# Multiagent Learning and Empirical Methods

by

Erik P. Zawadzki

B.Sc. Honours, The University of British Columbia, 2005

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

The Faculty of Graduate Studies

(Computer Science)

The University Of British Columbia

(Vancouver)

October, 2008

# Abstract

Many algorithms exist for learning how to act in a repeated game and most have theoretical guarantees associated with their behaviour. However, there are few experimental results about the empirical performance of these algorithms, which is important for any practical application of this work. Most of the empirical claims in the literature to date have been based on small experiments, and this has hampered the development of multiagent learning (MAL) algorithms with good performance properties.

In order to rectify this problem, we have developed a suite of tools for running multiagent experiments called the Multiagent Learning Testbed (MALT). These tools are designed to facilitate running larger and more comprehensive experiments by removing the need to code one-off experimental apparatus. MALT also provides a number of public implementations of MAL algorithms—hopefully eliminating or reducing differences between algorithm implementations and increasing the reproducibility of results. Using this test-suite, we ran an experiment that is unprecedented in terms of the number of MAL algorithms used and the number of game instances generated. The results of this experiment were analyzed by using a variety of performance metrics—including reward, maxmin distance, regret, and several types of convergence. Our investigation also draws upon a number of empirical analysis methods. Through this analysis we found some surprising results: the most surprising observation was that a very simple algorithm—one that was intended for single-agent reinforcement problems and not multiagent learning—performed better empirically than more complicated and recent MAL algorithms.

# Table of Contents

**Abstract** . . . . . ii

**Table of Contents** . . . . . iii

**List of Tables** . . . . . v

**List of Figures** . . . . . vi

**Acknowledgments** . . . . . viii

**Dedication** . . . . . ix

**1 Introduction** . . . . . 1

**2 Background and Related Work** . . . . . 3
2.1 Preference and Utility . . . . . 3
2.2 One-shot Games . . . . . 5
2.2.1 Solution Concepts . . . . . 7
2.3 Repeated Games . . . . . 9
2.4 Multiagent Learning Algorithms . . . . . 11
2.4.1 Fictitious Play . . . . . 12
2.4.2 Determined . . . . . 13
2.4.3 Targeted Algorithms . . . . . 14
2.4.4 Q-learning Algorithms . . . . . 14
2.4.5 Gradient Algorithms . . . . . 15
2.4.6 Previous Experimental Results . . . . . 16

**3 Platform** . . . . . 18
3.1 The Platform Architecture . . . . . 18
3.1.1 Definitions . . . . . 18
3.1.2 Platform Structure . . . . . 19
3.1.3 Algorithm Implementations . . . . . 21

**4 Empirical Methods and Setup** . . . . . 27
4.1 Experimental Setup . . . . . 27
4.2 Bootstrapping . . . . . 28
4.3 Statistical Tests . . . . . 29
4.3.1 Kolmogorov-Smirnov Test . . . . . 29
4.3.2 Spearman's Rank Correlation Test . . . . . 31
4.4 Probabilistic Domination . . . . . 31

**5 Empirical Evaluation of MAL Algorithms** . . . . . 33
5.1 Reward-Based Metrics . . . . . 33
5.1.1 Average Reward . . . . . 33
5.1.2 Maxmin Distance . . . . . 53
5.2 Regret . . . . . 56
5.3 Convergence-Based Metrics . . . . . 64
5.3.1 Strategic Stationarity . . . . . 67
5.3.2 Stage-Game Nash Equilibria . . . . . 71
5.3.3 Repeated-Game Nash Equilibria . . . . . 71
5.4 Links Between Metrics . . . . . 73
5.4.1 Linking Reward With Maxmin Distance . . . . . 73
5.4.2 Linking Reward With Regret . . . . . 77
5.4.3 Linking Reward With Nash Equilibrium Convergence . . . . . 80

**6 Discussion and Conclusion** . . . . . 88

**7 Future Work: Extension to Traffic** . . . . . 91
7.1 Wardrop Equilibrium . . . . . 92
7.2 Congestion Games . . . . . 92
7.3 Our Game . . . . . 93
7.3.1 Other Models . . . . . 95
7.3.2 Experimental Directions . . . . . 96

## Appendices

**A Stratified Sampling** . . . . . 97

**Bibliography** . . . . . 99

# List of Tables


2.1 Previous experiments . . . . . 17

3.1 Design decisions for `fictitious play` . . . . . 22
3.2 Design decisions for `determined` . . . . . 22
3.3 Design decisions for `meta` . . . . . 23
3.4 Design decisions for `meta` . . . . . 23
3.5 Design decisions for `GIGA-WoLF` . . . . . 24
3.6 Design decisions for `GSA` . . . . . 24
3.7 Design decisions $\texttt{RV}_{\sigma(t)}$ . . . . . 25
3.8 Design decisions for `Q-learning` . . . . . 25

4.1 The number and name of each game generator. . . . . 28

5.1 The different algorithms and their best-response sets . . . . . 38
5.2 The proportion of grand distribution subsampled algorithm games where each algorithm was strictly or weakly dominated. . . . . 40
5.3 The set of best algorithms for each generator. . . . . 49

A.1 Two schemes for sampling. . . . . 97

# List of Figures


2.1 von Neummann and Morgenstern's six preference axioms. . . . . 4
2.2 A partial ordering example . . . . . 5
2.3 A game of *Prisoner's Dilemma* in normal-form with canonical payoffs. . . . . 7
2.4 A game of *Battle of the Sexes* in normal-form with canonical payoffs. . . . . 8
2.5 A game of *Traveler's Dilemma* with 100 actions. . . . . 11
2.6 A *Sidewalk* or *Dispersion* game, where two agents try to miscoordinate where they step. . . . . 13
2.7 A game with a convincing Stackelberg outcome . . . . . 14

3.1 The five steps in running an analyzing an experiment using MALT. . . . . 20

4.1 Kolmogorov-Smirnov test example. . . . . 30

5.1 Lens plot for reward . . . . . 35
5.2 Bootstrapped mean estimator distributions for `Q-learning` and $\mathtt{RV}_{\sigma(t)}$ . . . . . 36
5.3 A heatmap showing the mean reward for each algorithm . . . . . 37
5.4 The mean reward algorithm game . . . . . 39
5.5 The mean reward algorithm game for D4 . . . . . 42
5.6 Reward distribution probabilistic domination between the algorithms . . . . . 43
5.7 Self-play lens plot for reward . . . . . 44
5.8 Algorithm-generator heatmap for reward. . . . . 46
5.9 Normalized algorithm-generator heatmap for reward. . . . . 47
5.10 Bootstrapped mean estimate distribution for D7 . . . . . 48
5.11 Correlation between size and reward . . . . . 50
5.12 Similarity among different algorithms . . . . . 52
5.13 The sign of the safety distance of each run, by algorithm. . . . . 54
5.14 The distribution of maxmin distances for `AWESOME`, `minimax-Q` and `Q-learning`. . . . . 55
5.15 The proportion of enforceable runs, blocked by opponent. . . . . 57
5.16 The distribution of negative maxmin distances for `GIGA-WoLF` and $\mathtt{RV}_{\sigma(t)}$. . . . . 58
5.17 Lens plot for regret. . . . . 60
5.18 The number of runs for each algorithm that have negative, zero, or positive regret. . 61
5.19 The distribution of regret for `Q-learning` and `GIGA-WoLF`. . . . . 62

5.20 Mean average regret, blocked by opponent. . . . . 63
5.21 The number of opponents for which the algorithm on the ordinate probabilistically dominates the algorithm on the abscissa. For example, `Q-learning` probabilistically dominates `fictitious play` on PSMs involving ten out of eleven possible opponents. . . . . 65
5.22 The number of generators for which the algorithm on the ordinate probabilistically dominates the algorithm on the abscissa. . . . . 66
5.23 Proportion of stationary runs, blocked on opponent . . . . . 68
5.24 Proportion of non-stationary runs, blocked on generator and protagonist . . . . . 69
5.25 Convergences proportions . . . . . 70
5.26 Self-play convergences proportions . . . . . 72
5.27 Proportion of PSMs with enforceable payoffs and payoffs profiles achieved, by algorithm. . . . . 74
5.28 Sign of correlation between reward and maxmin distance . . . . . 75
5.29 The distribution of reward for $\mathsf{RV}_{\sigma(t)}$ when conditioning on different maxmin distances. . . . . 76
5.30 The distribution of reward for `Q-learning` when conditioning on different enforceability. Runs from D10 were excluded. . . . . 78
5.31 Bivariate histogram showing reward and maxmin distance for $\mathsf{RV}_{\sigma(t)}$ . . . . . 79
5.32 The sign of correlation between reward and regret . . . . . 81
5.33 `GIGA-WoLF`'s CDF curves for positive and non-positive reward . . . . . 82
5.34 Reward and regret for `AWESOME` . . . . . 83
5.35 Reward and regret for `Q-learning` . . . . . 84
5.36 The sign of correlation between reward and $\ell_\infty$-distance to the closest Nash equilibrium . . . . . 86
5.37 Reward and $\ell_\infty$-distance to the closest Nash equilibrium . . . . . 87

7.1 A sample road graph with four intersections. . . . . 91
7.2 A sample road graph with two intersections that has no Wardrop equilibrium for a system with two atomic drivers. . . . . 92

# Acknowledgments

This thesis would not have been possible without the excellent support that I received from all of my friends, colleagues, and professors at the University of British Columbia. I cannot name everyone that has helped me with my work in the past seven years and two degrees at UBC, but I will try to name those who directly helped with this dissertation.

There were three students that worked on this project with me. All of this work stems from a previous thesis by Asher Lipson, and reading his thesis and paper helped me get a grip on the problem. David Ludgate was very helpful with the coding of this version of MALT and bore with me when I was still figuring out the rudimentary details of this project. Alice Gao was an essential part of the traffic application of this work.

My fellow students were a constant source of ideas, suggestions, and advice. I'm very thankful to Baharak Rastegari, Albert Xin Jiang, David Thompson, James Wright, Samantha Leung, Damien Bargiacchi, Frank Hutter, Lin Xu, and Tristram Southey for their insights. I am particularly thankful for James' careful read over the traffic part of the thesis. Both the GTDT and EARG reading groups listened and commented on version iterations of my talk and watched it grow from a summer project, to a course project and finally to a Master's thesis.

The entire direction for the analysis of results changed after taking the empirical algorithmics course given by Dr. Holger Hoos, and I am indebted to him for his interesting presentation of the material and useful suggestions for my problem. Dr. Nando de Frietas, beyond agreeing to being a reviewer for this thesis (on his sabatical, no less), was very helpful throughout the process—particularly with some of the statistics for this project.

Dr. Kevin Leyton-Brown, my supervisor, showed me what it meant to do research and essentially taught me how to write from scratch. He constantly challenged me and was a great person to work with. Three pieces of his advice will always stick with me: be the step-mother of your work—not its mother; research is a dialog and you should see your work as adding to the conversation; and always use a sans serif font in presentations.

# Dedication

*To my parents, my Babcia, and Gwenn. Each one is as important as the last*[1].

[1]This can be seen as a preference relationship that does not satisfy completeness. See Table 2.1

# Chapter 1

# Introduction

Urban road networks, hospital systems and commodity markets are all examples of complicated multiagent systems that are essential to everyday life. Indeed, any social interaction can be seen as a multiagent problem. As a result of the prominence of multiagent systems, a lot of attention has been paid to designing and analyzing learning algorithms for multiagent environments. Examples include algorithms by Littman [30], Singh et al. [47], Hu and Wellman [24], Greenwald and Hall [19], Bowling [7], Powers and Shoham [38], Banerjee and Peng [5], and Conitzer and Sandholm [13]. As a result of this attention, a multitude of different algorithms exist for a variety of different settings.

Before introducing a new algorithm, no matter the problem, a fundamental question needs to be asked: how does it compare with previous approaches? In particular, there is little point in suggesting a method that does not offer some form of improvement. Such a question rarely can be answered with theoretical guarantees alone, and a full answer typically involves an extensive set of experiments.

This thesis follows the *Artificial Intelligence agenda* [46] for multiagent learning: a good multiagent learning (MAL) algorithm is one that gets high rewards for its actions in a multiagent system. However, past work in MAL has primarily focused on proving theoretical guarantees about different algorithms and it has been difficult to prove results about raw average reward. Instead there have been numerous theorems about a variety of alternative performance metrics—for which results are provable—that are intended to 'stand in' for reward. So how good are the numerous MAL algorithms in terms of the Artificial Intelligence criterion for performance?

While most results for MAL algorithms have been theoretical, some experiments have been conducted. Most of these experiments were small in terms of game instances, opposing algorithms and performance metrics. The small number of metrics is particularly important. Authors have focused on many different aspects of performance for both theoretical and empirical work. The abundance of possible metrics and lack of comprehensive experiments mean that given any two algorithms it is unlikely that their empirical results can be meaningfully compared—the experiments that have been done investigated many incomparable aspects of performance. As a result, there are still opportunities to expand our empirical understanding of how MAL techniques interact and how one could design an algorithm with improved empirical characteristics.

Part of the reason for the relative paucity of large-scale empirical work is that neither a centralized algorithm repository nor a standardized test setup exists. This is unfortunate, not only because needless work has to be invested in designing one-off testbeds and reimplementing algorithms, but also because a centralized and public repository increases reproducibility and decreases the poten-

tial for implementation differences. Publicly available and scrutinized implementations will make experiments easier to run, reproduce, and compare.

In this thesis we make two contributions. First, we describe the design and implementation of a platform for running MAL experiments (§ 3). This platform offers several advantages over one-off setups and we hope that this initial investment in architecture will facilitate larger and more comprehensive empirical work.

Our second contribution is a large experiment that we set up and executed with our platform that is, to our knowledge, unprecedented in terms of scale (§ 4). Our discussion of this experiment focuses on suggesting empirical methods for analyzing MAL performance data and engaging in a detailed discussion of the results. In particular, we show that there are some interesting relationships between the different performance metrics and that some very basic algorithms out perform more sophisticated algorithms on several key metrics.

# Chapter 2

# Background and Related Work

Game theory is a way to model the interactions of multiple self-interested agents. Each game maps the decisions made by the agents to outcomes which are in turn mapped to rewards for each of the agents. Everyone's reward depends on the decisions made by other agents and vice versa. This powerful representation not only captures everything that is commonly thought of as a game, like rock-paper-scissors and backgammon, but also many other complicated situations in politics, economics and other social interactions.

Games are a mathematical object. While all game formalizations share the broad characterization that the agent's rewards are tied to the decisions made by others, specific games can vary greatly in their detail. For instance, is the game played once, or many times? How much about the game and the other players is known? While it is not the intention of this thesis to give an exhaustive survey of all game forms, we will describe the model for the type of game that we use in this dissertation—the repeated game. However, repeated games are built upon a more basic type of game, the one-shot game, which in turn relies on having a crisp mathematical formalization of the intuitive notion of having preferences. We will give some background for each of these topics, starting with building a formal idea of preference.

## 2.1 Preference and Utility

Game theory is founded on the idea that agents have preferences over the *outcomes* of the game—the state of the world after the game has been played. Game outcomes are a natural concept with many examples. Rock-paper-scissors can be seen as having three outcomes: player one might win, player two might win, or they might both tie. Poker can be seen as having many more possible outcomes, one for each possible distribution of chips to the various players. Each agent has preferences over these outcomes. Using the example of rock-paper-scissors, each agent prefers any outcome where they win to one where they lose or tie. Since all agents prefer the outcome where they win there is some natural tension.

Formalizing this intuition requires some light notation. Let $O$ be a set of outcomes and $\succeq \subset O \times O$ be some relationship over them. The notation $o_1 \succeq o_2$ denotes that $o_1$ is weakly preferred to $o_2$. Strict preference is denoted $o_1 \succ o_2$ and neutrality is denoted $o_1 \sim o_2$. A game might have some random component or the agents themselves might do something random, and this leads to the idea of a *lottery* over outcomes. A lottery is simply a multinomial probability distributions over a set of outcomes. We denote a lottery over outcomes as $\{o_1, \ldots, o_n\}$ as $[p_1 : o_1, \ldots, p_n : o_n]$ where $p_i$ is the probability that the outcome of the lottery will be $o_i$.

Not every relationship among the outcomes makes sense as a preference relationship. We want preference relationships to capture the intuitive sense of 'preferring'. In particular, we want to rule out relationships that make incomprehensible statements about preference. Consider the relationship '$\triangleright$': $o_1 \triangleright o_2$ and $o_2 \triangleright o_3$, but not $o_1 \triangleright o_3$—while '$\triangleright$' is a perfectly valid relationship, but it is hard to imagine having non-transitive preference over outcomes so '$\triangleright$' does not make sense as a preference relationship.

The most accepted account of what special properties make a relationship a preference is due to the six von Neummann and Morgenstern axioms [52]. These axioms are provided here for flavour and completeness. We will largely take them as writ in this thesis with one exception noted below.

I **Completeness**: Pick two outcomes. Either one is preferred to the other or they are equally preferred.

II **Transitivity**: If outcome $o_1 \succeq o_2$, and $o_2 \succeq o_3$, then necessarily $o_1 \succeq o_3$.

III **Substitutability**: If $o_1 \sim o_2$, then
$[p : o_1, p_3 : o_3, \ldots, p_n : o_n] \sim [p : o_2, p_3 : o_3, \ldots, p_n : o_n]$.

IV **Decomposability**: If two lotteries give equal probabilities to all outcomes then they are equally preferred. Essentially there is "no fun in gambling"—a game of craps and a slot machine are equally preferred if they pay out the same amounts with the same probabilities.

V **Monotonicity**: If $o_1 \succeq o_2$ and $p > q$
then $[p : o_1, (1-p) : o_2] \succeq [q : o_1, (1-q) : o_2]$.

VI **Continuity**: If $o_1 \succ o_2$, and $o_2 \succ o_3$, then there has to be a lottery such that $o_2 \sim [p : o_1, (1-p) : o_3]$

Figure 2.1: von Neummann and Morgenstern's six preference axioms.

Of course, some of these axioms could be disputed. For instance one might want to have a model of preference that does not assert completeness. Such a system would be able to account for situations where one might not be sure which of two outcomes they like better. This is different than merely being indifferent between two outcomes. One might be simultaneously trying to optimize along multiple dimensions: for example, a hospital administrator wants to save both money and increase the quality of care but might have trouble spelling off alternatives that trade off one against the other[2]. This idea of incomparability can be captured with a partial ordering over outcomes.

As an example, if we define a partial ordering through a partial preference relationship $\succeq$ over $\Re^2$ where outcomes are real-valued pairs and $(a, b) \succeq (a', b')$ if and only if $a \geq a'$ and $b \geq b'$. With such an ordering, $(2, 4)$ is preferred to $(1, 1)$, but there is no preference relationship between $(2, 4)$

[2]Although, estimates for the *Value of Statistical Life*—the projected marginal cost of a human life [51]—are based on the fact that money and care quality do get traded off in practice.

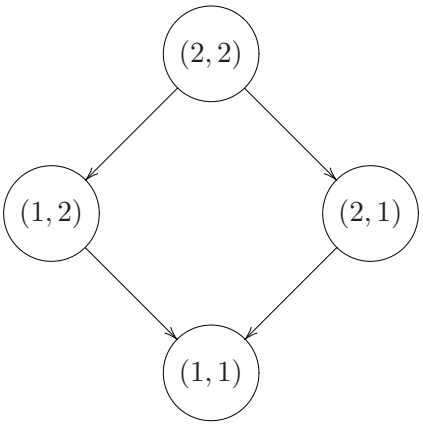


Figure 2.2: An example of a partial ordering over elements from $\Re^2$, where the edges indicate that the source node is preferred to the sink node.

and $(3, 0)$. Partial orderings can be conveniently visualized using a graph, such as in Figure 2.2. We will later argue that there are strong reasons for not having a preference between two algorithms even when we have extensive experimental data for their performance—they might be good in different situations.

If we have a preference relationship that satisfies the von Neummann and Morgenstern axioms it can be shown that there exists a *utility function* $u : O \mapsto [0, 1]$ that captures the preference relationship exactly [52]. This means that all of the structure of a preference relationship can be distilled to a mapping of outcomes to real numbers such that $o_1 \succeq o_2$ if and only if $u(o_1) \geq u(o_2)$; we can express our preference for any outcome in terms of a one-dimensional number.

We will call the value of the utility function for a particular outcome the *reward* for that outcome. This is for historical reasons and is not a substantive claim about our setting. 'Reward' is a more common term for the generic measure of worth in than 'utility' in the single-agent reinforcement learning and artificial intelligence communities, and so we will inherit this piece of terminology.

## 2.2 One-shot Games

As mentioned earlier, agents might have different preference orderings over the outcomes of a game, so what happens when agents with different preferences interact? If the agents only interact once then this conflict can be modeled as a *one-shot game*. The distinguishing characteristic of the one-shot game is that there is no notion of time or sequence of actions: the agents simultaneously act once and only once. Therefore the agents need not think about souring or fostering future relationships when acting in the game. They only need to concern themselves with how their

action affect their immediate one-off reward.

A one-shot game can be modeled with a 5-tuple: $G = \langle N, A, O, o, u \rangle$. Each element of this tuple has a simple interpretation:

- $N$ is the set of players.
- $A = \prod_{i \in N} A_i$ is the set of actions sets, where each action set $A_i$ is set of possible action that agent $i \in N$ can take. These action sets do not have to be finite or even countable but we will focus on finite actions sets in this thesis. A particular member $a \in A$ is called an *action profile*, and is a joint action decision for all players. A useful piece of notation to have is $a_{-i} \in A_{-i}$ which denotes the action choices by the opponents of a particular agent: $A_{-i} = \prod_{j \in N \setminus \{i\}} A_j$.
- $O$ is the set of outcomes.
- $o$ is a function $o : A \mapsto O$ that maps action profiles to outcomes.
- $u$ is a vector function that map outcomes of the game to rewards for each of the players *i.e.* for each agent $i \in N$, they have a utility function $u_i : O \mapsto \Re$. In this thesis, we are solely concerned with the reward associated with an outcome and so we will abbreviate $u_i(o(a))$ as $u_i(a)$.

If a player decides to play a particular action $a_i \in A_i$ in $G$, it is said to be playing a *pure strategy*. In many settings agents are allowed to randomize over decisions giving rise to *mixed strategies*. These strategies are denoted as $\sigma_i \in \Delta(A_i)$, where $\Delta(A_i)$ is the $|A_i|$-dimensional probability simplex—$\sigma_i$ is a multinomial distribution over agent $i$'s action set. Note that these probabilities are absolutely not allowed to depend on what the opponent does: the agents unfurl their fully-specified pure or mixed strategies simultaneously and independently.

The probability that any action $a_i$ will be played is $\sigma_i(a_i)$. Any action $a_i$ that has non-zero probability in $\sigma_i$ is said to be in its *support*. When playing a mixed strategy in a one-shot game, agents are concerned with maximizing their expected reward: $u_i(\sigma) = \sum_{a \in A} p(a|\sigma) u_i(a)$.

If, given the opponents' strategies $\sigma_{-i}$, agent $i$ plays a strategy $\sigma_i$ such that $u_i(\sigma_i, \sigma_{-i}) = \max_{\sigma_i'} u_i(\sigma_i', \sigma_{-i})$, then $\sigma_i$ is said to be a *best response*. The best response against a particular set of opponent strategies is not necessarily unique—there may be many strategies that attain the maximum reward.

Simple one-shot games are represented in *normal-form* tables where the rewards are explicitly written in $|N|$-dimensional vectors (one utility entry for each player) in a table with one entry per outcome. For two-players this is a 'matrix' where each element belongs to $\Re^2$. Game 2.3 is an example of a famous game, *Prisoner's Dilemma*, in normal form.

Explicitly writing out all the reward entries is wasteful in games that have highly-structured payoffs. While more parsimonious representations exist, the games examined in this thesis are largely restricted to having two players and typically small. All the games that we look at in this these are in normal form, although the discussion of a family of large $n$-player traffic games looked at in § 7 will primarily focus on issues of representation.

| | $C$ | $D$ |
|---|---|---|
| $C$ | $3, 3$ | $0, 4$ |
| $D$ | $4, 0$ | $1, 1$ |

Figure 2.3: A game of *Prisoner's Dilemma* in normal-form with canonical payoffs.

### 2.2.1 Solution Concepts

Once a game has been defined, a natural question is: how it should the agents act? Game theory is devoted to answering this question in the context of rational agents. A rational agent is an entity that ruthlessly and perfectly maximizes expected utility given its knowledge of the game and its opponent. Unfortunately, there is no single 'solution' to a game, but rather different families of *solution concepts* which are sets of strategies or strategy profiles that satisfy some game theoretic property. The most famous of these is the *Nash equilibrium*, but we will first look at two other solution concepts that will be important to later discussions: non-dominated outcomes and maxmin strategies.

#### Non-dominated Outcomes

Some actions make never make any sense to play. An important class of these are the dominated actions. An action is (strictly) dominated if there is another pure or mixed strategy that yields higher reward regardless of what the opponents do. This is formally captured as

$$a_i \text{ Dominates } a_i' \equiv \forall \sigma_{-i}, u_i(a_i, \sigma_{-i}) \geq u_i(a_i', \sigma_{-i}). \tag{2.1}$$

No rational agent ever plays a dominated action and we can discard any outcome that result from dominated action profiles. The idea of weak domination is similar, but the agent can be neutral between the dominating and dominated strategy for some (but not all) profiles of opponent strategies.

For example, in *Prisoner's Dilemma*, the row player's action $C$ is strictly dominated: if the column player plays $C$, row is better off playing $D$ than $C$, and if column plays $D$ row would still be better playing $D$. There is no reason that row should ever play $C$ and so $C$ can be discarded as a possible action.

While strict domination is easy to spot in a two-player two-action game, it generally has to be found with a series of linear feasibility programs—one for every action by every player. For any agent and action, we can use a feasibility program to look for some mixed strategy over the other actions that has higher expected utility for all of the opponent's actions. If there is a pure strategy like this, then the corresponding action can be thrown out.

In a two-player game if we assume that the opponent believes that the protagonist agent is rational—or reward maximizing—then the opponent must also believe that the protagonist will never play any strategy that has a strictly dominated action in its support. If we furthermore assume that the opponent is, itself, rational, then it will never play any action that is strictly dominated with respect to the protagonist's non-dominated actions. This in turn means that the protagonist will

| | $A$ | $B$ |
|---|---|---|
| $A$ | $2, 1$ | $0, 0$ |
| $B$ | $0, 0$ | $1, 2$ |

Figure 2.4: A game of *Battle of the Sexes* in normal-form with canonical payoffs.

never play any strategy that is strictly dominated with respect to the opponent's surviving actions and so forth. This process of removing actions is called *iterated domination removal* (IDR) and is repeated until no more actions can be removed for either agent. If an action has survived IDR it has attained a minimal certificate of sensibility: there is at least some belief about what the opponent will do for which this strategy is a best response[3].

**Maxmin Strategies**

If an agent has no idea what the opponents will do it can still guarantee some level of reward. One way of thinking about this problem is in terms of playing against an adversary: no matter what strategy an agent picks, the opponent just-so-happens to be playing the action that minimizes the protagonist's reward. The highest reward that an agent can get in such a situation is called the *maxmin value* or the *security value*, which is formally stated as

$$maxmin_i(G) = \max_{\sigma_i \in \prod (A_i)} \min_{a_{-i} \in A_{-i}} u_i(\sigma_i, a_{-i}). \tag{2.2}$$

Any strategy that achieves the maxmin value is called a maxmin strategy. While for some games the maxmin value is a pessimistic lower-bound, in other games agents are actually playing against opponents that are trying to minimize their reward and the maxmin strategy is well motivated. Indeed, any game with two agents competing for a common finite resource has this feature: if there is only one cake, any cake that you have is a slice that I do not have. In two-player constant sum games, the strategy profile where both players adopt a maxmin strategy is a member of the next solution concept that we will discuss—the Nash equilibrium.

**Nash Equilibria**

A Nash equilibrium is a strategy profile where all agents are playing a mutually optimal strategy—everyone is best responding to everyone else. Formally, the set of Nash equilibria is

$$NE(G) = \left\{ \sigma \in \Delta(A) \mid \forall i \in N, \forall \sigma_i' \in \Delta(A_i), u_i(\sigma) \geq u_i(\sigma_i', \sigma_{-i}) \right\}. \tag{2.3}$$

This set is always non-empty [35], but is not necessarily a singleton. For example, the *Battle of the Sexes* (Game 2.4) has three Nash equilibria: both agents playing $A$, both playing $B$ and some mixture between the two.

[3]This is not necessarily true when the number of players is greater than two.

The Nash equilibrium is a positive concept and not a predictive or normative one. If there are several Nash equilibria there is no prescription for picking which one an agent should or will play. There has been a lot of work done on restricting the set of Nash equilibria to sets of equilibria that seem 'natural' or predictive. Much of the existing work in repeated game dynamics can be seen as describing restricted classes of Nash equilibria in terms of simple adaptation rules—see, for example, Kalai and Lehrer [25], Kalai and Lehrer [26], Hart and Mas-Colell [21] and Hart and Mansour [20].

One of Nash equilibrium restrictions that we look at in this thesis is the set of Pareto-optimal Nash equilibria. Pareto-domination is a partial ordering over outcomes, where one outcome Pareto-dominates another if all the agents get weakly higher reward in the former outcome than the latter:

$$a \text{ Pareto Dominates } a' \equiv \forall i \in N, u_i(a) \geq u_i(a'). \tag{2.4}$$

Pareto-optimality refers to any outcome that is not Pareto-dominated by any other outcome: in every other outcome at least one of the agents is worse off. Indeed, the partial ordering example in Figure 2.2 can be seen as an example of Pareto-domination for two agents.

A Pareto-optimal Nash equilibrium is an equilibrium that is Pareto-optimal when outcomes are restricted to the Nash equilibria. In a Pareto-dominated equilibrium, all agents would do better if everyone switched to another equilibrium. This suggests that Pareto-optimal Nash equilibria should be a particularly stable appealing set of Nash equilibria for the agents. Note that a Pareto-optimal Nash equilibrium is not necessarily a Pareto-optimal outcome: the difference can be seen in *Prisoner's Dilemma*. $(C, C)$ is a Pareto-optimal outcome, but $(D, D)$ is the unique Nash equilibrium and so is also the Pareto-optimal equilibrium.

## 2.3 Repeated Games

One-shot games are the foundation of repeated games, in which two or more agents repeatedly (for either a finite or infinite number of iterations) play a one-shot *stage-game*. Unlike the one-shot game where the agents play the game once and then never interact again, the history of play in a repeated game is kept and agents can condition their strategies on it. This suggests that agents need to worry about how their current choice of action will effect future reward.

*Tit-for-Tat* (TfT) is an example of a strategy in a repeated game and is perhaps the most famous repeated-game strategy for *Prisoner's Dilemma* (Game 2.3). TfT begins by cooperating and then plays whatever the opponent did the past iteration. Therefore, if the opponent is obliging and cooperates TfT will continue to cooperate. If ever the opponent defects TfT will defect the next round, but will start cooperating whenever the opponent starts feeling cooperative again. TfT is a relatively simple strategy, but one could imagine constructing more and more elaborate strategies with trigger conditions and complicated modes of behaving.

Formally, let us denote the action played in iteration $t$ by agent $i$ as $a_i^{(t)}$, and the reward that $i$ receives is $r_i^{(t)}$. For the model of repeated games that we will use, agents are allowed to submit

mixed strategies, denoted $\sigma_i^{(t)}$, but agents do not receive the expected utility. Instead, an action is sampled from the mixed strategy, and payoff is calculated using the sampled action.

Agents in finitely repeated games are interested in maximizing their average reward, and agents in an infinitely repeated game are interested in maximizing their limit average reward:

$$\bar{r}_i = \lim_{T\to\infty} \left[ \frac{\sum_{t=1}^{T} r_i^{(t)}}{T} \right]. \tag{2.5}$$

In this thesis, we will exclusively focus on simulating infinitely repeated games in a finite number of iterations: agent do not believe that the game will ever end even though it does after a set number of iterations.

One interpretation of these repeated-game strategies is that they are strategies in a one-shot *supergame* where instead of making a decision each iteration conditioned on the past history, agents make a single policy decision. In this interpretation, TfT is a single action in the *Prisoner's Dilemma* supergame. There are many more policies than there are actions in the stage-game: if all players have $n$ actions and the game is repeated for $T$, then there are $O(n^{T^2})$ possible pure-action policies (policies that only ever make pure-strategy decisions). While the supergame is clearly not a compact way of representing a repeated game, it is an intuitive way of seeing that finitely repeated games must have Nash equilibria.

Infinitely repeated games also necessarily have equilibria and, furthermore, each infinitely repeated game has infinitely many equilibria. We cannot explicitly characterize every repeated game equilibria but we can instead say something about the the payoff profiles that these equilibria achieve. If all agents have an average reward above their respective security values (recall from § 2.2.1) then there exists some Nash equilibrium that attains the same profile of payoffs. This is the celebrate *Folk Theorem*[4]. Any payoff profile where all agents are attaining reward higher that their security values is said to be *enforceable*.

To gain some insight about the Folk Theorem, let us again examine the game of *Prisoner's Dilemma* and construct a repeated game Nash equilibrium where both players repeatedly play $C$. The security value for this game is $1$ (if an agent always plays $D$, then it gets a utility of at least $1$ regardless of what the opponent does), and so $(C, C)$ is clearly enforceable. Therefore, $(3, 3)$ is a payoff profile of the repeated game Nash equilibrium. What are the equilibrium strategies? There are a number of strategy profiles that can attain this outcome but one of the simplest profiles is *Grim Trigger* in self play.

Grim Trigger plays $C$ until the opponent plays $D$ and then plays $D$ forever: upon the first sign of deviation from the $(C, C)$ outcome the Grim Trigger strategy starts a merciless and unending program of punishment. Grim Trigger in self play is an equilibrium because if either player tries to deviate to another strategy that ever plays $D$ against Grim Trigger, the deviator will be worse off: the deviant strategy strategy can, at best, attain an average reward of $1$. If either player switches to a strategy that never plays $D$ against Grim Trigger, then this strategy differs only in the

[4]There are actually several Folk Theorems, but we will only look at one for average rewards.

| | 100 | 99 | ... | 1 |
|---|---|---|---|---|
| 100 | $100, 100$ | $99-\delta, 99+\delta$ | | $1-\delta, 1+\delta$ |
| 99 | $99+\delta, 99-\delta$ | $99, 99$ | | $1-\delta, 1+\delta$ |
| $\vdots$ | | | $\ddots$ | |
| 1 | $1+\delta, 1-\delta$ | $1+\delta, 1-\delta$ | | $1, 1$ |

Figure 2.5: A game of *Traveler's Dilemma* with $100$ actions.

'off-equilibrium' punishment details and receives the same average reward as Grim Trigger—the deviant strategy has not improved reward.

The Folk Theorem does not claim that if two agents are achieving an enforceable payoff profile then the agents are playing a Nash equilibrium. The idea of equilibrium is tied deeply to the threat of punishment. Grim Trigger in self play is an equilibrium while two strategies that blindly play $C$ is not, even if their behaviour looks the same to an outside observer. In the latter case if either agent switched to the simple strategy of blindly playing $D$, than their average reward would be higher.

While we might have been unsatisfied by the potential multitude of Nash equilibria in the one-shot game, the predictive power of the Folk Theorem is even less: there are games where nearly every outcome arises under a Nash equilibrium. For instance, consider an extension of *Prisoner's Dilemma*, the *Traveler's Dilemma* (Game 2.3; *Prisoner's Dilemma* is the special case with only $2$ actions—apparently travelers have more options available to them than prisoners). The security value is $1$ for both players, and so all outcomes—except for outcomes of the form $(1-\delta, 1+\delta)$ or $(1+\delta, 1-\delta)$—are potentially the result of a Nash equilibrium.

