ. 2.1).

#### 4.4.0.5 Pareto optimal

Playing a Best response or playing your part of a NE are concepts that stand in for high rewards, which works best depends on the game and opponent. There are also algorithms that only use the rewards. **Pareto optimal (PO)** is one of them. Although Zawadzki, Lipson, and Leyton-Brown (2014) do not mentioned it in their paper, their framework contains a PO algorithm which plays its part of the PO profile that results in the highest reward: the competitive PO outcome. However, most algorithms do not allow exploitation, therefore we also implemented a cooperative version. The cooperative PO algorithm plays its part of the PO profile that gives both players the highest reward, as described in Chapter 2. In case of a tie the profile with the highest personal reward is played.

#### 4.4.0.6 Fictitious play

**Fictitious player** (FP) assumes the opponent is playing a stationary, potentially mixed strategy and uses the opponent's empirical action distribution to estimate its next move (Fudenberg et al. 1998). This translates to playing the best response to the action the opponent played most often. The initial action counts are set to '0', the first action is chosen random and a random choice is used as tiebreaker when an opponent's action counts are the same.

#### 4.4.0.7 No-regret

**No-regret** tries to minimize its regret (Young 2004). After each round the regret of the played action is increased with the maximal reward the algorithm could have earned, given the opponents action, minus the reward it earned. The action with the lowest accumulated regret is played in the next iteration. Similar to FP the initial regret is set to '0' and a random choice is used in the first iteration and as tiebreaker.

#### 4.4.0.8 Markov

**Markov** is introduced in a bachelor thesis of a student at Utrecht university (Haan 2018). The algorithm is explained through an example two player, two action game in which Markov is the first player. The window size, a parameter of the algorithm, is set to three. This means the last three action pairs are combined to form a state, as shown in Table 4.1. The state is multiplied with the corresponding powers of two for a two action game, summing the results gives the actual state: 30 for the example window shown in Table 4.1.

Table 4.1: An example window consisting of the actions of both players of the last three iterations.

| *Round* | 1 | 2 | 3 |
|---|---|---|---|
| *Window* | 01 | 11 | 11 |

Increasing the occurrence count of state 30 by one completes the updating process. The next action is selected by first removing the oldest action pair in the window and then adding all possible action combinations. The new windows are the potential next states, the occurrence counts of these new states are looked up in the list of states and the reward of the action pairs is looked up in the game matrix as shown in Figure 4.2.

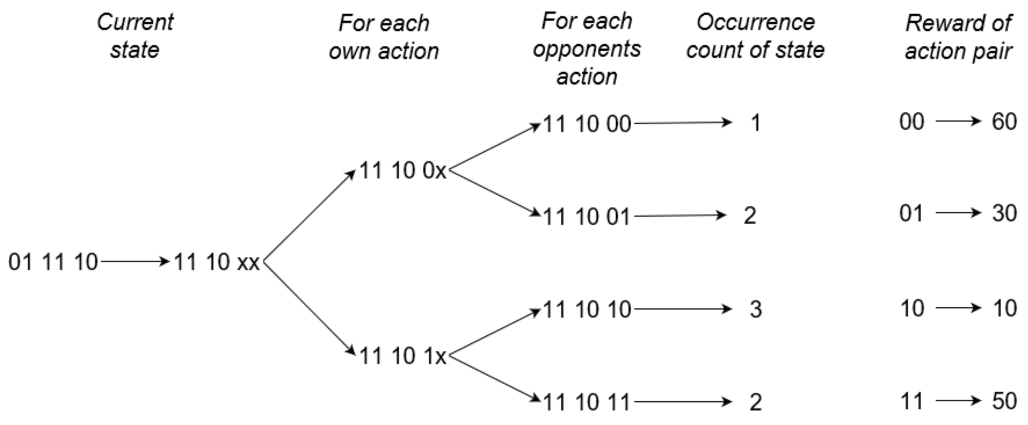


Figure 4.2: Markov action selection

The occurrence counts are converted to transition probabilities and multiplied with the rewards to calculate the expected reward of each action:

- Expected reward for playing '0' = $\frac{1}{1+2} * 60 + \frac{2}{1+2} * 30 = 40$.
- Expected reward for playing '1' = $\frac{3}{3+2} * 10 + \frac{2}{3+2} * 50 = 26$.

The action with the highest expected reward, in this example action '0', is played.

Setting the windows size is difficult: a bigger window gives a higher prediction precision at the expense of more states and more training iterations. The number of states is calculated with the following formula:

$$\text{Number of states} = \text{number of actions}^{\text{number of players * window size}} \tag{4.1}$$

The number of learning iterations is set to 9,000, this means that theoretically even for a window size of three not all 15,625 states will be played in a two player five action game. In practice, some combinations will not be played at all. It is even possible that only one state is played throughout the game. To test how many states are created Markov is set up against Random, the algorithm that is likely to visit most states. The algorithms compete on ten random game instances for 10,000 iterations, as done in the real tournament. Table 4.2 shows the number of states and average reward of Markov after the tournament. The results show a bigger window does not necessarily lead to higher rewards. The test also showed that Markov is most likely to switch action if the actions of the game return similar rewards. As a result it got more states playing against itself in game four then it did against Random, this cell is marked with a '*'. Finding the optimal window size is left to the parameter tuner. To avoid zero divisions the initial counts of all states are set to '1' and the initial windows is filled with zero's. The occurrence count of the initial window is not increased to avoid a bias.

Table 4.2: The number of states and the rewards of the Markov player

| *Game number* | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* | *9* | *10* |
|---|---|---|---|---|---|---|---|---|---|---|
| *States (window size 3)* | 24 | 23 | 42 | 41 | 64 | 8 | 43 | 10 | 21 | 45 |
| *Row player reward* | 74.85 | 62.12 | 83.57 | 50.43 | 55.63 | 77.66 | 48.07 | 84.31 | 66.14 | 94.32 |
| *Column player reward* | 51.40 | 82.31 | 17.93 | 82.88 | 52.13 | 59.56 | 49.99 | 70.74 | 46.84 | 40.38 |
| *States (window size 5)* | 436 | 257 | 458 | 776 | 911* | 32 | 565 | 36 | 179 | 598 |
| *Row player reward* | 74.29 | 61.46 | 83.57 | 49.60 | 55.63 | 77.66 | 48.07 | 84.31 | 66.14 | 94.32 |
| *Column player reward* | 51.40 | 82.31 | 17.70 | 82.88 | 50.70 | 59.56 | 49.79 | 70.74 | 46.89 | 39.6 |

#### 4.4.0.9 QL

The **stateless Q-Learner** is used by Littman and Stone (2001) and Zawadzki, Lipson, and Leyton-Brown (2014). The algorithm was originally designed for a changing environment, our games do not change during the round which means the Q-learner only needs one state. This simplifies the Q-value update formula with learn rate ($\alpha$), reward (R), and discount rate ($\lambda$) to:

$$Q(a) = (1 - \alpha t) * Q(a) + \alpha t * (R + \lambda * 1) \tag{4.2}$$

Each iteration the Q-learner used in Zawadzki, Lipson, and Leyton-Brown (2014) has a decaying chance of playing a random action: the exploration rate. The learning rate also decays over time. Iterations in which the algorithm is not exploring, it plays the action with the highest Q-value. While fairly simple the algorithm has five parameters that have to be tuned: the learning rate, decay of the learning rate, the exploration rate, decay of the exploration rate and the discount rate. The initial Q values are set to '0'.

