Empirically Evaluating Multiagent Reinforcement Learning Algorithms

Asher Lipson and Kevin Leyton-Brown
Department of Computer Science, University of British Columbia, Vancouver, Canada
{*alipson,kevinlb*}*@cs.ubc.ca*

Abstract. This article makes two contributions. First, we present a platform for running and analyzing multiagent reinforcement learning experiments. Second, to demonstrate this platform we undertook and evaluated an empirical test of multiagent reinforcement learning algorithms from the literature, which to our knowledge is the largest such test ever conducted. We summarize some conclusions from our experiments, comparing algorithms on a variety of metrics including reward, regret, convergence to a Nash equilibrium and behavior in self play[1].

Keywords: Game theory, multiagent systems, reinforcement learning

1. Introduction

Recently, there has been much interest in the design and analysis of algorithms for game theoretic settings, for example by Singh et al. (2000), Bowling and Veloso (2001a), Tesauro (2003), Bowling (2004) and Powers and Shoham (2004). Such work has typically introduced new algorithms, comparing them against one or two existing algorithms on a small set of well-known repeated games. Algorithm performance is then evaluated using one or two metrics. This focus has led to a wealth of different algorithms for multiagent learning in repeated games, but a relative lack of general understanding of these algorithms' strengths and weaknesses.

The research described in this article makes two contributions. First, we describe the design and implementation of a platform for running large experiments on multiagent reinforcement learning algorithms (Section 2). As we argue in this article, a standardized platform offers several advantages over one-off solutions for running experiments (though clearly the creation of a platform requires a large up-front expenditure of effort). Second, we present an analysis of an empirical test that was conducted using our platform (Section 3). We make a series of claims and argue for them on the basis of our experimental evidence. For example, we show how algorithm performance differs depending on the game and opponent. Furthermore, we identify algorithms which perform well according to different measures of performance

[1] We would like to gratefully acknowledge a variety of contributions to this work by Nando De Freitas, and helpful comments from three anonymous reviewers.

such as the amount of reward attained by the agent and whether the agent converges to playing a Nash equilibrium strategy. This article describes work which is presented in more detail in Lipson (2005).

1.1. SETTING

In this article we consider two-player repeated games—an environment in which a single normal-form game (or "stage game") is played repeatedly by a pair of self-interested agents[1]. Our environment is thus game theoretic; readers desiring an introduction to the topic should consult e.g., (Fudenberg and Tirole, 1991; Osborne and Rubinstein, 1994; Fudenberg and Levine, 1999). A game can be understood as having a set of game-theoretic properties including the set of Nash equilibria, the set of strategies that survive iterated removal of dominated strategies, Pareto-optimal outcomes and safety levels. In a given stage game agents are aware of the past history (actions that were played in previous stage games) but are not aware of which strategy their opponent will play in the current stage game or which algorithm the opponent is using to determine this strategy. This assumption is based on the principle that an agent's strategy should be based on observing the opponent's play, rather than on knowledge of the opponent's internal state.

1.2. RELATED WORK: MULTIAGENT REINFORCEMENT LEARNING ALGORITHMS

Fundamentally, a main goal of our work is to deepen our understanding of the empirical behavior of multiagent reinforcement learning algorithms. In this section we give high-level descriptions of the algorithms we studied along with pointers to papers which give more information about each algorithm.

1.2.1. *Fictitious Play*

Fictitious play is the earliest example of a learning algorithm for repeated games (Brown, 1951). The agent observes the actions of her opponent and estimates the opponent's mixed strategy by counting the number of times the opponent has played each of her actions so far. The agent then chooses to play the action (i.e., the pure strategy) that maximizes her expected payoff given her estimate of the opponent's strategy. The updates and strategy are as follows:

[1] One might also be interested in broader classes of stochastic games, or games with incomplete information. We did not pursue either direction, since in neither setting does there exist a widely-used distribution of benchmark games upon which to test. Considering such games would be a worthwhile topic for future work.

– At time $t+1$, the player selects a best response pure strategy $overlinea_{t+1}$ to the estimate of the opponent's strategy $\hat{C}_t$:

$$\overline{a}_{t+1} = br_i(\hat{C}_t) = \arg\max_{a_i} E[u_i(\hat{C}_t, a_i)]$$

– The player then updates her estimate of the opponent's strategy based on the opponent's action o_{t+1} :

$$C_{t+1} = C_t + e_{o_{t+1}};\ \hat{C}_{t+1} = \text{normalize}(C_{t+1}).$$

1.2.2. *Q-Learning*

One of the most popular reinforcement learning approaches in single agent environments is Q-learning (Watkins and Dayan, 1992), in which the agent does not know what rewards are assigned to actions. Q-learning repeatedly takes actions and obtains rewards, thereby learning a policy. Specifically, at each iteration/time step the agent does an update based on the state s she is in, the action a taken and the obtained reward $r(s,a)$. Let $Q(s,a)$ be the discounted value of taking action a in state s and s' is the new state that is the agent finds herself in after taking action a. The update at each step is then:

$$Q(s,a) \ = \ (1-\alpha)Q(s,a) + \alpha[r(s,a) + \gamma \max_{a'} Q(s',a')].$$

The value of state s is $V(s) = \max_a Q(s,a)$ and the policy is $\pi(s) = \arg\max_a Q(s,a)$. γ is the discount factor and α is the learning rate, which is traditionally decayed over time. It can be shown that if α is decayed and each action is played in each state an infinite number of times, then $Q(s,a)$ is guaranteed to converge to the optimal $Q^*(s,a)$ (Kaelbling et al., 1996).

Q-learning can be used as a learning algorithm in multiagent environments. However, note that doing so ignores the presence of an opponent, and so violates the stationarity assumption necessary to the theoretical guarantees upon which the algorithm is built. Nevertheless, this is not an altogether unreasonable idea—for example, if Q-learning converges in self play, it is easy to show that it will have converged to a Nash equilibrium.

1.2.3. *Minimax-Q*

There have also been efforts to extend the ideas behind Q-learning to multiagent environments by taking the opponent's actions into account and storing discounted Q-values for the joint actions of the agent and its opponent. Minimax-Q (Littman, 1994) is one such approach. Instead of computing a Q-value $Q(s,a)$ over the state and action, the algorithm computes a Q-value $Q(s,a,o)$ over the state, action and opponent's action, with o representing the opponent's action at a given iteration. The $\max_a$ operation in Q-learning

is replaced by a $\max_a \min_o$ operation in minimax-Q, which means that the agent plays a safety level (or "max-min") strategy.

Minimax-Q-IDR is an algorithm which iteratively removes dominated strategies from the game matrix and then applies the minimax-Q algorithm to the reduced game. We are not aware of any other published work that discusses this algorithm, though the idea is straightforward.[2]

1.2.4. *Win-or-Learn-Fast*

The Infinitesimal Gradient Ascent (IGA) algorithm (Singh et al., 2000) is a policy search reinforcement learning algorithm for single-agent settings. Bowling and Veloso (2001a) used a so-called Win-or-Learn-Fast (WoLF) heuristic with IGA to produce the WoLF-IGA algorithm. The idea is to use a variable step size that changes according to how well an agent is doing. If an agent is achieving high payoffs (relative to a Nash equilibrium) she uses a small step size, thereby slowing down her learning rate. If the agent is doing badly then she uses a larger step size, allowing her to adapt faster.

An improved version of WoLF-IGA does not compare performance to a Nash equilibrium, but rather keeps track of the agent's regret with respect to the counterfactual possibility of having played a stationary pure strategy (Bowling, 2004). This version, GIGA-WoLF, offers the theoretical guarantee that long-run regret will not be positive. This algorithm does not directly utilize a variable step size, but rather the step size is affected by the magnitude of the gradient.