Again, repeated game Nash equilibria are positive statements and not normative, but in many cases we want a normative claim: how should we behave in a repeated game? How should we go about selecting a particular strategy for a multiagent system? This thesis is largely devoted to evaluating one approach that computer science has suggested for this problem: multiagent learning algorithms.

## 2.4 Multiagent Learning Algorithms

MAL algorithms have been studied for a long time (57 years at the time of writing) and many different algorithms exist. Not only is there a profusion of algorithms but there are also several different settings for multiagent learning. Does an algorithm know the game's reward functions before the game starts? Some authors assume yes, while others assume that these reward functions need to be learned. There are other questions. What signals of the opponent actions can an algorithm observe? Are stage-game Nash equilibria and other computationally-expensive game properties assumed to be computable? Each of these assumptions changes the learning problem. A setting where the rewards are known *a priori* is fundamentally different than a setting where the rewards are not known *a priori* and algorithms have no ability to observe the opponents' rewards.

The algorithms that we describe in this section were designed with a variety of different goals

in mind and this reflects a general disagreement over what these MAL algorithms should be trying to do. Should they be trying to converge to a stage-game Nash equilibrium? Should they try to avoid being exploited by other algorithms? Or are they trying to maximize their sequence of reward? There is no single answer (but these issues are examined at length in Shoham et al. [46] and Sandholm [44]).

Each of these goals poses different empirical questions. For instance, if we are primarily interested the Bowling and Veloso [10] criterion—all algorithms should converge to a stationary strategy and if the opponent converges to a stationary policy all algorithms should converge to a best response—one should analyze experiments using performance metrics that are sensitive to strategic stationarity and to the difference between the current strategy and the best response strategy.

In this dissertation we focus on two-player repeated games with many (*i.e.* more than two) actions per player. Other learning settings have been investigated. Some of these settings are further restrictions that insist, for example, on two-action games [47] or constant-sum games [30]. Other work looks at learning in generalizations of two-player repeated games: stochastic games or $N$-player games [53]. There are also MAL experiments that have been conducted in settings that are neither generalizations nor restrictions, such as the population-based work by Axelrod [3] and Airiau et al. [2]. Of these games, the repeated two-player game setting is the best studied and there are many recent algorithms designed for such games.

In the remainder of this section we will discuss a selection of algorithms intended for two-player repeated games and look at some previous MAL experiments. We do not mean to give an exhaustive survey of this literature but we do want to build intuition about this set of algorithms, look at the assumptions that they make and indicate some of the relationships between them.

### 2.4.1 Fictitious Play

`Fictitious play` [11] is probably the earliest example of a learning algorithm for two-player games repeated games. Essentially, `fictitious play` assumes that the opponent is playing an unknown and potentially mixed stationary strategy, and tries to estimate this strategy from the opponent's empirical distribution of actions—the frequency counts for each of its actions normalized to be probabilities. Clearly, in order to collect the frequency counts `fictitious play` must be able to observe the opponent's actions. The algorithm then, at each iteration, best responds to this estimated strategy. Because `fictitious play` needs to calculate a best response, it also assumes complete knowledge of its own payoffs.

Fictitious play is guaranteed to converge to a Nash equilibrium in self play for a restricted set of games. These games are said to have the *fictitious play property* (see, for instance Monderer and Shapley [34]; for an example of a simple $2 \times 2$ game without this property see Monderer and Sela [33]). `Fictitious play` will also eventually best respond to any stationary strategy. This algorithm's general structure has been extended in a number of ways, including *smooth fictitious play* [17], and we will see later that `fictitious play` provides the foundation for `AWESOME` and `meta`, two more modern algorithms. These algorithms are described later in Section 2.4.3.

`Fictitious play` is known to have miscoordination issues, particularly in self play. For

example, consider the *Sidewalk Game* (Game 2.6), where two identical `fictitious play` agents are faced with the issue of trying to get by each other on the sidewalk by either passing to the *West* or the *East*. Since both algorithms are identical and deterministic, these algorithms will cycle between $(W, W)$ and $(E, E)$. There are some clever measures that can be taken to avoid some of these kinds of problems (for instances, special best response tie-breaking rules and randomization), but miscoordination is a general issues with the `fictitious play` approach.

| | $W$ | $E$ |
|---|---|---|
| $W$ | $-1, -1$ | $1, 1$ |
| $E$ | $1, 1$ | $-1, -1$ |

Figure 2.6: A *Sidewalk* or *Dispersion* game, where two agents try to miscoordinate where they step.

### 2.4.2 Determined

`Determined` or 'bully' (see, for example, Powers and Shoham [38]) is an algorithm that solves the multiagent learning problem by ignoring it. MAL algorithms typically change their behaviour by adapting to signals about the game. However `determined`, as its name suggests, stubbornly does not change its behaviour and relies on other algorithms adapting their strategies to it.

`Determined` enumerates the stage-game Nash equilibria and selects the one that maximizes its personal reward at equilibrium. Certainly, `determined` is not a final solution to the MAL problem: for instance, two `determined` agents will stubbornly play different equilibria (unless there is a an equilibrium that is best for both agents), possibly leading to a situation where both algorithms receive sub-equilibrium reward. Additionally, enumerating all the Nash equilibria not only requires complete knowledge of every agents' reward functions, but also is a costly computational activity that is infeasible on anything but the smallest stage games. With that said, it is certainly an interesting learning approach to test and compare. Slight variations of `determined` are, like `fictitious play`, at the heart of `meta` and `AWESOME`.

Using a stage-game Nash equilibrium is only one way of being stubborn and getting an opponent to adapt. One could also imagine aiming for convergence to other outcomes: for instance looking for the outcome with the highest reward given that the opponent is best responding. Note that this differs from a stage-game Nash equilibrium because the `determined` algorithm does not have to be best responding itself. This amounts to an equilibrium of the Stackelberg version of the game: imagine the same game, but instead of moving simultaneously, the `determined` agent moves first. Clearly, a sensible opponent will best-respond to whatever `determined` does, and so `determined` should pick the action that gives maximum reward given that the opponent will best respond.

As an example of a Stackelberg outcome: in Game 2.7 the unique Nash equilibrium is $(B, R)$. Indeed, this is the only outcome that survives IDR. However, there is something very appealing

about the Pareto-optimal outcome at $(T, L)$: if the row player can teach the column player that it will, in fact, play $T$ then the row player will be much better off.

| | $L$ | $R$ |
|---|---|---|
| $T$ | $1-\epsilon, 1$ | $0, 0$ |
| $B$ | $1, 0$ | $\epsilon, 1$ |

Figure 2.7: A game showing a situation where a `determined`-style algorithm might be better off not best responding to its opponent.

### 2.4.3 Targeted Algorithms

We will next focus on a class of algorithms called the *targeted* algorithms. Targeted algorithms focus on playing against a particular class of opponents. For example, `AWESOME` [13] guarantees convergence when playing itself or any stationary opponent. Both these algorithms are based around identifying what the opponent is doing, with particular attention paid to stationarity and Nash equilibrium, and then changing their behaviour based on this assessment.

Meta [38] switches between three simpler strategies: a strategy similar to `fictitious play` (there are some small differences in how best responses are calculated), a `determined`-style algorithm that stubbornly plays a Nash equilibrium, and the maxmin strategy. Average reward and empirical distributions of the opponents' actions are recorded for different periods of play. Based on these histories one of the three algorithms is selected. `Meta` was theoretically and empirically shown to be nearly optimal against itself, close to the best response against stationary agents, and to approach (or exceed) the security level of the game in all cases.

`AWESOME` also tracks the opponent's behaviour in different periods of play and tries to maintain hypotheses about their play. For example, it attempts to determine whether the other algorithms are playing a special stage-game Nash equilibrium. If they are, `AWESOME` responds with its own component of that special equilibrium. This special equilibrium is known in advance by all implementations of `AWESOME` to avoid equilibrium selection issues in self play. There are other situations where it acts in a similar fashion to `fictitious play`, and there are still other discrete modes of play that it engages in depending on what hypotheses it believes.

Because both of these algorithms switch between using simpler strategies depending on the situation, these algorithms can be viewed as portfolio algorithms. Here, they both manage similar portfolios that include a `determined`-style algorithm and a `fictitious play` algorithm.

### 2.4.4 Q-learning Algorithms

A broad family of MAL algorithms are based on `Q-learning` [55]: an algorithm for finding the optimal policy in Markov Decision Processes (MDPs; can be thought of as single-agent stochastic games). This family of MAL algorithms does not explicitly model the opponent's strategy choices.

They instead settle for learning the expected discounted reward for taking an action and then following some set policy: the $Q$-function. In order to learn the $Q$-function, algorithms typically take random exploratory steps with a small (possibly decaying) probability.

Each algorithm in this family has a different way of selecting its strategy based on this $Q$-function. For instance, one could try a straight forward adaptation of the single-agent `Q-learning` to the multiagent setting by ignoring the impact that the opponent's action makes on the protagonist's payoffs. The algorithm simply updates its reward function whenever a new reward observation is made, where the new estimate is a convex combination of the old estimate and the new information:

$$Q(a_i) = (1 - \alpha_t)Q(a_i) + \alpha_t \left[ r + \gamma \max_a Q(a) \right]. \tag{2.6}$$

This algorithm essentially considers the opponent's behaviour to be an unremarkable part of a noisy and non-stationary environment. The non-stationarity of the environment makes learning difficult but this idea is not entirely without merit: `Q-learning` has been shown to work in other non-stationary environments (see, for instance, Sutton and Barto [49]).

`Minimax-Q`[30] is one of the first explicitly multiagent applications of this idea. The $Q$-function that it learns is based on the action profile and not just the protagonist action: it learns $Q(a_i, a_{-i})$. Minimax-Q uses the mixed maxmin strategy calculated from the $Q$-function as its strategy:

$$\begin{aligned} Q(a_i, a_{-i}) = {} & (1 - \alpha_t)Q(a_i, a_{-i}) \\ & + \alpha_t \left[ r + \gamma \max_{\sigma_i \in \prod(A_i)} \left[ \min_{a_{-i} \in A_{-i}} \sum_{a_i} \sigma_i(a_i) Q(a_i, a_{-i}) \right] \right]. \end{aligned} \tag{2.7}$$

It should be noted that since its maxmin strategies are calculated from learned $Q$-values, they may not be the game's actual maxmin strategies and thus fail to attain the security value. Like `Q-learning`, `minimax-Q` also takes the occasional exploration step.

There are further modifications to this general scheme. `NashQ` [24] learns $Q$-functions for it and its opponents and plays a stage-game Nash equilibrium strategy for the game induced by these $Q$-values. `Correlated-Q` [19] does something similar except that it chooses from the set of correlated equilibria using a variety of different selection methods. Both of these algorithms assume that they are able to observe not only the opponents' actions but also their rewards, and additionally that they have the computational wherewithal to compute the necessary solution concept.

### 2.4.5 Gradient Algorithms

Gradient ascent algorithms, such as `GIGA-WoLF` [7] and $\texttt{RV}_{\sigma(t)}$ [5], maintain a mixed strategy that is updated in the direction of the payoff gradient. The specific details of this updating process depend on the individual algorithms, but the common feature is that they increase the probability of actions with high reward and decrease the probability of unpromising actions. This family of algorithms is similar to `Q-learning` because they do not explicitly model their opponent's strategies and instead treat them as part of a non-stationarity environment.

GIGA-WoLF is the latest algorithm in the line of gradient learners that started with IGA [47]. GIGA-WoLF uses an adaptive step length that makes it more or less aggressive about changing its strategy. It compares its strategy to a baseline strategy and makes the update larger if it is performing worse than the baseline. GIGA-WoLF guarantees non-positive regret in the limit (regret is discussed in greater detail in § 5.2) and strategic convergence to a Nash equilibrium when playing against GIGA [57] in two-player two-action games.

There are two versions of GIGA-WoLF. The first version assumes prior knowledge of personal reward and the ability to observe the opponent's action—this is the version used in the proofs for GIGA-WoLF's no-regret and convergence guarantees. There is also a second version—on which all the experiments were based—that makes limited assumptions about payoff knowledge and computational power. Instead, like Q-learning, it merely assumes that it is able to observe its own reward.

RV$_{\sigma(t)}$ [5] belongs to a second line of gradient algorithms initiated by ReDVaLeR [4]. This algorithm also uses an adaptive step size when following the payoff gradient, like GIGA-WoLF, but this is done on a action-by-action basis. This means that, unlike GIGA-WoLF, RV$_{\sigma(t)}$ can be aggressive in updating some actions while being cautious about updating others, and it does this by comparing its reward to the reward at a Nash equilibrium. Therefore, RV$_{\sigma(t)}$ requires complete information about the game and sufficient computational power to discover at least one stage-game Nash equilibrium. RV$_{\sigma(t)}$ also guarantees no-regret in the limit and additionally provides some convergence results for self play for a restricted class of games.

GIGA-WoLF and RV$_{\sigma(t)}$ differ in the way that they ensure that their updated strategies are still probabilities. GIGA-WoLF retracts: it maps an unconstrained vector to the vector on the probability simplex that is closest in $\ell_2$ distance. This approach has a tenancy to map vectors to the extreme points of the simplex, reducing some action probabilities to zero. RV$_{\sigma(t)}$ normalizes, which is less prone to removing actions from its support. This difference may explain some of the experimental results later on.

### 2.4.6 Previous Experimental Results

Setting up a general-sum repeated two-player game experiment requires a number of design choices. Say that one has an algorithms to be evaluated in terms of a particular performance metric: what set of games should these algorithms be run on? What other algorithms should this performance be compared to? If one is dealing with randomized algorithms (which includes any algorithm that is able to submit a mixed strategy), how many different runs should be simulated? For a particular game, how many iterations should a simulation be run for? As can be seen in Table 2.4.6, existing literature varies in all of these dimensions. Additionally, some papers do not even discuss parameters used which makes it difficult to reproduce experiments.

Overall, most of the tests performed in these papers considered few algorithms. In most of these experiments, the newly proposed algorithms were only evaluated by playing against one or two opponents. Some papers—like Littman [30] and Greenwald and Hall [19]—seemed to use many algorithms, but in fact these algorithms were quite similar to each other and varied only in

| Paper | Algorithms | Distributions | Instances | Runs | Iterations |
|---|---|---|---|---|---|
| Littman [30] | 6 | 1 | 1 | ? | ? |
| Claus and Boutilier [12] | 2 | 3 | 1 - 100 | ? | 50-2500 |
| Greenwald and Hall [19] | 7 | 5 | 1 | 2500 - 3333 | $1 \times 10^5$ |
| Bowling [8] | 2 | 6 | 1 | ? | $1 \times 10^6$ |
| Nudelman et al. [36] | 3 | 13 | 100 | 10 | $1 \times 10^5$ |
| Powers and Shoham [38] | 11 | 21 | ? | ? | $2 \times 10^5$ |
| Banerjee and Peng [5] | 2 | 1 | 1 | 1 | 16000 |
| Conitzer and Sandholm [13] | 3 | 2 | 1 | 1 | 2500 |

Table 2.1: This table shows a summary of the experimental setup for a selection of papers. The summary includes the number of algorithms, the number of game distributions, the number of game instances drawn from these distributions, the number of runs or trials for each instance, and the number of iterations that the simulations were run for. In some cases, the setup was unclear, indicated with a '?'. In many cases, fewer than $[Algorithms \times Distributions \times Instances \times Runs]$ runs were simulated, due to some sparsity in the experimental structures.

some small details. For example, in Littman [30] two versions of `minimax-Q` and two versions of `Q-learning` were tested and each version varied only by its training regime. In Greenwald and Hall [19], four versions of `Correlated-Q` were tested against `Q-learning` and `Friend-Q` and `Foe-Q` (the last two are from Littman [29]). Powers and Shoham [38] implemented the greatest variety of opposing algorithms out of these experiments. While four of the eleven tested were simple stationary strategy baselines, the remaining seven were MAL algorithms including `Hyper-Q` [50], `WoLF-PHC` [10], and a joint action learner [12].

Experiments have also tended to involve small numbers of games instances, and these instances have tended to have been drawn from an even smaller number of game distributions. For example, Banerjee and Peng [5] used only a single $3 \times 3$ action "simple coordination game" and Littman [30] probed algorithm behaviour with a single grid-world version of soccer. For earlier papers, this partially reflected the difficulty of creating a large number of different game instances for use in tests. However with the creation of GAMUT [36], a suite of game generators, generating large game sets is now easy and involves little investment in time. Indeed, Nudelman et al. [36] performed a large MAL experiment using three MAL algorithms (`minimax-Q`, `WoLF` [9], and `Q-learning`) on $1300$ game instances drawn from thirteen distributions. Some recent papers have taken full advantage of the potential of GAMUT, such as Powers and Shoham [38], but adoption has not been universal.

Experiments have also differed substantially in the number of iterations considered ranging from 50 [12] to $1 \times 10^6$ [8]. Iterations in a repeated game are usually divided into "settling in" (also calleda "burn-in" period) and "recording" phases , allowing the algorithms time to settle or adapt before results are recorded. Powers and Shoham [38] recorded the final $20\,000$ out of $200\,000$ iterations and Nudelman et al. [36] used the final $10\,000$ iterations out of $100\,000$.

# Chapter 3

# Platform

Unfortunately, empirical experiments have largely been run with one-off code tailored to showing a particular feature of an algorithm. This has a number of negative consequences. First, it decreases the reproducibility of experiments by, for instance, obscuring the details of algorithm implementation. Even when source code for the original experiment is available, it might be difficult to extend to new experimental settings; having to recode apparatus reduces the flexibility of experimental design. If one experiment hints at an unexpected result it is more difficult to flesh out this behaviour with a new experiment if it involves recoding the platform. Finally, rewriting the same code again and again wastes time that could be spent running more comprehensive experiments.

## 3.1 The Platform Architecture

In this section, we describe an open and reusable platform that we call MALT 2.0 (Multiagent Learning Testbed) for running two-player, general-sum, repeated-game MAL experiments. Basic visualization and analysis features are also included in this platform. This is the second version of the platform (the original version is described in Lipson [28]) and this new version is a complete recoding of the platform[5].

What we intend with MALT is not a finished product, but a growing repository of tools, algorithms and experimental settings (such as $N$-player repeated games or stochastic games). Essentially we want this version of MALT to be a base upon which other researchers can add and share tools.

### 3.1.1 Definitions

In order to clarify our discussion of running a experiment on a particular game with a particular set of algorithms, it is useful to define some terms. An ordered pair of two algorithms is a *pairing*. This pair is ordered because many two-player games are asymmetric: the payoff-structure for the row player is different than the payoff structure for the column player. The case where an algorithm is paired with a copy of itself (but with different internal states and independent random seeds) is *self play*.

[5] In this version, we recoded each one of the algorithms carefully from the original pseudo-code, completely redesigned the repeated game simulator, and created an entirely new visualization interface. In short, none of the original source code remains.

We largely concentrate on drawing games from distributions called *game generators*. A particular sample from a game generator is a *game instance*. *Prisoner's Dilemma* is a game generator and an example game instance is a particular set of payoffs that obey the *Prisoner's Dilemma* preference ordering. However, *Prisoner's Dilemma* is a simple example and not all game instances from the same generator are as closely tied.

A pairing and a game instance, taken together, are called a *match*. A match with one of the algorithms in the pairing left unspecified is a *partially specified match* (PSM). If two algorithms play the same PSM, we will conclude that any differences between their performances are due to the algorithms themselves (including any internal randomization) because everything else was held constant between the two matches.

A particular simulation of a match is called a *run* or *trial*. For deterministic algorithms, a single run is sufficient to understand the performance of that match, but for randomized algorithms (including any algorithm that plays a mixed strategy) multiple runs may each display different behaviour. In such cases, the solution quality distribution (SQD)[6]—the empirical distribution of a performance metric—for the match should be compared. Each run consists of a number of *iterations*. During an iteration, the algorithms submit their actions to a stage game and receive some feedback—such as observing their reward or what action the opponent played. Algorithms are allowed to submit a mixed strategy in which case a single action is sampled from the mixing distribution by the game. The iterations are separated into *settling-in iterations* and *recorded iterations*.

### 3.1.2 Platform Structure

In this section we give an overview of the structure of the platform. The five steps in running an experiment with the platform are summarized in Figure 3.1. There are three major components to this platform: the configuration GUI, the actual experiment engine (the piece that simulates the repeated games) and the visualization GUI.

The first step to running an experiment is to specify its parameters. There are three parts to this and the configuration GUI guides the process. The first step is to pick a group of algorithms and set their parameters. The second step is to select the GAMUT game distributions used and choose the parameters for these games. The third step is to establish general experimental parameters, such as the number of iterations for each simulation.

We have tried to make it as easy as possible to add new algorithms to MALT. Adding a new algorithm to the GUI is as simple as providing a text file with a list of parameters. Adding the algorithm to the actual engine requires minimal additional coding beyond the implementation of the algorithm.

The performance of many algorithms are likely very dependent on these parameters, however it is out of the scope of this thesis to conduct a sensitivity analysis or to tune these parameters. Along with each algorithm, we have provided some default parameters for these implementations

[6]We call these distribution "solution quality distributions" despite the fact that in MAL there is no clear idea of a 'solution' to a game. These distributions could be more meaningfully called 'metric distributions', however, 'SQD' is the terminology traditionally used in empirical algorithmics and so we adopt this language.

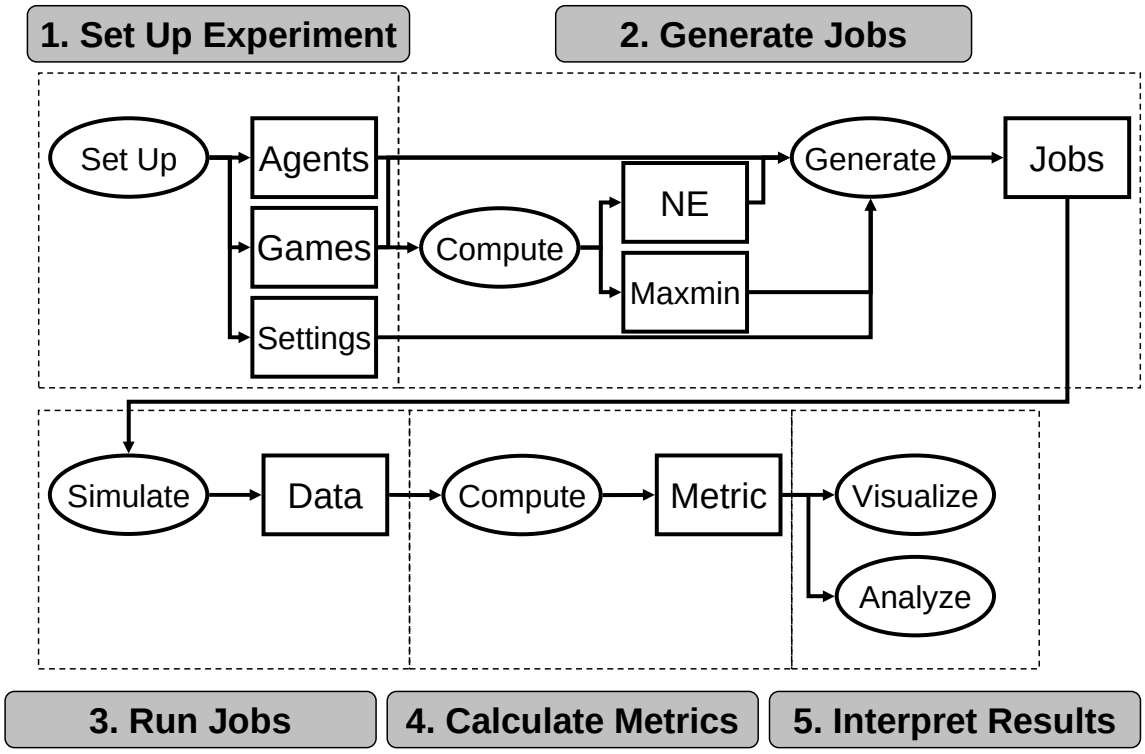


Figure 3.1: The five steps in running an analyzing an experiment using MALT.

that were taken from the original papers or were set to values that seemed reasonable from the description of the algorithm when this was not possible. These parameter settings can be easily changed using the GUI.

Once the experimental setup has been finalized, text files are generated for the agent configurations and the game instances. These files are easy to edit and they are also easy to generate using scripts that bypass the GUI. MALT uses GAMBIT's [32] implementation of Lemke-Howson [27] to find the set of Nash equilibria for each game instance and an internal linear program finds the maxmin strategies (however, MALT requires CPLEX to solve this and other linear programs). A job file is generated for each match. Each job file references the agent, game, equilibrium, and maxmin-strategy files. These files are referenced and this makes altering the job files simple even after the job files have been generated.

This set of job files may be run a number of ways. The most basic is to run them in a batch job, however for large experiments this may be prohibitively expensive. Because each job is independent, a cluster may be used. Each job creates an individual data file upon completion that records the history of play, so they may be run in any order. For each recorded iteration and for each agent in the pair, the strategy, sampled action, reward received, and beliefs about the opponents are recorded.

After the data files have been generated the performance metrics are calculated. A plain-text file describes the metrics to be calculated: *e.g.* if we are looking at some kind of convergence, we might want to specify that the $\ell_\infty$ sense of distance should be used. Calculating the metrics can proceed in serial or it can be run on a cluster. MALT includes basic tools for analyzing and visualizing these results, and there is a visualization GUI that guides the use of these tools.

### 3.1.3 Algorithm Implementations

To carry out this study, we selected and implemented eleven MAL algorithms. A brief description of each is useful for intuition.

#### `Fictitious play`

In our implementation of **`fictitious play`**, the initial action frequencies are set to one for each action, which is a uniform and easily overwhelmed prior. Tie-breaking (selecting among members of the best-response set) favours the previous action to encourage stability. For instance, if `fictitious play` plays $a_i$ in iteration $t$ and at iteration $t+1$ the best response set include $a_i$, the algorithm will chose $a_i$. If the best response set at $t+1$ does not include $a_i$, then the algorithm uniformly mixes between best responses.

| Design Decision | Setting |
|---|---|
| BR Tie-Breaking | Previous action if still BR |
| | Uniform otherwise |
| Initial Beliefs | Unit virtual action count |

Table 3.1: Design decisions for `fictitious play`

### `Determined`

Our implementation of **`determined`** repeatedly plays the Nash equilibrium that obtains the highest personal reward, but if there are multiple equilibria with the same protagonist reward, then the equilibrium with the highest opponent reward is selected. If there are any equilibria that are still tied we use the one found first by GAMBIT's implementation of Lemke-Howson.

| Design Decision | Setting |
|---|---|
| NE Tie-Breaking | Highest opponent utility |

Table 3.2: Design decisions for `determined`

### `AWESOME`

**`AWESOME`** is implemented according to the pseduo-code in Conitzer and Sandholm [13]. We use the parameter settings suggested in Conitzer and Sandholm [13] as its default. For picking the 'special' equilibrium we use the first equilibrium found by GAMBIT's implementation of Lemke-Howson. It would be interesting to compare our implementation of `AWESOME` to one that used the more computationally expensive approach of picking, say, a socially optimal equilibrium.

There is a a small performance difference between our implementation of `AWESOME` and the original implementation from Conitzer and Sandholm [13][7]. A small test—involving ten different game instances from a variety of generators and 100 runs against the random agent—showed that on three instances there was a significant difference between their solution quality distributions. A two-sample Kolmogorov-Smirnov independence test (see § 4.3.1) with $\alpha = 0.05$ was used to check for significance. For these three game instances, our implementation probabilistically dominated (see § 4.4) the original implementation in terms of reward (every reward quantile was higher for our implementation). We were not able to track down the source of this behaviour difference; however we spend a considerable amount of time verifying our implementation against the pseudocode in the paper, and we are convinced that it is correct.

[7] The original implementation was in C and MALT 2.0 is written in Java, so the original implementation could not be used directly.

| **Design Decision** | **Setting** |
|---|---|
| Special Equilibrium($\pi_p^*$) | First found |
| Epoch period ($N(t)$) | $\left\lceil \frac{\lvert A\rvert_\Sigma}{\left(1-\frac{1}{2^{t-2}}\right)(\epsilon_e^t)^2} \right\rceil$ |
| Equilibrium threshold ($\epsilon_e(t)$) | $\frac{1}{t+2}$ |
| Stationarity threshold ($\epsilon_s(t)$) | $\frac{1}{t+1}$ |

Table 3.3: Design decisions for `meta`

### `meta`

**`Meta`** is implemented according to the pseduo-code in Powers and Shoham [38]. The Powers and Shoham [38] implementation of `meta` used a distance measure based on the Hoeffding Inequality, even though the pseudo-code called for using an $\ell_2$ norm. We follow the pseudo-code and use the $\ell_2$ norm. We do not adjust the default threshold level ($\epsilon_3$) for distance and left it at the original value.

| **Design Decision** | **Setting** |
|---|---|
| Security threshold ($\epsilon_0$) | $0.01$ |
| Bully threshold ($\epsilon_1$) | $0.01$ |
| "Generous" BR parameter ($\epsilon_2$) | $0.005$ |
| Stationarity threshold ($\epsilon_3$) | $0.025$ |
| Coordination/exploration period ($\tau_0$) | $90\ 000$ |
| Initial period ($\tau_1$) | $10\ 000$ |
| Secondary period ($\tau_2$) | $80\ 000$ |
| Security check period ($\tau_3$) | $1\ 000$ |
| Switching probability ($p$) | $0.00005$ |
| Window ($H$) | $1\ 000$ |
| $\lVert\cdot\rVert$ | $\ell_2$ |

Table 3.4: Design decisions for `meta`

#### Gradient Algorithms

Our implementation of **`GIGA-WoLF`** follows the original pseudo-code and uses the learning rate and step size schedules from the original experiments as defaults. These step sizes, however, were set for drawing smooth trajectories and not necessarily for performance. Additionally, the original experiments for `GIGA-WoLF` involved more iterations than we simulated: we used $10^5$ iterations in our experiments as opposed to $10^6$ in Bowling [7]. It is possible that a more aggressive set of parameters (e.g. larger $\eta_t$) might improve some facets of performance. We, however, stuck with the original parameter settings for our implementation and defer parameter tuning questions to future work.

For GIGA-WoLF's retraction map operation (the function that maps an arbitrary vector in $\Re^n$ to the closest probability vector in terms of $\ell_2$ distance) we used an algorithm based on the method described in Govindan and Wilson [18]. GIGA-WoLF has two variants: in one it assumes that it can counterfactually determine the reward for playing an arbitrary action in the previous iteration, and in the other it only knows the reward for the the action that it played and has to approximate the rewards for the other actions. The formula for this approximation is given by

$$\forall \dot{a} \in A_i \; \hat{r}_{\dot{a}}^{(t+1)} = (1-\alpha) r^{(t)} \mathbb{I}_{\dot{a}=a^{(t)}} + \alpha(\hat{r}_{\dot{a}}^{(t)}). \tag{3.1}$$

In this equation, $r^{(t)}$ is the reward that the algorithm experienced while playing action $a^{(t)}$ in iteration $t$. The vector $\hat{r}^{(t)}$ is an $|A_i|$-dimensional vector that reflects the algorithm's beliefs about rewards. We implemented the latter approach, as all of GIGA-WoLF's experimental results are produced by this version.

| Design Decision | Setting |
|---|---|
| Learning rate ($\alpha(t)$) | $\frac{1}{\sqrt{\frac{t}{10}+100}}$ |
| Step size ($\eta(t)$) | $\frac{1}{\sqrt{10^4 t+10^8}}$ |

Table 3.5: Design decisions for GIGA-WoLF.

We also tested an algorithm, **GSA** (Global Stochastic Approximation Spall [48]; to our knowledge this was first suggested for use in a MAL setting by Lipson [28]), which is a stochastic optimization method that resembles GIGA, but takes a noisy, rather than deterministic, step. The GSA strategy is updated according to Equation 3.2. In

$$x^{(t+1)} = P(x^{(t)} + \eta^{(t)} r^{(t)} + \lambda^{(t)} \zeta^{(t)}), \tag{3.2}$$

$x_t$ is the previous mixed strategy, $r_t$ is the reward vector, $\zeta_t$ is a vector where each component is sampled from the standard normal distribution (with variance controled by the parameter $\lambda^{(t)}$), and $P(\cdot)$ is the same retraction function used for GIGA-WoLF.

| Design Decision | Setting |
|---|---|
| Learning rate ($\alpha(t)$) | $\frac{1}{\sqrt{\frac{t}{10}+100}}$ |
| Step size ($\eta(t)$) | $\frac{1}{\sqrt{10^4 t+10^8}}$ |
| Noise Weight ($\lambda(t)$) | $\frac{1}{\sqrt{10^5 t+10^8}}$ |

Table 3.6: Design decisions for GSA.

**RV**$_{\sigma(t)}$ is a implementation of the algorithm given in Banerjee and Peng [5]. Some initial experiments showed that the settings of the algorithm used in the paper performed poorly, and we consequently used some hand picked parameter settings that were more aggressive and seemed to perform better.

| **Design Decision** | **Setting** |
|---|---|
| $\sigma$-schedule ($\sigma(t)$) | $\frac{1}{1+\frac{1}{25}\sqrt{t}}$ |
| Step size ($\eta(t)$) | $\frac{1}{\sqrt{1000t+10^5}}$ |

Table 3.7: Design decisions $\mathsf{RV}_{\sigma(t)}$.

### `Q-Learning`

Our implementation of **`Q-learning`** is very basic. Since in a repeated game there is only one 'state', `Q-learning` essentially keeps track of $Q$-values for each of its actions. We use an $\epsilon$-greedy exploration policy (perform a random action with probability $\epsilon$) with a decaying $\epsilon$. $400$ exploration steps are expected for this $\epsilon$-schedule, and $\epsilon$ drops below a probability of $0.05$ at approximately iteration $2800$. It is negligible at the end of the settling-in period (less than $3E{-}9$). The learning rate ($\alpha$) decays to $0.01$ at the end of the settling in period. The discount factor of $\gamma = 0.9$ was set rather arbitrarily. There is no need to trade off current reward with future reward: all actions take the algorithm back to the same state.

| **Design Decision** | **Setting** |
|---|---|
| Learning rate ($\alpha(t)$) | $\left(1-\frac{1}{2000}\right)^t$ |
| Exploration rate ($\epsilon(t)$) | $\frac{1}{5}\left(1-\frac{1}{500}\right)^t$ |
| Future discount factor ($\gamma$) | $0.9$ |

Table 3.8: Design decisions for `Q-learning`.

### `Minimax-Q` and `Minimax-Q-IDR`

For **`minimax-Q`**, we solved a linear program to find the mixed maxmin strategy based on the $Q$-values. This program was

$$\begin{aligned} &\textbf{Maximize} && U_1 \\ &\textbf{subject to} && \textstyle\sum_{j\in A_1} u_1(a_1^j, a_2^k)\cdot\sigma_1^j \geq U_1 \quad \forall k \in A_2 \\ & && \textstyle\sum \sigma_1^j = 1 \\ & && \sigma_1^j \geq 0 \qquad\qquad\qquad\quad \forall j \in A_1 \end{aligned}$$

(see, for example, Shoham and Leyton-Brown [45]). The learning rate, exploration rate, and future discount factor are identical to `Q-learning`. We also look at a variant of `minimax-Q` called **`minimax-Q-IDR`** that iteratively removes dominated actions. In each step of the iterative IDR algorithm a mixed-strategy domination linear program (see, for example, Shoham and Leyton-Brown [45]). Both programs are solved with CPLEX 10.1.1.

### `Random`

The final algorithm, **`random`**, is an simple baseline that uniformly mixes over the available actions. Specifically, it submits a mixed strategy $\sigma$ where $\forall a \in A,\ \sigma(a) = \frac{1}{|A|}$.

# Chapter 4

# Empirical Methods and Setup

The primary purpose of MALT is to facilitate the creation of experiments. The next two chapters demonstrate what MALT can do. In this chapter, we describe the setup of a large experiment—indeed this experiment is the largest MAL experiment on many dimensions—that is aimed at comparing the performance of different MAL algorithms using a variety of different metrics. We also focus on building some tools that allow us to make our second contribution: comprehensively comparing existing MAL algorithms using a variety of different metrics. This analysis also shows the relationship between reward and some of the alternative metrics that other authors have used. The following chapter ($\S$ 5) is devoted to explaining these results.