#### 4.4.0.10 Satisficing play

**Satisficing play** (SAT) is a variation of reinforcement learning for situation in which the opponents actions and payoffs are not observable (Karandikar et al. 1998). SAT starts with an initial aspiration level and a random chosen action, it keeps playing the same action as long it returns a reward above the aspiration level. When the reward drops below the aspiration level SAT switches to one of its other actions. Each iteration the aspiration level ($\alpha$) is updated using the persistence rate ($\lambda$):

$$\alpha(t+1) = \lambda * \alpha(t) + (1 - \lambda) * R(t) \tag{4.3}$$

Both the initial aspiration level and persistence rate need tuning. The optimal initial aspiration level probably lies near the highest reward of the game. If a value below the lowest reward action is chosen the algorithms will keep playing its initial action, which might be sub optimal. The persistence rate is harder to tune, any value between 0 and 1 can be the optimal value.

#### 4.4.0.11 Greedy algorithms

**$\epsilon$-greedy** is a simple greedy algorithm (Sutton and Barto 2011). After each iteration the algorithm increases the total reward it earned by playing action 'x' with the reward it earned and increases the number of times it played action 'x' by one. Each iteration the algorithm has a chance ($\epsilon$) of playing a random action, exploring. The iterations $\epsilon$-greedy is not exploring it exploits by playing the action with the highest average reward:

$$\text{Action(t)} = \underset{x}{\text{argmax}} \frac{\text{Sum of the rewards when x was played prior to t}}{\text{Number of times x was played prior to t}} \tag{4.4}$$

To initialize the algorithm each action is played at least once, the tuning of the exploration rate is described in the next section.

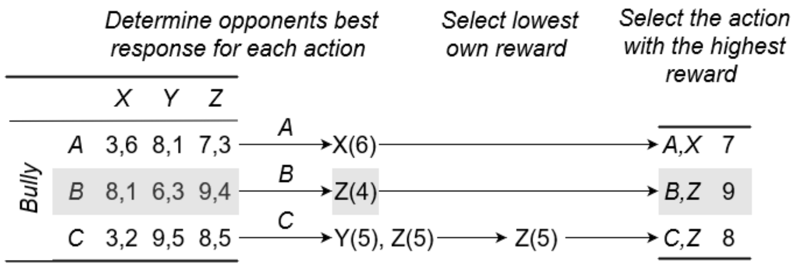
**N-greedy** is based on the same concept, the only difference is the decreasing exploration rate used by N-greedy:

$$\text{Exploration rate} = \frac{\text{Number of actions}}{\text{Current iteration}} \tag{4.5}$$

N-greedy also plays each action at least once to initialize the algorithm, this is done because the exploration rate is bigger than one as long as the current iteration is lower than the number of actions of the game. After the initialization the exploration rate decreases rapidly, after 100 iterations in a two action game it is only: $\frac{2}{100}$. We expect N-greedy will perform better than $\epsilon$-greedy because it exploits more.

#### 4.4.0.12 Teachers

Both **Bully and Godfather** are created by Littman and Stone (2001). After reading the paper we thought Bully assumes all opponents play a best response to the past action and Bully plays the best response to all the opponents best responses. Instead, Bully is a stationary strategy. Before the first iteration it calculates which action all its opponents would play as response to its action, assuming they are best response players. Bully then plays the action that results in the highest reward. An example is given in Figure 4.3, Bully player is the row player and action 'B' is the action with the highest reward played persistently by Bully.



Determine opponents best response for each action
Select lowest own reward
Select the action with the highest reward
Bully
X Y Z
A 3,6 8,1 7,3
B 8,1 6,3 9,4
C 3,2 9,5 8,5
A → X(6) → A,X 7
B → Z(4) → B,Z 9
C → Y(5), Z(5) → Z(5) → C,Z 8

Figure 4.3: Bully action selection

The static version works in two player N-action games, but when the number of players is increased there are multiple interactions: assuming Bully is player one it can model the effect it has on player two and three, but the effect player two and three have on each other complicates the prediction. The dynamic version of Bully adapts to actions played by other players, avoiding the complicated model. As a result the dynamic Bully might end up playing a different action than the static version. Despite this we expect that both versions earn similar rewards, this theory is put to the test in the tuning section.

The last algorithm, **Godfather** uses a different approach. Godfather plays its part of a targetable pair, an action pair that rewards both players with at least their maxmin value, as long as the opponent does. If the opponent plays a different action Godfather punishes it by playing his part of the opponents maxmin reward. Godfather only works in a two player setting, when the number of players is increased only the player which is not playing its part of the targetable pair should be punished instead of all opponents. Similarly, who to punish if multiple opponents deviate? For this reason Godfather is not participating in the tournament, but it is used as training algorithm for the parameter tuning.

If there are multiple targetable pairs Godfather uses a random choice as tiebreaker, the downside of this approach is the chance of miscommunication in self-play. A solution to this would be playing the targetable pair that gives both players the highest reward, this solution is not implemented because it does not influence the parameter tuning.

### 4.4.1 Parameter tuning

As far as we known parameter tuners have not been used in MAL before, therefore the parameters it finds are compared to those of previous experiments to verify the tuning process. In the first subsection the evaluation function is described and in the second subsection the parameters are tuned.

#### 4.4.1.1 The evaluation function

While we said we would use ten algorithms, we introduced sixteen algorithms in the previous section. The six additional algorithms are used to train the algorithms with parameters on a subset of the games, this is done to avoid unfair advantages: If the same algorithms and games are used the algorithms get a chance to learn over the various games through the parameter optimization, which gives them an advantage over algorithms without parameters.

A small tournament is used as evaluation function, each round is played for 10,000 iterations of which the last 1,000 are recorded as done in the real tournament. This does not give an advantage because the other algorithms also get 9,000 burn-in iterations. One game generator is chosen for each gametype from Table A.4: Random, Prisoner's Dilemma, Matching Pennies and the Coordination Game. The created two player, two action games are shown in Table 4.3. The results of the games are scaled to the 0-1 range and most of the stationary algorithms are used as opponents: Best response, Determined, Maxmin, Nash, TFT and the two player Godfather. The stationary algorithms are chosen to lower the variation in the outcome of different rounds. Since we are only interested in the rewards of the algorithm that is being tuned, only the comparisons which contain this algorithm are played. With six opponents thirteen combinations can be made with the algorithm under test: self-play, six with it as column player and six with it as row player. On the symmetric Prisoner's Dilemma game there are only seven different combinations. The rewards the algorithm earned in these games are averaged to a single value which MAX LIPO +TR tries to optimize. As most of the algorithms are stationary we decided to play each game for two rounds instead of five to keep the computation time down.