1.2.5. *Global Stochastic Approximation*

Global Stochastic Approximation (GSA) (Spall, 2003) draws on ideas from annealing and stochastic approximation. It is very similar to IGA, but adds a random perturbation term to the stochastic approximation update equation. This adds some "jumps" to the update equation in order to explore the space and avoid local maxima. To our knowledge, the GSA algorithm has not previously been applied to a repeated game setting.

1.2.6. *Alternative Algorithms*

We have not implemented all algorithms that have been used in repeated game research. This includes the work of Yoav Shoham (Shoham et al., 2004; Shoham et al., 2006), Rob Powers (Powers and Shoham, 2004; Powers and Shoham, 2005) and Thuc Vu (Vu et al., 2005) who have presented algorithms for learning against specific classes of opponents, in addition to critically looking at the direction of research in multiagent learning.

Though we have presented a number of different metrics, we have not covered all of the possible equilibrium types. Algorithms such as Nash-Q

[2] Note that while minimax-Q generalizes straightforwardly to stochastic games, minimax-Q-IDR does not have this property.

(Hu and Wellman, 1998) which focus on learning strategies that converge to playing Nash equilibrium[3] and Correlated-Q Learning (Hart and Mas-Colell, 2000; Greenwald and Hall, 2003) which use correlated equilibrium, as opposed to Nash equilibrium. We have presented research based on the concept of an agent's regret, but these are not the only algorithms that utilize this concept (Freund and Schapire, 1996; Foster and Vohra, 1997). These algorithms should be looked at in the future.

1.3. Related Work: Experimental Methods

A wide variety of choices about experimental setup must be made before an experiment can be run. Unfortunately, papers in the literature vary widely in the way they make these choices. Furthermore, some papers do not even discuss the parameters used in their implementations of the algorithms, which can make it difficult to reproduce experiments. Table I shows how six recent research papers ran experiments, illustrating the fact that the researchers used very different experimental setups.

Table I. *Testing methodologies in different research papers.*

Paper	**# iterations**	**# games**	**# runs/trials**	**Settling in/ recording period**
Littman (1994) - minimax-Q	1.1M	1	3	1M/100k
Claus and Boutilier (1997) - Joint Action Learners	50 or 2500	3	100	0/50 or 2500
Bowling (2004) - GIGA-WoLF	1M	1	unknown	0/1M
Nudelman et al. (2004) - GAMUT	100k	13 GAMUT generators	100 instances/game 10x per instance	90k/10k
Powers and Shoham (2004) - MetaStrategy	200k	21 or 35 GAMUT generators	unknown	180k/20k
Tesauro (2003) - Hyper-Q	1.6M	1	unknown	0/1.6M

Overall, most of the tests performed in these papers (and in other papers from the literature) considered quite small numbers of algorithms. Tesauro (2003) and Bowling (2004) reported tests of their new algorithms against two and one algorithms respectively. Nudelman et al. (2004) used three agents but

[3] Bowling (2000) discusses some of the convergence results of Nash-Q

many games;[4] however, the purpose of that paper was primarily the demonstration of the GAMUT software. The experiments by Littman (1994) were performed with four algorithms, two minimax-Q and two Q-learning algorithms. Claus and Boutilier (1997) utilized two algorithms. Powers and Shoham (2004) conducted one of the largest experiments but only used reward as a metric.

Tests have also tended to consider small numbers of different games. For example, many papers (Bowling and Veloso (2001b), Tesauro (2003), Bowling (2004)) tested on a single repeated game. Some used a single new game to demonstrate properties of an algorithm, e.g., Littman (1994), Claus and Boutilier (1997), Littman (2001) and Bowling and Veloso (2002). To some extent, this reflects the difficulty of creating a large number of different games for use in tests. Recently, some papers including Nudelman et al. (2004) and Powers and Shoham (2004) have tested on larger sets of games, using the GAMUT software (Nudelman et al., 2004) to produce test data.

Experiments also differed substantially in the number of iterations considered (i.e., the number of repetitions of the normal form game). First, note that the iterations in a repeated game are often split into "settling in" and "recording" phases, allowing the algorithms time to determine a strategy before results are recorded. This is done to reduce the sensitivity of experimental results to different algorithms' learning rates. Littman (1994) used a simple two-player soccer game that took a variable number of iterations to play once; nevertheless, tests ran for a fixed number of iterations rather than a fixed number of games. The experiments in Claus and Boutilier (1997) were run with different numbers of iterations (though far fewer than were used by any other researchers). In Powers and Shoham (2004), different results were shown in the paper for a 21 game distribution and an "all" game distribution from GAMUT, which currently includes 35 generators.

An alternative style for a tournament would be to run one that is based around evolutionary ideas, as in Axelrod (1987). Though such an approach would try and play the "genetically best" agents from a population against one another, the initial population has strategies defined by specific actions for each stage games. It is not clear how such an approach could utilize learning algorithms as described in this paper, where the learning strategies do not specify what actions to take in each stage game prior to the stage game being played.

[4] The term *generator* in Table I refers to an algorithm which produces random normal-form games satisfying certain constraints (e.g., n-player prisoner's dilemma games). An instance is a particular normal form game produced by such a generator.

2. A Platform for Multiagent Reinforcement Learning

One common approach to running experiments is to write one-off code tailored to the particular experiment. Often the code is not used again. While this approach is appropriate for quick and dirty experiments meant to test specific features of a new algorithm, it can make replication of the experiment difficult, and can discourage the sort of exploratory experimentation that is needed to gain understanding of the complex interactions between multiagent reinforcement learning algorithms. This section describes an open and reusable platform that we have implemented for conducting experiments with multiagent reinforcement learning algorithms in repeated games.

The platform software is available for download at `http://www.cs.ubc.ca/~kevinlb/malt`. This website also gives the configuration files used to run all the experiments described in this article.

2.1. The Platform Architecture

Our multiagent learning platform was written in object oriented Matlab. We chose Matlab for its rapid prototyping ability, excellent collection of numerical algorithms and built-in data visualization routines. The user does not need to interact with Matlab's command line, as all parameter setting, metric specification and visualization routines are conducted through GUIs.

Experiments are run in a tournament fashion, with each algorithm running against all other algorithms, including in self play. Each pairing of algorithms is run twice, with each agent taking turns to play as the row and the column player against the other agent. While in zero-sum and symmetric games it makes no strategic difference to an agent whether she plays as the row or column player, in general games the role each agent plays is important. Allowing each algorithm to play both as the row and the column player prevents bias. Each pairing of algorithms plays on a set of instances of game generators, with each pairing playing on the same instances.

If there are n agents, g games and i instances of each game, then the total number of runs is $2 \times \left[\sum_{j=1}^{n} j\right] \times g \times i$. This shows the degree of parallelism available, as each of these runs can occur in parallel. Individual runs cannot be split up as every iteration in the run relies on the previous iteration.

Each run can be identified by the following characteristics: the pair of algorithms involved, which algorithm is the row player and which is the column player, which game generator is being used and which instance of that generator is being played.

2.1.1. *Algorithms/Agents*
Figure 1 gives the class structure for the algorithms described in Section 1.2 and implemented in the platform.