## 4.1 Experimental Setup

We used MALT to set parameters for the eleven algorithms described previously. The goal of this experiment was to find the algorithms that are best against particular opponents on different types of games. In order to do this, we ran each of the eleven algorithms on a number of matches, and compared their results.

To test a variety of game instances we used $13$ game generators from the GAMUT game collection (see Table 4.1). From these generators we generated a total of $600$ different game instances. The generators selected were diverse and created instances that belonged to many families of games. We do not describe the details of each generator (these descriptions are available in GAMUT's online documentation), but we do discuss their relevant features when they are important for understanding the results. The game instances rewards were normalized to the $[0, 1]$ interval, in order to make the results more interpretable and comparable.

We examined five different action set sizes: $2 \times 2$, $4 \times 4$, $6 \times 6$, $8 \times 8$ and $10 \times 10$. For each size, we generated 100 game instances, drawing uniformly from the first twelve generators. An additional 100 instances were drawn from the last distribution, D13, which is a distribution of all strategically distinct $2 \times 2$ games [41]. The distribution induced by mixing over all 13 GAMUT generators is called the *grand distribution*.

With eleven algorithms and $600$ game instance there were $11 \times 11 \times 600 = 72\,600$ matches. Each match was run once for $100\,000$ iterations, although the first $90\,000$ iterations that were spent adapting were not recorded for analysis. Each match could have been run multiple times instead of just once, and indeed doing so would have been essential to understanding how any particular match behaves if at least one of the algorithms is randomized. However, conducting more runs per matches would mean that for the same amount of CPU time we would have had to either

| | |
|---|---|
| D1 | A Game With Normal Covariant Random Payoffs |
| D2 | Bertrand Oligopoly |
| D3 | Cournot Duopoly |
| D4 | Dispersion Game |
| D5 | Grab the Dollar |
| D6 | Guess Two Thirds of the Average |
| D7 | Majority Voting |
| D8 | Minimum Effort Game |
| D9 | Random Symmetric Action Graph Game |
| D10 | Travelers Dilemma |
| D11 | Two Player Arms Race Game |
| D12 | War of Attrition |
| D13 | Two By Two Games |

Table 4.1: The number and name of each game generator.

experiment with fewer games or fewer algorithms. We chose not to do this and traded a better understanding of how particular matches behaved in return for more data from more varied game instances and algorithms. Furthermore, we show in Appendix A that not stratifying (holding one experimental variable fixed while varying another; as opposed to varying both) on game instances reduces variance for many estimates of summary statistics like mean and median. Since we only ran each algorithm once on each PSM (a partially specified match), we use the terms 'run' and 'PSM' interchangeably when we discuss the results.

This experiment generated a lot of data. In order to interpret the results precisely we used several different empirical methods. Each is motivated by a particular problem that we encountered in the analysis.

## 4.2 Bootstrapping

If we conduct an experiment where two algorithms are run on a number of PSMs then a natural way to compare their performance is to compare the sample means of some measure of their performance (average reward, for example). However, if we have the conclusion that 'the sample mean of algorithm $A$ is higher than the sample mean of algorithm $B$', how robust is this claim? If we ran this experiment again are we confident that it would support the same conclusion?

A good way to check the results of an experiment is to run it multiple times. If the conclusion is the same each time than we can be fairly confident that the conclusion is true. Let's say we run an experiment $100$ times, and we found that $95\%$ of the experiments had a sample mean for algorithm $A$ of between $[\underline{a}, \overline{a}]$, that $95\%$ of the experiments had a sample mean for algorithm $B$ of between $[\underline{b}, \overline{b}]$. If $\underline{a} > \overline{b}$ (the lower bound of $A$'s interval is great than the upper bound of $B$'s) then we can be confident that $A$ is better in terms of mean. These intervals are the $95\%$ percentile intervals

of the mean estimate distribution, and the fact that they do not overlap will be taken as sufficient evidence for there to be a significant relationship between the means.

While this repeated experimentation is sufficient to allay concerns of insignificant results, it is also expensive. To verify the summary statistics from one experiment, we had to run many more ($99$ in the above example). This is not always possible (our experiments took 7 days on a large computer cluster, so to rerun them a hundred more times would have taken the better part of two years) and is certainly never desirable. Is there a way to use the data from one experiment and still construct confidence intervals of summary statistics? The answer is yes, and one way to do this is through the powerful technique of bootstrapping.

Given an experiment with $m$ data points, we can 'virtually' rerun the the experiment by subsampling from the empirical distribution defined by those $m$ points. For example, if we have a sample with $100$ data points, we could subsample $50$ data points (with replacement) from these $100$ and look at the statistic for this subsample. We can cheaply repeat this procedure as many times as we like, creating a distribution for each estimated statistic. From these bootstrapped estimator distributions we can form bootstrapped percentile intervals and check for overlap. This is exactly what we would do if we were rerunning experiments, although bootstrapping does not involve running a single new experiment and is just a manipulation of the data that was already collected.

There are two parameters that control the bootstrapped distribution: we form the distribution by subsampling $l$ points from the original $m$, and we repeat this process $k$ times. For this thesis we will chose $l$ to be $\lfloor m/2 \rceil$ and $k$ to be around $2\ 500$. These particular parameters were chosen to ensure that there would be diversity among the subsamples (this explains the moderate size of $l$) and that the empirical distributions would be relatively smooth (this explains the large $k$).

## 4.3 Statistical Tests

### 4.3.1 Kolmogorov-Smirnov Test

While bootstrapping is useful for seeing if summary statistics are significantly different or not, will also want to check if two distributions are themselves significantly different. A beta distribution and a Gaussian distribution might coincidentally have the same mean, but they are are not the same distribution. So how can we distinguish between distributions that are different?

The most common way of doing this for general functions is a statistical test called the KolmogorovSmirnov (KS) independence test. This test is nonparametric, which means that it does not assume that the underlying data is drawn from some known probability distribution. The KS test checks the vertical distance between two CDFs (see Figure 4.3.1) and if the maximum vertical distance is large enough then the distributions are significantly different. 'Large enough' is controlled by the significance level, $\alpha$, and we will use the standard $\alpha = 0.05$ unless otherwise noted.

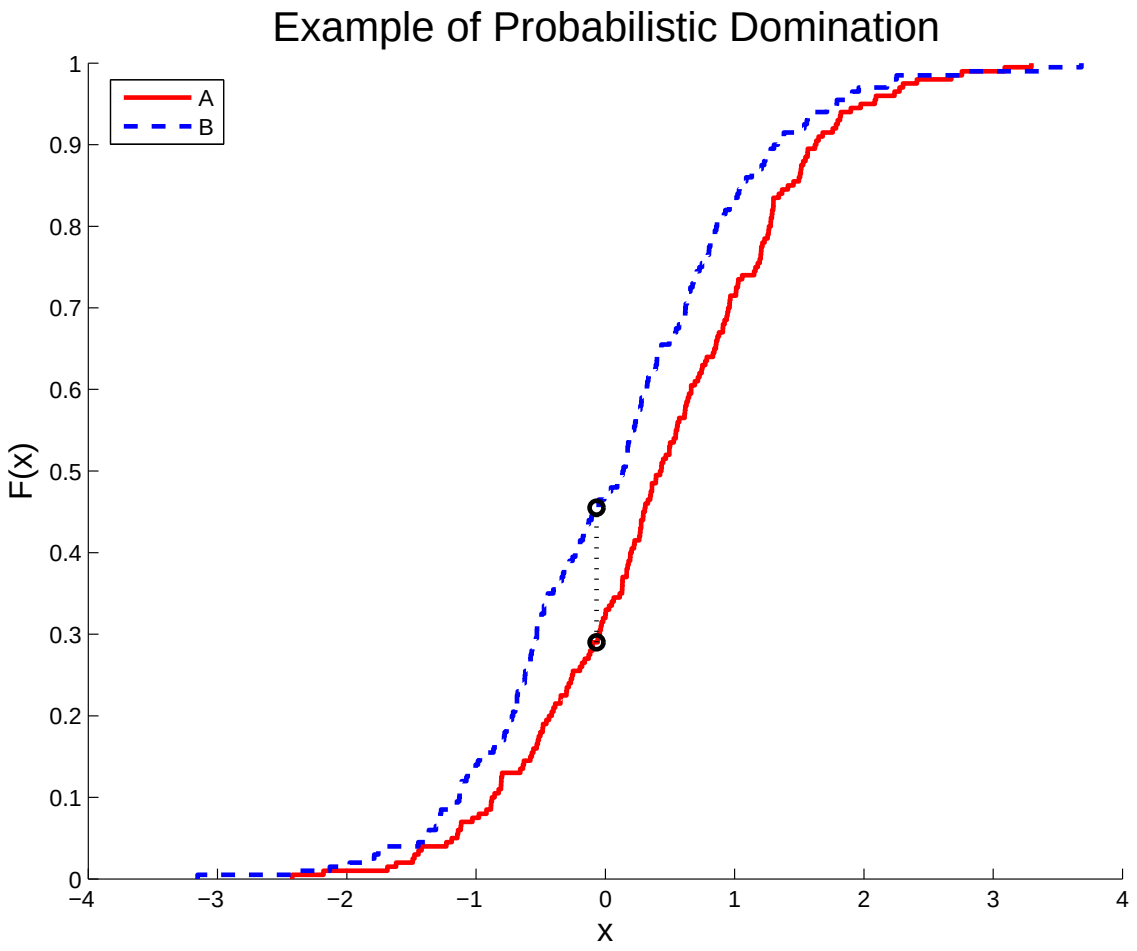


Figure 4.1: An illustration of the KS statistic in terms of two CDFs. The KS statistic is the vertical dotted black line between the CDF curves.

### 4.3.2 Spearman's Rank Correlation Test

Spearman's rank correlation test is a way to establish whether or not there is a significant monotonic relationship between two paired variables. For example, we might want to show that there is some significant monotonic relationship between the size of a game's action set size and the reward that a learning agent receives on it.

This test is, like the KS test, non-parametric: it does not assume any parametric form of the underlying data. The relationship between the two variables can be positive (high values of one variable are correlated with high values of the other variable) or negative (high values of one variable are correlated with low values of the other).

## 4.4 Probabilistic Domination

Say that we have two algorithms and that one has both a higher mean and more 'bad' runs than the other (say, runs below $0.1$). Looking solely at the sample means would lead us to conclude that this algorithm is better. Should we be happy with the account of performance given by the sample mean?

We might worry, for example, that the distribution of game instances that we have in the experiment is not representative of a practical problem that we want deploy a MAL algorithm on: the 'bad' runs might be more common in practice. Or perhaps we do not know the reward function but instead we observe a monotonic transformation of the reward. For example, in traffic we may observe trip time but not the exact reward function—it probably is not the case that a $60$ minute route is exactly $12$ times worse than a trip that takes $5$ minutes, but we do know that it is worse.

However, sometimes performance results are clear enough that these sorts of objections do not matter and we can claim that one algorithm is better than the other without exploring these issues. These are situations where performance conclusions can be drawn without making subjective and problem-specific judgments, and they are particularly compelling. These situations can be captured using *probabilistic domination*—a robust partial ordering for distributions.

A solution quality distribution $A$ (SQD; the distribution of a performance metric) dominates another SQD $B$ if $\forall q \in [0, 1]$, the $q$-quantile of $A$ is higher than the $q$-quantile of $B$. If there are two algorithms, $A$ and $B$, that are trying to maximize reward, and $A$'s SQD probabilistically dominates $B$ then regardless of the reward value $r$ that one picks, there are more runs of $A$ than of $B$ that attain a reward higher than $r$. Notice that this means that probabilistic domination is unaffected if all rewards are shifted by some monotonic function: there will still be more runs of $A$ than $B$ that attain a higher reward than $f(r)$. Probabilistic domination is stronger than a claim about the mean of the distributions: domination implies higher means.

Checking for probabilistic domination between two samples, with a maximum size of $n$, can be done in $O(n \log n)$ time (sorting the two samples is the dominating step), but it can also be checked visually by looking at the CDF plots. If one of the CDF curves is below the other curve everywhere, than the former dominates the latter. Intuitively, this is because the better SQD has

less probability mass on low solution qualities, and more mass on higher solution qualities: better distributions are right-shifted (in Figure 4.3.1, SQD $A$ dominates SQD $B$).

# Chapter 5

# Empirical Evaluation of MAL Algorithms

The experiment described in Chapter 4 was big: recording all $72\,600$ matches for $10\,000$ iterations generated $143\,GB$ of data. Furthermore, the experiment took approximately 126 CPU days to run. This is an incredible amount of information to sort through and extract meaning from, and to make sense of the experiment at all we had to summarize much of this information. First, we used performance metrics that map the $10\,000$ recorded iterations of each match into a single number (such as average reward or average regret). Secondly, we tended to use summary statistics to examine and analyze the distribution of these performance metrics. In this section, we only comment on properties of distributions as a whole when the properties are especially strong: *e.g.* probabilistic domination. It is an understatement to say that some information is lost in this process, but this is inevitable.

Work in MAL has focused on many different metrics and our experiment evaluated different algorithms using several of these measures of performance. We used average reward and a selection of other metrics that addressed other aspects of empirical performance. We took these metrics from two broad families of metrics that either measure performance based on aggregated reward or check for various types of strategic convergence. For each of these metrics we, as much as possible, try to relate our results to the agent's algorithmic structure.

Additionally, many authors have proven results about these alternative metrics instead of directly proving results about reward. However, the Artificial Intelligence [46] agenda for MAL learning states that the fundamental goal of all agents should be to achieve high reward. Because of the mismatch between this goal and existing results, not only do we evaluate the algorithms with each metric, but we also investigate the general connections between reward and the alternative performance metrics. For example, if an algorithm frequently attains low regret, does it also frequently attain high reward?

## 5.1 Reward-Based Metrics

### 5.1.1 Average Reward

As argued earlier, reward is the most fundamental of all metrics as agents are explicitly trying to maximize reward given what the other agents are doing. Because of this, we will engage in a detailed discussion of the reward results. Merely looking at the average reward attained in all the

run over the $10\,000$ iterations is an extremely coarse summary of a distribution of games, and so our analysis does not stop there. In particular, there are interesting trends when we consider varying the different opposing algorithms and game instance distributions. However, a broad summary of the results undifferentiated by run features is a sensible place to start and gives a gross ranking of the algorithms.

The average reward that we look at is with respect to the sampled actions, and not the submitted mixed strategy. This is formally stated in Equation 5.1, where the iterations $1$ to $T$ refer to the $10\,000$ recorded iterations:

$$\bar{r}_i^{(T)} = \frac{\sum_{t=1}^{T} r_i^{(t)}}{T}. \tag{5.1}$$

**Observation 1** *`Q-Learning` and $\mathsf{RV}_{\sigma(t)}$ attained the highest rewards on the grand distribution.*

`Q-learning` had the highest mean reward at $0.714$, although $\mathsf{RV}_{\sigma(t)}$ was close with an average of $0.710$ (see Figure 5.1). We noticed considerable variation within the reward data, and all of the other algorithms' sample means still were within one standard deviation of `Q-learning`, including `random` (which obtained a sample mean of $0.480$).

The distribution of reward was definitely not symmetric and tended to have negative skewness (`random` was the only exception). `Q-learning`'s distribution had the highest skewness, $-0.720$, indicating that the proportion of runs that attained high reward was larger than the proportion of runs that attained low reward.

These ranking were not all significant. The slight difference in means between `Q-learning` and $\mathsf{RV}_{\sigma(t)}$ does not in fact indicate that `Q-learning` was a better algorithm (in terms of means) on the grand distribution of games and opponents. These two algorithms attained significantly higher reward than any other algorithm, however. This was determined by looking at the $95\%$ percentile intervals on bootstrapped mean estimator distributions (see § 4.2) and seeing which intervals overlapped (see Figure 5.2). The distributions were obtained by subsampling $2\,500$ times, where each subsample had $6\,600$ runs (half of the $13\,200$ runs that each algorithm participated in).

**Observation 2** *Algorithm performance depended substantially on which opponent was played.*

We blocked the runs based on the opponent for a more detailed analysis of the reward results. Not only is this useful from an algorithm design perspective (why is my algorithm particularly weak against algorithm $A$?), but it is also useful for lifting results from our experiment to other experimental settings. For instance, if one was looking for an algorithm that only operated on games where the payoffs were unknown (excluding, for example, $\mathsf{RV}_{\sigma(t)}$), blocking could be used to restrict attention to algorithms that satisfied this constraint.

Figure 5.3 shows the mean reward for each algorithm against every possible opponent. The most salient feature of this figure is that `minimax-Q`, `minimax-Q-IDR` and `random` were all relatively weak against a broad range of opponents.

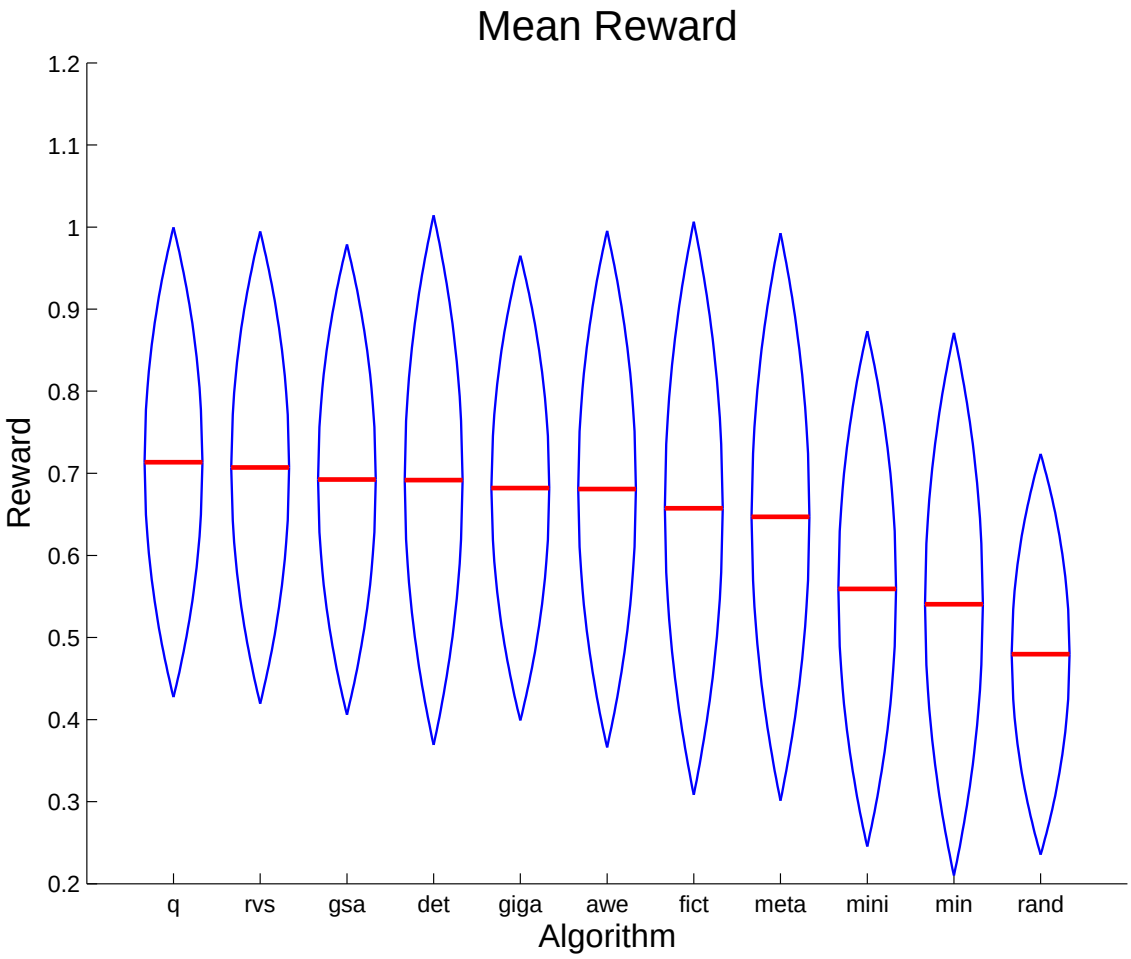


Figure 5.1: A plot that shows the mean reward (bar) for each algorithm and one standard deviation in either direction (indicated by the size of the lens).

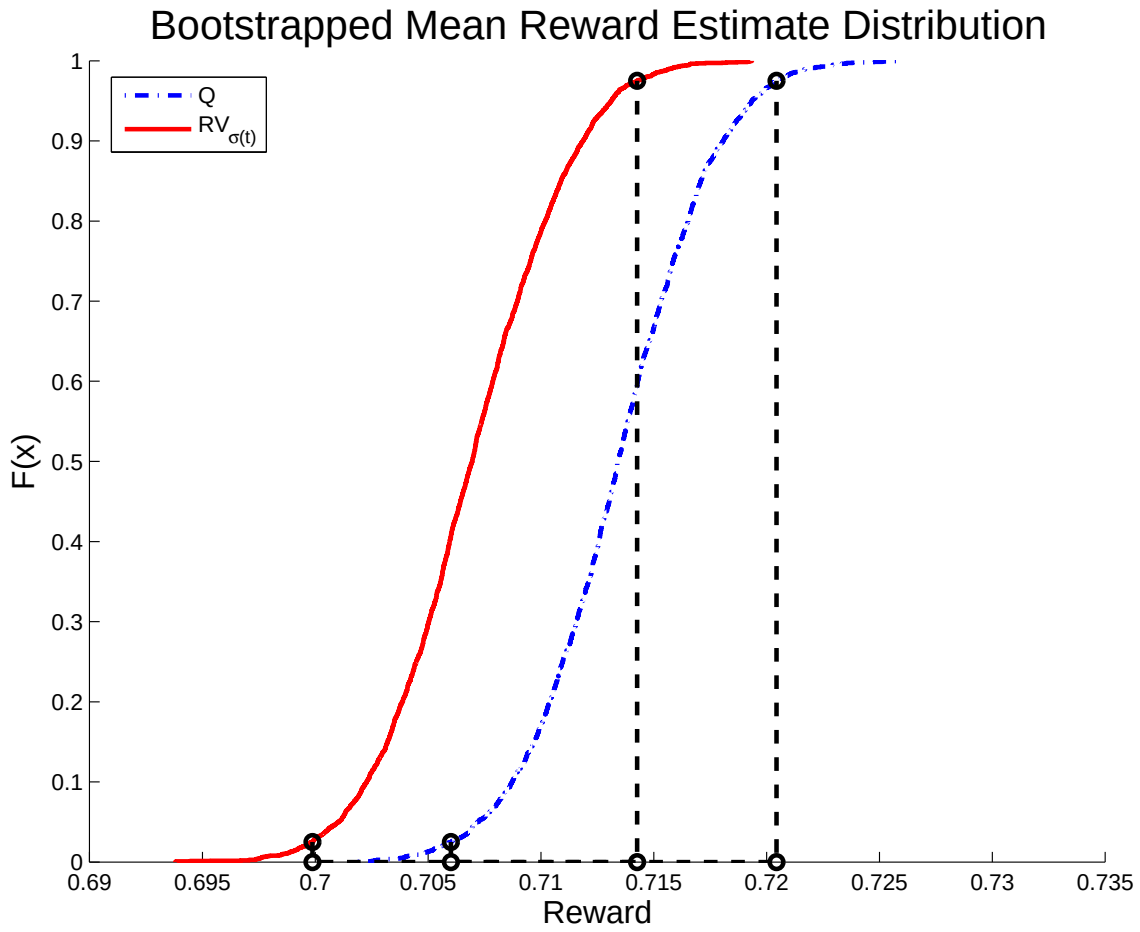


Figure 5.2: The distribution of mean reward estimates for `Q-learning` and $\texttt{RV}_{\sigma(t)}$, constructed by bootstrapping. The $95\%$ confidence intervals are indicated by the dark circles and dashed-lines.

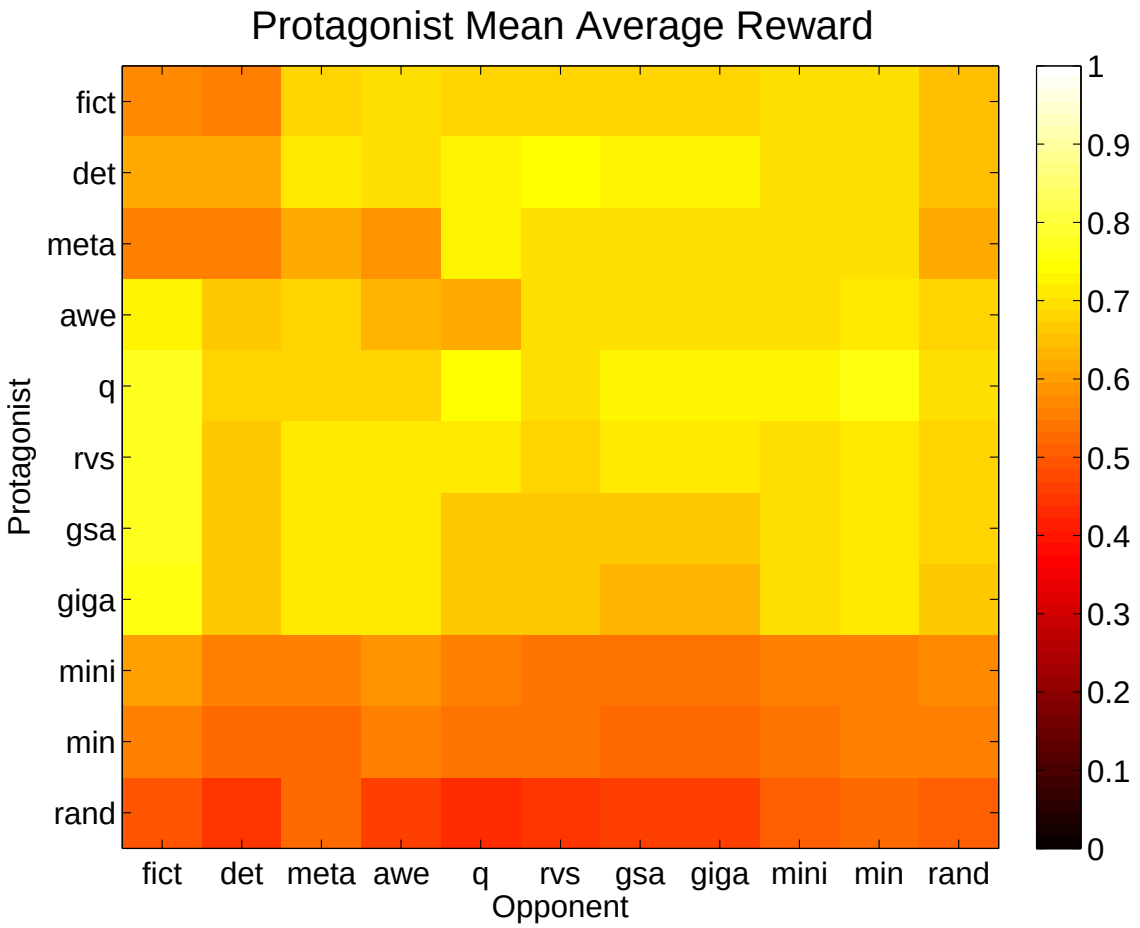


Figure 5.3: A heatmap showing the mean reward for each protagonist algorithm (ordinate) playing against each opposing algorithm (abscissa).

| **Opponent** | **Best-Response Set** |
|---|---|
| AWESOME | GIGA-WoLF, GSA and $\mathsf{RV}_{\sigma(t)}$ |
| Determined | AWESOME, GIGA-WoLF, GSA, Q-learning and $\mathsf{RV}_{\sigma(t)}$ |
| Fictitious play | GSA, Q-learning and $\mathsf{RV}_{\sigma(t)}$ |
| GIGA-WoLF | determined, Q-learning and $\mathsf{RV}_{\sigma(t)}$ |
| GSA | determined, Q-learning and $\mathsf{RV}_{\sigma(t)}$ |
| Meta | determined, GIGA-WoLF, GSA and $\mathsf{RV}_{\sigma(t)}$ |
| Minimax-Q | Q-learning |
| Minimax-Q-IDR | Q-learning |
| Q-Learning | determined, Q-learning and $\mathsf{RV}_{\sigma(t)}$ |
| Random | determined, Q-learning and $\mathsf{RV}_{\sigma(t)}$ |
| $\mathsf{RV}_{\sigma(t)}$ | determined |

Table 5.1: The different algorithms and their best-response sets

Another interesting feature is that fictitious play and determined tend to get lower reward against themselves (self-play) and each other than against other opponents. Meta—an algorithm that manages a profile of algorithms including fictitious play and determined—also inherited these performances issues, while AWESOME—the other portfolio algorithm—largely avoided them.

If we know what algorithm the opponent is using, which algorithm should we use? We constructed best-response sets for each possible opponent using bootstrapped percentile intervals. We call the algorithm with the highest mean against a particular opponent a best response, but any algorithm with a overlapping bootstrapped $95\%$ percentile interval was also in this set—we cannot significantly claim that these algorithms do worse than the apparent best algorithm. These best reponse sets are summarized in Table 5.1.1. Q-learning and $\mathsf{RV}_{\sigma(t)}$ are most frequently best responses, while fictitious play, meta, minimax-Q, minimax-Q and random never are best responses.

An interesting interpretation of these best-response results is to consider the one-shot 'algorithm' game where a player's action space is the set of algorithms and the player picks one of these as a proxy for playing the repeated game. The payoff for using algorithm $A$ against algorithm $B$ is the mean reward that algorithm $A$ attained against $B$.

**Observation 3** *Determined and Q-learning both participate in pure strategy Nash equilibria of the algorithm game.*

With this interpretation, what can we say about this algorithm game? There were three algorithms that were strictly dominated in this grand distribution algorithm game: minimax-Q, minimax-Q-IDR and random. Strict domination means that regardless of what algorithm the other player is using, we could use an algorithm that is strictly better than the one that we are using.

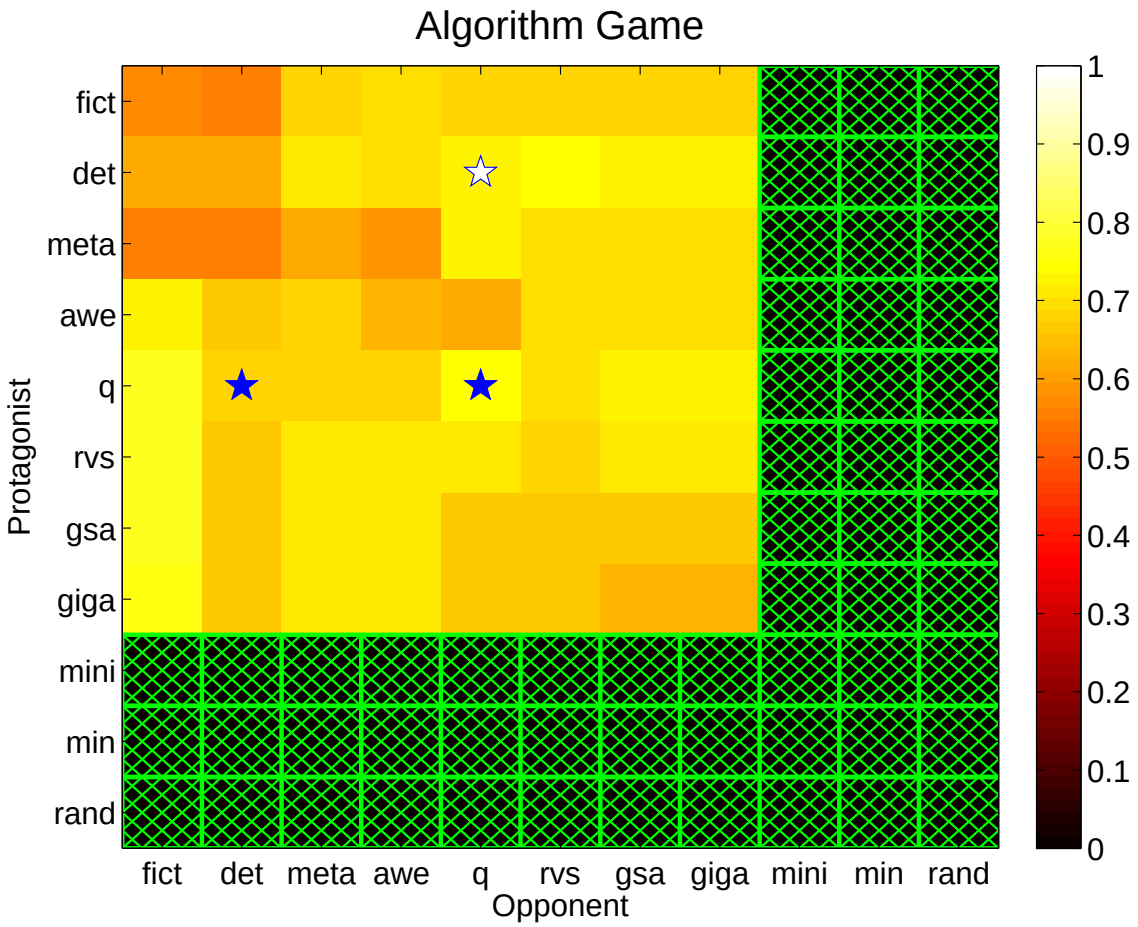


Figure 5.4: Interpreting the mean reward results as a one-shot game. The cells that are cross-hatched are dominated and the '⋆'s indicate pure-strategy Nash equilibria. The `determined` and `Q-learning` equilibrium shows up twice since the protagonist can either select `Q-learning` or `determined`, and we indicate this symmetry by making one of the corresponding stars hollow.

| Algorithm | Strictly Dominated | Weakly Dominated |
|---|---|---|
| AWESOME | 10.8% | 11.7% |
| Determined | 0.0% | 0.0% |
| Fictitious play | 35.9% | 36.4% |
| GIGA-WoLF | 54.1% | 55.1% |
| GSA | 0.4% | 0.4% |
| Meta | 28.8% | 28.2% |
| Minimax-Q | 100.0% | 100.0% |
| Minimax-Q-IDR | 100.0% | 100.0% |
| Q-Learning | 0.0% | 0.0% |
| Random | 100.0% | 100.0% |
| $\text{RV}_{\sigma(t)}$ | 0.0% | 0.0% |

Table 5.2: The proportion of grand distribution subsampled algorithm games where each algorithm was strictly or weakly dominated.

The domination must be significant: we want to be confident that if this experiment were repeated, we would get a similar result. We used bootstrapping to check this: we subsampled $6\,600$ PSMs $10\,000$ times and from these formed $10\,000$ 'subsampled' games. We checked for strict domination in each game, and considered an algorithm dominated if it was dominated in at least $95\%$ of the subsampled games. The proportion of subsampled games where each algorithm was dominated on is shown in Table 5.2.

There were only two pure-strategy Nash equilibria that ever occurred in the subsampled games for the grand distribution: `Q-learning` in self-play, and `Q-learning` against `determined`. `Q-Learning` in self-play is particularly convincing because it was symmetric and did not require that the players coordinate to playing different strategies, and it occurred in $90.2\%$ of the sub-sample games. The other equilbrium occurred in the remaining $9.8\%$ of games.

We also looked at the algorithm games formed by restricting attention to individual generators. For these per-generator algorithm games `Determined` in self-play was the most common symmetric pure strategy Nash equilibrium. It was a significant Nash equilibrium in seven of the generator games, *i.e.* `determined` in self play was a pure strategy Nash equilibrium in more than $95\%$ of the subsampled games for these each of these seven generator games. `Q-Learning` was the second most common symmetric pure strategy Nash equilibrium, and existed in four generator games.