Table 4.3: The game instances used for the parameter tuning, from left to right: a Random game, Prisoner's Dilemma, Matching Pennies and the Coordination Game

| | *A* | *B* |
|---|---|---|
| *A* | 100/21 | 29/78 |
| *B* | 0/70 | 55/11 |

| | *C* | *D* |
|---|---|---|
| *C* | 83/83 | 0/100 |
| *D* | 100/0 | 36/36 |

| | *H* | *T* |
|---|---|---|
| *H* | 100/0 | 0/100 |
| *T* | 0/100 | 100/0 |

| | *L* | *R* |
|---|---|---|
| *L* | 100/100 | 0/0 |
| *R* | 51/51 | 85/85 |

#### 4.4.1.2 The parameters

There are four algorithms with parameters. The first, $\epsilon$-greedy has one parameter: the exploration rate. Sutton and Barto (2011) compare an exploration rate of 0.1 and 0.01 (Figure 2.2 of their book). They show that while the optimal value was found earlier with an exploration rate of 0.1, playing it only 91% of time leads to a lower average reward in the long run than the exploration rate of 0.01. While the games differ, they used ten actions while we use two, we still expect an exploration rate near 0.01. The optimizer is run twice for 50 iterations, which we think is enough for one variable between 0 and 1. It returned 0.01 as optimal exploration rate both times as shown in Table 4.4. This value is consistent with previous works and will therefore be used in the tournament.

Table 4.4: Results of both optimization runs of $\epsilon$-greedy's exploration rate.

| | *Run 1* | *Run 2* |
|---|---|---|
| *Average reward* | 0.6402 | 0.6561 |
| *Exploration rate* | 0.01 | 0.01 |

Markov also has one parameter, the window size. The referenced paper used a slightly different version of Markov that looks two states ahead instead of one with a window size of two. In the initial experiment against Random Markov got a slight higher reward with a window size of three, therefore we expect the optimal window size lies between two and ten. To give the optimizer room to work the maximal window size is set to 50. The optimizer is run twice for 50 iterations as done for $\epsilon$-greedy, the results are shown in Table 4.5. Both runs returned a window size of eight which is used in the tournament.

Table 4.5: Results of both optimization runs of Markov's window size.

| | *Run 1* | *Run 2* |
|---|---|---|
| *Average reward* | 0.6970 | 0.6970 |
| *Window size* | 8 | 8 |

The third algorithm, SAT has two parameters: the initial aspiration and the persistence rate. Karandikar et al. (1998) advice a persistence level close to one and since the algorithm keeps playing the action as long as it returns a reward above the aspiration level we expect an initial aspiration level close to or above the maximum reward of the game. The optimizer has two parameters so it is given twice as many iterations, 100. The results are shown in Table 4.6. As expected, the value for the initial aspiration level were high, the average rewards are also similar. We decided to set the initial aspiration level at 100, the persistence level is set to 0.93 because it resulted in the slightly higher reward.

Table 4.6: Results of both optimization runs of SAT's parameters.

| | *Run 1* | *Run 2* |
|---|---|---|
| *Average reward* | 0.7079 | 0.7121 |
| *Initial aspiration* | 100 | 100 |
| *Persistence rate* | 0.91 | 0.93 |

Q-learning is the last algorithm, with five parameters it is also the most difficult to tune. Littman and Stone (2001) used a version without decay factors and Zawadzki, Lipson, and Leyton-Brown (2014) used the version we implemented, based on their code. The values of both papers and the results of the optimizer, which is run for 250 iterations, are shown in Table 4.7. The optimal values not only differ a lot between rounds, but are also quite far from the values used by Zawadzki, Lipson, and Leyton-Brown (2014). Therefore, we decided to run a comparison with their parameters as well, (un)surprisingly it resulted in a lower average reward. Where the optimizer did what we asked: find the values that give the highest reward, the real tournament is played on more games against different algorithms. This might explain why different values give a higher reward. Since the optimizer performed as expected for the previous algorithms we decided to use the values which resulted in the slightly higher reward (run 2) in the tournament.

Table 4.7: Results and previously used parameters for the Q-learner.

| | *Littman and Stone (2001)* | *Zawadzki, Lipson, and Leyton-Brown (2014)* | *Run 1* | *Run 2* |
|---|---|---|---|---|
| *Average reward* | - | 0.6179 | 0.6343 | 0.6344 |
| *Exploration rate* | 0.1 | 0.2 | 0.64 | 0.96 |
| *Decay exploration rate* | - | 0.999941 | 0.98 | 0.99 |
| *Learning rate* | 0.1 | 0.9 | 0.02 | 0.44 |
| *Decay learning rate* | - | 0.999913 | 0.99 | 0.84 |
| *Discount rate* | 0.9 | 0.9 | 0.64 | 0.75 |

The two Bully algorithms are compared on the same games, playing against the same opponents. The dynamic version of Bully earned an average reward of 0.529 and 0.577, while the stationary, original, Bully earned a reward of 0.569 and 0.573. The average rewards for the different games are shown in Table 4.8, the Coordination Games are the main cause of the different rewards. Since the dynamic version scored lower on one run, we expect the lower reward is caused by miscommunication. We do not intend to do a full scale comparison of the algorithms, this comparison was merely to test if both algorithms perform similar. The dynamic N-player version of bully is used in the tournament.

Table 4.8: Rewards earned by the bully algorithms.

| | *Dynamic Bully 1* | *Dynamic Bully 2* | *Stationary Bully 1* | *Stationary Bully 2* |
|---|---|---|---|---|
| *Random* | 0.4539 | 0.4586 | 0.4507 | 0.4486 |
| *Prisoner's Dilemma* | 0.36 | 0.36 | 0.36 | 0.36 |
| *Matching Pennies* | 0.5753 | 0.5744 | 0.4978 | 0.4997 |
| *Coordination Game* | 0.6473 | 0.8164 | 0.8711 | 0.8855 |
| *Average* | 0.529 | 0.577 | 0.569 | 0.573 |

### 4.4.2 Information usage

Algorithms are based on different concepts that use different information, as shown in Table 4.9. The own action is presumed known, we think this is a reasonable assumption. The opponents reward is only used to find the PO outcomes and NE of the game, in real situations the opponents rewards might be difficult to asses: How much do they really prefer this outcome? The opponents action on the other hand is used by multiple algorithms, it remains a question whether using this additional information gives them an advantage. The iteration is marked for all algorithms that use a different action selection method in the first iteration(s) to to start the learning process. Most of the algorithms play a random action in the first iterations, which is probably the main reason for the different outcomes of the various rounds.