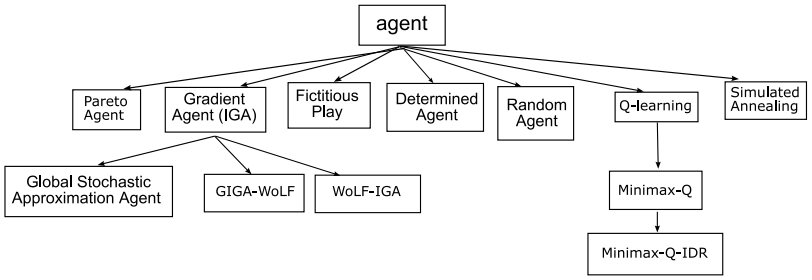

Figure 1. Class hierarchy for the agent/algorithm classes. The root node of agent *contains the basic information relevant to all agents, including the algorithm name, number of actions and strategy.*

Fictitious play is an implementation of the algorithm described in Section 1.2.1. *Pareto agent* and *determined agent* are algorithms that play the strategy corresponding to the most beneficial Pareto outcome or Nash equilibrium respectively. The *gradient agent* is an implementation of the Infinitesimal Gradient Ascent (IGA) algorithm from Singh et al. (2000). *GIGA-WoLF* (Bowling, 2004) and *WoLF-IGA* (Bowling and Veloso, 2001a) are the (gradient-based) algorithms described in Section 1.2.4. The *global stochastic approximation agent* is an implementation of the stochastic approximation algorithm of Spall (2003), which was discussed in Section 1.2.5. The *random agent* is an algorithm that selects among all actions with uniform probability. The *Q-learning* algorithm (Watkins and Dayan, 1992) applies a single agent algorithm to the multiagent setting as discussed in Section 1.2.2. The *minimax-Q* algorithm differs slightly from the implementation in Littman (1994) by randomizing over actions that return the safety level of the game. The *minimax-Q-IDR* plays a minmax strategy on the reduced game after iteratively removing dominated strategies as discussed in Section 1.2.3.

2.1.2. *Games*
The testing platform obtains all of its normal form games from the game generator GAMUT (Nudelman et al., 2004). A user inputs specific parameter values for the generators in GAMUT and these values are checked to ensure they are valid. If invalid, the platform returns the GAMUT error message.

Each game has its payoffs normalized to the range $[-1,\ 1]$. Normalization is done for two reasons. Firstly, it allows easier comparisons to be made across games. All games have different default payoff bounds and it is difficult to obtain average statistics across such varied bounds. Second,

normalization standardizes the performance of the gradient-based algorithms by ensuring that gradients remain well-behaved.

2.1.3. *Performance Metrics*

A key difference between the single- and multiagent reinforcement learning settings is that there is no single way of evaluating the performance of a multiagent reinforcement learning algorithm. That is, in a multiagent setting there is no well-defined notion of optimality. For example, while it is desirable for an agent to play a best response to its opponent's strategy, the fact that the opponent may also adapt raises the possibility that an agent could be better off forgoing immediate rewards in order to condition her opponent to behave in a desirable way.

The most basic metric, *reward*, is the sum of the payoffs that the agent received from the sequence of stage games. However, using reward as an isolated metric is difficult. (For example, is it good or bad for an agent to obtain a reward of 0.5?).

Regret considers how much better off the agent would have been if she had played the best among some family of candidate strategies, making the assumption that the opponent's sequence of actions would not have been affected. Here we consider regret with respect to stationary pure strategies, as did Bowling (2004) for the GIGA-WoLF algorithm. This is a fairly weak notion of regret, because it is stated with respect to a very restricted class of strategies. Positive regret means the agent would have been better off playing some strategy from the candidate set (e.g., some pure strategy). Negative regret means the agent's strategy outperformed every strategy from the candidate set. Equation (1) gives the formula for calculating *total regret*. a_i is a pure strategy for agent i and $u_i(\sigma_{-i}, a_i)$ is the payoff received by agent i from playing the pure strategy against the opponent's strategy σ_{-i}.

$$\mathcal{R} = \max_{a_i \in A_i} \sum_{t=1}^{T} (u_i^{(t)}(\sigma_{-i}, a_i) - u_i^{(t)}(\sigma_{-i}, \sigma_i)) \tag{1}$$

Incentive to deviate is based on the ideas of Walsh et al. (2003). This value is the difference between the reward that an agent would have received if she had played a best response at every stage game and the reward she actually received. This can be seen as a more extreme version of regret, because it is stated with respect to any possible strategy that an agent could play. Equation (2) gives the formula for calculating agent i's best response strategy to an opponent's mixed strategy σ_{-i}. $\mathbb{E}[u_i(\sigma_{-i}, \sigma_i)]$ is the expected payoff to agent i, with σ_i being agent i's strategy and σ_{-i} being the opponent's strategy. Equation (3) gives the formula for calculating the *incentive to deviate* from a mixed strategy σ at time t; in this equation, $br_i(a_{-i})$ is agent i's best response to her opponent's action a_{-i}.

$$br_i(\sigma_{-i}) = \arg\max_{\sigma_i} \mathbb{E}[u_i(\sigma_{-i}, \sigma_i)] \tag{2}$$

$$ID_{\sigma_t} = \max(0, u_i(a_{-i}, br(a_{-i})) - u_i(a_{-i}, a_i)) \tag{3}$$

The *number of wins* metric is calculated each stage game. If agent *A* receives a higher reward than agent *B*, then *A* is said to have won that stage game. This metric has the drawback that it implicitly assumes that the agent cares about receiving a greater reward than its opponent. This ought not to be the case as by definition an agent should care only about maximizing its own reward, regardless of the reward that its opponent receives. However, this metric is good for indicating pairwise dominance amongst algorithms.

Walsh et al. (2003) uses the ℓ_2 norm as a measure of convergence to a Nash equilibrium. The result is the minimum of the distances between the agent's current strategy and any equilibrium from the set of equilibria E. The formula for this is given in Equation (4). One problem with this metric is that it measures how close an individual agent's strategy is to a Nash equilibrium strategy, without considering whether the opponent is playing close to the same Nash equilibrium. We thus introduce a second measure of convergence. This uses the ℓ_1 norm and takes into account how far both players are from all equilibria. When the value of this metric is 0, we have convergence to an exact Nash equilibrium. The formula for the joint ℓ_1 norm is given in Equation (5). e_i denotes player i's strategy in the Nash equilibrium e, σ_i is player i's mixed strategy, N_a is the total number of actions for each agent.

$$l_{2_i}(t) \;=\; \min_{e \in E} \sum_{a=1}^{N_a} \sqrt{(\sigma_i(a) - e_i(a))^2} \tag{4}$$

$$l_1(t) \;=\; \min_{e \in E} \left(\sum_{a=1}^{N_a} |\sigma_1(a) - e_1(a)| \;+\; \sum_{a=1}^{N_a} |\sigma_2(a) - e_2(a)| \right) \tag{5}$$

2.2. Setting Up and Running an Experiment

The pipeline for running an experiment is split into three phases. The first is the set up phase: selecting the agents, games, number of runs or instances, number of iterations and the metrics to be recorded. The second phase is the actual running of the experiment. This phase can be done on a single machine in a batch process, or in parallel on a cluster. The final phase is to visualize the metrics and to analyze the resulting data. The entire pipeline is shown in Figure 2.

The GUI for setting up an experiment, phase 1 of the pipeline, is shown in Figure 3. The user selects the agents, games, metrics and parameters for the

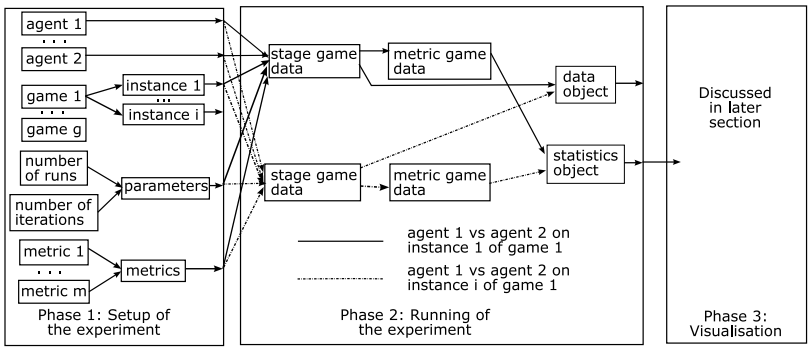

Figure 2. The pipeline for running experiments on the platform. The illustration is shown for a single combination of two agents and a single game generator.

experiment. A configuration file describing this setup can be saved prior to running the experiment. This file contains all of the information necessary to rerun the experiment. This file can be made available for download, allowing researchers to reproduce one another's experiments.