The generators varied substantially in their pure strategy Nash equilibria. For instance D1 (*A Game with Normal Covariant Payoffs*) had no significant pure strategy Nash equilibrium. D4, *Dispersion Game*, is the other extreme and had 22 pure strategy Nash equilibria (see Figure 5.5). Part of the reason for the vast number of equilibria in $D4$ is that majority of runs for many of the algorithms yielded reward of $1$ (*e.g.* $84.6\%$ of `AWESOME`'s runs yielded a reward of $1$). This meant that in many of the subsampled games, the majority of payoffs were exactly $1$ and so many of these Nash equilibria are weak. For example, both $\text{RV}_{\sigma(t)}$ and `Q-learning` attained a re-

ward of 1 against `fictitious play`, and `fictitious play` itself attained a reward of 1 against $\mathsf{RV}_{\sigma(t)}$ and `fictitious play`. Therefore both $\mathsf{RV}_{\sigma(t)}$ and `fictitious play`, and `Q-learning` and `fictitious play`are Nash equilibria.

Using the concept of probabilistic domination (§ 4.4), we can make more robust statements about performance than we could by using mean reward.

**Observation 4** *`Q-Learning` was the only algorithm that was never probabilistically dominated by any other algorithm when playing any opponent.*

`Determined` and $\mathsf{RV}_{\sigma(t)}$ were the next-least dominated: `determined` was only probabilistically dominated by `AWESOME` against a `fictitious play` opponent, which was in turn dominated by `Q-learning`. $\mathsf{RV}_{\sigma(t)}$ was dominated by `Q-learning` when playing against the `minimax-Q` variants, and also by `determined` when playing against $\mathsf{RV}_{\sigma(t)}$. Indeed, being dominated by another algorithm in self-play seemed to be common: only `AWESOME`, `determined` and `Q-learning` avoided being dominated by another algorithm when playing themselves. The fact that `determined` was not dominated should be seen as a property of the games distributions that we chose.

It should be noted that while there are some strong domination relationships, these are the exceptions and ambiguity the rule: for most algorithm pairs on most opponents no probabilistic domination relationship exists (see Figure 5.6). Furthermore, there is no opponent for which one algorithm probabilistically dominates all others.

**Observation 5** *Most algorithms were worse in self-play than in general.*

We noticed above in the probabilistic domination section (see § 4.4) that self-play is a difficult situation for many algorithms. We also noticed that there is a slight tendency towards 'cool' cells on the main diagonal of Figure 5.3. A closer analysis shows that for most algorithms there is indeed a significant relationship between self-play instances and low reward. The distribution of reward in the self-play runs for `AWESOME`, `determined`, `fictitious play` and `meta` were probabilistically dominated by the distribution of reward in the non-self-play runs.

There were no domination results of this kind for the gradient algorithms because they had a tendency to get fewer low-reward runs in self-play, but their self-play means were significantly lower than their non-self-play means. We verified this by looking at the 95% bootstrapped percentile intervals. There was no significant relationship for `minimax-Q` and `minimax-Q-IDR`, and this self-play trend was reversed for `Q-learning`: its self-play runs probabilistically dominated its non-self-play runs. Furthermore, `Q-learning` has the highest mean reward in self-play (see Figure 5.7).

Interestingly, `AWESOME` was one of the algorithms with poorer self-play runs, despite its machinery for converging to a special equilibrium in self-play. One might wonder whether this is because `AWESOME` does not converge due to an overly conservative threshold for detecting whether its opponent is playing part of an equilibrium, or an indication that `AWESOME` was converging to the special equilibrium but it was just not associated with high reward (our implementation used

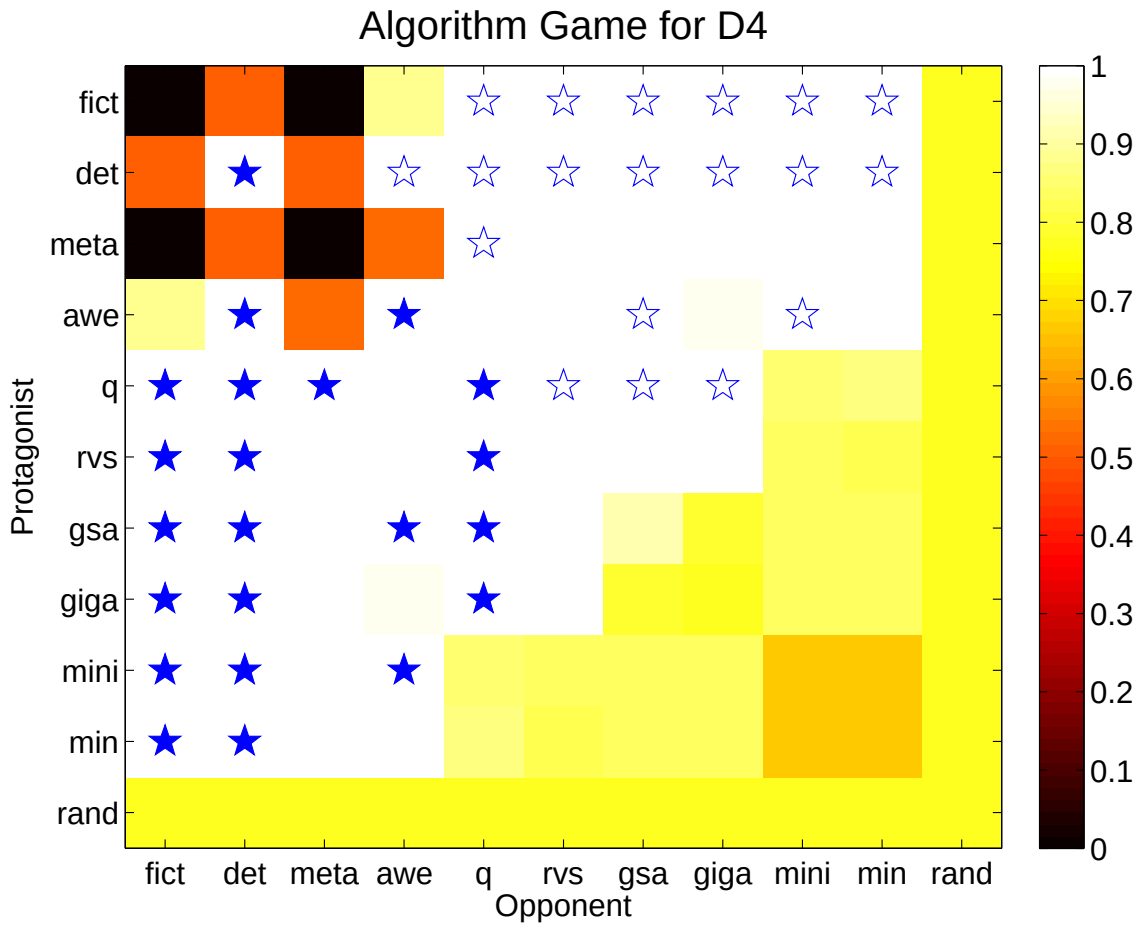


Figure 5.5: Interpreting the mean reward results for D4 (*Dispersion Game*) as a one-shot game. The cells that are cross-hatched are dominated, and the '⋆'s indicate pure-strategy Nash equilibria. Some equilibria show up twice since some of the equilibria are asymmetric, and we indicate this symmetry by making one of the corresponding stars hollow.

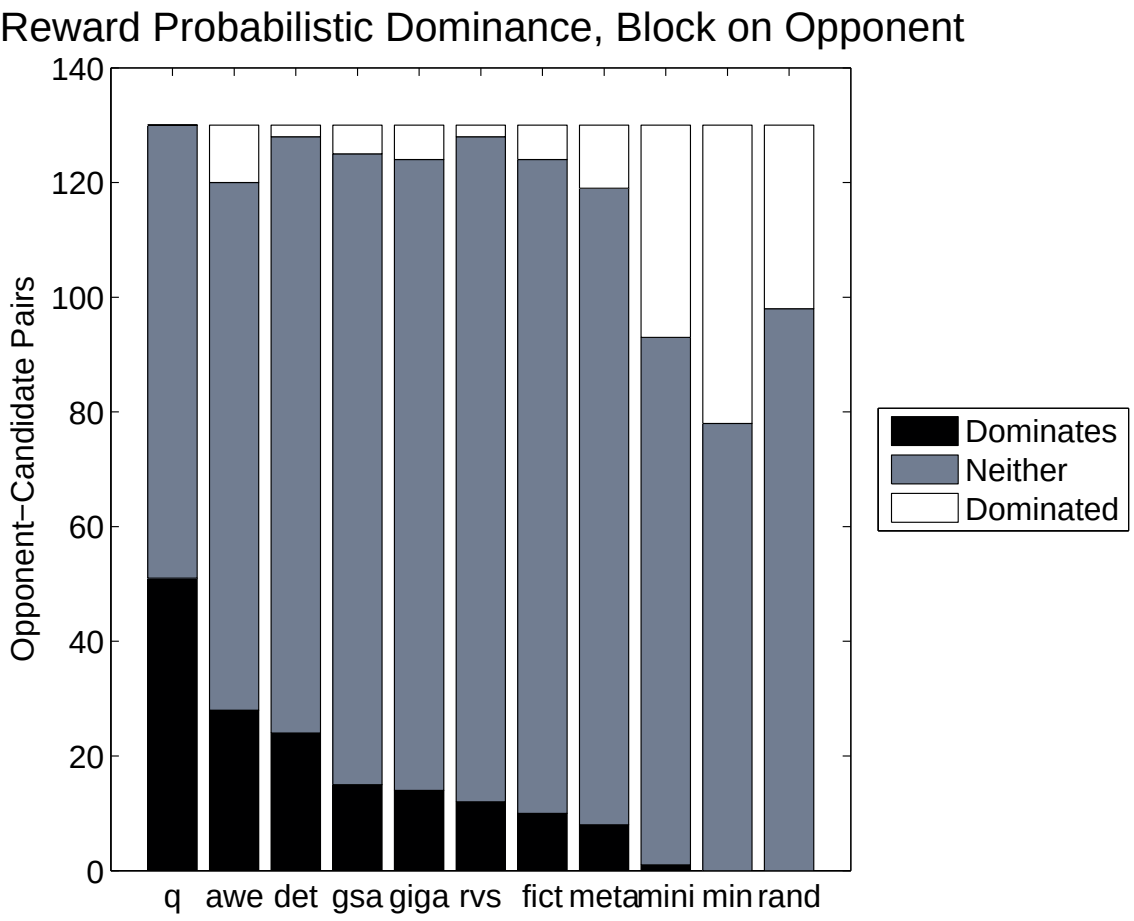


Figure 5.6: The number of opponents and candidate algorithm that each algorithm dominates, is dominated by, or is in a non-dominance relationship with.


Figure 5.7: A plot that shows the mean reward (bar) for each algorithm in self-play and one standard deviation in either direction (indicated by the size of the lens).

the first Nash equilibrium found by GAMBIT's implementation of Lemke-Howson). At the risk of keeping the reader in suspense, we defer the answer to § 5.3.2 where we look at convergence results.

Blocking on the opponent yields many trends and quirks but the 'problem' faced by an algorithm is not completely described by the opponent. The instance generator also matters: one could easily imagine a case where a particular algorithm is excellent on one type of game and poor on another, and this information is lost if the data is solely examined in terms of the opponent (or, indeed, if the results are not blocked by any feature of the match). Analyzing each generator separately is particularly useful from an algorithm design perspective. Weak distributions can be identified, and hopefully any poor behaviour can be isolated and rectified. Per-generator results are also useful from an deployment perspective; there is little point in deploying a MAL algorithm on a particular distribution of games instances if the algorithm is weak on it, even if the algorithm seems to be good 'in general'.

**Observation 6** *`Q-Learning` is the best or one of the best algorithms to use for most generators.*

Generators are an important part of the reward story. As can be seen in Figure 5.8, the results for any algorithm vary considerably between the different game generators. However, it helps to take these per-generator reward results and normalize them, dividing the results for each algorithm on a particular generator by the maximum reward attained by any algorithm. Some familiar trends emerge: `minimax-Q`, `minimax-Q-IDR` and `random` are all worse than the other algorithms in a broad range of generators, and `Q-learning` and $\mathtt{RV}_{\sigma(t)}$ tend to be good.

`Q-Learning` was the best algorithm or was one of the best algorithms for $10$ generators (see Table 5.3). It was the only algorithm that was a unique best response of any generator in terms of mean reward, and was the only best response for generators D1, D4, and D9). Furthermore `Q-learning` also belonged to the set of best algorithms in generators D2, D3, D7, D10, D11, D12 and D13—this is the set formed by algorithms whose bootstrapped mean estimator $95\%$ percentile intervals overlaped with the algorithm with the best sample mean. While `Q-learning` most frequently was a member of a generator's best algorithm set, `fictitious play` and `determined` were also frequently in these sets ($6$ and $7$ generators respectively).

The gradient algorithms were especially strong on D7 and this was the only generator where all three gradient algorithms were in the best algorithm set (Figure 5.10). D5, D6, and D8 were interesting distributions for `AWESOME` and `meta`. In D5, neither `AWESOME` nor `meta` managed to be one of the best algorithms despite the fact that both `fictitious play` and `determined`—two of the algorithms that they manage—were. In D6, `AWESOME` joins `fictitious play` and `determined` but `meta` does not, and in D8 the reverse happens: `meta`, `fictitious play` and `determined` were the three best algorithms. These three generators illustrate situations where at least one of the portfolio algorithms failed to capitalize on one of their managed algorithms. It would be interesting to run further experiments to determine why this is the case for these distributions and if this issues could be remedied.

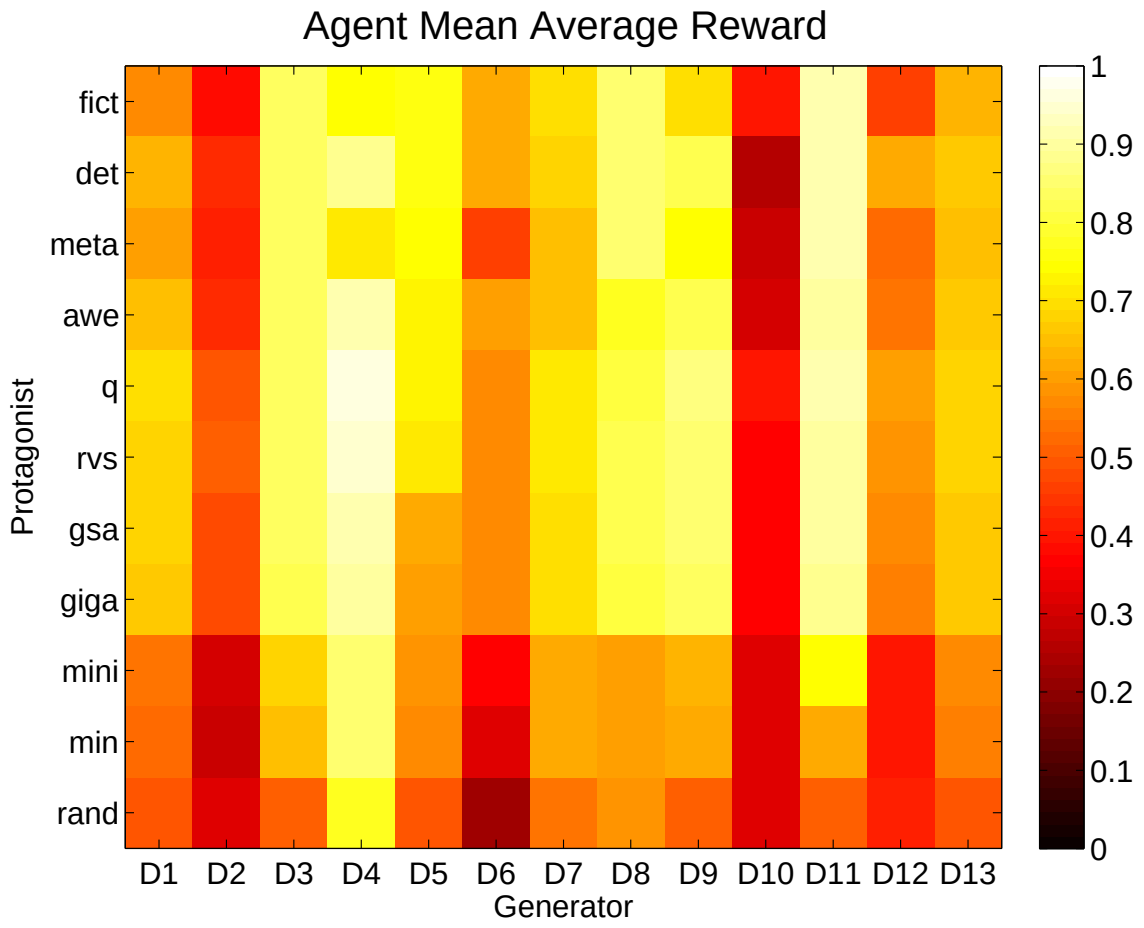


Figure 5.8: A heatmap showing the reward for the protagonist algorithm playing PSMs from a particular generator, averaged over both iterations and PSMs.

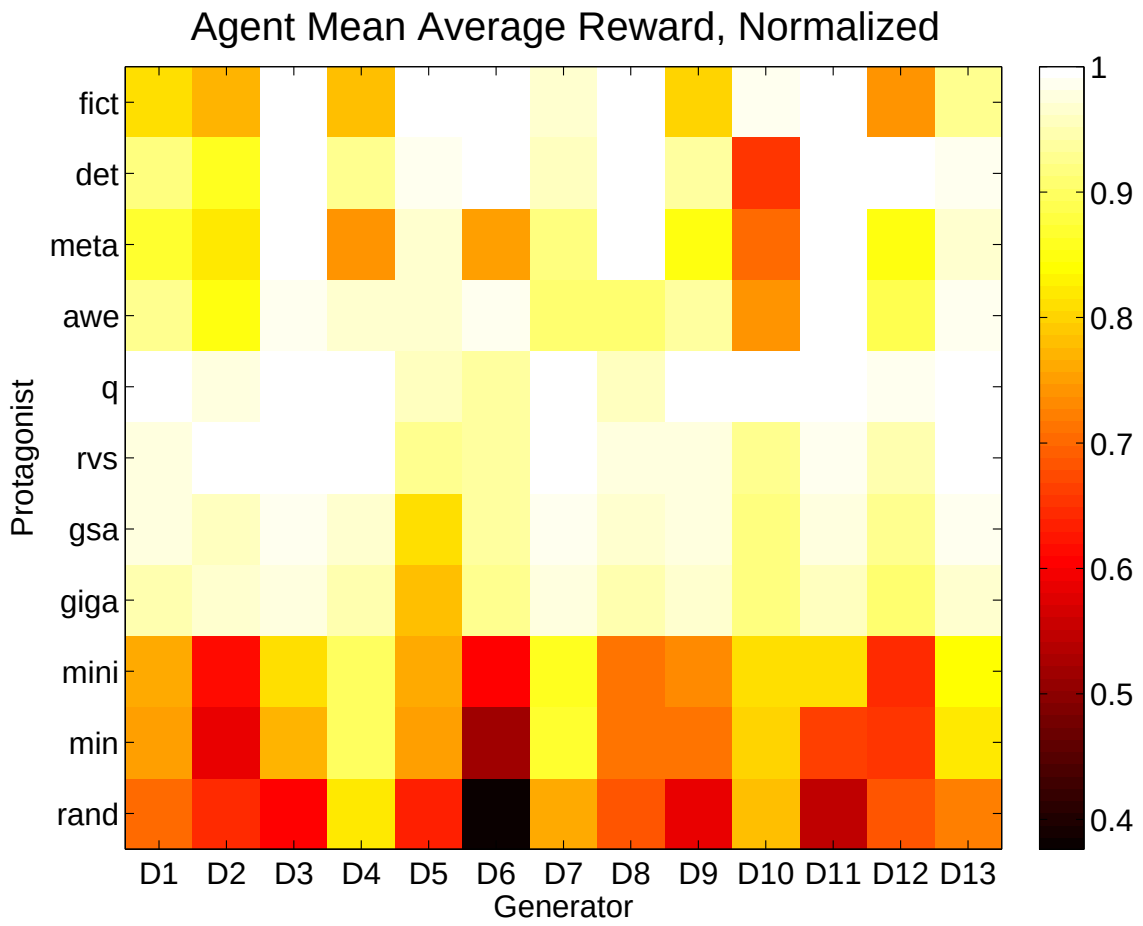


Figure 5.9: A heatmap showing the mean reward for the protagonist algorithm, playing against the opposing algorithm. These cells have been normalized. Each column has been divided by the maximum average reward attained by any algorithm on that particular generator.

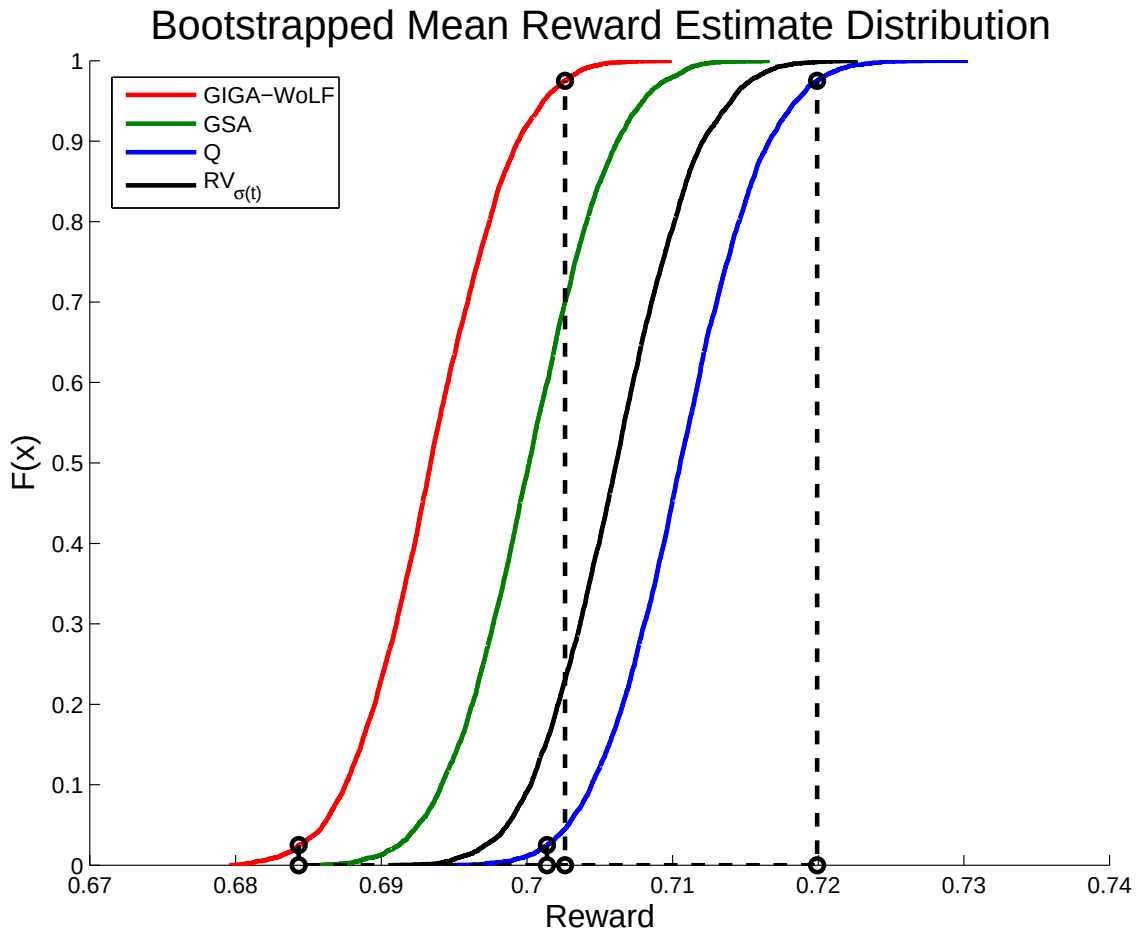


Figure 5.10: The bootstrapped mean estimate distribution for D7. Four algorithms are shown; they are the algorithms that have a $95\%$ confidence interval that overlaps with `Q-learning`, the algorithm with the highest mean. The $95\%$ confidence intervals for `Q-learning` and `GIGA-WoLF` (the algorithms with the highest and lowest mean) are indicated with a dashed black line and circles.

| Generator | Best Algorithm |
|---|---|
| D1 | Q-learning |
| D2 | Q-learning and $\text{RV}_{\sigma(t)}$ |
| D3 | AWESOME, determined, fictitious play, GSA, meta, Q-learning and $\text{RV}_{\sigma(t)}$ |
| D4 | Q-learning |
| D5 | determined and fictitious play |
| D6 | AWESOME, determined and fictitious play |
| D7 | GSA, Q-learning and $\text{RV}_{\sigma(t)}$ |
| D8 | determined, fictitious play and meta |
| D9 | Q-learning |
| D10 | fictitious play and Q-learning |
| D11 | determined, fictitious play, meta and Q-learning |
| D12 | determined and Q-learning |
| D13 | AWESOME, determined, GSA, Q-learning and $\text{RV}_{\sigma(t)}$ |

Table 5.3: The set of best algorithms for each generator.

**Effect of Game Size on Reward**

How does the size of a game's action set effect performance? Larger action spaces entail the possibility of more complicated game dynamics that take longer to learn about and adapt to, so it is natural to assume that average reward will decrease as the size of the game increases. Are there clear trends in this respect?

**Observation 7** *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.*

Our experiment shows that these intuitions do not always hold. For many algorithms on many generators we could not reject the null hypothesis of Spearman rank correlation test—that there was no significant correlation between size and performance—at a significance level of $\alpha = 0.05$. For instance, in D7 only GSA and GIGA-WoLF had significant trends (both exhibited negative correlation; reward was lower in larger games).

As can be seen in Figure 5.11, D2 and D12 were the only two distributions on which we could reject the null hypothesis for all algorithms, and they supported opposite conclusions. On instances from D2, correlation was completely and strongly negative: the larger the game, the worse everyone did. The least correlated algorithm was random with a Spearman's coefficient of correlation $\rho = -0.329$. Correlation was entirely positive for D11, although some of the coefficients were smaller. Fictitious play was the least sensitive to size ($\rho = 0.07$), but it was anomalous. The algorithm with the next smallest coefficient was GIGA-WoLF, at $\rho = 0.267$.

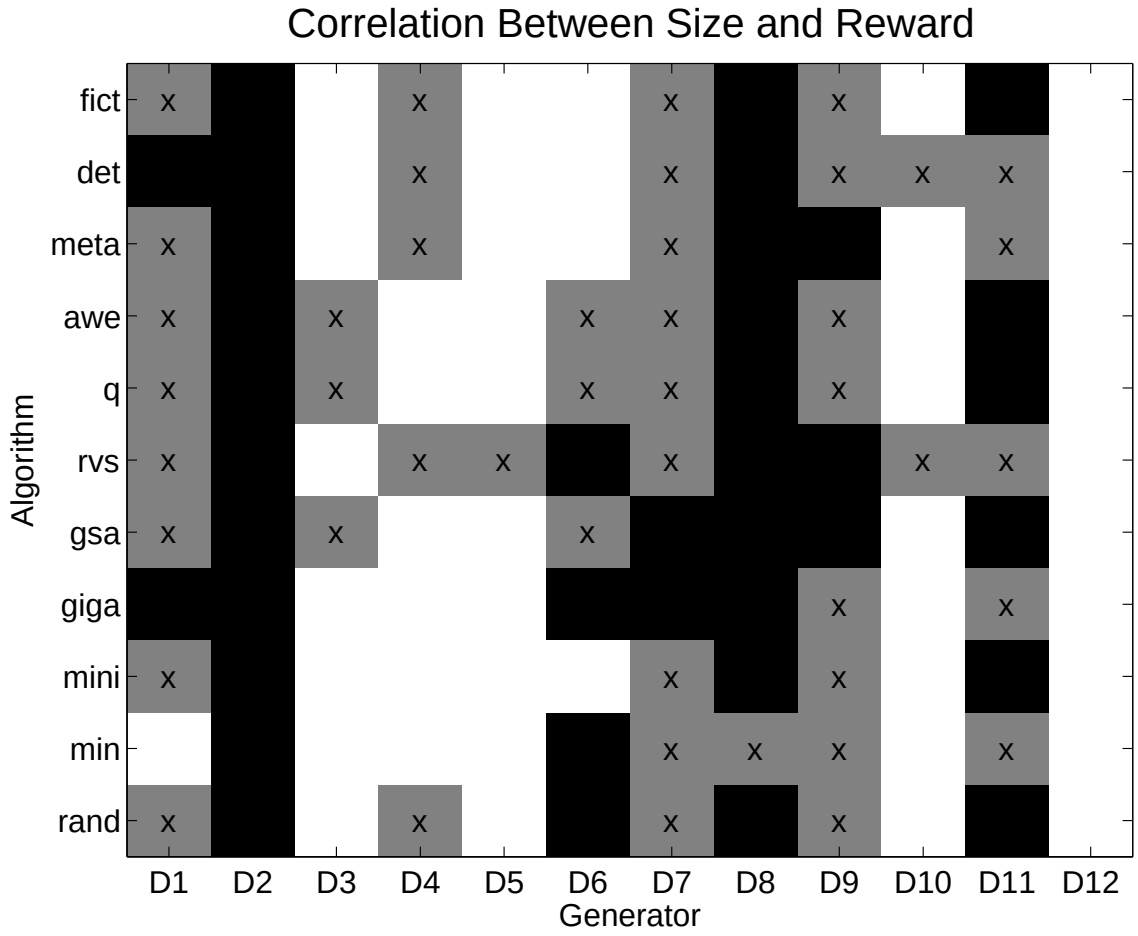


Figure 5.11: A heatmap summarizing the correlations between size and reward for different agents on different generators. A white cell indicates positive correlation, a black cell indicates negative correlation, and a gray cell with an 'x' indicated an insignificant result.

Distributions tended to either support entirely negative significant correlations, or entirely positive significant correlations. The only exceptions to this were D1 and D6, which supported both kinds of correlation. D2, D7, D8, D9 and D11 were negatively correlated distributions; the remainder were positive. These results were less clear than might be expected, and we are not sure why. It could be the case that when the action spaces increase in size, important game features tied with high reward become more common, or it could be that larger actions spaces make it easier for MAL algorithms to miscoordinate, which is desirable for some games. Indeed, D4—*Dispersion Games*—are show positive correlation between the number of actions and reward, and this is a game where agents need to miscoordinate to do well.

**Observation 8** *Similar algorithms tended to have similar performance.*

Several of the algorithms that we implemented have common approaches to learning. Are these similarities reflected in the reward results? There are three major blocks of algorithms with programmatic similarities: `AWESOME` and `meta` are similar because they both manage portfolios with versions of `fictitious play` and `determined`; `GIGA-WoLF`, `GSA` and $\texttt{RV}_{\sigma(t)}$ are similar as they are all variations on following the reward gradient; and `minimax-Q` and `minimax-Q-IDR` are similar as the latter is the same as the former except for the addition of an IDR preprocessing step. We call these the portfolio, gradient, and minimax blocks. We also might suspect that `Q-learning`, an algorithm that does not explicitly model the opponent, might bear some performance similarities to the gradient algorithms.

The algorithms were tested for similarity on PSMs that had the same generator and opponent, and the results are aggregated by summation. There are a possible $13 \times 10 = 130$ cases where similarity could occur—algorithms are of course similar to themselves and we did not bother checking these cases. Failing to reject the null hypothesis of the KS test (the hypothesis that both samples were drawn from the same population) is some evidence for the samples being similar. This rough-and-ready approach does not establish significant similarity and is merely suggestive of similarity; failing to reject a null hypothesis is not the same as having shown that the null hypothesis is true. However, with this caveat in mind, there are some interesting trends.

All three predicted blocks emerge, as can be seen in Figure 5.12. `Meta`, `AWESOME`, `fictitious play` and `determined` were all similar to each other on a number of opponent and generator pairs. Both `meta` and `AWESOME` are similar in more cases to `determined` than to `fictitious play`. For instance, `AWESOME` is similar to `determined` in $101$ out of $130$ cases while being similar to `fictitious play` in only $81$ cases. `Meta` and `AWESOME` also look quite similar to each other, being similar in $88$ cases. `Q-learning` is similar to the algorithms in this block, especially with `determined` and `AWESOME`, which we had not expected. `AWESOME` is more similar to `Q-learning` than to any other algorithm: they were similar in $103$ cases—even `determined` and `AWESOME` were only similar in $101$ cases.

`Q-Learning` also bears similarities to the gradient-algorithm block. The block of algorithms consisting of $\texttt{RV}_{\sigma(t)}$, `GIGA-WoLF` and `GSA` were all similar in a number of instances and there is a particularly tight relationship between `GIGA-WoLF` and `GSA` (they were similar in $111$ cases).

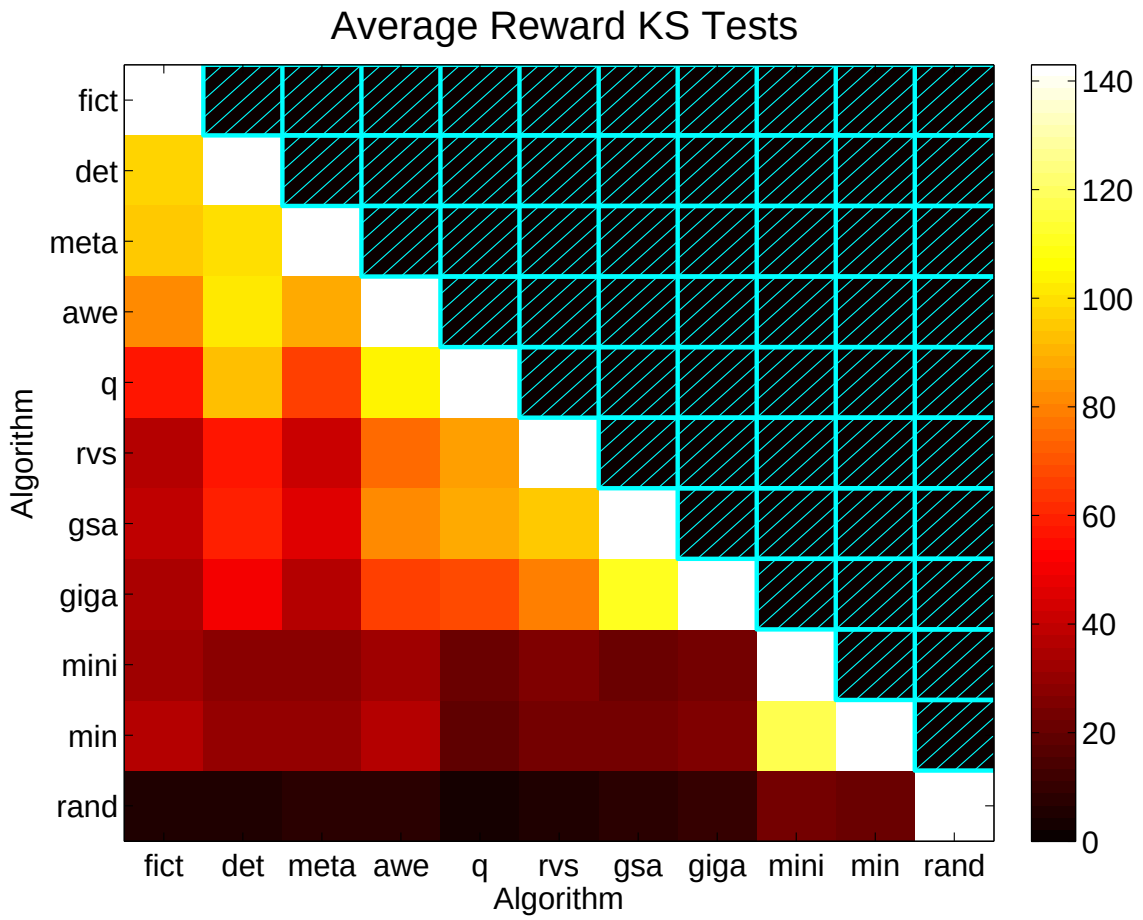


Figure 5.12: A heatmap that summarizes the number of opponent/generator pairs two algorithms are similar on in terms of reward distribution. This relationship is symmetric, so only the lower half of the plot is presented. The hotter the cell, the more situations the two algorithms are similar in.