Table 4.9: Information types used by algorithms.

| *Algorithm* | *Own reward* | *Opp action* | *Opp reward* | *Game matrix* | *Iteration* | *NE's* | *PO's* |
|---|---|---|---|---|---|---|---|
| *Bully* | | x | | x | x | | |
| *$\epsilon$-greedy* | x | | | | x | | |
| *Fictitious play* | | x | | x | x | | |
| *Markov* | x | x | | x | x | | |
| *N-greedy* | x | | | x | x | | |
| *No-regret* | x | x | | x | x | | |
| *Pareto optimal* | | | x | x | | | x |
| *QL* | x | | | | x | | |
| *Random* | | | | | | | |
| *Satisficing play* | x | | | | x | | |
| | | | | *Parameter Tuners* | | | |
| *Best response* | | x | | x | x | | |
| *Determined* | | | x | x | | x | |
| *Godfather* | x | | | x | x | | |
| *Maxmin* | | | | x | | | |
| *Nash* | | | x | | | x | |
| *TFT* | | x | | | x | | |

## 4.5 Results

The game and algorithm names are shortened to the first letters in the images, Table 4.9 and A.4 can be used to check the full names. In the following sections the grand table, average rewards, average regret, summary statistics, and the replicator results are shown and analyzed.

### 4.5.1 Grand table

The grandtable is shown in the appendix, Table A.5. It has no pure NE, only three mixed NE:

- NE1,Partner: $\frac{1}{8}$ N-greedy, $\frac{25}{32}$ Pareto optimal, $\frac{3}{32}$ Satisficing play. Actor: $\frac{1}{6}$ $\epsilon$-greedy, $\frac{1}{6}$ Markov, $\frac{2}{3}$ N-greedy.
- NE2, Partner: $\frac{11}{21}$ N-greedy, $\frac{22}{63}$ Pareto optimal, $\frac{8}{63}$ Satisficing play. Actor: $\frac{1}{19}$ $\epsilon$-greedy, $\frac{10}{19}$ Fictitious play, $\frac{8}{19}$ N-greedy.
- NE3, Partner: $\frac{5}{6}$ Pareto optimal, $\frac{1}{6}$ Satisficing play. Actor: $\frac{1}{2}$ Fictitious play, $\frac{1}{2}$ N-greedy.

Some algorithms are a part of each NE but for this grand table the PO outcomes might say more, they favor Satisficing play as Actor. Satisficing play also scored PO against itself which makes it a promising candidate for the replicator dynamic.

### 4.5.2 Average reward

The heatplots enable us to quickly compare the rewards the algorithms earned, the colors in Figure 4.4 show the averaged reward the Actor earned playing as row and column player against that Partner. Whilst the significance tests use the rewards the Actor earned as column player versus that given Partner as row player. The asterices indicate the Actor scored significantly higher than the Partner.

While some of the Actors scored high against Pareto optimal this difference was not significant, Random on the other hand is dominated significantly by all its opponents. To our surprise Satisficing play is the only algorithm that scored 'red' against itself. Based on the number of algorithms Fictitious play, Bully, Markov, and Satisficing play dominate we would expect them to have higher average rewards, but these turn out to be fairly close as shown in Figure 4.5.

Ignoring the results of Random, Figure 4.6 shows the games in which the algorithms struggled to earn high rewards: the Prisoner's and Travelers Dilemma and it also shows the algorithms all performed well on the LEG games.

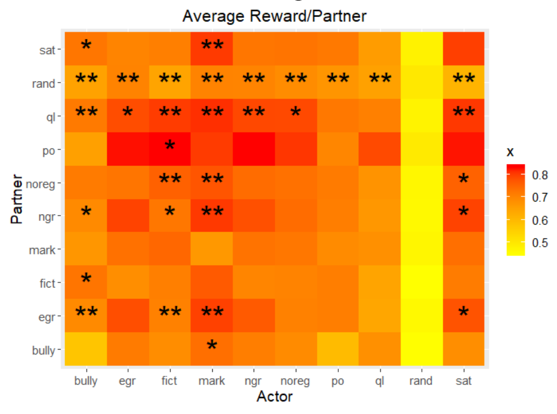


Figure 4.4: Heatplot of the average reward of the Actors against the Partners, one asterisk shows significance at 0.05 level. Two asterisks indicate significance at the 0.001 level.

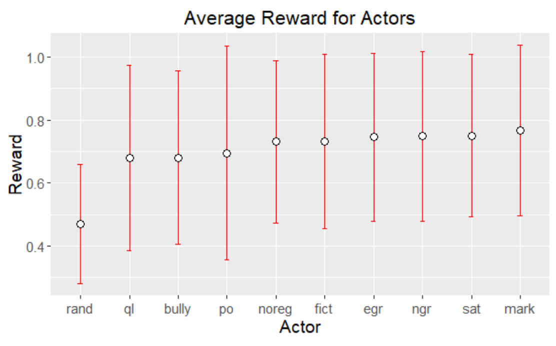


Figure 4.5: Error plot of mean Actor reward with standard deviation

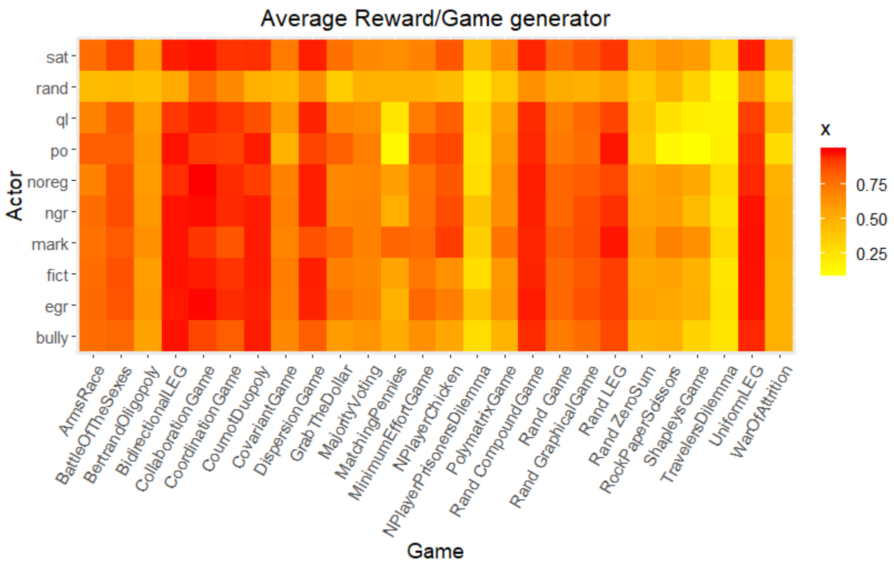


Figure 4.6: Heatplot of the Actors average reward on each gametype

### 4.5.3 Average regret

Similar to the average rewards, the average regrets are also fairly close as shown in Figure 4.8. Fictitious play has the least average regret but did not earn the highest average reward, confirming that algorithms with more regret can score higher than algorithms with less regret. Figure 4.7 looks similar to the reward plot, with only a few minor differences. Figure 4.9 shows all algorithms had more regret on the Matching pennies, Rock Paper Scissors and Shapleys game which is to be expected. There was also little the algorithms could have improved on the Travelers and Prisoner's Dilemma.