Once an experiment has been run, we need to visualize the results. Due to the volume of data collected in a typical experiment, we would not want to see a statistic for each run, but instead want to be able to visualize statistics over multiple runs. Although visualization could be performed by the user at the command line, it can be cumbersome to specify to the system which data to vary in a visualization, which data to aggregate, and which data to omit. For this reason we built a GUI-driven visualization system. This system is described in an appendix.

3. Our Empirical Evaluation of Multiagent Reinforcement Learning Algorithms

3.1. Experimental Setup

The experiment was set up using a subset of the available algorithms, metrics and games. In total, six algorithms, seven metrics and thirteen game generators were chosen. The experiment was conducted in a tournament fashion, with each algorithm playing each other algorithm. The algorithms and their parameters are listed in Table II. The parameters for each algorithm were either set to values from the paper in which they were introduced or to decay below a certain value after 100 000 iterations. Each agent's beliefs about its opponent's strategy was initialized to a uniform distribution over the opponent's actions. The initial belief update rate was set to 0.9 for all agents that used a discounted stochastic approximation technique to estimate an oppo-

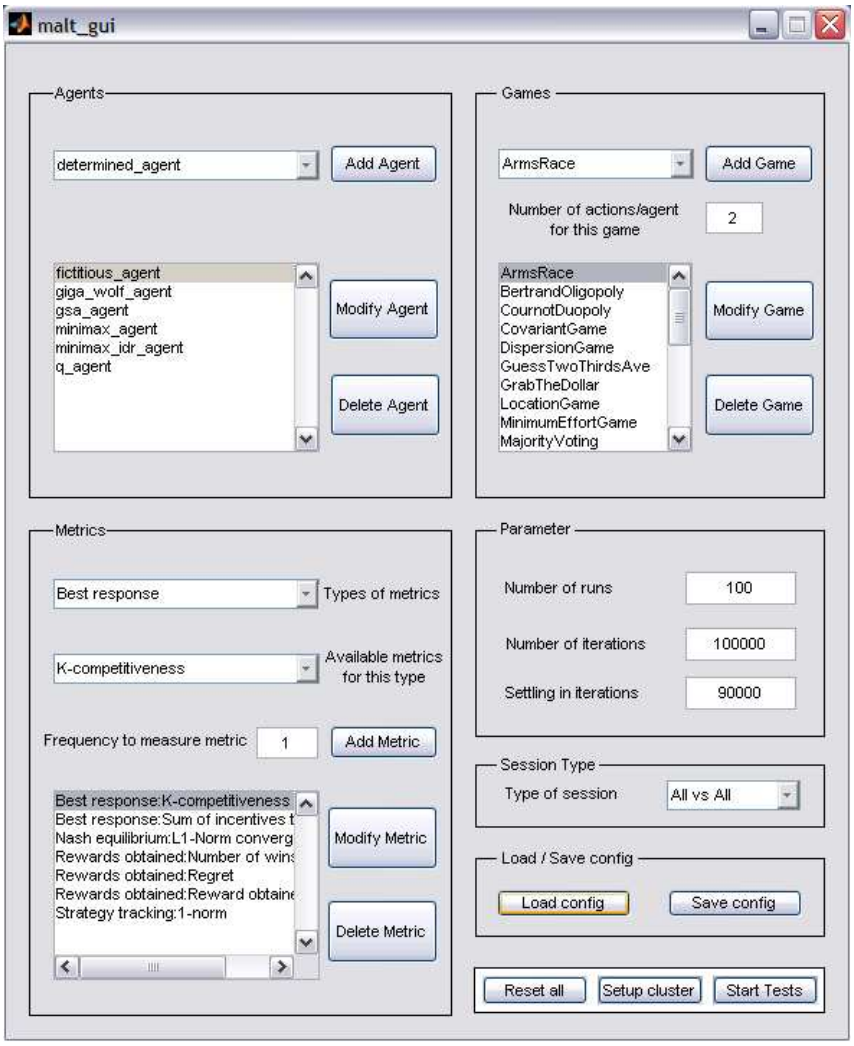

Figure 3. The graphical user interface for phase one of an experiment, where the parameters are set.

nent's strategy. The decay rate for this parameter was set to decay the initial belief update rate to 10^{-6} after 100 000 steps (i.e., a decay rate of 0.999886). The update values for GIGA-WoLF and GSA that are displayed in Table II are based on information supplied in Spall (2003).

The algorithms were tested on thirteen game generators, which are split into two groups. The first twelve generators were used to produce games with ten actions; these generators are listed in Table III. We generated 100 instances from each of these generators. The thirteenth generator was the

Table II. *The six algorithms used in the experiment and the values of their associated parameters. t reflects the value of the current iteration. Minimax-Q, minimax-Q-IDR and Q-learning have the same set of parameters. Fictitious play has no parameters.*

Algorithm	**Parameters**
GIGA-WoLF	$\gamma_t : \frac{10}{(A+t)^{0.602}}$; $A : 5000 = 5\% \; of \; 100000$ This ensures that the update is large at the start and gradually decays over time to $< 10^{-3}$ after 100 000 iterations.
GSA-Agent	$\alpha_t : \frac{10}{(t+1+A)^{0.602}}$; $A : 5000$; $\beta_t : \frac{0.3}{(t+1)^{0.301} log(t+1)}$ This provides the same decrease rate for α as in GIGA-WoLF. We want β_t to decrease at a faster rate and this setting causes it to be $< 10^{-4}$ after 100 000 iterations.
minimax-Q minimax-Q-IDR Q-learning	Initial exploration rate: 0.2; Exploration rate decay: $10^{(log 0.01)/100000}$ $\alpha : 1$; α_{decay}: $10^{(log 0.01)/100000}$; γ: 0.9 Values taken from the minimax-Q paper (Littman, 1994). The decay rates were adapted from the paper, which used one million iterations.
Fictitious play	None

Table III. *The twelve game generators that were used to generate the games with ten actions.*

Generators of games with ten actions		
Arms Race	Bertrand Oligopoly	Cournot Duopoly
Covariant Game	Dispersion Game	Guess Two Thirds Average
Grab The Dollar	Location Game	Minimum Effort Game
Majority Voting	Traveler's Dilemma	War Of Attrition

TwoByTwo game generator, which generates from all of the 85 distinct 2×2 games, see Rapoport et al. (1976). The goal with the 2×2 games was to obtain results over a random distribution of 2×2 games. The ability to sample from the set of possible 2×2 games comes at the expense of being unable to view results for a specific 2×2 game, for example Prisoner's Dilemma. Such experiments would be a worthwhile avenue for future work. Using the 2×2 generator we generated twelve times as much data as we did with each of the twelve 10×10 generators, giving us a total of 1200 10×10 and 1200 2×2 game instances.

A subset of the possible metrics were recorded during the experiment. The metrics were selected based on them measuring different aspects of an algorithm's performance. The metrics are listed in Table IV.

A single run on a game instance consisted of 100 000 stage games, with the first 90 000 allowing the agent's learning parameters to settle and statistics

Table IV. *List of the seven metrics recorded during the experiment.*

Metrics		
K-competitiveness	Sum of incentives to deviate	ℓ_1 norm convergence sum
Number of wins	Regret	Reward obtained
ℓ_1 norm strategy estimation		

being recorded on the last 10 000 stage games. Each instance of a game was played twice, with the agents taking turns to be the row and column players.

We used Gambit (McKelvey et al., 2004) to compute sets of actions that survive iterated dominance removal and Nash equilibria. We used the Kolmogorov-Smirnov Z test to test for statistical similarity between distributions. (This test was performed by the SPSS statistics package, with the data imported from Matlab.) We used a p-value ≤ 0.05 to indicate a statistically significant difference between the distributions being compared.