The gradient-block algorithms also tended to be similar to `determined` and `AWESOME` on some cases.

The connection between `minimax-Q` and `minimax-Q-IDR` was particularly strong. These two algorithms were similar in $118$ cases. They were also the most similar algorithms to `random`. Indeed, these two algorithm were almost twice as likely to be similar to `random` than the next most similar algorithm (`AWESOME`: it was similar in $11$ cases to `minimax-Q`'s $21$ cases).

### 5.1.2 Maxmin Distance

Looking at the difference between the reward that an agent acquires and the maxmin value of the underlying game instance is a way of placing reward results in context:

$$MaxminDistance(\vec{r}_i) = \frac{\sum_{t=1}^{T} r_i^{(t)}}{T} - \max_{a_i \in A_i} \min_{a_{-i} \in A_{-i}} u(a_i, a_{-i}). \tag{5.2}$$

We call this difference *maxmin distance* despite the fact that it can be negative. One can always play a maxmin strategy without fear of exploitation, so getting above the maxmin value is a minimal requirement of sensible MAL behaviour. Having a enforceable payoff (having a non-negative maxmin distance) is also a necessary condition achieving payoffs consistent with some repeated game equilibrium, and we will examine this more in § 5.3.

**Observation 9** *`Q-Learning` attains an enforceable payoff more frequently than any other algorithm.*

`Q-Learning` is the algorithm that most frequently attained an enforceable payoff; it attained a negative maxmin distance in only $1.8\%$ of its runs. The runs where `Q-learning` failed to attain an enforceable payoff mostly came from either D4 (*Dispersion Game*; accounted for $37.6\%$ of the unenforcable runs) or D13 (*Two by Two Game*; accounted for $33.3\%$ of the runs). They also occured prodominently against `random` ($29\%$ of the unenforceable runs), `minimax-Q` ($17.3\%$) and `minimax-Q-IDR` ($16.0\%$). There is a sharp jump in the number of non-enforceable runs between `Q-learning` and the next best algorithm, `AWESOME`, which attained a negative maxmin distance in $7.4\%$ of its runs.

`Minimax-Q` and `minimax-Q-IDR` were the algorithms least likely to attain enforceable payoffs (with the exception of `random`). They failed to attain enforceability in $28.9\%$ and $27.7\%$ of their runs respectively. While they look for the maxmin value of the game they do this with respect to the payoffs they they have learned. This result suggests that they might have difficulty attaining their maxmin value due to having inaccurate beliefs.

`Minimax-Q` and `minimax-Q-IDR` were especially poor in self-play, where conservative play can retard payoff learning. There is also a greater proportion of enforceable runs on $2 \times 2$ games ($75.2\%$) than on $10 \times 10$ games ($68.5\%$)—larger games have more payoffs to learn. Working on a more sophisticated exploration scheme looks like an especially promising place to improve our implementation of `minimax-Q` and its variant.

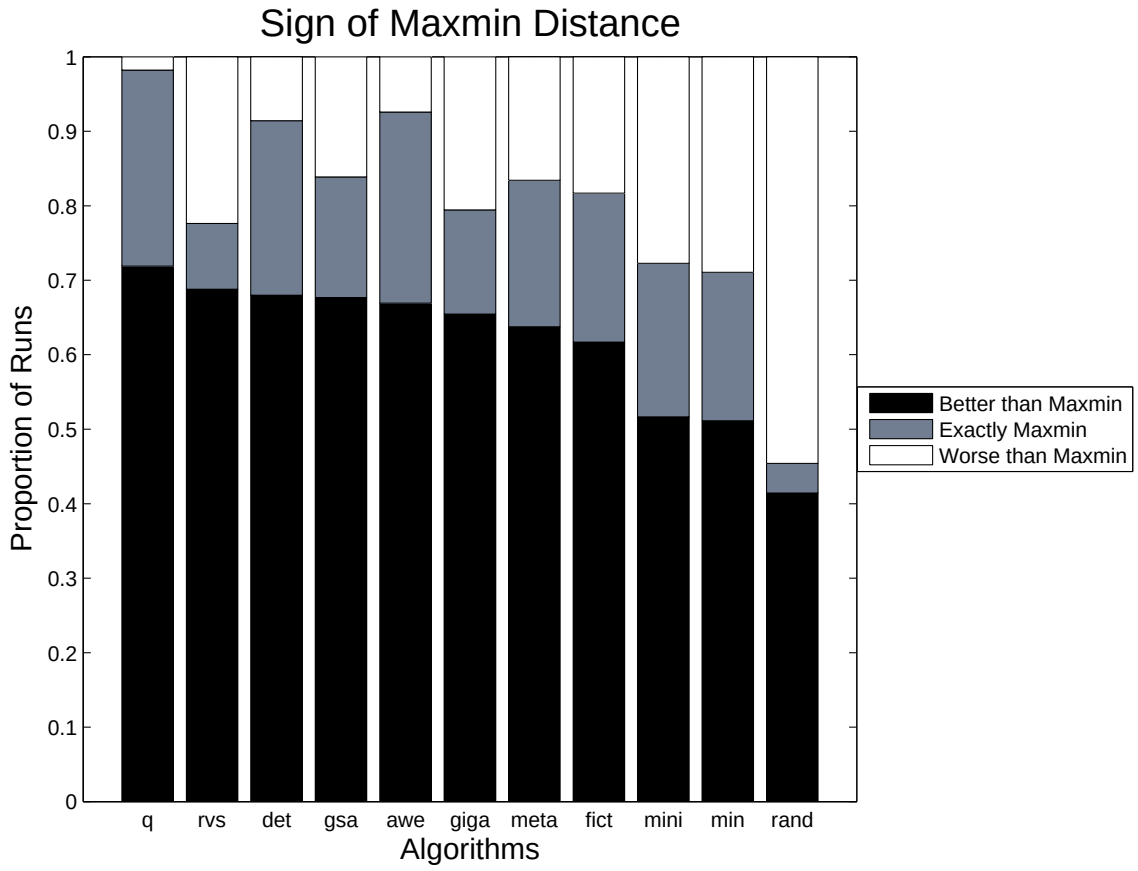


Figure 5.13: The sign of the safety distance of each run, by algorithm.

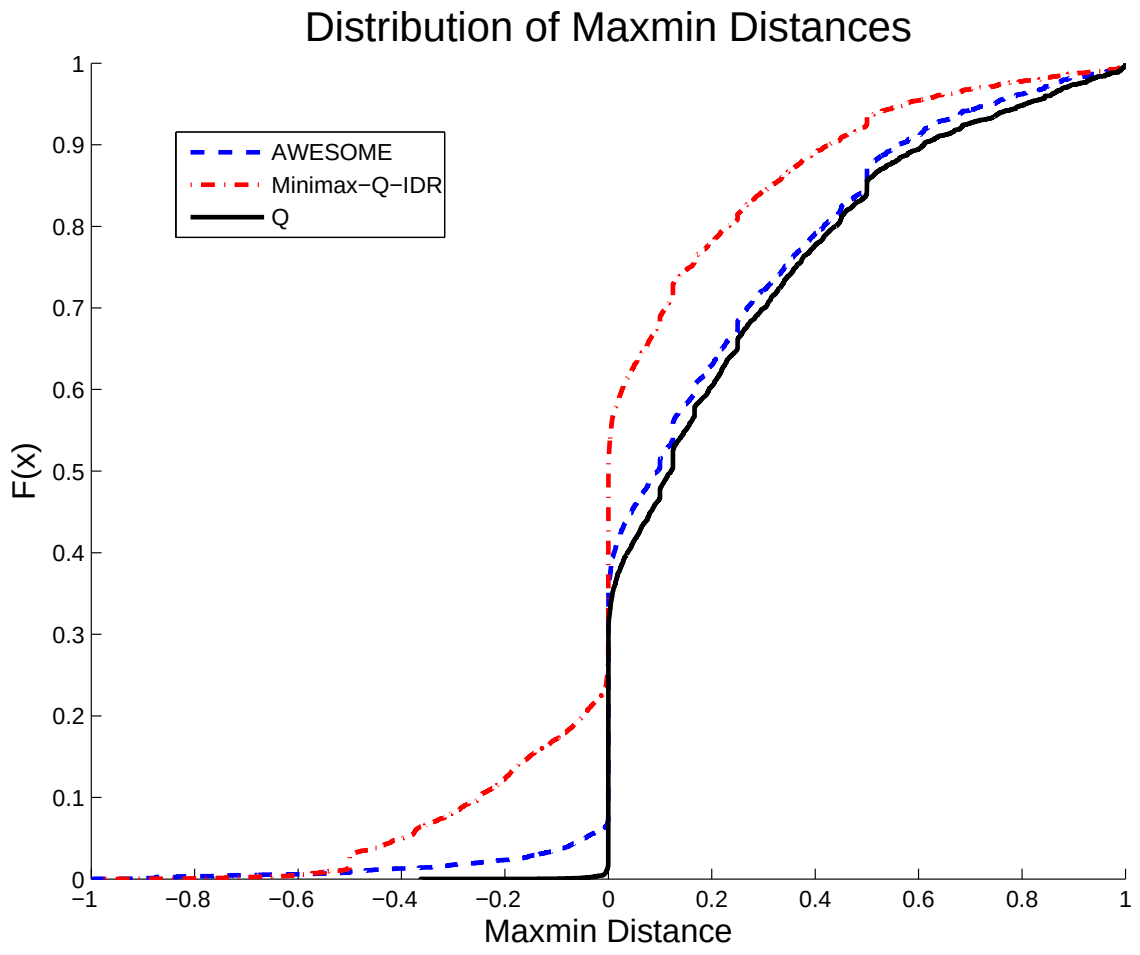


Figure 5.14: The distribution of maxmin distances for `AWESOME`, `minimax-Q` and `Q-learning`.

While Q-learning is successful against a broad range of opponents, there are other algorithms that have problems against certain algorithms. In particular, meta is quite good against all algorithms except for fictitious play, determined, AWESOME and itself. It is especially bad against fictitious play, where only $68.0\%$ of its runs are enforceable. This should be contrasted with its excellent performance against Q-learning: enforceability is attained in $97.7\%$ of it runs. Fictitious play also has issues playing against meta, determined and itself. We notice that AWESOME and determined did not share this problem.

RV$_{\sigma(t)}$ had problems attaining enforceable runs too, and although it received payoffs well above the maxmin value frequently (it had the second highest proportion of runs with strictly positive distances at $68.8\%$) it had a large number of instances that were close to but below zero. This is in contrast to the minimax distance distribution of GIGA-WoLF, which had fewer non-enforceable runs with greater negative minimax distance (see Figure 5.1.2). We speculate that these runs were caused by RV$_{\sigma(t)}$ maintaining a small amount of probability mass on all of its actions, causing it to 'tremble'. RV$_{\sigma(t)}$, like all gradient algorithms, maintains a mixed strategy which is updated in the direction of the reward gradient. This updated vector needs to be mapped back to the probability simplex (the action weight might not sum to one after an update). RV$_{\sigma(t)}$ does this by normalizing the updated vector, while GSA and GIGA-WoLF use a retraction operator that is biased toward the extreme points on the probability simplex, and has a bias toward dropping actions from the mixed strategy's support. An interesting tweak of RV$_{\sigma(t)}$ would be to use GIGA-WoLF's retraction operator instead of normalization, and see if this improves how frequently RV$_{\sigma(t)}$ attains enforceability.

## 5.2 Regret

Regret is the difference between the reward that an agent could have received by playing the best static pure strategy and the reward that it did receive by using the algorithm:

$$Regret(\vec{\sigma}_i, \vec{a}_{-i}) = \max_{a \in A_i} \sum_{t=1}^{T} \left[ r(a, a_{-i}^{(t)}) - \mathbb{E}\left[ r(\sigma_i^{(t)}, a_{-i}^{(t)}) \right] \right]. \tag{5.3}$$

The best pure strategy is chosen after the run assuming that the opponent's actions choices are frozen. Note that we are using the expected reward formulation of regret—as opposed to one that uses the actual actions that the algorithm played—following Bowling [7].

Regret has been suggested as a measure of how exploitable an algorithm is. If an agent accrues significant regret one possible explanation is that it has been 'tricked' into playing the wrong action by the opponent. However, there are situations, like in Game 2.7, where ignoring regret might lead better long-term reward.

Some algorithms, including GIGA-WoLF and RV$_{\sigma(t)}$, are *no-regret* learners: they have theoretic results which guarantee that they will accrue zero regret as the number of iterations approaches infinity. However, to our knowledge it has not been shown experimentally how the regret achieved by these algorithms compares to the regret that other algorithms achieve; nor has it been demonstrated whether these algorithms achieve better than zero regret in practice.

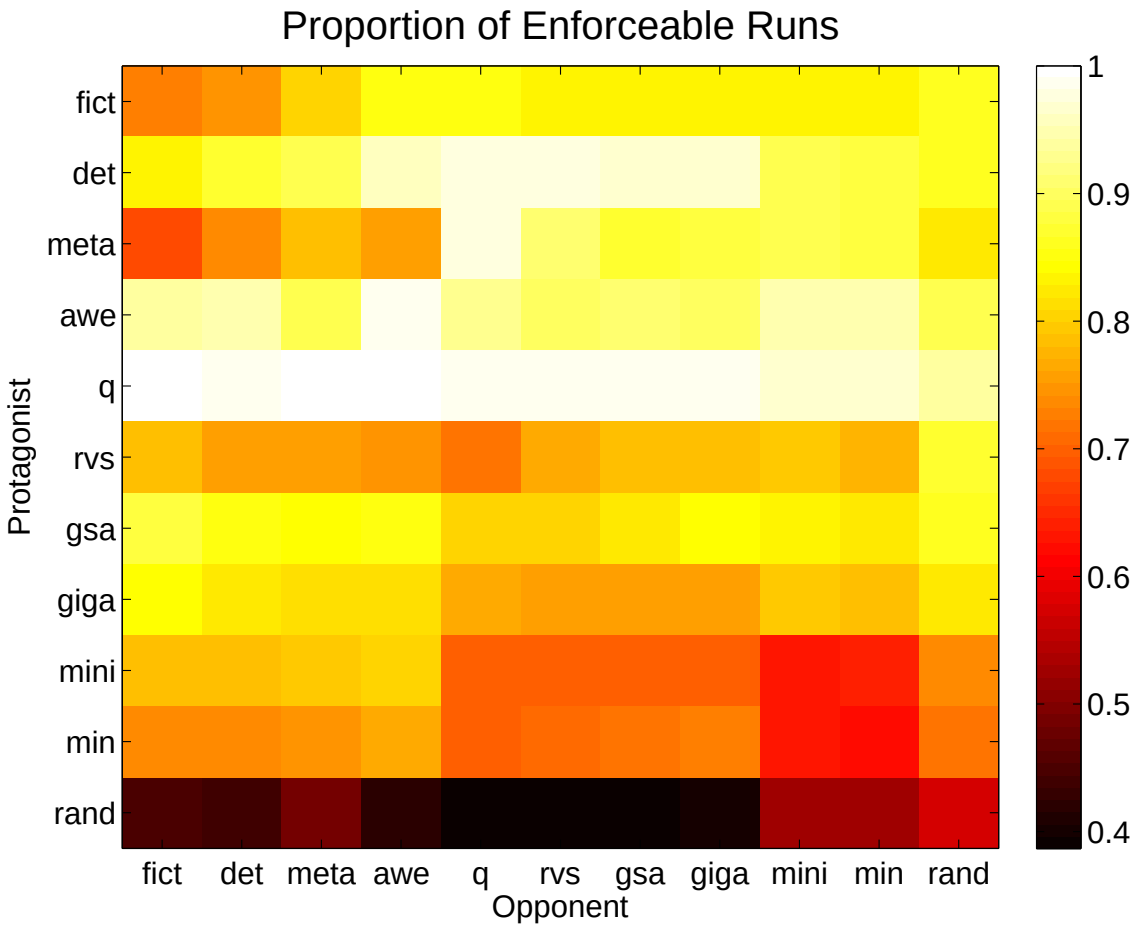


Figure 5.15: The proportion of enforceable runs, blocked by opponent.

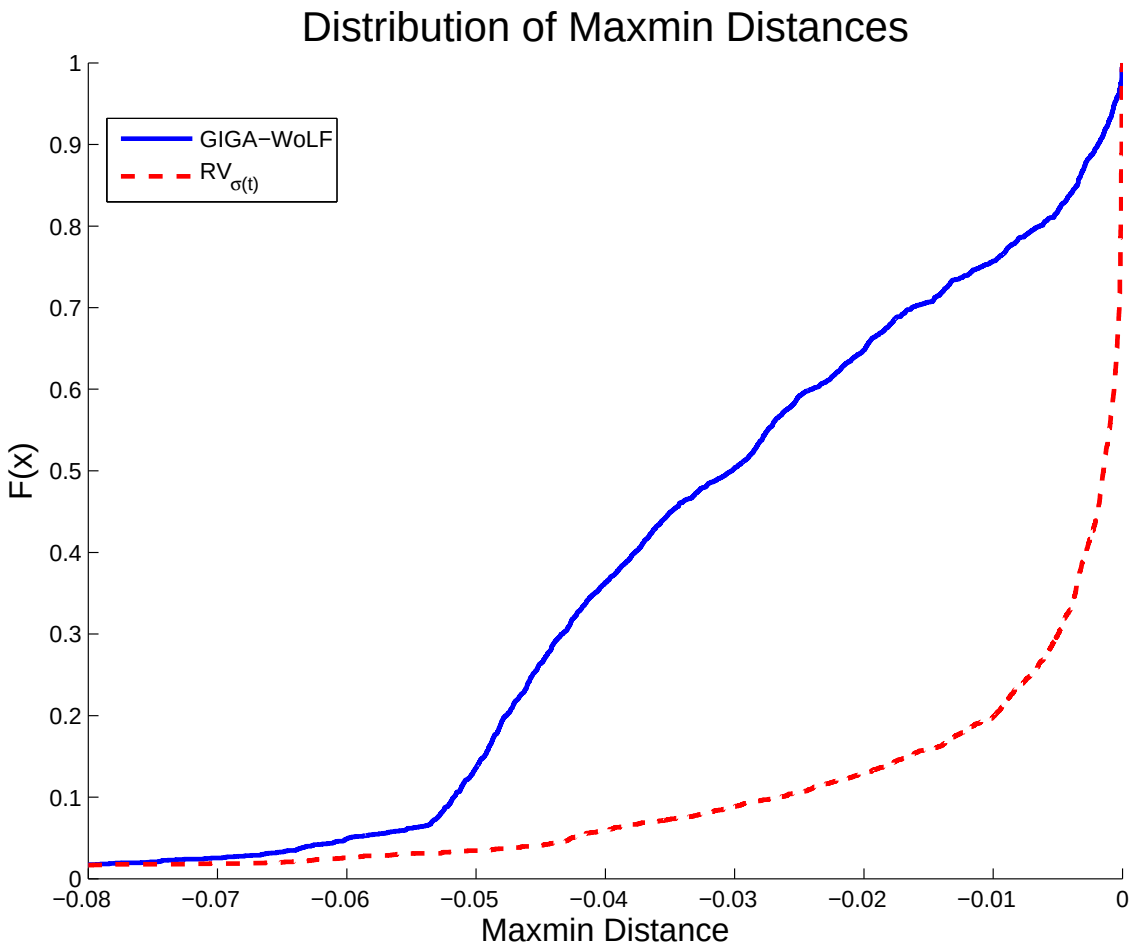


Figure 5.16: The distribution of negative maxmin distances for `GIGA-WoLF` and $\mathrm{RV}_{\sigma(t)}$.

Rather than looking at the total sum of regret over all $10\ 000$ recorded iterations, we will discuss the average regret over these iterations. Since player payoffs are restricted to the $[0, 1]$ interval, averaged regret can give a better sense of the magnitude of regret with respect to possible reward.

**Observation 10** *`Q-Learning` was the best algorithm in terms of minimizing regret. `GIGA-WoLF` was the algorithm that most frequently had negative regret runs.*

Based on the distribution of games and opponents used in this experiment all algorithms had positive mean regret (Figure 5.17). All the means were significantly different, which was shown by checking the $95\%$ percentile intervals for overlap (there was none). Of these, `Q-learning` had the lowest regret, at $0.008$. The gradient algorithms—`GIGA-WoLF`, `GSA` and $\mathsf{RV}_{\sigma(t)}$—had the next lowest regret after `Q-learning`. Among the gradient algorithms, $\mathsf{RV}_{\sigma(t)}$ had lower mean regret than `GIGA-WoLF`, but `GSA` had lower mean regret than either of them. These empirical results agree with `GIGA-WoLF` and $\mathsf{RV}_{\sigma(t)}$'s theoretical no-regret guarantees—not only were they predicted to get zero regret in the limit, but also they had good empirical regret results—although the best algorithm in terms of mean regret, `Q-learning`, has no guarantees about regret.

Mean regret masks an interesting difference between `Q-learning` and the gradient algorithms: they have low mean regret for different reasons. Most ($89.5\%$) of `Q-learning`'s runs attain zero regret. It has the fewest positive runs at $10.4\%$ (the next lowest is `AWESOME` at $18.2\%$), and has the second-fewest negative runs (only `fictitious play` has fewer at $0.1\%$). The gradient algorithms, on the other hand, tended to have many negative runs; the three algorithms with the most negative regret runs were `GIGA-WoLF` ($5.8\%$), $\mathsf{RV}_{\sigma(t)}$ ($3.2\%$) and `GSA` ($3.0\%$). The gradient algorithms also have few zero runs. The algorithms, in order, with the fewest zero runs are $\mathsf{RV}_{\sigma(t)}$, `GSA`, `random` and `GIGA-WoLF`.

The negative regret runs were only slightly negative: the run with the lowest regret had an average regret of $-2 \times 10^{-6}$. The same cannot be said for positive regret: in $440$ different runs an average regret of $1$ was attained. These runs indicate disastrously poor play since one of the algorithms has taken the exact wrong action at every possible step. $48.6\%$ of these runs involve both `fictitious play` or one of the algorithms that wrap around `fictitious play`( awesome or `meta`) in self-play, and are on D4 (*Dispersion Games*), which are games that encourage miscoordination. Indeed, *Dispersion Games* generalize the intuition of Game 2.6 to more than two actions. This behaviour suggests that `fictitious play` becomes stuck in pathological cycling between the symmetric outcomes (outcomes where both agents play the same action), which gets no reward in game instances from $D4$. This is a well known problem with `fictitious play` and a judicious application of noise to the `fictitious play` algorithm might break the above lockstep cycle and improve performance.

In terms of mean regret, `Q-learning` was the best algorithm to use for any generator except for D13 (all strategically distinct $2\times2$ games)—$\mathsf{RV}_{\sigma(t)}$ was the best algorithm there. `Q-learning` was also the best algorithm to use against almost every opponent. There were only two exceptions: one wants to use $\mathsf{RV}_{\sigma(t)}$ against `Q-learning` and `AWESOME` against itself. Another interesting

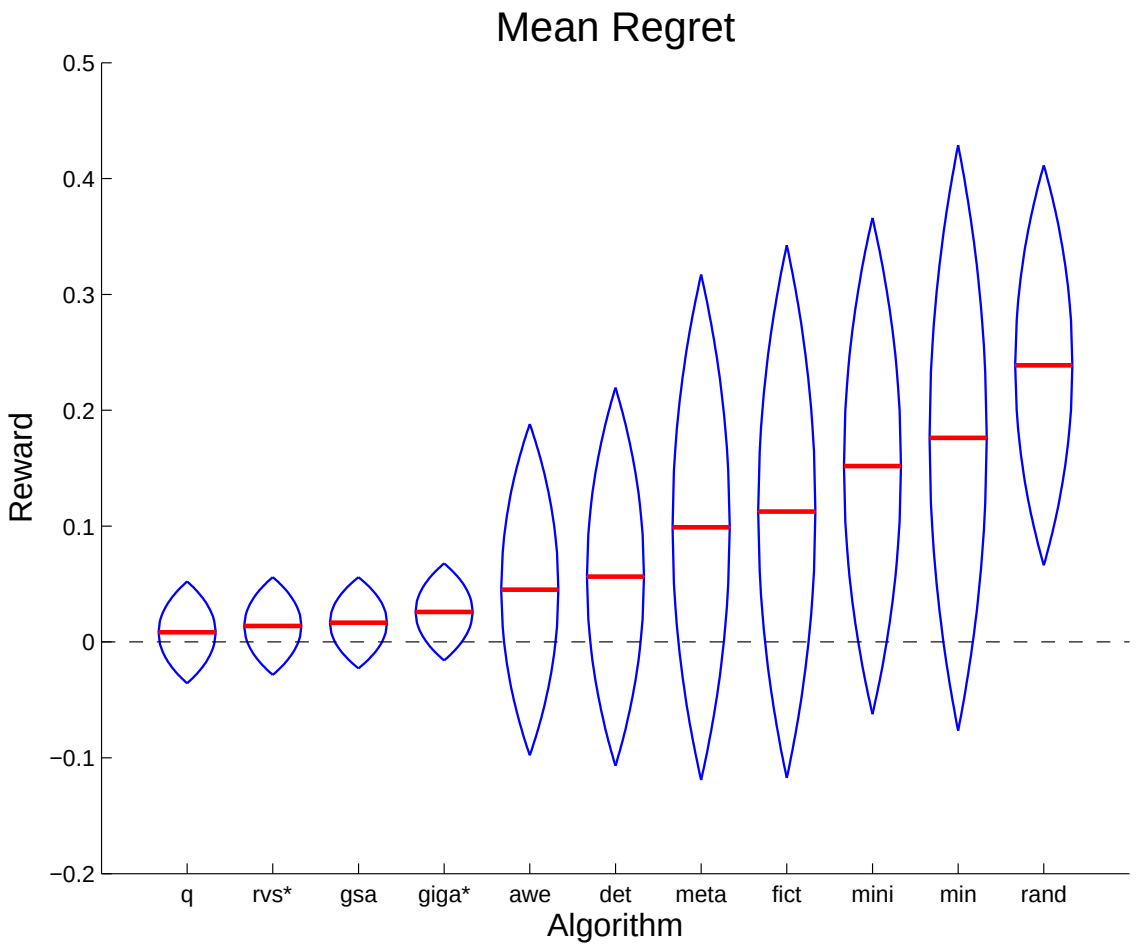


Figure 5.17: A plot that shows the mean regret (bar) for each algorithm and one standard deviation in either direction (indicated by the size of the lens). Algorithms with an asymptotic no-regret guarantee are indicated with a '$*$'.

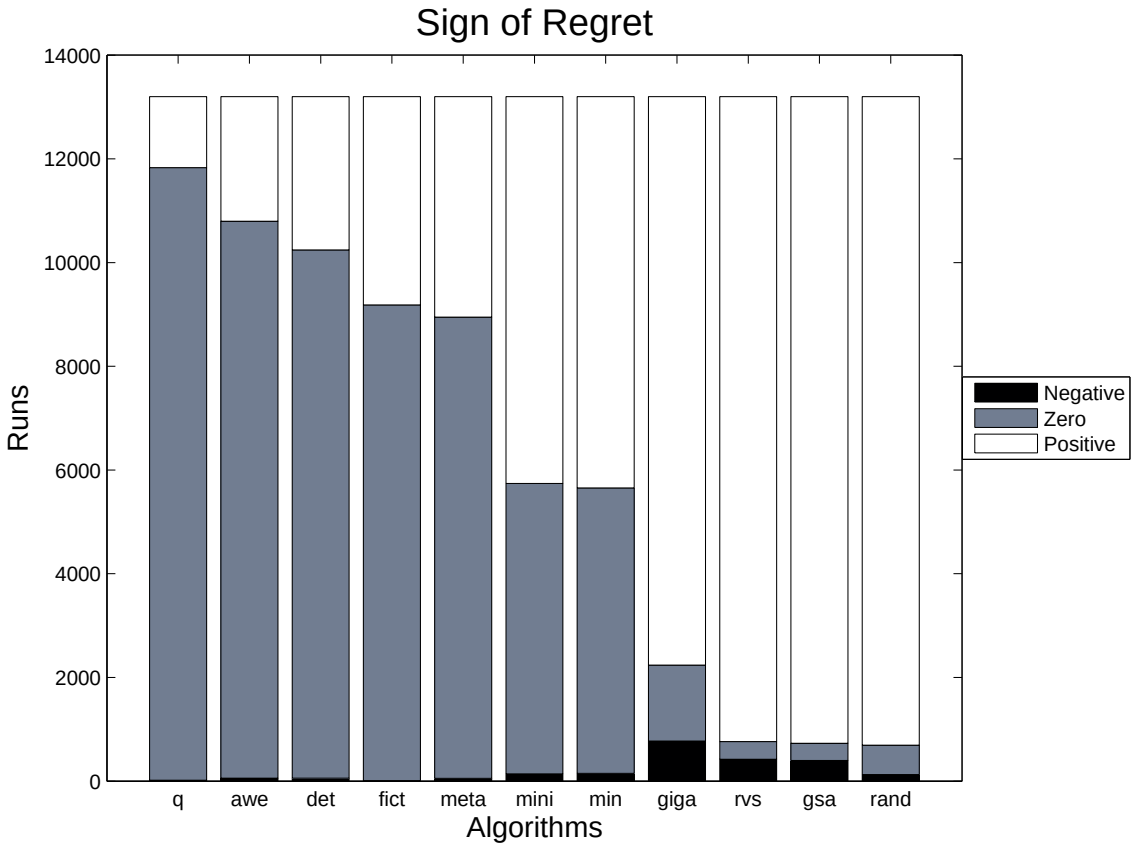


Figure 5.18: The number of runs for each algorithm that have negative, zero, or positive regret.

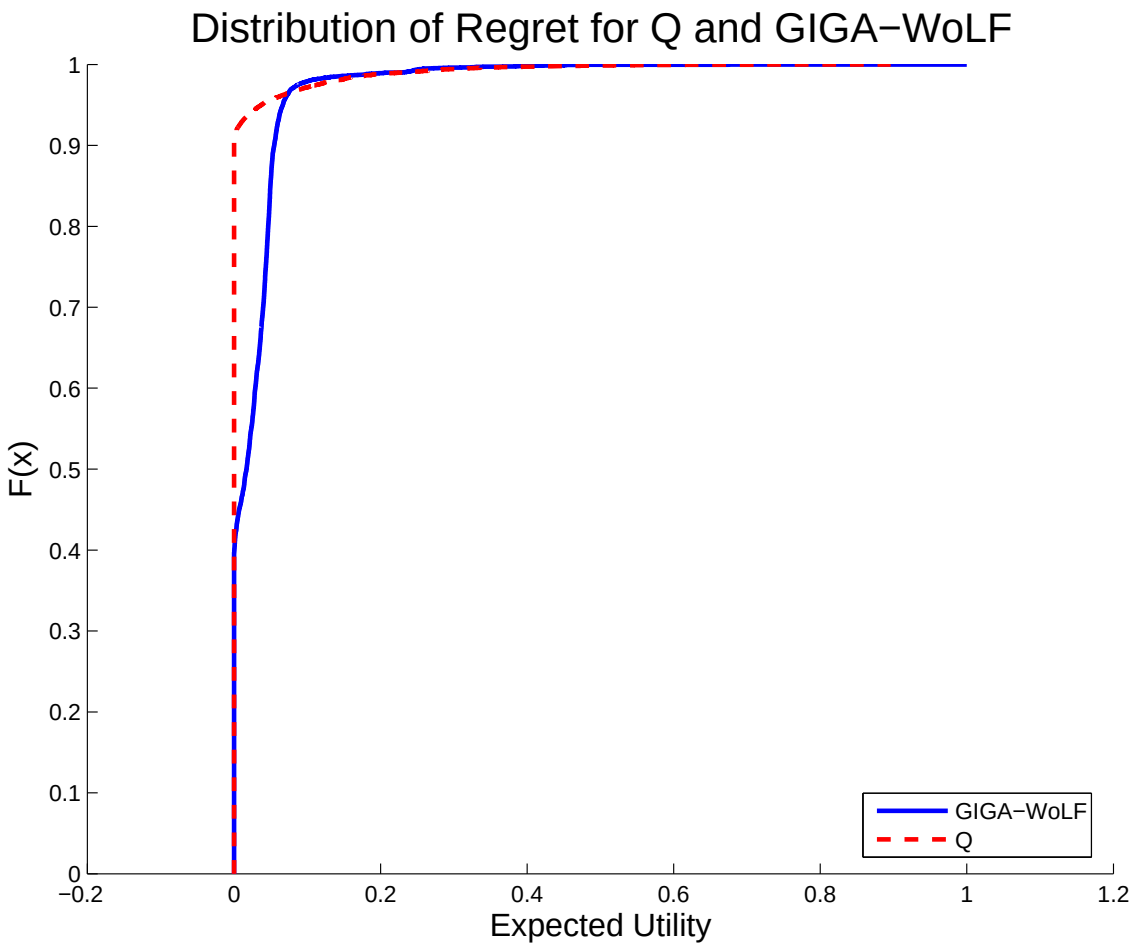


Figure 5.19: The distribution of regret for `Q-learning` and `GIGA-WoLF`.

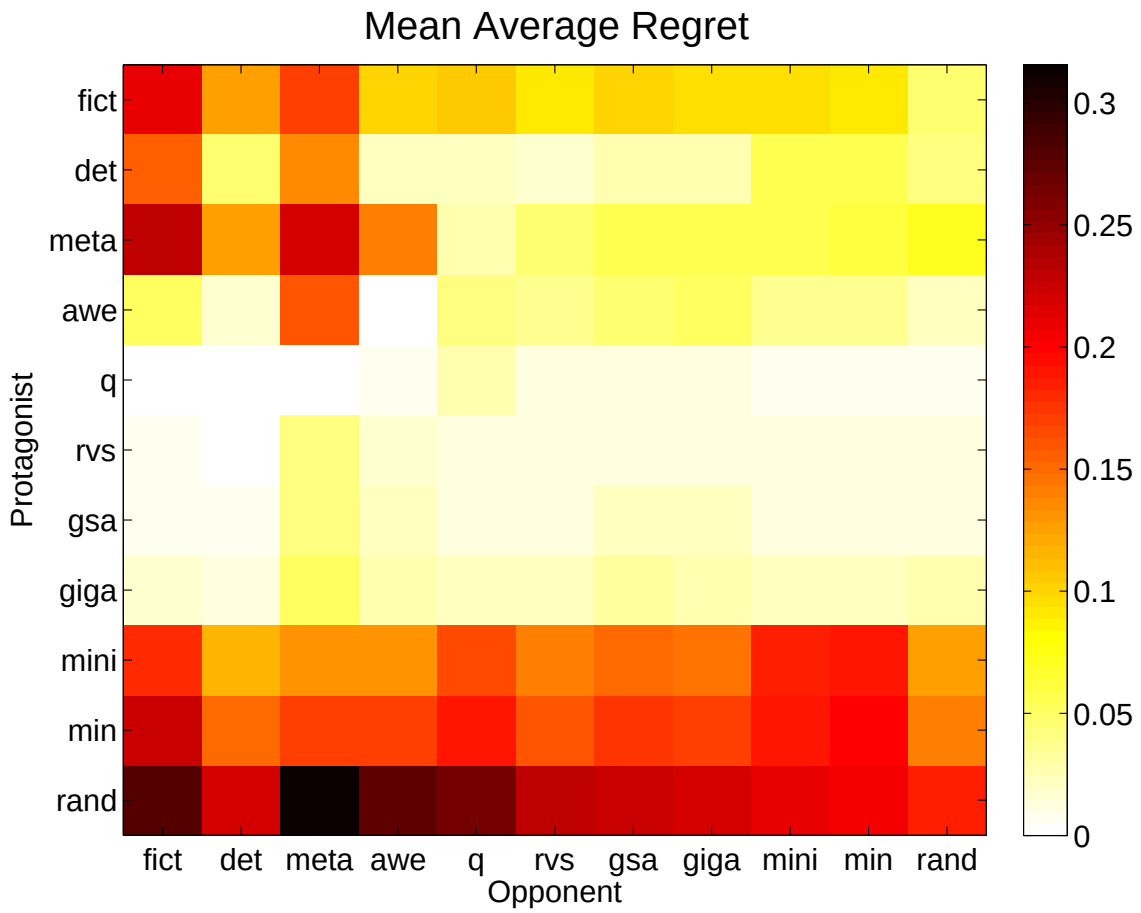


Figure 5.20: Mean average regret, blocked by opponent.

pairing was when Q-learning played against fictitious play: Q-learning attained zero regret in every single game. This seems to indicate that Q-learning converged to a pure-strategy best response in every game against fictitious play. Q-Learning was the only algorithm to do this.