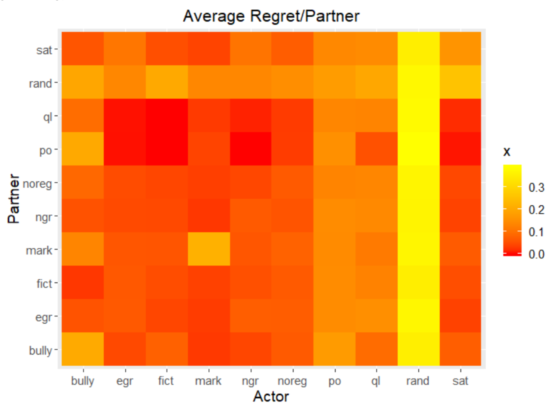


Figure 4.7: Heatplot of the average regret of the Actors against the Partners

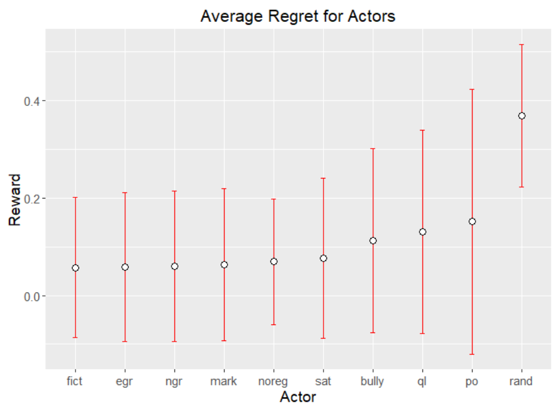


Figure 4.8: Error plot of mean Actor regret with standard deviation

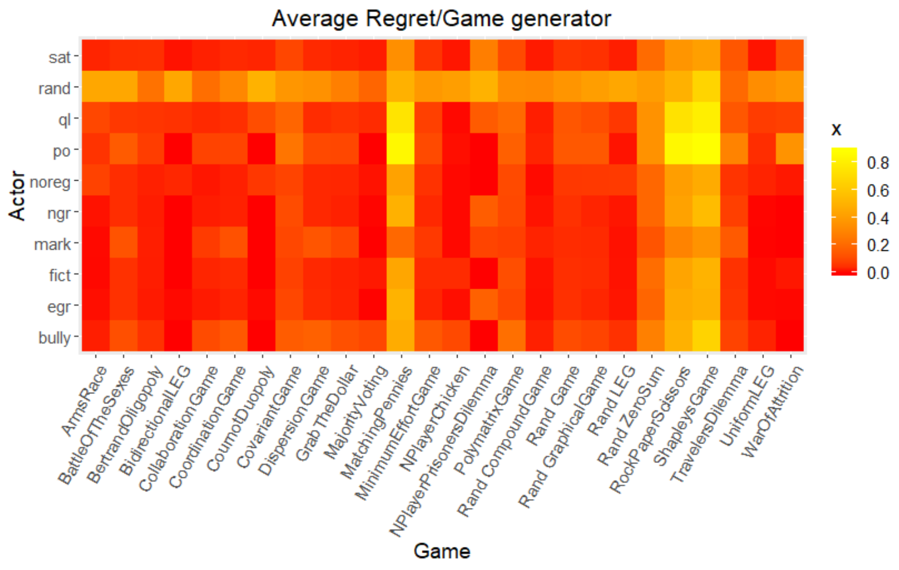


Figure 4.9: Heatplot of the Actors average regret on each gametype

### 4.5.4 Mean results

Where plots make it easy to compare results it is hard to see how much the values really differ. Excluding self-play, the algorithms all played 1818 comparisons. With a score of two for a win, one for a draw, and zero for a loss the maximum score in Table 4.10 is 1818 for both the score as Partner and Actor. Markov has both a high average rewards and score, this shows it did not earn the highest average reward by exploiting a few opponents by a large margin but outperformed most of the algorithms in the tournament. And where it is difficult to see in the rewards plots, Table 4.10 shows that $\epsilon$-greedy and N-greedy's score almost as good in self-play as Satisficing play.

Table 4.10: Average rewards and number of wins earned as Partner and Actor

| | *Partner* | | *Actor* | | *Self-play* |
|---|---|---|---|---|---|
| | *Reward* | *Score* | *Reward* | *Score* | *Reward* |
| *Bully* | 0.6755 | 1029 | 0.6859 | 1090 | 0.5735 |
| *$\epsilon$-greedy* | 0.7437 | 816 | 0.7477 | 883 | 0.7822 |
| *Fictitious play* | 0.7273 | **1076** | 0.7375 | 1120 | 0.7102 |
| *Markov* | **0.7639** | 1060 | **0.7705** | **1160** | 0.6657 |
| *N-greedy* | 0.7452 | 881 | 0.7514 | 950 | 0.7807 |
| *No-regret* | 0.7268 | 903 | 0.7356 | 954 | 0.7333 |
| *Pareto optimal* | 0.6983 | 900 | 0.6914 | 922 | 0.6980 |
| *QL* | 0.6788 | 734 | 0.6783 | 758 | 0.7090 |
| *Random* | 0.4664 | 432 | 0.4731 | 471 | 0.4979 |
| *Satisficing play* | 0.7473 | 987 | 0.7544 | 1054 | **0.7990** |

### 4.5.5 Replicator

The outcome of the replicator dynamic is shown in Figure 4.10, despite the high average reward of Markov and the high reward Satisficing play earned in self-play N-greedy and $\epsilon$-greedy have the largest proportions. The dynamic is ran for 1,000 iterations more than planned as N-greedy seems to be climbing around 1,000 iterations, but it drops again leaving $\epsilon$-greedy as the winner. An example that a higher average reward does not mean it is a better algorithm.