Most of the graphs presented in this article are box plots. The middle line of the box represents the median of the distribution, the leftmost and rightmost edges of the box are the 25^{th} and 75^{th} percentile values. These percentiles correspond to the values in the distribution, below which are 25% and 75% of the distribution respectively. The two "whiskers" are $1.5 * IQR$ from the edges of the box, with Inter Quartile Range $(IQR) = (75^{th}\ percentile - 25^{th}\ percentile)$. Any crosses $(+)$ are outliers that lie outside the $1.5 * IQR$ distance.

3.2. Reducing the Size of the Experiment Space

Earlier, we discussed the idea that the configuration of an individual experiment can be thought of as corresponding to a single cell of a four dimensional table indexed by (iteration, instance, game, algorithm pairing). For the experiments we conducted this table would have contained 21*24*100* 10 000 = 504 million cells. This would clearly have been too much data to permit us to examine every cell, so we needed to find a way to reduce the table's dimensionality. We consider how (and whether) each dimension can be reduced, examining each in turn.[5]

CLAIM 1. *The iterations dimension can be reduced by averaging metric values across iterations.*

We have observed that metric values often differ from one iteration to the next, which means that it would be inappropriate to keep only a single

[5] To keep this article to a reasonable length, we do not display graphs for the claims in this section. They are provided in Lipson (2005).

iteration as representative of the whole sequence. It is nevertheless desirable to keep only a single value for each metric and each run; therefore, we average over the iterations. We observed no cases where metric values are not either constant or drawn from a small set of values. For the instances where the metric values converged, averaging over these values is clearly suitable: it does not hide or remove any of the underlying information. For the cases where the metric values do not converge, but rather come from a small set of values, averaging over these values does hide some of the underlying detail, but nevertheless preserves information about all the values. Clearly we could choose other aggregation approaches: we could take both the average and the standard deviation, or we could take the median. This point notwithstanding, our choice agrees with choices made by others in the literature: averaging has also been used in the same way by e.g., Powers and Shoham (2004) and Nudelman et al. (2004).

CLAIM 2. *The instances dimension can be reduced by averaging metric values across instances.*

Instances from the same generator that are qualitatively the same can produce different behavior because of payoff differences. This means that we should not judge performance on a single instance, since reporting statistics from one instance would not be representative of an agent's general performance on that generator. Reporting results by aggregating over different instances reduces the effect of the payoffs from any single instance. As above, the choice of averaging rather than aggregating in another way strikes us as reasonable, although other aggregation methods could also be defended.

CLAIM 3. *Generators from the same game theoretic group did not exhibit consistently similar properties according to any of the metrics we measured.*

If similar games led to similar metric values, we would have been able to reduce the games dimension by discarding some of our games. One way of attempting to cluster games is according to their game theoretical properties. Nudelman et al. (2004) taxonomized the games GAMUT can generate according to categories such as dominance solvability and the absence of a pure strategy equilibrium. In our experiment we observed that games from the same category do not cause consistently similar dynamics among agents, but rather produce different behavior across the different generators. This was also true for other clustering approaches that we considered.

CLAIM 4. *No two of our algorithms were similar in performance against all opponents on all games.*

If two algorithms performed similarly against all opponents on all games, we would be able to avoid using both algorithms in an experiment. We could

use just one of the algorithms and take its performance as being representative of the other algorithm. Unfortunately, as will be evident from the results presented in the rest of this section, we did not find this to be the case.

3.3. Experimental Results: Reward-based Metrics

We now move on to considering different algorithms' performance according to the performance metrics we defined in Section 2.1.3. The first group of metrics we consider are those based on reward.

3.3.1. *Raw Reward*

In a sense the average amount of reward obtained by an agent is the most fundamental metric, as agents' explicit goals are to maximize this quantity.

CLAIM 5. *None of our algorithms obtained the highest average reward against every opponent in either the* 2×2 *or* 10×10 *sets of generators.*

Tables V and VI display the average reward obtained by each agent against each other agent in the set of 2×2 and 10×10 game generators respectively.[6] In the 2×2 set of games, GSA and minimax-Q-IDR each obtained the highest average reward against an opponent once and GIGA-WoLF and Q-learning each obtained the highest average reward twice.

In the 10×10 set of games shown in Table VI, the three algorithms that estimate their opponent's strategy—fictitious play, GIGA-WoLF and GSA—obtained the highest average reward. Fictitious play obtained the highest average reward once, GIGA-WoLF twice and GSA three times. Q-learning now obtained a higher average reward than minimax-Q-IDR against all opponents, which was not the case in the 2×2 set of games.

An interesting point to note is that no single algorithm achieved the best average reward against a given opponent for both 2×2 and 10×10 generators. For the 10×10 set of generators, no estimation algorithm clearly outperformed the other estimation algorithms, in the sense that each one of the estimation algorithms obtained the highest average reward against one of the other estimation algorithms.

Although we cannot identify a best algorithm, we do observe from Tables V and VI that the Minimax-Q algorithm performed consistently badly across opponents. We will examine the behavior of this algorithm in more detail in Section 3.3.4.

CLAIM 6. *Q-learning achieved the highest mean and median reward against all opponents in the* 2×2 *set of generators.*

[6] These two tables are reproduced as bar graphs in Figures 18 and 19 in Appendix B at the end of this article.

Table V. *Average reward obtained by each agent against each other agent in the set of* 2×2 *generators. Larger scores are better.*

	Fictitious play	GIGA-WoLF	GSA	minimax-Q	minimax-Q-IDR	Q-learning
Fictitious play vs	0.217	0.231	0.261	0.242	0.242	0.218
GIGA-WoLF vs	0.475	0.213	0.254	0.242	0.242	0.223
GSA vs	0.455	0.464	0.218	0.233	0.241	0.222
minimax-Q vs	0.235	0.235	0.229	0.099	0.070	0.113
minimax-Q-IDR vs	0.410	0.410	0.422	0.364	0.129	0.162
Q-learning vs	0.370	0.391	0.422	0.342	0.428	0.176

Table VI. *Average reward obtained by each agent against each other agent in the set of* 10×10 *generators. Larger scores are better.*

	Fictitious play	GIGA-WoLF	GSA	minimax-Q	minimax-Q-IDR	Q-learning
Fictitious play vs	0.339	0.297	0.383	0.379	0.247	0.383
GIGA-WoLF vs	0.357	0.284	0.304	0.379	0.247	0.423
GSA vs	0.290	0.357	0.257	0.384	0.248	0.403
minimax-Q vs	0.087	0.087	0.088	0.190	0.078	0.118
minimax-Q-IDR vs	0.154	0.148	0.153	0.210	0.122	0.176
Q-learning vs	0.257	0.236	0.204	0.301	0.162	0.344

Though we cannot say that a single algorithm beat all the other algorithms or that there is a consistent pattern to the ranking of all of the algorithms, we can identify a "best" algorithm based on average reward over all opponents. Q-learning obtained the highest median and mean reward in the 2×2 set of games when we average over all of the opponents. Figure 4 (top) displays the reward distributions that each agent received against all opponents on the 2×2 set of games. Recall that the middle line in a box plot is the median of the distribution. Table VII provides the results of a Kolmogorov-Smirnov Z test performed on the reward distributions obtained by Q-learning and the algorithm that achieves the next highest median reward, GSA. There is a

Table VII. *Kolmogorov-Smirnov test comparing the distribution of reward obtained by Q-learning and GSA in the set of* 2×2 *game generators.*

	Distributions are statistically different: $p \approx 0$	
Statistic	Q-learning	GSA
mean	0.355	0.305
median	0.490	0.356
25^{th} percentile	-0.040	-0.139
75^{th} percentile	0.804	0.086

statistically significant difference between the Q-learning and GSA reward distributions in the set of 2×2 game generators, $p \approx 0$.