**Observation 11** *Q-learning, GIGA-WoLF, GSA and $RV_{\sigma(t)}$ are rarely probabilistically dominated in terms of regret.*

When blocking on the opponent, some strong probabilistic dominance trends emerge between the different distributions of regret. For example, the gradient algorithms were never dominated by any other algorithm. Q-learning is also seldom dominated. It was only dominated once by AWESOME when playing against an AWESOME opponent. However, it is unsurprising that AWESOME attains lower regret than Q-learning in this case since it has special machinery for converging to a stage-game Nash equilibrium in self-play. In a Nash equilibrium both agents are best-responding so both accrue zero regret. Fictitious play, on the other hand, was frequently dominated, especially by AWESOME, determined, Q-learning and to a lesser degree meta. Both determined and Q-learning dominated fictitious play against $10$ opponents (Q-learning was the exception for determined and *vice versa*), and AWESOME dominated fictitious play on $9$ opponents (GIGA-WoLF and meta were the only opponents for which AWESOME did not dominate fictitious play).

Similar observations result from blocking on the game generators. Q-learning dominated other algorithms frequently—particularly fictitious play (on $9$ generators), meta ($8$ generators), and AWESOME (on $8$ generators)—while avoiding domination by another algorithm. Fictitious play was dominated frequently by Q-learning ($9$ generators), determined ($6$), AWESOME ($6$) and meta ($4$) on many different instance generators.

## 5.3 Convergence-Based Metrics

In this section we shift away from looking at metrics that are based on reward and instead look at metrics that are based on empirical frequency of action. We focus on various ideas of convergence, from a weak form that merely insists that the empirical distribution of actions is stationary to much stronger forms that insist on convergence to a restricted class of stage-game Nash equilibria. We will also consider whether average payoofs are consistent with the infinitely repeated game equilibrium.

When we look at convergence, we will consider a sequence of action to be either converged or not: we will not have an extended discussion about how some sequences are "closer" than others to convergence.

One issue in studying convergence based on empirical data is dealing with runs that appear "not quite" to have converged because of random fluctuations in the empirical action frequency. The Fisher exact test (FET) and Pearson's $\chi^2$-test can be used for checking whether two multinomial samples are drawn from a distribution. For example, we might check whether a later empirical

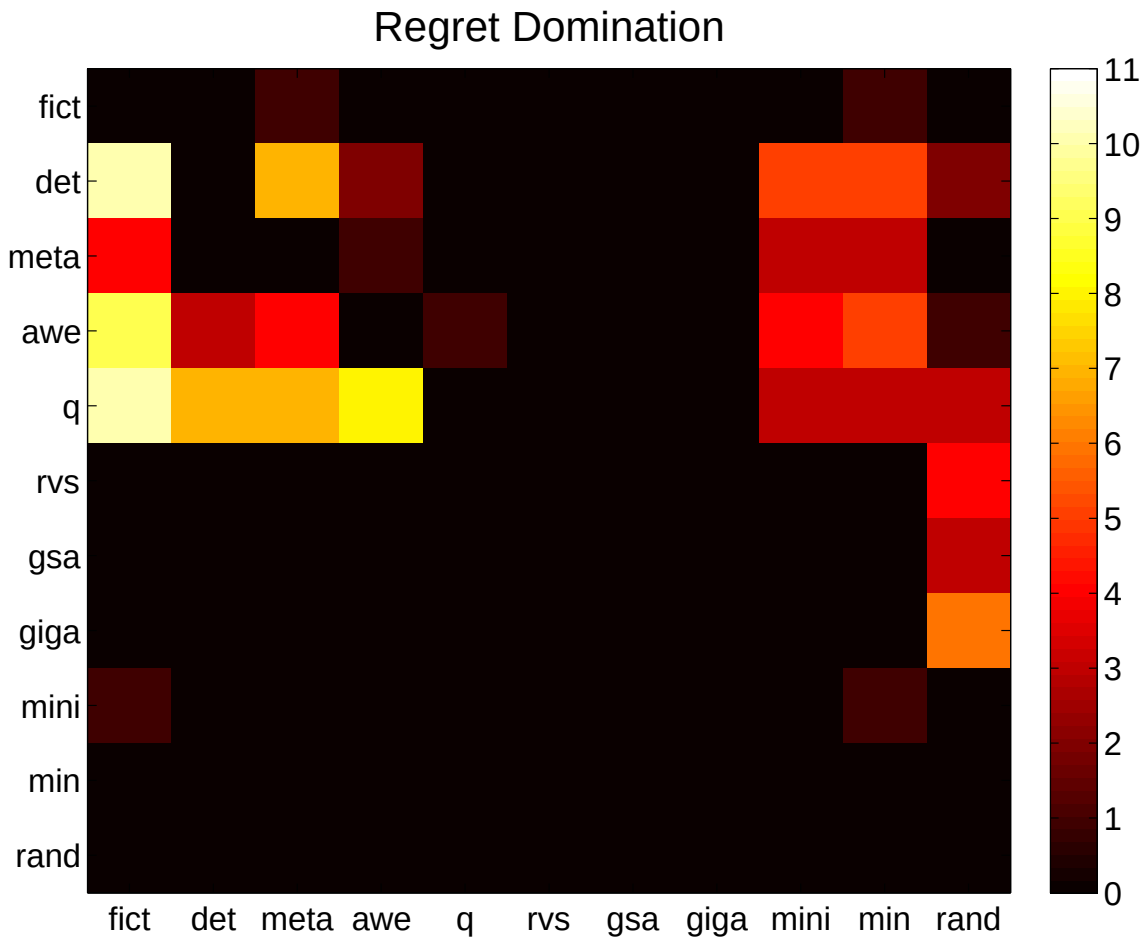


Figure 5.21: The number of opponents for which the algorithm on the ordinate probabilistically dominates the algorithm on the abscissa. For example, `Q-learning` probabilistically dominates `fictitious play` on PSMs involving ten out of eleven possible opponents.

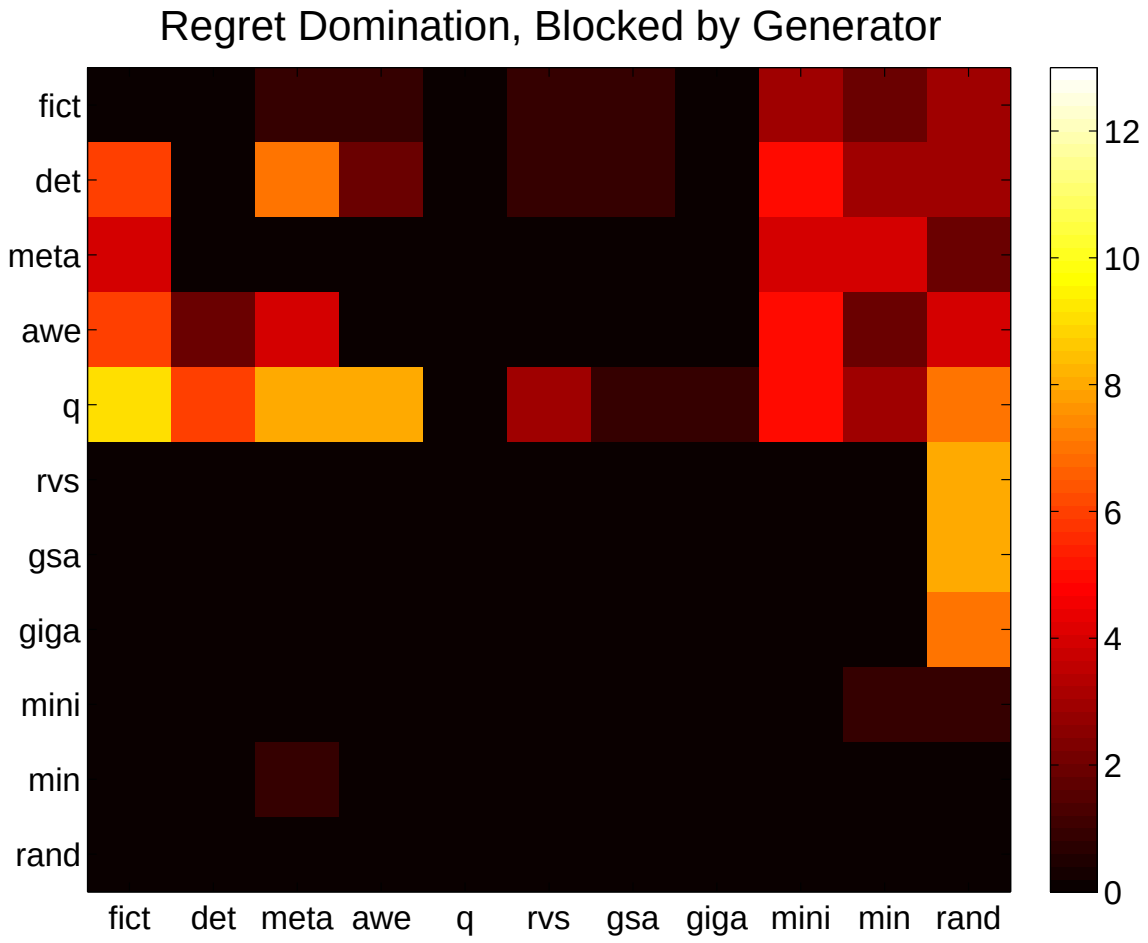


Figure 5.22: The number of generators for which the algorithm on the ordinate probabilistically dominates the algorithm on the abscissa.

action distribution was drawn from the same distribution as an earlier sample (establishing that the empirical mixed strategies were stationary) or that an empirical action distribution profile was drawn from the same distribution as a stage-game Nash equilibrium.

Each test was unfortunately inappropriate for the situations that we needed. The $\chi^2$ test does not handle situations where some of the actions are rare or not present and the FET was computationally expensive, and the implementation of it that we used [40] failed on some of the larger and more balanced action vectors (typically in the $10 \times 10$ case).

Instead, we used the incomplete set of FET results to calibrate a threshold based on vector distance where any two vectors that were closer than the threshold $\theta$ were considered to be the same. We calibrated $\theta$ using a receiver operating characteristic curve. The incomplete FET results were used as ground truth, and we plotted the change in true positive rate and false positive rate as we varied $\theta$. We picked the threshold that lead to equal number of false positives and false negatives. Based on this ROC analysis, we picked a $\theta$ of $0.02$.

### 5.3.1 Strategic Stationarity

The weakest form of convergence that we will look at is whether or not the algorithms converge to a stationary strategy profile. This is interesting in its own right, but is also a necessary condition for stronger forms of convergence. We consider a run to be stable if the joint distribution of actions is the same in the first and second half of the recorded iterations, using $\ell_\infty$-distance. This is a joint property of both algorithms, so while `determined` and `random` play stationary strategies they may still participate in runs that are not stable.

To check how successful our threshold criterion is at detecting stationarity we looked at the results for two algorithms that always use stationary strategies. `Determined` was found to be stable in $99.5\%$ of self-play matches and `random` was found to be stable in $92.0\%$ of self-play matches. When playing each other, they were found to be stable in $94.8\%$ of their runs. The differences that exist between these cases are likely because `determined` has weakly smaller supports than `random` and mixed strategies with smaller supports are more likely to produce empirical action distributions that are close to the original strategy. We note that a false positive rate of between $0.5\%$ and $8\%$ larger than might be hoped, but nevertheless defer improved criterion for empirical convergence to future work.

`GIGA-WoLF` and `GSA` were the least likely to be stable—particularly in self-play, against each other, or against `meta` (see Figure 5.23). Their striking instability with `meta` is potentially because they trip `meta`'s internal stability test and change its behaviour. However, `AWESOME` also has an internal check like this, but the stability of the `GIGA-WoLF` and `GSA` are not noticeably different between matches with `AWESOME` and with `Q-learning` (which has no such check). $\mathsf{RV}_{\sigma(t)}$, the other gradient algorithm, was more stable than `GIGA-WoLF` and `GSA`. This might be because $\mathsf{RV}_{\sigma(t)}$ had a more aggressive step length: the parameters used in this experiment for `GIGA-WoLF` and `GSA` were taken from [7] and they were intended to produce smooth trajectories and rather than fast convergence.

`Meta`, `determined`, `fictitious play` and `AWESOME` were, for the most part, quite

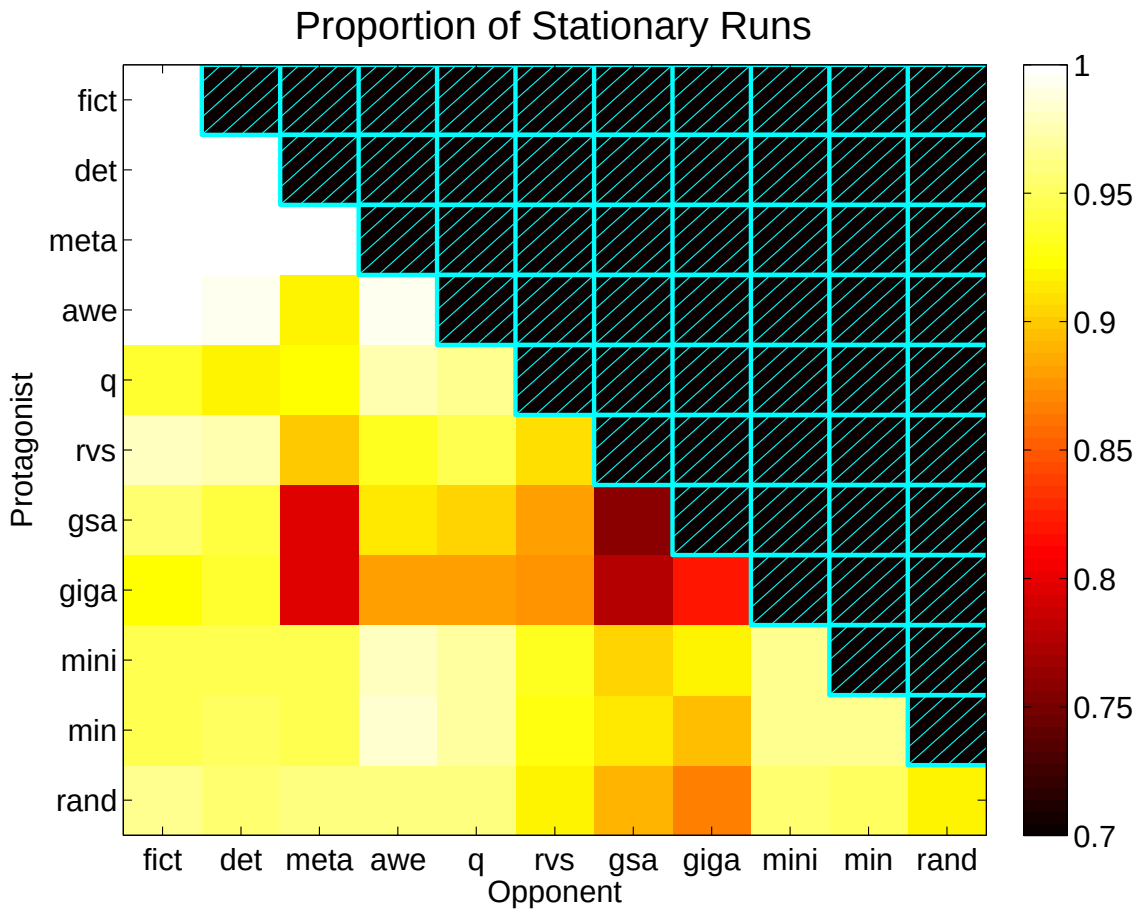


Figure 5.23: Proportion of stationary runs, blocked on opponent. This intensity map is symmetric and we removed redundant entries for clarity.)

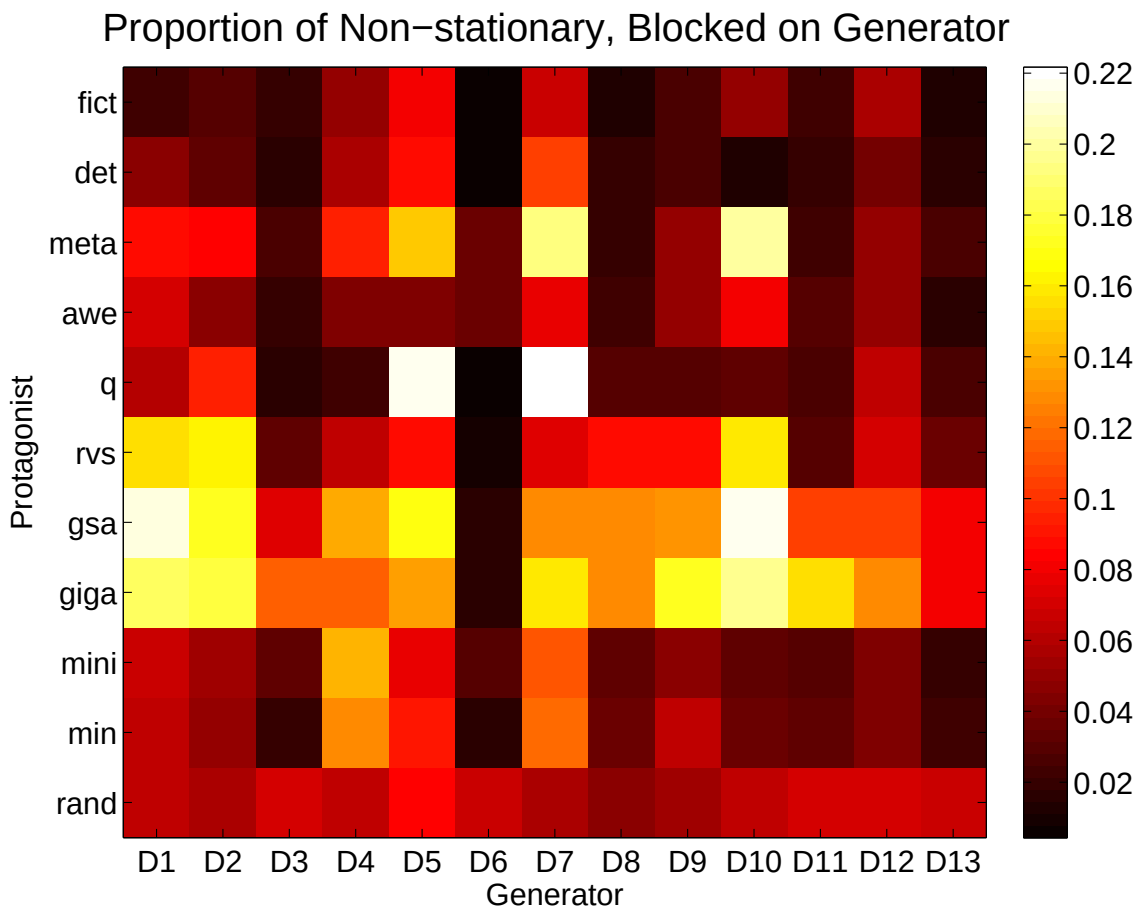


Figure 5.24: Proportion of non-stationary runs, blocked on generator and protagonist

good at achieving stationarity with each other and in general. `meta` and `fictitious play` were particularly strong against each other, and always reached a stationary strategy profile. The only exception to the stability in this group was `AWESOME` vs. `meta`; this pairing was unstable in $10.3\%$ of runs. We are not sure why this is, but it likely has to do with the discrete behavioural changes that both algorithms undergo when their internal states change.

There were a number of problem generators for the different algorithms (see Figure 5.24). For example: generators D1, D2, and D10 created instances that were particularly difficult for the gradient algorithm in terms of strategic stability; `Q-Learning` was weak on both D5 and D7; and `meta` tended to be unstable on D5, D7 and D10. However these unstable instances were rare regardless of the algorithm paring. The vast majority of runs found a stationary strategy profile. Even `GIGA-WoLF`, which was the algorithm least likely to stabilize, found stationarity in $87.0\%$ of its runs (see Figure 5.25).

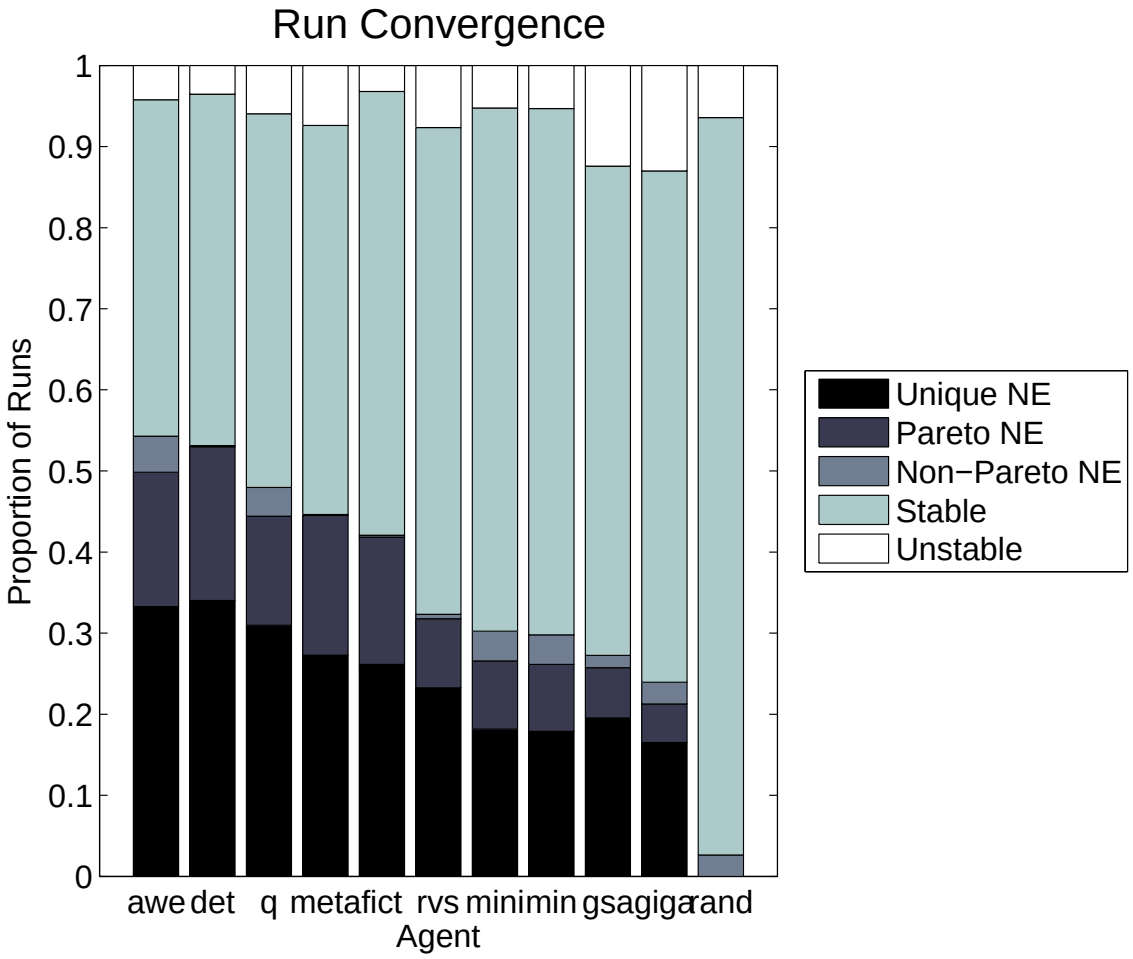


Figure 5.25: The proportion of runs that were stationary, converged to a Nash equilibrium or converged to a Pareto-optimal Nash equilibrium.

### 5.3.2 Stage-Game Nash Equilibria

A subset of the stable runs settled on one of the stage-game Nash equilibrium. For some algorithms, Nash equilibrium convergence was reasonably common: AWESOME converged in $54.3\%$ of its runs, and determined converged in $53.1\%$ of its runs. Determined was better at AWESOME at converging to a Pareto-optimal Nash equilbrium (a Nash equilbrium that is not Pareto-dominated by any other Nash equilibrium). Indeed, AWESOME most frequently converged to a Pareto-dominated equilibrium. This this likely has to do with the way that our implementation of AWESOME picked its 'special' equilibrium.[8] It was simply the first equilibrium found by the Lemke-Howson algorithm, without attention to whether it was *e.g.* Pareto-dominated. AWESOME also tended to attain lower reward when it is converged to a Pareto-dominated Nash equilbrium than when it did not converge or converged to a dominated Nash equilibrium.

Figure 5.26 gives the convergence results for self-play. One of the first things that jumps out is how often determined manages to converge to a Nash equilibrium in self-play. This indicates that the games we choose had an important characteristic: many possessed a single Nash equilibrium that was the best for both agents. Indeed, we can see that there is a surprisingly high number of games with a unique stage-game Nash equilibrium ($58.5\%$). This is not a general property of all games and so determined's convergence results could be radically different on another set of games. This property likely also affects the convergence properties for the other algorithms.

We see that AWESOME nearly always attains a stage-game Nash equilibrium. Yet, if we recall the discussion about self-play reward from Section 5.1.1, AWESOME received lower reward in self-play than non-self-play runs. Together, this indicates that while AWESOME was successful at converging to a Nash equilibrium, this was not enough to guarantee high rewards in self-play. An interesting tweak to AWESOME would be to use its special self-play machinery to converge to other outcomes that are not stage-game Nash equilibria, such as the socially-optimal outcome of the stage game or the Stackelberg-game equilibrium. The aim of this adjustment would be to improve self-play reward results while keeping AWESOME resistant to exploitation by other algorithms.

### 5.3.3 Repeated-Game Nash Equilibria

So far, we have been looking at the equilibria for the stage game. The algorithms are actually playing a repeated game, however, and we now turn to analyzing properties of this repeated game. We look at enforceability—achieving payoff profiles in which both payoffs exceed their respective maxmin values—since enforceability is a necessary condition for any repeated game Nash equilibrium. Unfortunately, a sufficient condition for repeated game equilibria would involve testing whether each algorithm has correct off-equilibrium behaviour—punishments, in particular—built into its strategy that prevents profitable deviation by its opponent. While the algorithms that we looked at lack off-equilibrium punishments, it is still interesting to see how frequently the algorithms converge to payoff profiles that are realizable by repeated game Nash equilibria.

[8] The original paper, Conitzer and Sandholm [13], left the method of picking the 'special' equilibrium unspecified.

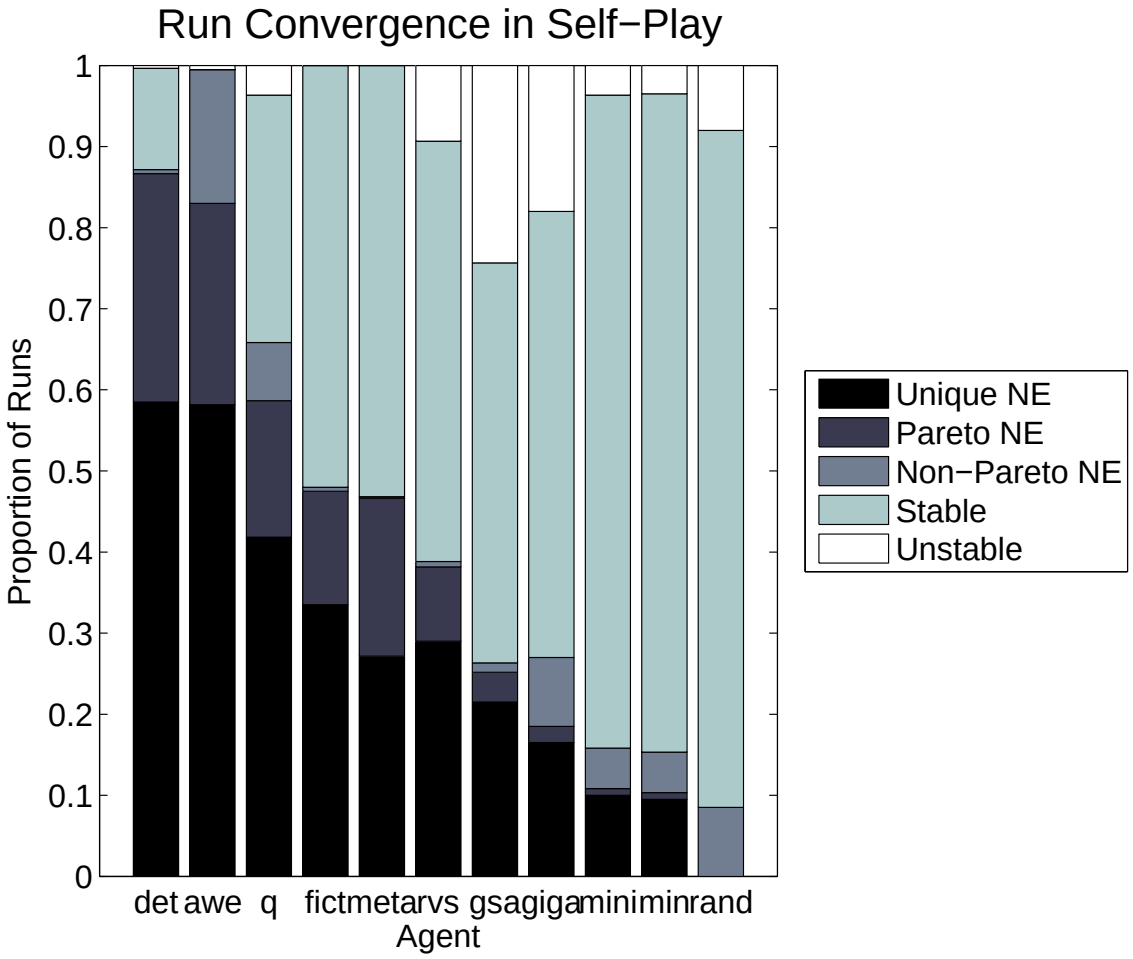


Figure 5.26: The proportion of self-play runs that were stationary, converged to a Nash equilibrium or converged to a Pareto-optimal Nash equilibrium.

**Observation 12** *Q-Learning was involved in matches with payoff profiles consistent repeated game Nash equilibrium more often than any other algorithm.*

Of the algorithms that we examined, Q-learning most frequently had runs that were consistent with a repeated game Nash equilibrium. It was consistent with a repeated game equilibrium in $76.8\%$ of its runs (see Figure 5.27). Determined and AWESOME were the next most frequently consistent, and were consistent in $75.0\%$ and $73.8\%$ of their runs, respectively. Consistency with a repeated game Nash equilibrium is common, but not universal. Even after $90\ 000$ adaptation iterations, no algorithm was completely successful at achieving enforceable payoff profiles. We note that if a protagonist algorithm is playing against a particularly unsuccessful opponent, such as random, it might fail to achieve an enforceable payoff profile simply because its opponent does not achieve an enforceable payoff. In particular, if the opponent fails to achieve an enforceable payoff, it would be unfair to conflate runs where the protagonist was able to achieve an enforceable payoff with runs where it also failed. We looked at the proportion of matches where each algorithm attained an enforceable payoff in Section 5.1.2, and we included these results in Figure 5.27 for comparison. Again, no algorithm achieved payoffs above the maxmin value in all of its runs.

## 5.4 Links Between Metrics

We argued earlier in this chapter that reward is the most fundamental metric and that the other metrics, like regret, can be seen as 'standing in' for reward. Therefore, it is important to compare reward results to the other metrics to see if these alternative metrics are reasonable substitutes for reward. For example, is high reward linked with converging to a Nash equilibrium? We saw in § 5.3.2 that while AWESOME was very good at converging to a Nash equilibrium in self-play, it did not get especially high reward in these runs. Is this a general trend, or a special property of AWESOME? What are the other links between the different aspects of performance?

### 5.4.1 Linking Reward With Maxmin Distance

**Observation 13** *Algorithms tend to receive larger rewards when runs are also enforceable.*

For most of the algorithms there is a clear and strong relationship between enforceability and reward. This is to be expected, because a run is only enforceable for an algorithm if the algorithm attains high reward. Maxmin distance is positively correlated with reward for all algorithms. This was tested with Spearman's rank correlation test (§ 4.3.2) at a significance level of $\alpha = 0.05$. If we block on game generator (Figure 5.28), the results are largely the same with a few differences. There are a few insignificant results, mostly on D11. For example, minimax-Q is negatively correlated on D11. Interestingly, minimax-Q-IDR still exhibits positive correlation.

For all algorithms the mean reward for enforceable runs is higher than the mean reward for unenforceable runs. However, when we compare reward SQDs based enforceable runs and unenforceable runs, the former does not always probabilistically dominate the latter. (Figure 5.29). The

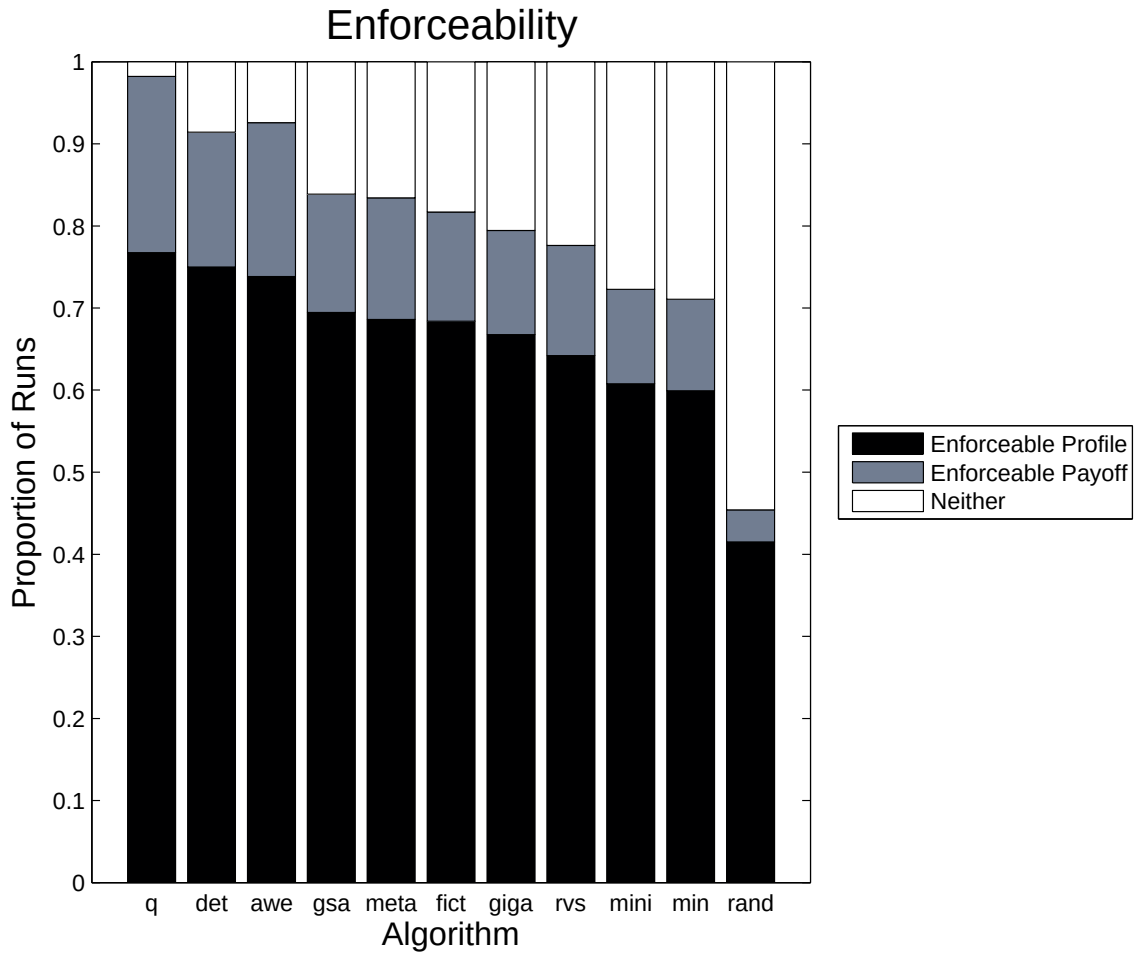


Figure 5.27: Proportion of PSMs with enforceable payoffs and payoffs profiles achieved, by algorithm.