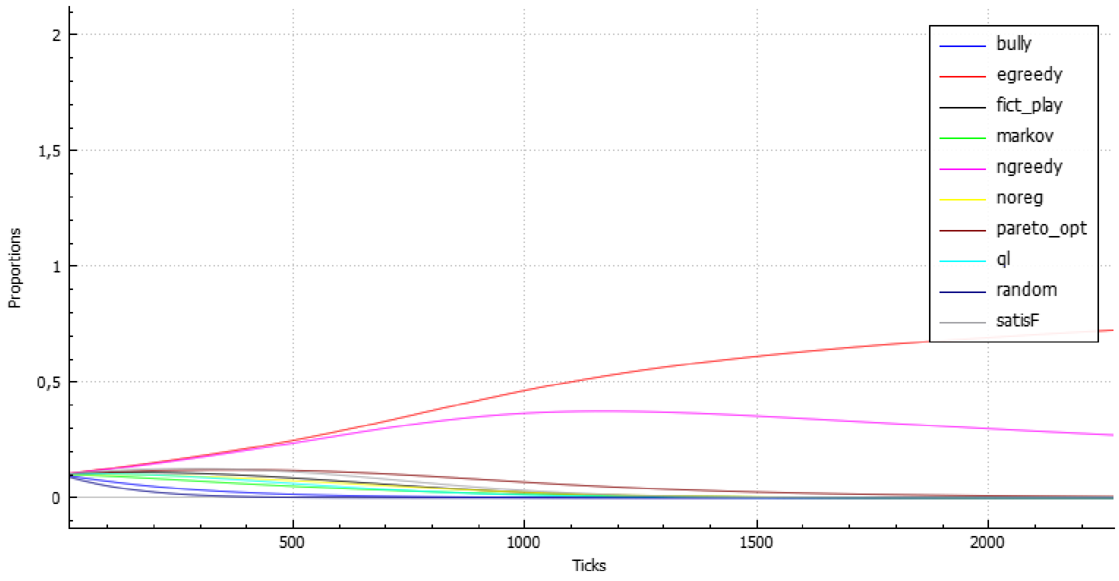


Figure 4.10: Replicator dynamic

# 5 Discussion and Conclusion

This chapter starts with a discussion, followed by the opportunities for further research and the conclusion.

## 5.1 Discussion

The results of the sample tournament are not the real result of this thesis, despite this we discuss these before moving on to the actual result: the methodological considerations.

Q-Learning was added because it outperformed other modern algorithms in the tournament of Zawadzki, Lipson, and Leyton-Brown (2014), surprisingly it did not even end up in the top three of our tournament. We suspect the main reason for this is the different parameters found through the new parameter tuning process, which gave the expected results for all algorithms except Q-learning. This suggests Q-learning either got an unfair advantage in previous works, or need a different tuning process. The newcomer, Markov, which is based on the idea of Bayesian learning, did surprisingly well. Bayesian learners are known to find the best solution if the prior probabilities are set right. In this tournament uniform priors were used, as a results Markov ends up in the NE in the Prisoners Dilemma instead of the action pair: cooperate, cooperate. Maybe the opponents' rewards can be used to set the priors differently to reach the optimal rewards.

The methodology can be studied in multiple ways, we made a shallow analysis of all the stages. The alternative is an in-depth analysis of one part, for instance the games as done by Nudelman et al. (2004). Both have their advantages, but our broad comparison gives more insight into the challenges that arise when choosing a methodology. In depth analysis of the various methods are interesting follow-up projects.

## 5.2 Future work

The analysis of past work led to new insights, too many to investigate and test in one thesis. They are described for each part of the methodology.

**Tournament setup**
The setup contains mostly settings without an optimal value: the number of rounds and iterations, but also bigger design choices like the tournament type vary between papers. We argued for tournaments with more algorithms and described a way to implement them: the Swiss tournament. Implementing this and comparing the results of various seeding methods with the results of a round robin tournament is beyond the scope of this thesis. Despite this, we think it has potential: it will save computation time and enable bigger tournaments as organized in the beginning of the field. An open tournament of which all algorithms are published online will help the field advance faster. For this to happen, it is important that everyone agrees on the tournament settings (and more algorithms can compete in the same tournament). A first step in this direction is the convergence test, the test can be used to determine how many iterations the current algorithms need to converge. For the number of rounds we argued against the bootstrap method, a comparison of bootstrapped results and the results of multiple rounds might help find a guideline to determine the number of rounds.

**Games**
While the games are often treated as a simple tool, the playing field has a big influence on the outcome. In our tournament we ended up excluding some generators, implementing them combined with different types of games is a logical follow-up project. For example, stochastic and extensive form games to broaden the playing field. During our tournament we also encountered one game with zero rewards for all actions, as a result of the random parameters used by Gamut. We wasted valuable computation time running this game. To prevent this in the future the framework can be expanded with a few checks to ensure the games are playable. The games can also be studied to create the interesting or playable random cases for all generators.

**Algorithms**
The algorithms are the main part of most papers, in ours they were only used for testing purposes. Despite this, parameter tuning is an important part of the methodology. We argued that the games, algorithms and settings used in the tuning process influence the outcome of the tournament. Verifying this and looking into how to balance between fairness and the evaluation function is an interesting follow-up project. The tournament also had a surprising winner: Markov, which makes us wonder how it scores against state of the art learning algorithms. Lastly, we noticed some algorithms use more information than others, which might give them an advantage. We think this can easily be tested by comparing the results of similar algorithms like Reinforcement learning and Satisficing play with each other. Satisficing play is a variant of Reinforcement learning created for games in which the opponents action cannot be observed.

**Result analysis**
It is an understatement to say that some information is lost by averaging results, but this is inevitable if we want to interpret the results ourselves. Machine learning algorithms might be a solution here, they can use the raw data to gain new insights. They might also be able to find a method that can determine the best overall performer combining all the currently separately used methods. The analysis of N-player games is also an interesting field. Where this is difficult for humans, a learning method that works for two player games might also be usable on bigger games.

## 5.3 Conclusion

In this thesis we gave an overview of the various methods used in previous tournaments along with their purpose. As stated in the introduction most methods are used to answer different questions, therefore there is no optimal methodology. During the comparison additional attention was payed to small design choices, like how to tune the parameters and why using the same games and algorithms for tuning and comparing can introduce a bias.

We chose the optimal values for our tournament settings where possible. For a few settings, like the number of rounds there is no optimal value, but based on past tournaments and some design criteria there is usually a preferred value. We highlighted a few design choices that can lead to misleading results, like averaging over games with different reward ranges, or comparing regret earned on these games. We also combined statistical methods and plots used in previous research and explained when they can be used. There is however still a lot of room for improvement and a need for insight.

# Bibliography

Airiau, Stéphane, Sabyasachi Saha, and Sandip Sen. 2004. "Dynamics of strategy distribution in iterated games."

Airiau, Stéphane, and Sandip Sen. 2003. "Tournament-based comparison of learning and non-leanring strategies in iterated single-stage games": 1–6.

Airiau, Stéphane, Sandip Sen, and Sabyasachi Saha. 2005. "Evolutionary Tournament-Based Comparison of Learning and Non-Learning Strategies for Iterated Games.": 449–454.