CLAIM 7. *Fictitious play obtained the highest median and mean reward in the* 10×10 *set of generators.*

Figure 4 (bottom) displays the reward distribution obtained by each agent against all opponents in the 10×10 set of generators. Fictitious play obtained the highest median reward, with the other strategy estimating algorithms, GIGA-WoLF and GSA performing similarly. Q-learning did not perform as well on the 10×10 set of generators as in the 2×2 set of games. The three estimation algorithms all obtained higher median, 25^{th} and 75^{th} percentile reward values than Q-learning. The estimation algorithms also obtained the three highest mean reward positions in the 10×10 set of generators. The larger game size appear to benefit the algorithms that estimate their opponent's strategy. However, we also observed that there is no trend in performance as the number of actions in the game increases (see Observation 3 in Section 3.5). This reminds us that it is the structure of the games rather than simply their size that affected the performance of the algorithms.

Figure 5 displays the reward distribution obtained by each agent against all opponents on all generators. (That is, the game generators are not divided into 2×2 and 10×10 games as in Figure 4.) Q-learning had the highest median reward, due to its performance in the 2×2 set. GSA obtained a higher 75^{th} percentile value, which caused it to achieve the highest mean reward.

3.3.2. *Regret*

We now consider other metrics that ask counterfactual questions about reward, considering whether the agent could have obtained a higher reward by playing a different action. First we consider regret, asking whether the agent would have gained more reward by playing a stationary pure strategy.

The GIGA-WoLF algorithm (Bowling, 2004) offers the theoretical guarantee that it will obtain at most zero average regret. However, to our knowl-

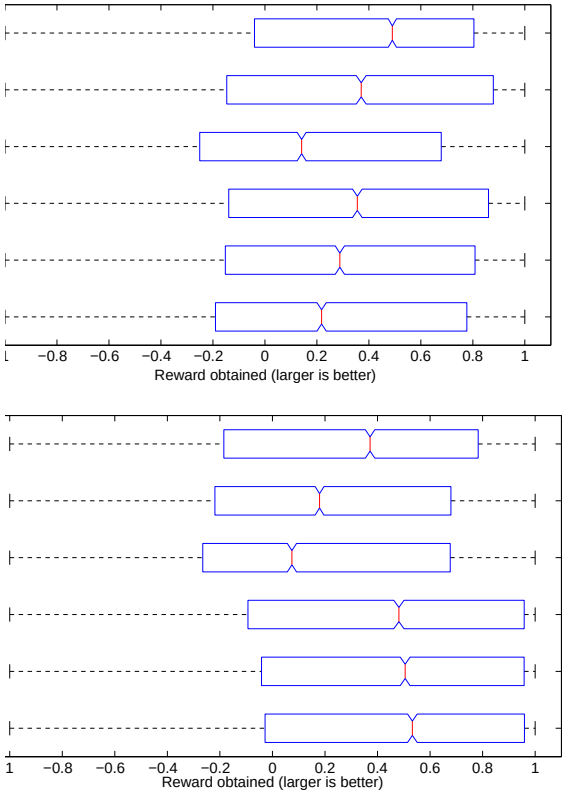

Figure 4. Distribution of reward obtained by each agent against all algorithms in the set of 2×2 *(top) and* 10×10 *(bottom) game generators.*

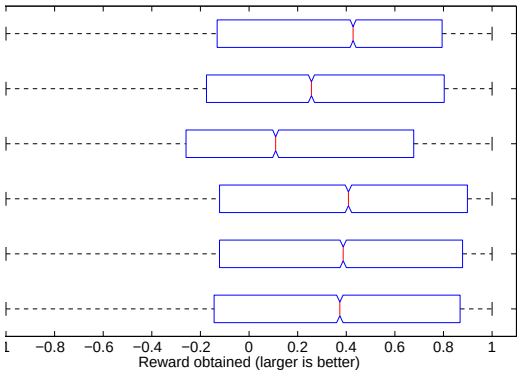

Figure 5. Distribution of reward obtained by each agent against all algorithms on both sets of game generators.

Table VIII. *Kolmogorov-Smirnov test comparing the distribution of regret obtained by GIGA-WoLF and Q-learning.*

	Regret in 2×2 game distribution		Regret in 10×10 game distribution	
	Distributions are statistically different: $p \approx 0$		Distributions are statistically different: $p \approx 0$	
Statistic	GIGA-WoLF	Q-learning	GIGA-WoLF	Q-learning
mean	-0.0004	0.0052	0.0003	0.0263
median	0	0	0	0
25^{th} percentile	0	0	0	0
75^{th} percentile	0	0	0.0001	0

edge it has not been shown experimentally how the regret achieved by this algorithm compares to the regrets achieved by other algorithms; nor has it been demonstrated whether GIGA-WOLF often achieves better than zero regret in practice.

CLAIM 8. *GIGA-WoLF achieved lower average regret than our other algorithms, and often achieves negative regret.*

Figure 6 shows a box plot of the regret distribution for each algorithm against all other algorithms on the set of 2×2 (top) and 10×10 (bottom) game generators. GIGA-WoLF had a substantially smaller variance than any other algorithm and consistently achieved negative regret. Averaging over all of the opponents, GIGA-WoLF achieved negative average regret in the 2×2 set of games and an average regret of ≈ 0 in the 10×10 set of generators. The other algorithms did tend to achieve zero median regret but had higher average regret. Q-learning was sometimes able to achieve negative regret on both sets of generators, and fictitious play did so on the 10×10 set of generators.

To confirm that there is a statistical difference between the regret distributions obtained by GIGA-WoLF and Q-learning, we performed a Kolmogorov-Smirnov Z test. Table VIII provides the results of this test. In both the 2×2 and 10×10 generator sets there is a statistically significant difference between the regret distributions achieved by GIGA-WoLF and Q-learning.

3.3.3. *Incentive to Deviate*

We now consider agents' incentive to deviate, which can be understood as their regret with respect to arbitrary nonstationary strategies. In other words, incentive to deviate measures how much extra reward an agent would have obtained had she managed to play a best response to her opponent's action in each stage game.

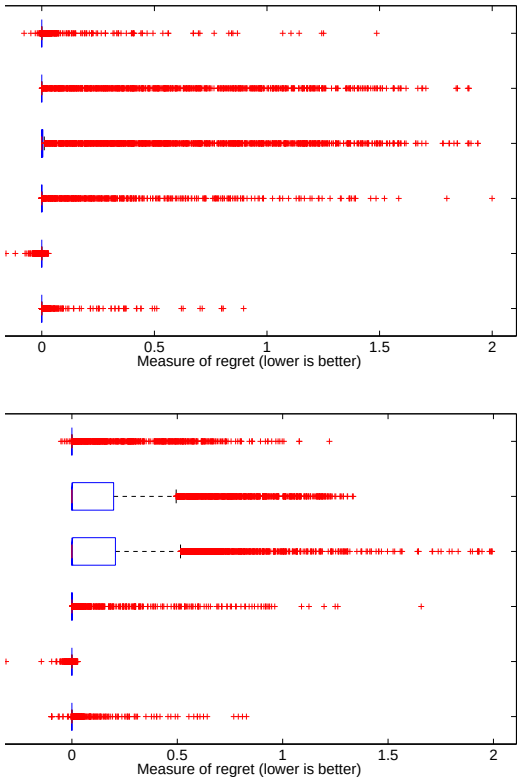

Figure 6. Box plot of the distribution of regret obtained by each agent against all other agents in the 2×2 *(top) and* 10×10 *(bottom) set of generators. In this plot all medians are at 0. The +'s indicate outliers—points that lie more than 1.5 times the interquartile range away from the edges of the box.*

CLAIM 9. *GIGA-WoLF consistently achieved a lower incentive to deviate score against its opponents than our other algorithms.*

Table IX displays the incentive to deviate score achieved by each agent against each other agent averaged over the set of 2×2 games[7] . The values are sorted according to ascending incentive to deviate values obtained across *all* games (i.e., the averages of the values in Table IX and Table 21.) Against all agents GIGA-WoLF achieved the lowest score, with fictitious play always a close second. The table also shows a fairly regular order of incentive to deviate: GIGA-WoLF, fictitious play, GSA, minimax-Q-IDR, Q-learning and

[7] Tables IX and X are shown in a bar graph format in Figures 20 and 21 in Appendix B at the end of this article.