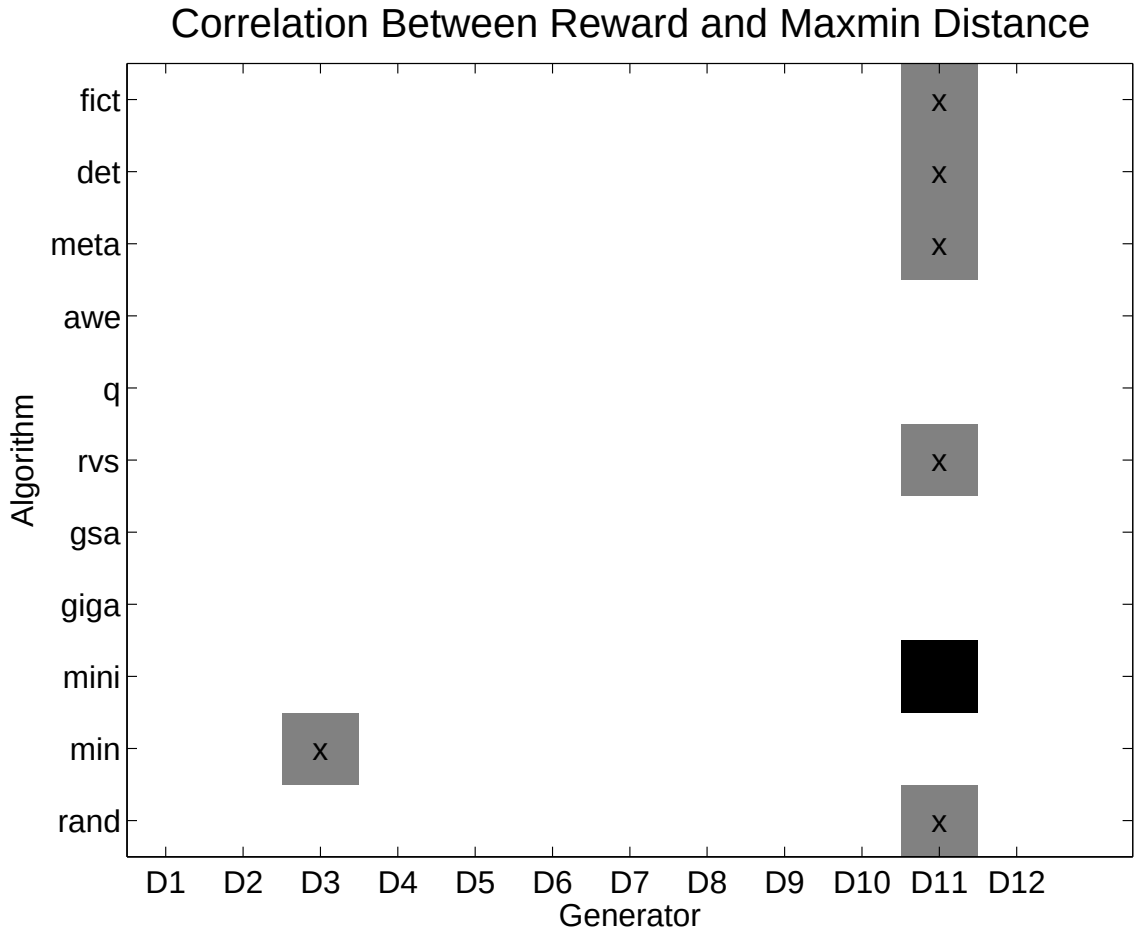


Figure 5.28: The sign of correlation between reward and maxmin distance for each algorithm and each game generator. A white cell indicates positive correlation, a black cell indicates negative correlation, and a grey cell with an 'x' indicates insignificant correlation.

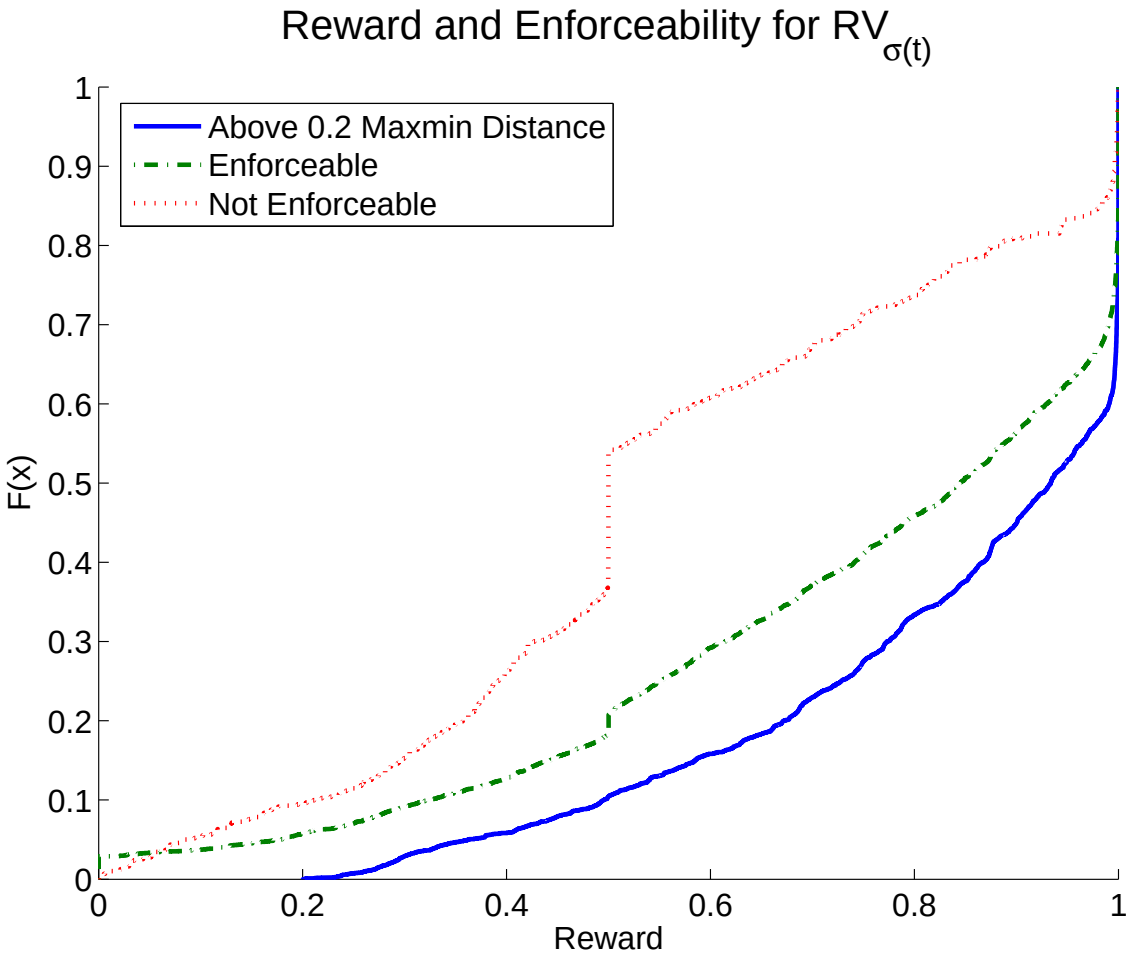


Figure 5.29: The distribution of reward for $\mathsf{RV}_{\sigma(t)}$ when conditioning on different maxmin distances. For example, the 'Above 0.2 Maxmin Distance' curve is the empirical CDF curve that represents the distribution of reward for all runs that have a maxmin distance greater than $0.2$.

exceptions here are the gradient algorithms and `Q-learning`. For both the gradient algorithms and `Q-learning` there tend to be many enforceable zero-reward runs. The majority of these zero-reward enforceable runs occur in D10—*Traveller's Dilemma*. All instances from D10 have a security value of zero so D10 is one of the few generators where algorithms can get an enforceable zero reward run. All algorithms frequently have zero-reward runs on instances of D10 but they tend to get unenforceable zero reward runs in other games more frequently than either the gradient algorithms or `Q-learning`—the enforceable zero-reward runs stand out more for the gradient algorithms or `Q-learning` since they have fewer unenforceable $0$ reward runs. If we exclude runs from D10 then `GIGA-WoLF`, `GSA` and $\mathsf{RV}_{\sigma(t)}$ also exhibit domination, but `Q-learning` still does not—though the cross-over is small (Figure 5.30). We also compared the reward SQDs for runs with reward strictly higher than the maxmin value to the reward SQDs for unenforceable runs, and we found for all algorithm except `Q-learning`, the strictly enforceable maxmin reward SQDs dominates the unenforceable SQDs.

A more detailed explanation for this relationship can be see in a bivariate histograms such as Figure 5.31. This figure is a representative example of the relationship that exists between reward and maxmin distance for all algorithms except `random`. Reward bounds maxmin distance: if one gets a reward of $x$, then maxmin distance must be between $x - 1$ (security value is $1$) and $x$ (security value of $0$). These constraints create a feasible regions that is a parallelogram with points $\{(1, 1), (1, 0), (0, 0), (0, -1)\}$, where reward is the first coordinate and maxmin distance the second.

There are two prominent ridges in this histogram. The first is the ridge formed by runs having zero or close to zero maxmin distance, and the second is formed by runs with a reward of close to $1$. The three bins with the most runs are all close to zero maxmin distance with rewards of $0$, $0.5$, and $1$. The first bin (reward of $0$) largely corresponds to runs from D10 (although a few runs came from D7 and one run came from D13), the middle bin is mostly composed of runs from D6, and the final bin(reward of $1$) consists of instances drawn from either D3, D7, or D11. Again, these observations are for $\mathsf{RV}_{\sigma(t)}$ but are echoed in histograms for the other algorithms.

These ridges tell us something rather surprising: runs tend to get either close to the maximum reward or to the security value. There seems to be little else in the way of a trend between the two metrics beyond the fact that one bounds the other. This observation is not limited to game instances from one generator, and many generators contribute to these ridges.

### 5.4.2 Linking Reward With Regret

**Observation 14** *There is a link between obtaining large reward and low regret.*

There is also a link between having low regret and high reward. Regret and reward are negatively correlated for all algorithms (Spearman's rank correlation test; $\alpha = 0.05$): high reward is linked with low regret. When blocking on generators, we see that D10 induces positive correlation for all algorithms except `determined`, and this is sensible: algorithms get better reward when they are not best responding in this game (the unique Nash equilibrium is one of the worst outcomes

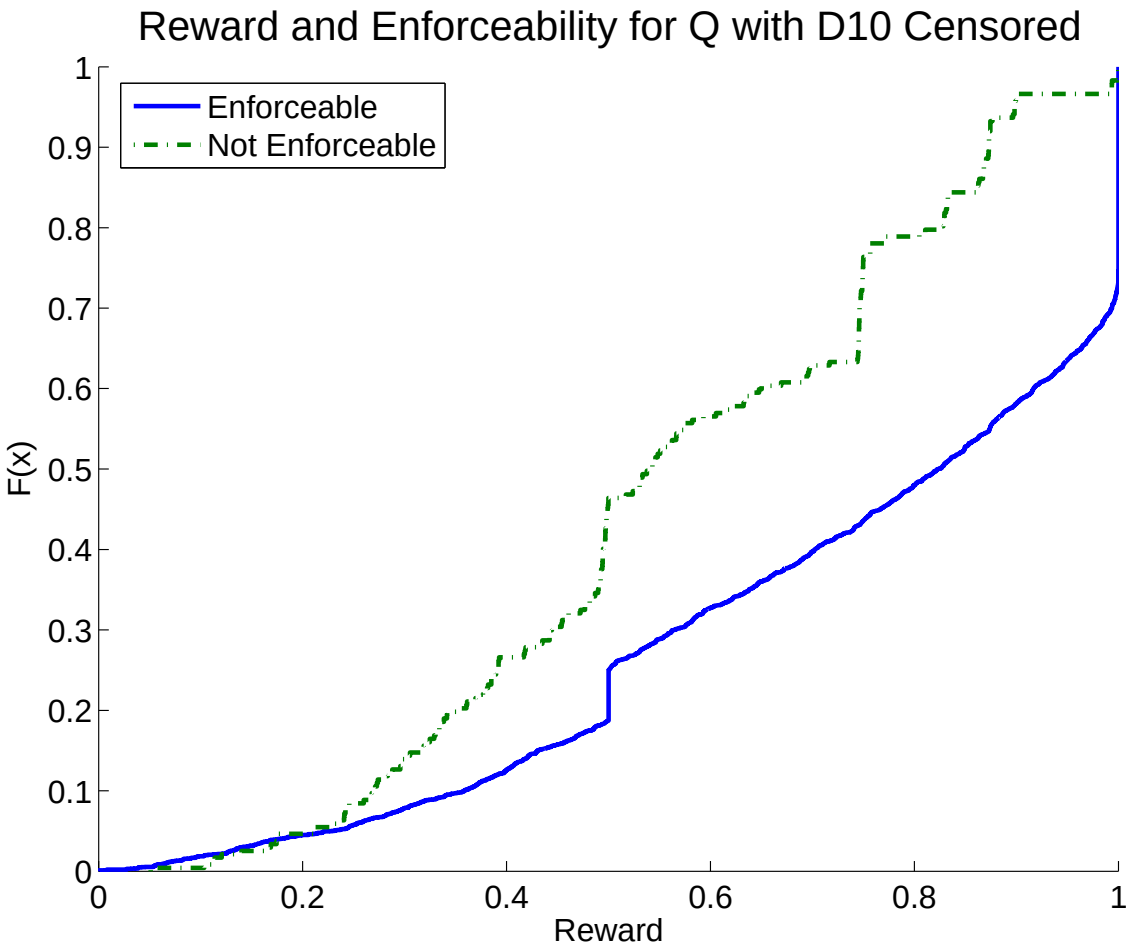


Figure 5.30: The distribution of reward for `Q-learning` when conditioning on different enforceability. Runs from D10 were excluded.

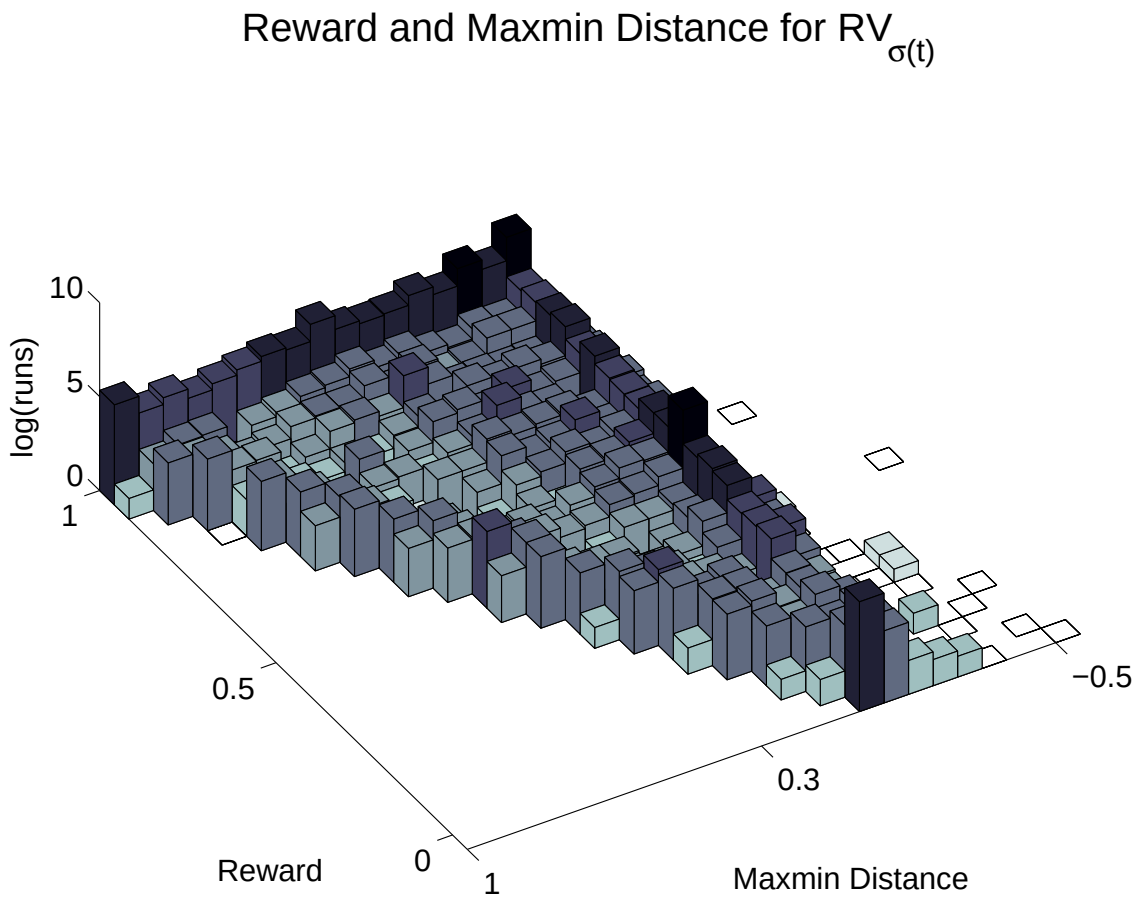


Figure 5.31: Bivariate histogram showing reward and maxmin distance for $\mathsf{RV}_{\sigma(t)}$. The $25 \times 25$ uniform bins were used. The height of the bins is shown on a log scale.

of the game).

Since relatively few runs accrued significant negative regret (see § 5.2), the most important division is between runs that attained positive regret and runs that attained non-positive regret. For most of the algorithms, their performance in non-positive runs probabilistically dominated their performance in positive regret runs. There were some exceptions; for example `Q-learning` and `GIGA-WoLF` experienced cross-overs. While for `Q-learning` the cross-over was relatively minor, the cross-over for `GIGA-WoLF` was more significant: runs that attained positive regret less often attained zero reward (Figure 5.33).

There is an even more dramatic result: the positive-regret runs dominated the non-positive runs for `GSA` and $\texttt{RV}_{\sigma(t)}$. These two gradient algorithms exhibited behaviour that none of the other algorithms displayed: runs with positive regret had better reward characteristics than runs with zero or negative regret. This phenomenon did not seem to be due to only one generator nor any one opponent. We can note that the probabilistic domination visually seemed to be the weakest when PSM involving *Traveler's Dilemma* were censored.

The bivariate histograms for the different algorithms show that there is a ridge for all of them where regret is zero (the histogram for `AWESOME` is presented in Figure 5.34). This ridge has a prominent bin for runs with regret close to zero and reward close to one. This bin indicates good runs where algorithms are best-responded and got the game's maximum reward. This bin is the largest in terms of runs for most of the algorithms. However, there are other interesting bins that are only observed in some of the algorithms. In particular, all algorithms except for the gradient algorithms and `Q-learning` had a number of runs with reward close to zero and regret close to one. These runs are horrible: not only did the algorithms get close to the minimum reward possible, but also they could have switched to a pure strategy and potentially received a reward close to one. Of course, it is possible that a reward of 1 was not attainable with the new action—the opponent could have adapt to the candidate's strategy—but it is hard to imagine that the new action would have done much worse: these runs were already getting close to the minimum possible reward. Furthermore, some algorithms were able to avoid these runs entirely. `Q-Learning`, for example, had no runs of this type and generally avoided high-regret low-reward runs (Figure 5.35). These runs should serve as a focal point for thinking about how existing algorithms should be improved.

### 5.4.3 Linking Reward With Nash Equilibrium Convergence

A lot of work in multiagent systems has focused on algorithms that try to converge to a stage-game Nash equilibrium. Indeed, many algorithms like `determined` and `AWESOME` explicitly try to converge to some stage-game Nash equilibrium. But if one is primarily interested in getting high reward, is converging to an equilibrium desirable? Or, more generally, is proximity to a stage-game Nash equilibrium correlated with obtaining high reward?

**Observation 15** *There is a link between obtaining large reward and being close to a stage-game Nash equilibrium for most algorithms.*

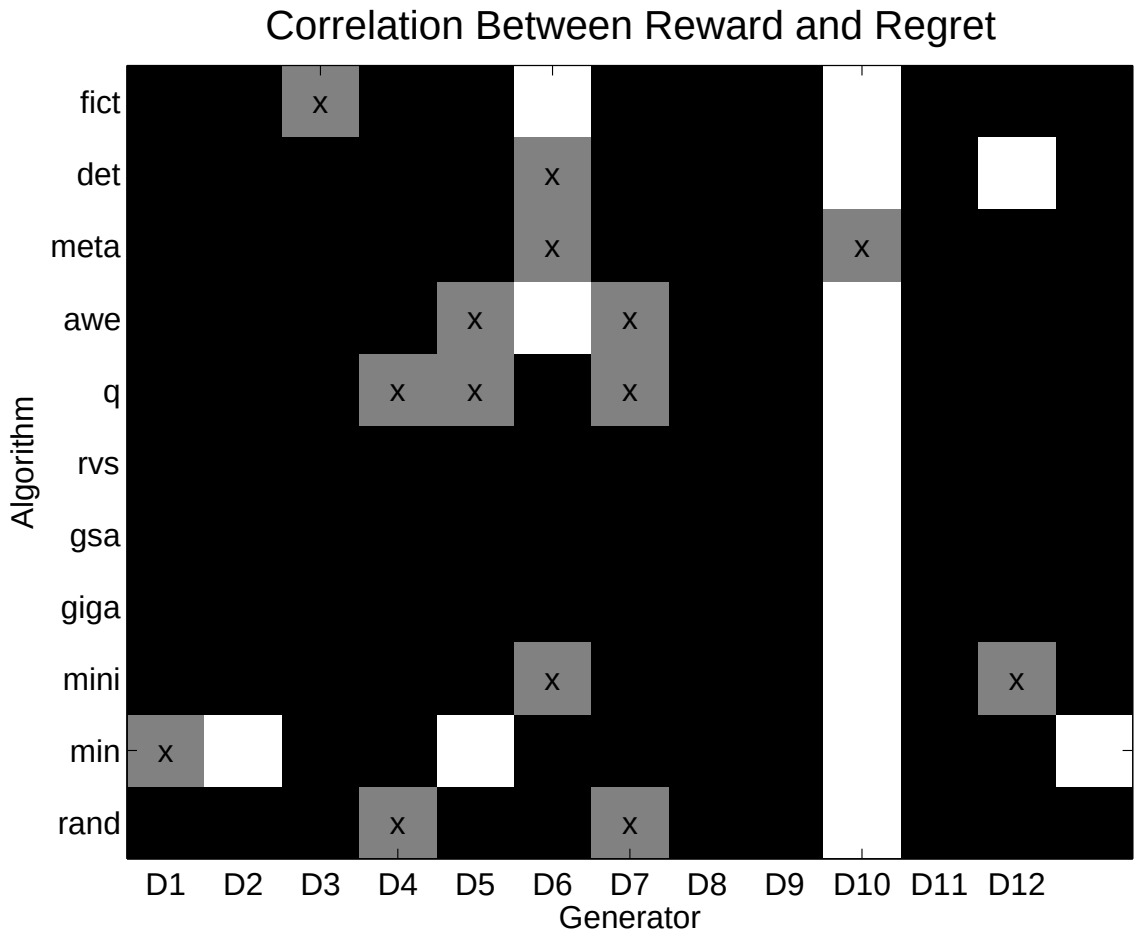


Figure 5.32: The sign of correlation between reward and regret for each algorithm and each game generator. A white cell indicates positive correlation, a black cell indicates negative correlation, and a grey cell with an 'x' indicates insignificant correlation.

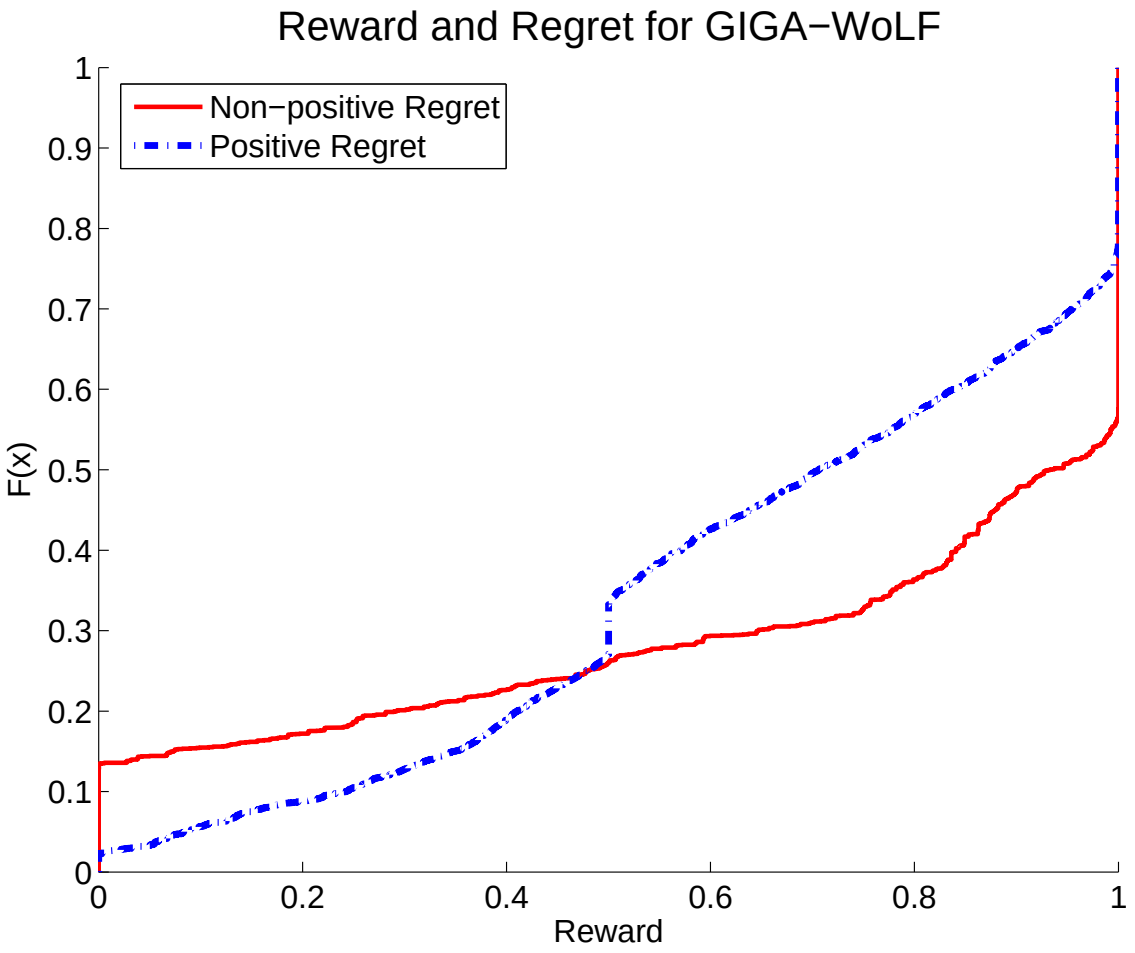


Figure 5.33: A CDF plot showing `GIGA-WoLF`'s performance conditioned on achieving either positive or non-positive reward. Notice that there is less probability mass on 0 reward when `GIGA-WoLF` attains positive regret.

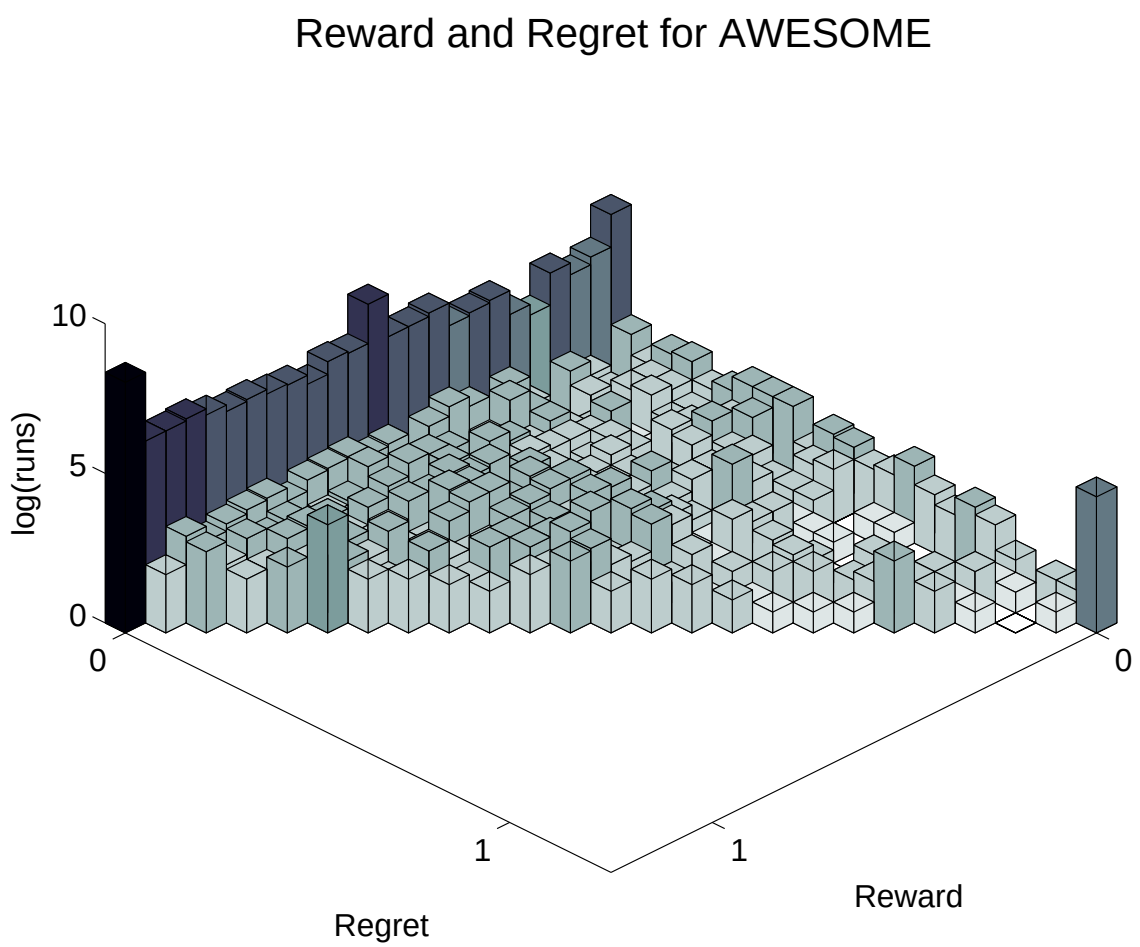


Figure 5.34: A bivariate histogram showing reward and regret for AWESOME. Reward bounds regret: if one gets a reward of $x$, since reward is on $[0, 1]$, one can at most attain a regret of $1 - x$.

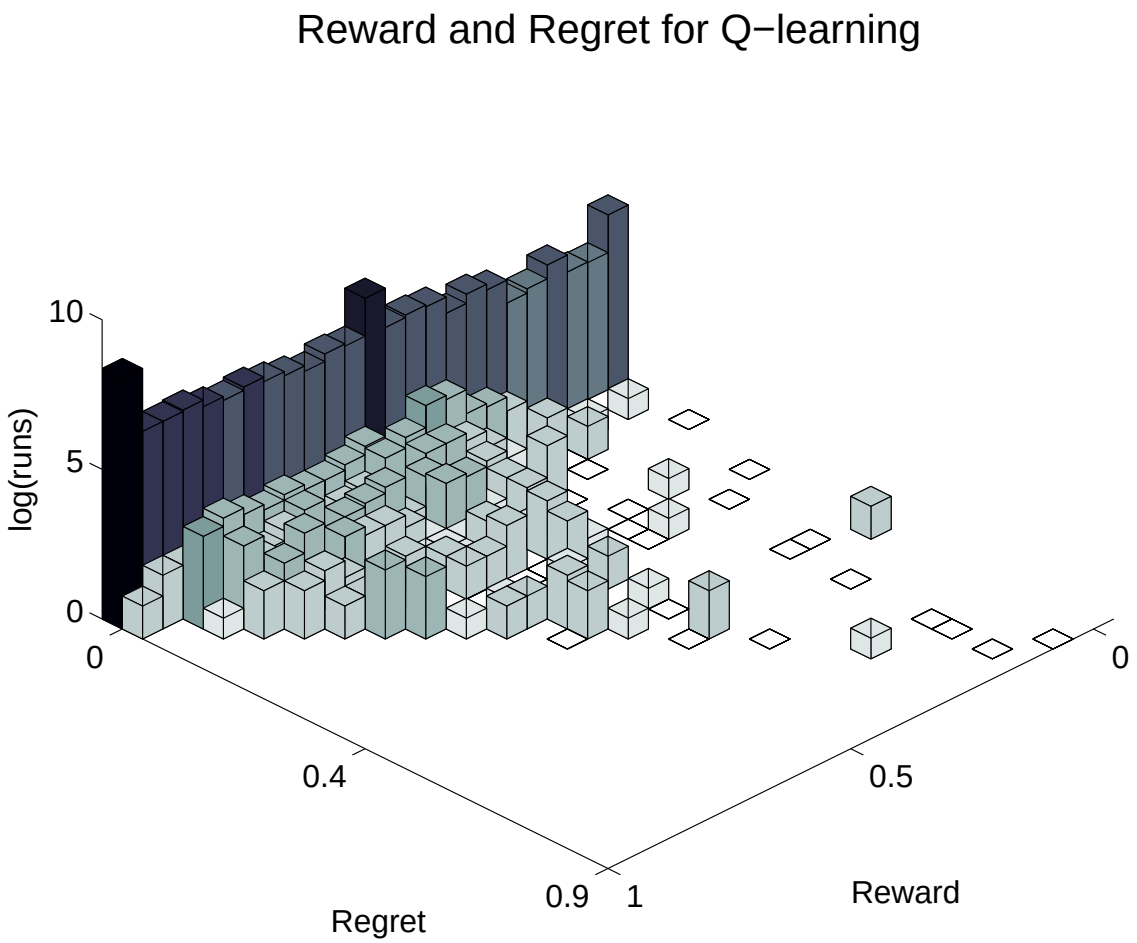


Figure 5.35: A bivariate histogram showing reward and regret for `Q-learning`. Notice that there are fewer low-reward high-regret runs than in Figure 5.34—there is less mass on the right of the plot.

All algorithms have reward that was negatively correlated with $\ell_\infty$-distance to the closest Nash equilibrium (Spearman's rank correlation test; $\alpha = 0.05$). Furthermore, most algorithms were negatively correlated even when we blocked on the game generators (Figure 5.36). There were some exceptions. The most noticeable exceptions were on D6, D10, and D12 there were a number of algorithms with positive correlation: it was better to be far away from the equilibrium. This is especially true on D10 and algorithms received much higher reward if they participated in some other outcome.

Bivariate histograms (Figure 5.37) reveal three major trends for all algorithms: runs are either high in reward, close to a Nash equilibrium, or far from a Nash equilibrium (this does not exclude being high in reward *and* being either close or far from the equilibrium). For most algorithms, being close to a Nash equilibrium and being high in reward is the most common bin. Between $8.1\%$ (`GIGA-WoLF`) and $26.2\%$ (`determined`) of runs have a reward greater than $0.96$ and are less than $0.04$ away from a Nash equilibrium (this is the right-most bin in Figure 5.34). Most algorithms have other strong modes at the other corners of the plots, but these are less promienent that the close and high reward bin.