Axelrod, Robert, and William Donald Hamilton. 1981. "The evolution of cooperation." *science* 211 (4489): 1390–1396.

Bergstra, James, and Yoshua Bengio. 2012. "Random search for hyper-parameter optimization." *Journal of Machine Learning Research* 13 (Feb): 281–305.

Bouzy, Bruno, and Marc Métivier. 2010. "Multi-agent learning experiments on repeated matrix games": 119–126.

Bouzy, Bruno, Marc Métivier, and Damien Pellier. 2018. "Hedging algorithms and repeated matrix games." *arXiv preprint arXiv:1810.06443.*

Byl, John. 2013. *Organizing Successful Tournaments, 4E.* Human Kinetics.

Chernick, Michael R. 2011. *Bootstrap methods: A guide for practitioners and researchers.* Vol. 619. John Wiley & Sons.

Crandall, Jacob W. 2014. "Towards minimizing disappointment in repeated games." *Journal of Artificial Intelligence Research* 49:111–142.

Crandall, Jacob W, Asad Ahmed, and Michael A Goodrich. 2011. "Learning in Repeated Games with Minimal Information: The Effects of Learning Bias."

Crandall, Jacob W, and Michael A Goodrich. 2011. "Learning to compete, coordinate, and cooperate in repeated games using reinforcement learning." *Machine Learning* 82 (3): 281–314.

Daskalakis, Constantinos, Paul W Goldberg, and Christos H Papadimitriou. 2009. "The complexity of computing a Nash equilibrium." *SIAM Journal on Computing* 39 (1): 195–259.

Fudenberg, Drew, Fudenberg Drew, David K Levine, and David K Levine. 1998. *The theory of learning in games.* Vol. 2. MIT press.

Gintis, Herbert. 2009. *Game Theory Evolving: A Problem-Centered Introduction to Modeling Strategic Interaction.* Princeton University Press.

Goldberg, David E, and Kalyanmoy Deb. 1991. "A comparative analysis of selection schemes used in genetic algorithms." In *Foundations of genetic algorithms,* 1:69–93. Elsevier.

Haan, I de. 2018. "Een lerende strategie gebruikmakend van Markov modellen om te gebruiken in de herhaalde vorm van bimatrix spelen." B.S. thesis.

Ishowo-Oloko, Fatimah, Jacob Crandall, Manuel Cebrian, Sherief Abdallah, and Iyad Rahwan. 2014. "Learning in repeated games: Human versus machine." *arXiv preprint arXiv:1404.4985.*

Jose B. Cruz, Jr., and Marwan A. Simaan. 2002. "Towards minimizing disappointment in repeated games."

Just, Tim, Daniel B Burg, et al. 2003. *United States Chess Federation's Official Rules of Chess.* Random House Incorporated.

Karandikar, Rajeeva, Dilip Mookherjee, Debraj Ray, and Fernando Vega-Redondo. 1998. "Evolving aspirations and cooperation." *journal of economic theory* 80 (2): 292–331.

Kilgour, D Marc, and Niall M Fraser. 1988. "A taxonomy of all ordinal 2 × 2 games." *Theory and Decision* 24 (2): 99–117.

King, Davis. 2009. “Dlib-ml: A Machine Learning Toolkit.” *Journal of Machine Learning Research* 10:1755–1758.

Littman, Michael L, and Peter Stone. 2001. “Leading best-response strategies in repeated games.”

Malherbe, Cédric, and Nicolas Vayatis. 2017. “Global optimization of lipschitz functions”: 2314–2323.

McKelvey, Richard D, Andrew M McLennan, and Theodore L Turocy. 2006. “Gambit: Software tools for game theory.”

Nash, John F, et al. 1950. “Equilibrium points in n-person games.” *Proceedings of the national academy of sciences* 36 (1): 48–49.

Nudelman, Eugene, Jennifer Wortman, Yoav Shoham, and Kevin Leyton-Brown. 2004. “Run the GAMUT: A comprehensive approach to evaluating game-theoretic algorithms”: 880–887.

Papadimitriou, Christos H. 1994. “On the complexity of the parity argument and other inefficient proofs of existence.” *Journal of Computer and system Sciences* 48 (3): 498–532.

Powell, Michael. 2009. “The BOBYQA algorithm for bound constrained optimization without derivatives.” *Cambridge NA Report NA2009/06, University of Cambridge, Cambridge:* 26–46.

Powell, Michael. 2012. “On the convergence of trust region algorithms for unconstrained minimization without derivatives.” *Computational Optimization and Applications* 53 (2): 527–555.

Powers, Rob, and Yoav Shoham. 2005a. “Learning against opponents with bounded memory.” In *IJCAI,* 5:817–822.

Powers, Rob, and Yoav Shoham. 2005b. “New criteria and a new algorithm for learning in multi-agent systems”: 1089–1096.

Shapiro, Samuel Sanford, and Martin B Wilk. 1965. “An analysis of variance test for normality (complete samples).” *Biometrika* 52 (3/4): 591–611.

Shapley, Lloyd S. 1953. “Stochastic games.” *Proceedings of the national academy of sciences* 39 (10): 1095–1100.

Shoham, Yoav, and Kevin Leyton-Brown. 2008. *Multiagent systems: Algorithmic, game-theoretic, and logical foundations.* Cambridge University Press.

Sutton, Richard S, and Andrew G Barto. 2011. “Reinforcement learning: An introduction.”

Taylor, Peter D, and Leo B Jonker. 1978. “Evolutionary stable strategies and game dynamics.” *Mathematical biosciences* 40 (1-2): 145–156.

Voraprateep, Jutharath. 2013. “Robustness of Wilcoxon signed-rank test against the assumption of symmetry,” University of Birmingham.

Vu, Thuc, Rob Powers, and Yoav Shoham. 2006. “Learning against multiple opponents”: 752–759.

Weibull, Jörgen W. 1997. *Evolutionary game theory.* MIT press.

Wu, Jianzhong, and Robert Axelrod. 1995. “How to cope with noise in the iterated prisoner’s dilemma.” *Journal of Conflict resolution* 39 (1): 183–189.

Young, H Peyton. 2004. *Strategic learning and its limits.* Chap. 2.4. OUP Oxford.

Zawadzki, Erik, Asher Lipson, and Kevin Leyton-Brown. 2014. “Empirically evaluating multiagent learning algorithms.” *arXiv preprint arXiv:1401.8074.*

## Websites

Bruns, Bryan. n.d. “Periodic Table of 2x2 Games.” `http://www.bryanbruns.com/2x2chart/`, Last accessed on 2019-5-10.

“Hyperparameter optimization.” n.d. `https://en.wikipedia.org/wiki/Hyperparameter_optimization`, Last accessed on 2019-5-10.