Table IX. *Average incentive to deviate values achieved by each agent in a table row against each of the agents listed in the columns in the set of* 2×2 *games. Smaller scores are better.*

	Fictitious play	GIGA-WoLF	GSA	minimax-Q	minimax-Q-IDR	Q-learning
GIGA-WoLF vs	0.034	0.036	0.014	0.045	0.018	0.054
Fictitious play vs	0.037	0.038	0.019	0.046	0.018	0.056
GSA vs	0.045	0.054	0.031	0.053	0.019	0.06
Q-learning vs	0.115	0.120	0.103	0.124	0.098	0.118
minimax-Q-IDR vs	0.093	0.096	0.097	0.102	0.131	0.100
minimax-Q vs	0.254	0.254	0.254	0.187	0.189	0.161

Table X. *Average incentive to deviate values achieved by each agent in a table row against each of the agents listed in the columns in the set of* 10×10 *games. Smaller scores are better.*

	Fictitious play	GIGA-WoLF	GSA	minimax-Q	minimax-Q-IDR	Q-learning
GIGA-WoLF vs	0.014	0.017	0.012	0.098	0.036	0.050
Fictitious play vs	0.023	0.024	0.017	0.097	0.036	0.058
GSA vs	0.026	0.030	0.022	0.098	0.036	0.062
Q-learning vs	0.147	0.159	0.132	0.177	0.114	0.187
minimax-Q-IDR vs	0.213	0.227	0.215	0.268	0.162	0.247
minimax-Q vs	0.265	0.266	0.278	0.288	0.206	0.274

minimax-Q. This order only differed when minimax-Q-IDR was an opponent, in which case Q-learning obtained a lower score than minimax-Q-IDR.

Table X displays the results for the 10×10 set of generators. Results are generally the same as in the previous table, but now the order of the algorithms' scores is the same for all opponents. Q-learning now always achieved a better score than Minimax-Q-IDR, which is the opposite of what occurred in the 2×2 games. Again, GIGA-WoLF consistently achieved the lowest incentive to deviate against all opponents. These results are interesting since incentive to deviate may be seen as a kind of regret, and GIGA-WoLF is designed to minimize regret of a different kind.

3.3.4. *Minimax-Q and Domination*

Although for the most part we cannot make broad claims about how effective each algorithm was at attaining reward, we can say more about one algorithm: Minimax-Q.

CLAIM 10. *Minimax-Q-IDR consistently achieved higher levels of reward than minimax-Q.*

Minimax-Q-IDR obtained higher median reward than minimax-Q against all opponents in every game drawn from the 2×2 generator. This was also true in the majority of 10×10 generators. Due to this performance difference, minimax-Q will sometimes be dropped from the results in the next section.

The games in which minimax-Q obtained higher reward are characterized generally by having a single equilibrium. The generators are Cournot Duopoly, Bertrand Oligopoly, Arms Race, Location game, Traveler's Dilemma, Covariant game, Dispersion game and War of Attrition. Minimax-Q-IDR assumes that its opponents will not play dominated actions, which of course is not always true. For example, in the Traveler's Dilemma game both players can achieve higher rewards by playing dominated strategies than they can by playing non-dominated strategies. (That is why this game is called a dilemma!) This suggests that while minimax-Q-IDR plays the Nash equilibrium strategy in these games, minimax-Q does not.

CLAIM 11. *Minimax-Q-IDR dominated GIGA-WoLF in Traveler's Dilemma when judged by percentage of wins.*

Nudelman et al. (2004) compared minimax-Q, Q-learning and one of the earlier variants of WoLF across a distribution of games. They found that for the percentage of wins metric, minimax-Q dominated WoLF in the Traveler's Dilemma generator: that is, that Minimax-Q *always* achieved the higher reward in interactions between the two algorithms. This was shown for a two-action version of the game.

We were interested in whether this claim would hold true for different sizes of the game, and for the updated version of WoLF. We ran a relatively small additional experiment that considered 20 instances of a 2×2 and 10×10 version of Traveller's Dilemma. The configuration file for the main experiment was used to ensure consistency among all parameters. The results showed that the claim that minimax-Q dominates WoLF does not extend to GIGA-WoLF. However, we saw that *minimax-Q-IDR* did substantially outperform both minimax-Q and GIGA-WoLF in both the 2×2 and 10×10 versions of Traveller's Dilemma.

We show a subset of the win percentages for the minimax-Q, minimax-Q-IDR and GIGA-WoLF algorithms on the two-action version of Traveler's Dilemma in Figure 7 (top). GIGA-WoLF now won 74% of the interactions

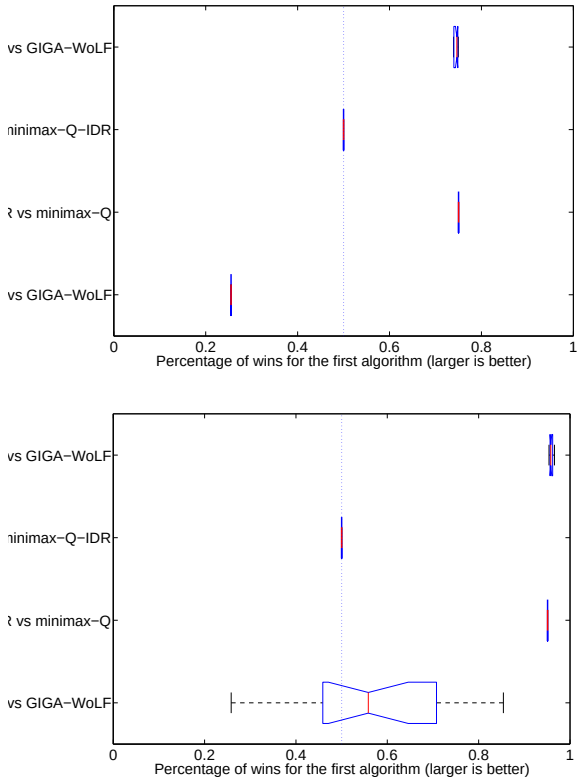

Figure 7. Percentage of wins that minimax-Q and minimax-Q-IDR obtain against minimax-Q, minimax-Q-IDR and GIGA-WoLF in a 2×2 *(top) and* 10×10 *(bottom) Traveler's Dilemma.*

against minimax-Q, but only 25% of the interactions against minimax-Q-IDR. GIGA-WoLF also won more than 50% of the games against Q-learning and GSA. Minimax-Q won less than 50% of the interactions against any opponent. Minimax-Q-IDR won more than 50% of the interactions against every opponent.

The results are slightly different in a ten-action version of Traveler's Dilemma, with the distributions of win percentage values shown in Figure 7 (bottom). Minimax-Q-IDR did even better in the larger game, winning more than 90% of the interactions against all non-self play opponents. The results changed most substantially for minimax-Q against GIGA-WoLF, with minimax-Q now having won 55% of their encounters, compared to 25% in the 2×2 version of Traveler's Dilemma. Other than against minimax-Q-IDR, minimax-Q won more than 50% of the games against all opponents. GIGA-WoLF only won more than 50% of the interactions against fictitious play. This again demon-

strates how testing algorithms on a game at one size can give results that are not representative of performance at other sizes of that game.