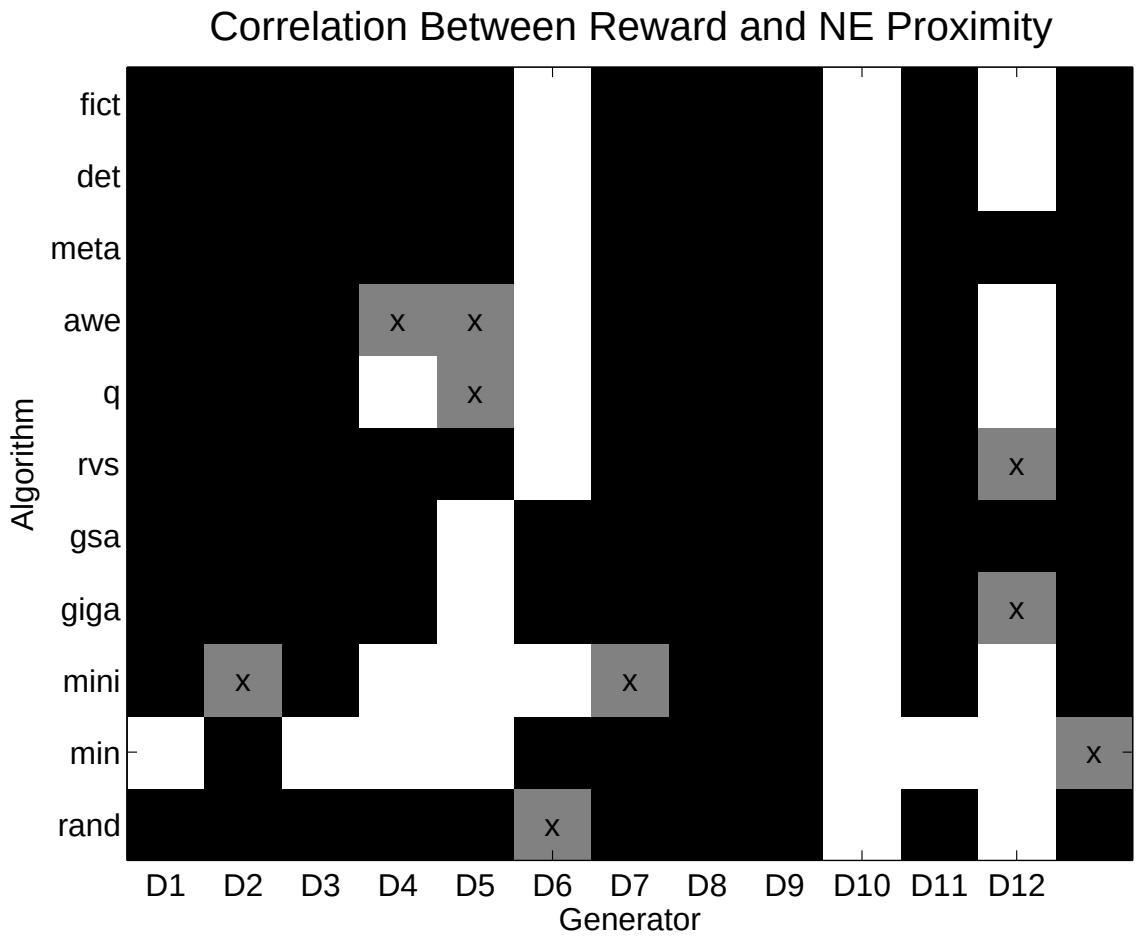


Figure 5.36: The sign of correlation between reward and $\ell_\infty$-distance to the closest Nash equilibrium for each algorithm and each game generator. A white cell indicates positive correlation, a black cell indicates negative correlation, and a grey cell with an 'x' indicates insignificant correlation.

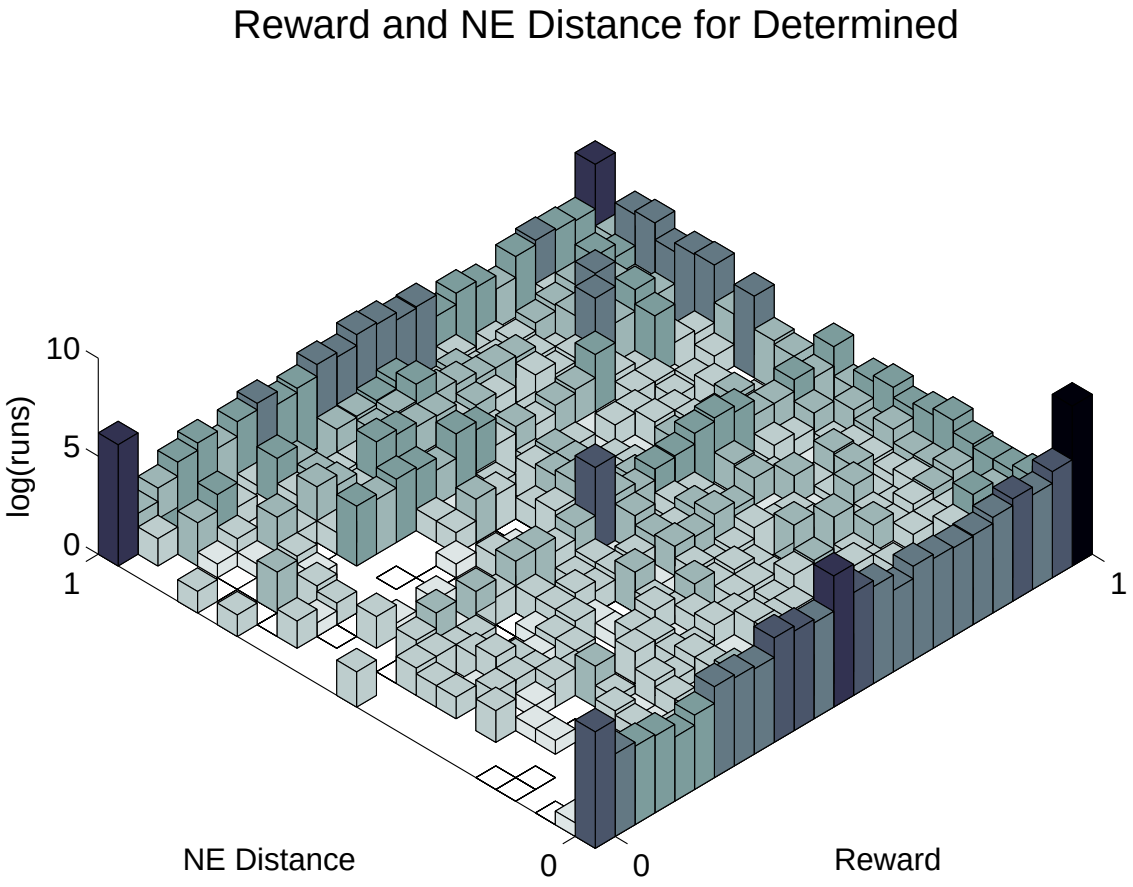


Figure 5.37: A bivariate histogram showing reward and $\ell_\infty$-distance to the closest Nash equilibrium for `determined`.

# Chapter 6

# Discussion and Conclusion

In this thesis, we argued for a standardized testbed for multiagent experimentation. This testbed, while initially more work to create, should offer lasting benefits: it will allow researchers to focus on experimental design and not implementation details. We built a version of this testbed, MALT 2.0, and conducted a sample experiment: we used MALT to evaluate a set of MAL algorithms. We analyzed this experiment in depth and we suggested some analytical methods that are intended not only to be useful for understanding and comparing existing algorithm behaviour but also will be useful for empirically-minded algorithm design.

We observed clear performance results. Firstly, `minimax-Q` and `minimax-Q-IDR` tended to perform poorly from a number of perspectives. These included reward, regret, minimax distance, strategic stationarity, and convergence to a stage-game Nash equilibrium. On the other hand `Q-learning` had excellent results and was frequently better than more recent and sophisticated learning algorithms—such as `GIGA-WoLF` and `meta`—on most game instance generators for most performance metrics. This was a surprising result, and it could be seen as embarrassing for the MAL community: an off-the-self algorithm designed for single-agent learning handily beat the latest multiagent learning algorithms in many respects. This suggests that there is a lot of room for improving the empirical performance of specially-designed MAL algorithm. Indeed, there are a number of areas where efforts should be focused.

Firstly, there simply needs to be more experimentation. Our experiment was large, but it is impossible to answer all empirical MAL questions in one experiment. Some promising directions for extension are:

- More careful examination of the relationship between performance and game properties like size;
- More detailed investigation of behaviour of instances from a single generator. This might give more insight into algorithm behaviour than an experiment with a broad focus like the one presented in this dissertation;
- Further evaluation studies involving more algorithms like Hyper-Q [50] and Nash-Q [23];
- Extension to $N$-player repeated games and stochastic games.

Secondly, many of the more recent and sophisticated algorithms have a lot of tunable parameters. It was beyond the scope of this paper to adjust them. Indeed, parameters might be one of the reason's that `Q-learning` did so well: it had only three parameters which were all easy to set. This does not excuse poor performance from algorithms with many parameters. Indeed, the

onus of finding a good set of parameters belongs to the architect of the algorithm. However, it is possible that some algorithms may have hidden potential that can be release through rigorous tuning. This tuning will require further experimentation—hopefully MALT will be of assistance here—and there are some interesting questions to ask:

- Is one parameter setting good for many problems, or is it the case that one parameter setting will be great on one set of matches and poor on another?
- Which of, for instance, `meta`'s parameters are the most important?
- How much better would a rigorously-tuned setting be for `AWESOME` be than the default values presented in Conitzer and Sandholm [13]?
- Does `AWESOME`'s performance change radically when it selects the socially optimal Nash equilibrium as its special equilibrium? How about the 'Stackelberg' equilibrium?
- For gradient algorithms, is it better to ensure that mixed strategies are feasible through retraction or normalization?
- Is a set of parameters that are good for reward also good for regret?
- What is the best way to automate parameter tuning?

.

Thirdly, we presented two different tweaks to existing algorithms: `minimax-Q-IDR` and `GSA`. In many situations, these algorithms offered improvements over the original algorithm, and in many cases probabilistically dominated the originals. Other modifications and preprocessing steps can be added to existing algorithms, and an interesting direction of would be to see which ones tend to work the best. For instance, does IDR-preprocessing always improve performance? When does it hinder rather than help? Hopefully, experiments of this nature will also build intuition for how to rectify problematic situations and for building new algorithms from scratch.

Finally, managing a portfolio of existing algorithms also seems like a promising approach for designing new algorithms with good empirical properties. `AWESOME` and `meta` both can be seen as portfolio algorithms: they look for features of their opponent's behaviour and switch between different algorithms accordingly. A more general framework for building these portfolio algorithms—especially ones where the portfolio is not explicitly written into the algorithm—could be a way to reuse existing MAL methods. Such an algorithm would switch between the different algorithms in a portfolio as the situation demands, depending on the empirical characteristics of the managed algorithms. Again, this direction of research leads to a whole host of empirical questions. What features of the game and game play are the best to look for? Does adding an algorithm to a portfolio strictly improve performance?

We leave the reader with a host of unanswered questions. This is a sign of vibrancy of the field: there is still a vast amount of research that needs to be done in this area. We will end this thesis with a discussion of urban traffic that shows that this field of research is not merely an abstract study of learning algorithms. MAL research has practical value, especially as societal interactions become more numerous and more difficult to navigate. We hope that this thesis has piqued curiosity and left

some tools that are useful in addressing these many issues associated with learning and behaving in multiagent environments.

# Chapter 7

# Future Work: Extension to Traffic

In this section, we will discuss how MAL algorithms can be used to understand urban traffic. There are two major goals of such work. The first is to characterize what behaviour occurs in a traffic system—called traffic modeling or prediction—and the second is to optimize various policy tools—called traffic management. In this chapter we focus on the first problem, while noting that the second problem is the ultimate goal that we want to tackle once we have a satisfactory model of traffic to work with.

We are certainly not the first people to study traffic modeling. Civil engineers, for example, have worked extensively on the problem. There are multiple textbooks including May [31] and de Dios Ortzar and Willumsen [14] that are devoted solely to modeling issues. We take a slightly different approach to the problem than is traditionally used in civil engineering. Specifically, we are more interested the incentives behind routing decisions rather than specific physical details of the system.

Let us clearly articulate one of the problems of traffic modeling: route selection. We assume that we start with a set of trips between two points in a road system (these trips are generated in the earlier phases of modeling). This system can be modeled as a graph like Figure 7 where nodes represent intersections and directed edges represent lanes. This is allowed to be a multi-graph where there are several distinct lanes between two intersections. What kind of behaviour will intelligent drivers engage in? What system properties can we predict from their interaction?

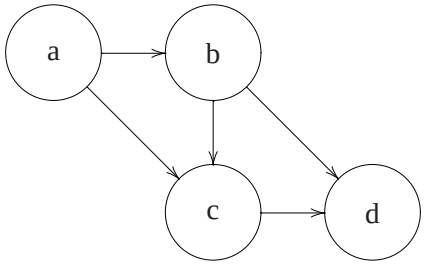


Figure 7.1: A sample road graph with four intersections.

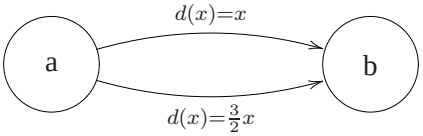


Figure 7.2: A sample road graph with two intersections that has no Wardrop equilibrium for a system with two atomic drivers.

## 7.1 Wardrop Equilibrium

One of the suggested route selection methods from the civil engineering community is to assign a delay function to each edge and look for a Wardrop equilibrium [54]. The Wardrop equilibrium is similar to the Nash equilibrium, but assumes that there is an uncountably infinite number of agents navigating the networks. The users of the road system are modeled as infinitely divisible flows that travel from a source node to a sink node. The delay for the flow along a particular path is equal to the sum of delays on each edge.

The traffic flow is assumed to be selfish so at equilibrium all paths with any flow must have the same delay and any paths without flow must be worse—this is the Wardrop condition. If we assume not, then there is some small fraction of flow on a more expensive path that would be better off on a cheaper path and so the system was not in equilibrium. Traffic engineers use these conditions to estimate traffic flow.

This model assumes that commuters are infinitely divisible. This is not the case in practice, where each driver is a discrete and atomic object that cannot be subdivided. This is a substantial assumption and Wardrop's conditions do not work when drivers are atomic. Consider the system in Figure 7.1 with two players: there is no way to make the two paths equal in cost. Additionally, atomic drivers could be in a Nash equilibrium but not a Wardrop equilibrium. In Figure 7.1 there is a Nash equilibrium when one driver takes the top route and the second driver takes the bottom but notice that the route costs are $\frac{3}{2}$ and 1 respectively. This shows that the condition for a Wardrop equilibrium does not work in the natural atomic agent case.

## 7.2 Congestion Games

*Congestion games* [43], roughly, are a discretization of the previous continuous commuter model of traffic. Each atomic agent $i \in N$ picks a path through a graph. The delay for a user is the sum of the delay along each edge of the path. This is formally stated in:

$$d(L) = \sum_{l \in L} d_l(\#_l). \tag{7.1}$$

Here, $\#_l$ is the number of agents using lane $l$ and $d_l(\cdot)$ is the delay function. Notice that the delay for a lane only depends on the number of agents using it; there is no inter-lane dependency.

Congestion games assume that the traffic conditions in one intersection do not affect the conditions in neighbouring intersections; this is an unrealistic assumption. In real traffic networks, when one lane becomes congested, cars entering from neighbouring lanes are impeded. This propagates congestion to nearby lanes and causes delays to cascade throughout the road network. However, in congestion games the congestion in each lane is independent, and so we cannot accurately represent this phenomenon.

## 7.3 Our Game

The inability to model interlane dependencies is a serious issue associated with both atomic and non-atomic congestion games. To model the propagation of congestion more accurately, we suggest our own model of traffic where these inter-lane dependencies can be expressed. We represent the traffic network as an extensive form game where agents make a series of turning decisions based on whatever observations they can make about the state. This mapping of observations to turns is called a policy. We explicitly model the position of each agent. Because of this, drivers may be unable to enter a lane that is heavily congested and either need to bypass jammed lanes or wait out the congestion. This is exactly the kind of delay propagation that we are interested in, where a jam in one lane creates blockages in or strains on other near by lanes.

While this model has the necessary richness to express inter-lane delay effects, it is also much more complicated than one-shot congestion games. In particular, our system dynamics make it practically impossible to write a closed-form function that maps from a profile of strategies for the extensive form game to a profile of delays. As a consequence we need to simulate the road network model to determine the outcome of a game.

Our simulator models both space and time as discrete. We want a simulator that can efficiently simulate large sections of a city and continuous models of time, space and motion are usually complicated and expensive to simulate. However, the discretization needs to be done with care, or important details will be lost, fundamentally altering the problem. We will call a physical quantum a ‘cell’ and a temporal quantum a ‘tick’. Cells are the fundamental unit of progress: in each tick an unimpeded car should be able to move at least one cell. Also, a cell should be a sufficient description of a car’s position: a car cannot be half-way through a cell.

We define a cell to be large enough to hold one and only one car. Cells that are larger than cars abstract space more aggressively and cells that are smaller model space and movement with greater precision. While our model could easily be extended to use larger or smaller-grained discretization, vehicle-sized cells offer a good trade-off between more efficient super-vehicular cells and more precise sub-vehicular cells (this raises an interesting empirical question: how sensitive is the system to more or less coarse discretization of space?).

Each car has a velocity (in cells-per-tick or ‘CPT’s) bounded by some upper limit and the some acceleration function (in $\text{CPT}^2$). For now, we take acceleration to be constant for simplicity

but our model could trivially be extended to more realistic acceleration functions. We offer no formal advice for how these numbers should be set, but there should be a qualitative difference between CPTs for fast-moving cars and slow moving cars. Also, cars should not be able to move too far in one tick. Moving larger distances in each tick means that a car could potentially interfere with many other cars and the point of simulating is to make these conflicts simple to resolve and relatively infrequent. Setting these kinematic numbers and a coarsening of space tacitly sets the coarsening of time and we need not consider it as a special topic.

The velocity of a car is bounded by the speed limit of each road, some internal maximum velocity, and by the velocity of the cars in front of it. All agents are assumed to perfectly decelerate to avoid accidents. While accidents are an important part of real traffic jams we do not model them in this thesis. In general we assume agents to be flawless and lawful: they are perfect drivers that always follow laws. Beyond our assumption of flawlessness and lawfulness, the agents have complete freedom to choose routes and adapt to traffic conditions. At each intersection, agents are able to make a turning decision based on some local features of the state. However, this policy of mapping observations to turning actions is complicated and needs to be learned. We use MAL algorithms to do this.

Indeed, the only difference between our traffic game and the earlier repeated game experiments is simply the complexity of the stage game: rather than repeatedly playing a simple two-player one-shot game like *Prisoner's Dilemma* the algorithms are repeatedly playing a large extensive-form game with $N$ players. Because the game is large and the payoffs are unknown—we are simulating exactly because we do not have a closed-form expression for the utility functions—we cannot use many of the algorithms looked at in earlier sections. The only algorithms that are able to function in such an environment are the gradient algorithms and `Q-learning`. Of these, we will focus on `Q-learning` because the gradient algorithms are only designed for one-shot games, and would requires serious redesign to work well in extensive-form games. Additionally, in Section 5 we saw that `Q-learning` was a good algorithm for learning in multigent systems and was better than the other MAL algorithms for most metrics.

One of the problems for any learning algorithm is that even our simple traffic simulator has a vast number of states. The state of the system is the position and velocity information for every vehicle (ignoring traffic lights for the moment). If there are $C$ cells, $N$ vehicles, and a maximum velocity of $\overline{v}$, then there are $\frac{C!}{(C-N)!} \times \overline{v}^N$ possible states.

This can be done efficiently through value approximation [6]. As an example, for `Q-learning` to learn efficiently with so many states it estimates the value of $Q(s, a)$ for new states by generalizing from similar states that it has seen before. Essentially, `Q-learning` needs to use non-linear regression to get an approximate $\hat{Q}(s, a)$ based on past observations. Techniques like forests of regression trees and Gaussian processes (see, for example, Rasmussen and Williams [42]) might be useful for this regression, but they need to be fast to update relative to the cost of simulating each round: $N$ learning algorithms need to update their model each round. There are a lot of interesting questions to explore: what is a good set of features? Are some regression techniques better than other for this problem? How should we explore to perform well in this non-stationary environment? Designing RL algorithms with good empirical properties for traffic

modeling is an exciting topic, and we hope to work more on this problem in the future.

### 7.3.1 Other Models

Let us summarize our model in four main points:

- Space is discrete and each cell is the same size as a car,
- Cars may have different velocities, and can accelerate,
- Agents are lawful and flawless drivers,
- Drivers are adaptive and are able to learn from past runs.

This model of traffic is not the only way to simulate simple urban road systems but it is a flexible model that can be extended in a number of directions. However, there have been other models of traffic suggested. In the following discussion we summarize a few of them and and indicate how they differ from ours.

There are a number of commercial simulators for simulating traffic including VISSIM [1], Paramics [39], and CORSIM [16]. Most commercial simulators are continuous time and space: each vehicle is a physical object in $\Re^3$ with real-valued velocity and acceleration. However, this makes simulations expensive to run. Indeed, most commercial simulators are meant for simulating short stretches of highways or single intersections to ensure that proposed alterations can accommodate predicted use. These simulators typically assume that the driver's routes are drawn from some static distribution. For example, a traffic engineer might want to simulate $100$ drivers per minute heading down a highway from North to South while $10$ drivers per minute attempt to enter the highway via a new on-ramp to ensure that merging is smooth.

Wiering [56], Porche and Lafortune [37] and de Oliveira et al. [15] all suggest discrete simulators that are intended for simulating intersections for optimizating light times. These simulators are much more closely aligned with our own goal of exploring policy tools than the commercial simulators.

Wiering [56] uses vehicle-sized cells but does not model acceleration. Vehicles move one cell forward every tick iff that cell is unoccupied. This understates the difference between an unimpeded fast moving car and a slow car moving on a congested road. Agents either randomly pick one of the shortest $\ell_1$-distance paths or co-learn: they pick the shortest $\ell_1$-distance path with the lowest estimated waiting time. This estimated waiting time is not a subjective estimate learned by the agent. It is an estimate maintained by the system that each car has access to.

This type of learning is unconvincing. There is no particular reason why any shortest $\ell_1$-distance paths will be the best path. For example, one might want to skirt around the downtown core of a city, even if it is directly between the start and end locations. Also, this model assumes that all driving agents obediently follow the advice of some central congestion-tracking service. This ignores the question of whether it is always in the best interest of the agent to follow this service's advice.

de Oliveira et al. [15] use a model very similar to our own: cells are vehicle-sized and cars can accelerate. Agents are assumed to be lawful but not flawless: indeed the agents are overly

enthusiastic about breaking and with probability $p$ slow down by one CPT. Random shocks like this might be an important part of why congestion forms, but we will ignore it for this phase of the work. The drivers in this model are simple path following algorithms that are unintelligent and do not adapt.

Porche and Lafortune [37] use super-vehicular cells where one cell is an entire block. Cars move at the constant rate of one CPT and they incur a delay that is not physically modeled: even if a road is heavily congested all cars still move at one CPT. This bears some striking similarities to a congestion game. Like congestion games, the Porche and Lafortune simulator fails to model how delays can propagate through a network. As we argued earlier, this omits an important feature of the traffic problem. In particular, roads in this model can never 'jam' and affect neighbouring road segments. The drivers in this model are unintelligent and unadaptive. The vehicles' routes are simply picked from some distribution and executed.

We also note that the empirical experiments are small: Porche and Lafortune [37] conduct an experiment with a uniform $4 \times 4$ grid of intersections, de Oliveira et al. [15] conduct a smaller one with a $3 \times 3$ grid and Wiering [56] also conducts an experiment on a $3 \times 3$ grid. These experiments are too small to realistically represent genuine route choices, although this does not matter because the drivers are largely unadaptive. Clearly, there is a lot of room for improvement in the empirical simulation of road systems, especially with adaptive and indeed strategic agents.

### 7.3.2 Experimental Directions

We have not yet run any experiments on this model. However, we have indicated particularly promising areas for empirical work.

For the system:

- How frequently does the system converge to a stationary or relatively stationary state?
- Of these stationary states, are they in equilibrium?
- How sensitive is the system to random accidents or driver mistakes?

For the MAL algorithm:

- For reinforcement algorithms, what is a good regression technique and what is a good set of state features?
- What is a good way to explore without introducing too much noise into the system?
- Can gradient learning algorithms be extended to extensive-form games like traffic?
- What are good performance metrics for driving agents?
- What are the best learning algorithm for different situations?

All these questions are important and interrelated, and we look forward to eventually answering them.

# Appendix A

# Stratified Sampling

In this thesis we need to evaluate with a limited computational budget a large number of algorithms and a large number of game instances. However, we want to get the maximum information from the simulations that we did perform. How should we go about running these experiments?

For all of our experiments, we are concerned with the expected performance of a match, denoted by $f(\mu, \zeta)$. Here, $f$ is some metric function, $\mu \sim M$ is a match, and $\zeta \sim Z$ is a random seed that completely determined any non-deterministic behaviour in both algorithms. The game instance/seed pairing uniquely define a run.

When designing our experiment, we must choose whether or not to stratifying runs based on the match. For instance, if we have enough computational time to run $100$ simulations, we can either sample $100$ matches and perform a single run on each, or we can sample only $10$ matches and run $10$ runs for each. Stratification clearly leads to a more detailed understanding of the role that randomization plays in each match and is critical information for algorithm design. However, for two kinds of common summary statistics—means and quantiles—one should avoid any stratification.

**Lemma A.0.1** *If we are trying to obtain an estimate of $\mathbb{E}\left[f_{(M,Z)}\right]$ and we have a limited budget of samples, it reduces variance to sample from $M$ and $Z$ independently rather than to stratify based on $M$.*

**Proof** Consider two schemes of sampling from $M$ and $Z$, as seen in Table A.1. In the first scheme, $M$ and $Z$ are sampled separately each time. In the second scheme $k$ samples are taken from $M$ and for each sample of $M$, $Z$ is sampled $s_i$ times.

| | |
|---|---|
| **Independent** | $\{(M_1, Z_1), \ldots, (M_n, Z_n)\}$ |
| **Stratified** | $\{(M_1, Z_{1,1}), \ldots, (M_1, Z_{1,s_1}), \ldots, (M_k, Z_{k,s_k})\}$ |

Table A.1: Two schemes for sampling.

In both cases, the sample mean is used as the point estimator for the population. Since $G$ and $Z$ are sampled independently, both schemes yield unbiased estimators.

However, the first scheme yields lower variance as can be seen in Equations A.1-A.3. Equation A.2 follows from the fact that completely independent random variables have no covariance (Equation A.5) and so if two samples share the same strata (the same sample $\mu \sim M$) then they have weakly higher covariance (Equation A.4).

$$Var\left[\sum_i f(M_i, Z_i)\right] = \sum_{i,j} Cov\left[f(M_i, Z_i), f(M_j, Z_j)\right] \tag{A.1}$$

$$\leq \sum_{i,j,k,l} Cov\left[f(M_i, Z_{i,j}), f(M_k, Z_{k,l})\right] \tag{A.2}$$

$$= Var\left[\sum_{i,j} f(M_i, Z_{i,j})\right] \tag{A.3}$$

$$Cov\left[f(M_k, Z_{k,l}), f(M_k, Z_{k,m})\right] \geq Cov\left[f(M_i, Z_i), f(M_j, Z_j)\right] \tag{A.4}$$

$$Cov\left[f(M_i, Z_i), f(M_j, Z_j)\right] = Cov\left[f(M_k, Z_{k,l}), f(M_m, Z_{m,n})\right] \tag{A.5}$$

Additionally, stratifying increases the variance of quantile point estimation. This result can be found in Heidelberger and Lewis [22], but it is provided without proof.

# Bibliography


[1] PTV AG. Vissim 5.00, 2008. URL `www.ptvag.com`.

[2] Stéphane Airiau, Sabyasachi Saha, and Sandip Sen. Evolutionary tournament-based comparison of learning and non-learning algorithms for iterated games. *Journal of Artificial Societies and Social Simulation*, 10(3):7, 2007. ISSN 1460-7425.

[3] R. Axelrod. The evolution of strategies in the iterated prisoner's dilemma. In L. Davis, editor, *Genetic Algorithms and Simulated Annealing*, pages 32–41. Morgan Kaufman, Los Altos, CA, 1987.

[4] B. Banerjee and J. Peng. Performance bounded reinforcement learning in strategic interactions. In *AAAI 11*, 2004.

[5] B. Banerjee and J. Peng. RV: a unifying approach to performance and convergence in online multiagent learning. In *AAMAS '06*, pages 798–800, 2006.

[6] A.G. Barto, S.J. Bradtke, and S.P. Singh. Learning to act using real-time dynamic programming. Technical Report UM-CS-1993-002, University of Massachusetts, Amherst, 1993.

[7] M. Bowling. Convergence and no-regret in multiagent learning. In *NIPS 17*, 2004.

[8] M. Bowling. Convergence and no-regret in multiagent learning. Technical Report TR04-11, University of Alberta, 2004.

[9] M. Bowling and M Veloso. Rational and convergent learning in stochastic games. In *IJCAI 17*, August 4 – 10 2001.

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

[11] G. Brown. Iterative solution of games by ficticious play. In *Activity Analysis of Production and Allocation*, New York, 1951.

[12] C. Claus and C. Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *AAAI 4*, pages 746 – 752, July 28 1997.

[13] V. Conitzer and T. Sandholm. AWESOME: A general multiagent learning algorithm that converges in self-play and learns a best response against stationary opponents. *Machine Learning*, 67(1):23–43, 2007.

[14] Juan de Dios Ortzar and Luis G. Willumsen. *Modelling Transport*. Wiley, 2001.

[15] Denise de Oliveira, Ana L. C. Bazzan, Bruno Castro da Silva, Eduardo W. Basso, and Lus Nunes. Reinforcement learning based control of traffic lights in non-stationary environments. In *EUMAS*, volume 223 of *CEUR Workshop Proceedings*, 2006.

[16] Center for Microcomputers in Transportation. TSIS-CORSIM 6.0, 2008. URL `mctrans.ce.ufl.edu`.

[17] Drew Fudenberg and David M. Kreps. Learning mixed equilibria. *Games and Economic Behavior*, 5(3):320–367, July 1993.

[18] S. Govindan and R. Wilson. A global newton method to compute Nash equilibria. *Journal of Economic Theory*, 110(1):65 – 86, 2003.

[19] A. Greenwald and K. Hall. Correlated-Q learning. In *ICML 20*, 2003.

[20] S. Hart and Y. Mansour. How long to equilibrium? The communication complexity of uncoupled equilibrium procedures . In *ACM Symposium on Theory of Computing*, volume 39, pages 345 – 353, 2007.

[21] Sergiu Hart and Andreu Mas-Colell. Stochastic uncoupled dynamics and nash equilibrium: extended abstract. In *TARK '05: Proceedings of the 10th conference on Theoretical aspects of rationality and knowledge*, pages 52–61, 2005.

[22] P. Heidelberger and PAW Lewis. Quantile estimation in dependent sequences. *Operations Research*, 32(1):185–209, 1984.

[23] J. Hu and M. Wellman. Multiagent reinforcement learning: theoretical framework and an algorithm. In *ICML 15*, pages 242 – 250, 1998.

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

[25] Ehud Kalai and Ehud Lehrer. Rational learning leads to Nash equilibrium. *Econometrica*, 61 (5):1019–1045, 1993.

[26] Ehud Kalai and Ehud Lehrer. Subjective games and equilibria. *Games and Economic Behavior*, 8(1):123–163, 1995.

[27] C. Lemke and J. Howson. Equilibrium points of bimatrix games. In *Journal of the Society for Industrial and Applied Mathematics*, volume 12, page 413423, 1964.

[28] A. Lipson. An empirical evaluation of multiagent learning algorithms. Master's thesis, University of British Columbia, Vancouver, Canada, 2005.

[29] M. Littman. Friend-or-foe Q-learning in general-sum games. In *ICML 18*, pages 322 – 328, June 28 – July 1 2001.

[30] M. Littman. Markov games as a framework for multi-agent reinforcement learning. In *ICML 11*, pages 157 – 163, 1994.

[31] A.D. May. *Traffic Flow Fundamentals*. Prentice-Hall, Upper Saddle River, N.J., 1990.

[32] R.D. McKelvey, A.M. McLennan, and T.L. Turocy. Gambit: software tools for game theory. Version 0.97.0.6. `http://econweb.tamu.edu/gambit`, 2004.

[33] D. Monderer and A. Sela. A $2\times$ 2 game without the fictitious play property. *Games and Economic Behavior*, 14:144–148, 1996.

[34] D. Monderer and L.S. Shapley. Fictitious play property for games with identical interests. *Journal of Economic Theory*, 68(1):258–265, 1996.

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

[36] E. Nudelman, J. Wortman, K. Leyton-Brown, and Y. Shoham. Run the GAMUT: a comprehensive approach to evaluating game-theoretic algorithms. In *AAMAS 3*, July 19 – 14 2004.

[37] I. Porche and S. Lafortune. Adaptive look-ahead optimization of traffic signals. Technical report, University of Michigan, 1997.

[38] R. Powers and Y. Shoham. New criteria and a new algorithm for learning in multi-agent systems. In *NIPS*, volume 17, pages 1089–1096, 2005.

[39] *Paramics Modeller*. Quadstone Paramics Ltd., 21. URL `www.paramics-online.com`.

[40] R Development Core Team. *R: a language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria, 2006. URL `http://www.R-project.org`.

[41] A. Rapoport, M. Guyer, and D. Gordon. *The 2x2 Game*. Univeristy of Michigan Press, 1976.

[42] Carl E. Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, December 2005.

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

[44] T. Sandholm. Perspectives on multiagent learning. *Artificial Intelligence*, 171(7):382–391, 2007.

[45] Yoav Shoham and Kevin Leyton-Brown. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, New York, 2008.

[46] Yoav Shoham, Rob Powers, and Trond Grenager. If multi-agent learning is the answer, what is the question? *Artificial Intelligence*, 171(7):365–377, 2007.

[47] S. Singh, M. Kearns, and Y. Mansour. Nash convergence of gradient dynamics in general-sum games. In *UAI 16*, 2000.

[48] J. C. Spall. *Introduction to Stochastic Search and Optimization: Estimation, Simulation and Control*. John Wiley & Sons, Hoboken, New Jersey, 2003.

[49] R.S. Sutton and A.G. Barto. *Reinforcement Learning, An Introduction*. The MIT Press, Cambridge, Massachusetts, 1999.

[50] G. Tesauro. Extending Q-learning to general adaptive multi-agent systems. In *NIPS 16*, 2004.

[51] W.K. Viscusi. The value of life: estimates with risks by occupation and industry. *Economic Inquiry*, 42(1):29–48, 2004.

[52] J. von Neumann and O. Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, 1944.

[53] T. Vu, R. Powers, and Y. Shoham. Learning against multiple opponents. In *AAMAS*, 2005.

[54] J. Wardrop. Some theoretical aspects of road traffic research. *Proceedings of the Institution of Civil Engineers, Part II*, 1(36):352–362, 1952.

[55] C.H. Watkins and P. Dayan. Q-learning: technical note. *Machine Learning*, 8:279–292, 1992.

[56] Marco Wiering. Multi-agent reinforcement learning for traffic light control. In *Proc. 17th International Conf. on Machine Learning*, pages 1151–1158. Morgan Kaufmann, San Francisco, CA, 2000.

[57] M. Zinkevich. Online convex programming and generalized infinitesimal gradient ascent. In *ICML'03*, 2003.