King, Davis. 2017. “A Global Optimization Algorithm Worth Using.” `http://blog.dlib.net/2017/12/a-global-optimization-algorithm-worth.html`, Last accessed on 2019-5-10.

“List of games in game theory.” n.d. `http://www.gametheory.net/Dictionary/games/`, Last accessed on 2019-5-10.

“MATLAB release history.” n.d. `https://en.wikipedia.org/wiki/MATLAB`, Last accessed on 2019-5-10.

# Appendix

## A.4 Types of games

Explanation for removing the Location game is given in Section 4.3.

Table A.4: Number of players (N pl), action (N act), and other parameters of 31 of the Gamut generators. The excluded generators are marked bold. The Congestion Game, Greedy Game, Guess Two Thirds Ave and the Simple Inspection Game are not analyzed as they are not supported by the framework. The mixed NE column indicates if the created instances had a Mixed NE or not, 'Y' is an abbreviation for 'Yes' and 'C' for 'Can' an empty cell indicates a no.

| *Generator* | *N pl* | *N act* | *Symmetric* | *Mixed NE* | *Pure NE=PO* | *Common interest* | *Conflicting interest* | *Constant sum* |
|---|---|---|---|---|---|---|---|---|
| *Arms Race* | 2 | - | Y | C | | | | |
| *Battle of the Sexes* | 2 | 2 | Y | Y | | | Y | |
| *Bertrand Oligopoly* | - | - | Y | C | | | | |
| *Bidirectional LEG* | - | - | Y | C | | | | |
| ***Chicken*** | 2 | 2 | Y | C | Highest | | | |
| *Collaboration Game* | - | 2 | | C | Y | Y | | |
| *Coordination Game* | - | 2 | | C | Highest | Y | | |
| *Cournot Duopoly* | 2 | - | | C | Highest | | | |
| *Covariant Game* | - | - | | C | Highest | | | |
| *Dispersion Game* | - | - | Y | Y | Y | Y | | |
| *Grab The Dollar* | 2 | - | Y | Y | Y | | Y | |
| ***Hawk and Dove*** | 2 | 2 | Y | | | | Y | |
| ***Location Game*** | 2 | - | | | Y | | | |
| *Majority Voting* | - | - | | | | | | |
| *Matching Pennies* | 2 | 2 | | Y | | | Y | Y |
| *Minimum Effort Game* | - | - | Y | C | Highest | Y | | |
| *N-Player Chicken* | - | 2 | Y | C | Highest | | | |
| *N-Player Prisoners Dilemma* | - | 2 | Y | | | | Y | |
| *Polymatrix Game* | - | - | | | | | | |
| ***Prisoners Dilemma*** | 2 | 2 | Y | | | | Y | |

| *Generator* | *N pl* | *N act* | *Sym-met-ric* | *Mixed NE* | *Pure NE=PO* | *Common interest* | *Con-flicting interest* | *Con-stant sum* |
|---|---|---|---|---|---|---|---|---|
| *Random Game* | - | - | | | | | | |
| *Random Compound Game* | - | 2 | Y | C | Highest | | | |
| *Random LEG* | - | - | Y | | | | | |
| ***Random Graphical Game*** | - | - | | | | | | |
| *Random Zero Sum* | 2 | - | | | | | | Y |
| *Rock Paper Scissors* | 2 | 3 | Y | Y | | | | Y |
| *Shapleys Game* | 2 | 3 | Y | Y | | | | |
| *Travelers Dilemma* | - | - | Y | | | | Y | |
| ***Two by Two Game*** | 2 | 2 | | | | | | |
| *Uniform LEG* | - | - | Y | | | | | |
| *War Of Attrition* | 2 | - | | | Inverse | | | |

## A.5 Grand table

Some of the players names have been shortened to save space: Satisficing play (Sat), Fictitious play (Fict) and Pareto optimal (PO)

Table A.5: Grand table of the main tournament, the pure NE are underlined and the PO action pairs are shown bold

| | | *Actor* | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | *Bully* | *ϵ-greedy* | *Fict* | *Markov* | *N-greedy* | *No-regret* | *PO* | *Ql* | *Random* | *Sat* |
| *Partner* | *Bully* | 0.57/0.58 | 0.71/0.72 | 0.77/0.67 | 0.71/0.62 | 0.66/0.76 | 0.69/0.75 | 0.65/0.63 | 0.77/0.66 | 0.57/0.57 | 0.75/0.71 |
| | ***ϵ-greedy*** | 0.70/0.72 | 0.78/0.75 | 0.72/0.67 | 0.73/0.79 | **0.74/0.82** | 0.76/0.68 | 0.72/0.80 | 0.74/0.68 | 0.58/0.57 | 0.70/0.76 |
| | *Fict* | 0.66/0.74 | 0.68/0.70 | 0.70/0.70 | 0.68/0.73 | 0.66/0.75 | 0.72/0.73 | 0.72/0.80 | 0.74/0.59 | 0.57/0.56 | 0.71/0.77 |
| | *Markov* | 0.70,0.68 | 0.76/0.71 | 0.72/0.64 | 0.66/0.64 | 0.73/0.78 | 0.73/0.62 | 0.73/0.77 | 0.73/0.75 | 0.60/0.56 | **0.79/0.74** |
| | *N-greedy* | 0.70/0.69 | 0.77/0.79 | 0.70/0.72 | 0.72/0.81 | 0.78/0.77 | 0.69/0.78 | 0.73/0.80 | 0.62/0.68 | 0.59/0.54 | 0.74/0.78 |
| | *No-regret* | 0.71/0.73 | 0.77/0.68 | 0.74/0.69 | 0.73/0.64 | 0.72/0.77 | 0.70/0.74 | 0.73/0.80 | 0.65/0.72 | 0.60/0.48 | 0.78/0.70 |
| | *PO* | 0.64/0.65 | 0.75/0.78 | 0.76/0.79 | 0.73/0.75 | 0.77/0.77 | 0.82/0.71 | 0.75/0.63 | 0.64/0.72 | 0.59/0.54 | **0.80/0.73** |
| | *QL* | 0.66/0.62 | 0.67/0.73 | 0.70/0.72 | 0.72/0.80 | 0.73/0.77 | 0.70/0.77 | 0.78/0.77 | 0.75/0.72 | 0.64/0.59 | 0.65/0.67 |
| | *Random* | 0.61/0.65 | 0.59/0.68 | 0.57/0.57 | 0.61/0.71 | 0.56/0.69 | 0.57/0.71 | 0.59/0.71 | 0.57/0.71 | 0.59/0.62 | 0.59/0.69 |
| | *Sat* | 0.72/0.74 | 0.81/0.66 | 0.78/0.67 | 0.74/0.66 | 0.75/0.77 | 0.72/0.72 | 0.76/0.76 | 0.76/0.73 | 0.65/0.60 | **0.78/0.79** |

# List of Figures

# List of Tables