3.4. Experimental Results: Convergence to a Nash Equilibrium

We now examine whether algorithms converge to a Nash equilibrium. Note that we cannot ask this question about a single algorithm because the concept of equilibrium depends on both players' strategies.

There has been some debate in the literature about whether it is reasonable to evaluate algorithms according to their ability to converge to an equilibrium. For example, Shoham et al. (2003) argue that an unconditional focus on Nash equilibrium is misguided. They suggest that the focus should be on meeting an objective (e.g., achieving high reward) given information about the types of agents who inhabit the environment. In this scenario, "the equilibrium concept [is not considered] central or even necessarily relevant at all" Shoham et al. (2003). However, the Nash equilibrium retains broad appeal as the "solution" to a game. These are at least "focal" strategies in some sense; if two algorithms are playing Nash equilibrium strategies, both are best responding to each other and neither can obtain a higher payoff by unilaterally changing her strategy.

3.4.1. *Linking Reward and Convergence*

The first experimental question to ask is whether it turns out that good performance under reward- and equilibrium-based metrics are correlated; if so, the philosophical debate about choosing between the metrics would appear less important.

CLAIM 12. *There was no relationship between obtaining large reward and converging to a Nash equilibrium.*

Our experiments do not support the idea that there is a link between the reward that an agent receives and its convergence to a Nash equilibrium. Two agents can be equally far from an equilibrium and not obtain similar rewards. Two different pairs of agents could also converge to different equilibria that have different payoffs.

At a high level, Figure 8 displays the combined ℓ_1 score for convergence to a Nash equilibrium and the reward obtained by each agent in the set of 2×2 (top) and 10×10 (bottom) game generators.[8] The results for each agent are aggregated against all opponents. Minimax-Q is not included in these tests. In these figures, we compare the reward and convergence scores between agents, rather than comparing these scores for the same agent. Note

[8] Recall the definition of this performance measure in Section 2.1.3.

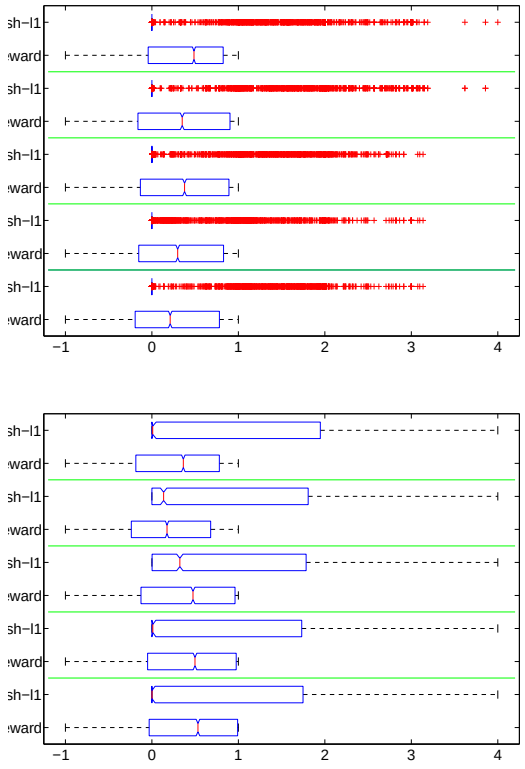

Figure 8. Distributions of the reward and ℓ_1 distance to a Nash equilibrium obtained in the 2×2 (top) and 10×10 (bottom) set of generators. FP represents fictitious play.

that in the top figure all convergence box plots have their mass at zero, with the pluses indicating outliers. This means that the vast majority of data points had a value of zero, contrary to the visual impression given by the graph.

Focusing on the 10×10 plot of Figure 8 (bottom), it can be seen that GSA obtains a higher median reward than Q-learning but has a worse convergence rate. GIGA-WoLF obtains a similar reward distribution to GSA, but has a very different Nash convergence distribution.

In Figure 9 we examine only the Arms Race generator, showing rates of convergence to a Nash equilibrium and the reward obtained for Q-learning and fictitious play in self play. (This configuration is provided only as an example; qualitatively similar results were obtained for many other algorithm-generator combinations.) Both agents converge to an equilibrium, but obtain very different reward distributions.

We did find one robust connection between reward and convergence to a Nash equilibrium. If algorithm *A* plays *B* and *A* also plays *C* and in both

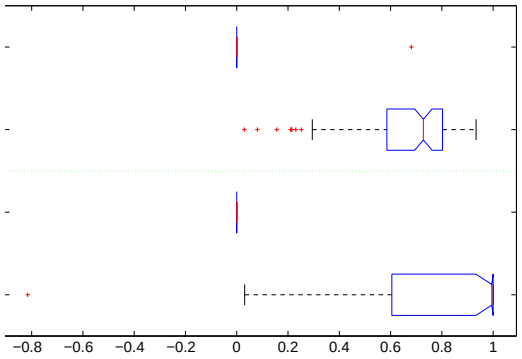

Figure 9. Distributions of the reward and ℓ_1 distance to a Nash equilibrium obtained in the Arms race game by fictitious play and Q-learning in self play.

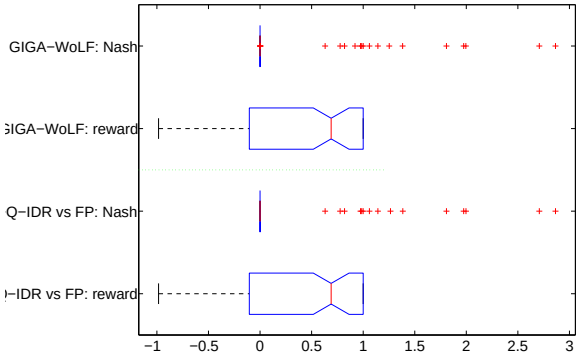

Figure 10. Minimax-Q-IDR obtains similar distributions of the ℓ_1 distance to a Nash equilibrium against two opponents. The resulting reward distribution is also similar.

cases A obtains similar convergence rates, then A can be expected to achieve a similar reward distribution in both cases. An example that follows this rule is shown in Figure 10, where the distributions for the reward obtained and distance to the Nash equilibrium are shown for minimax-Q-IDR against fictitious play and GIGA-WoLF. The algorithms play similar strategies leading to the same set of rewards. We performed a Kolmogorov-Smirnov test comparing the resultant Nash and reward distributions, which showed that there was no significant difference between the metric values that minimax-Q-IDR obtained against fictitious play and against GIGA-WoLF ($p \approx 1$ in both cases). Similar examples of this connection can be found for all of the algorithms tested in our experiment.

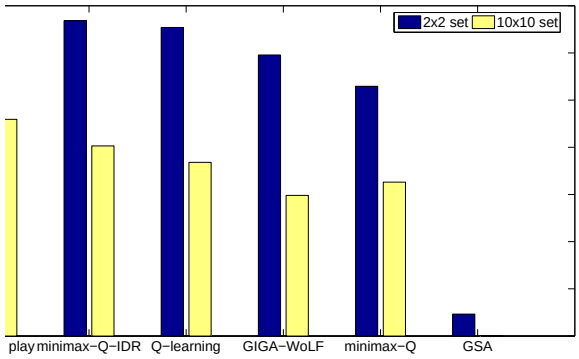

Figure 11. Percentage of repeated games where an algorithm and its opponent converge exactly to a Nash equilibrium in the two generator sets.

Interestingly, our experimental evidence does *not* allow us to generalize the rule described above to say that when algorithm *A* plays *C* and algorithm *B* plays *D*, if *A* and *B* obtain similar Nash convergence distributions, then they should also obtain similar reward distributions.

The results of this experiment suggest that convergence to Nash equilibrium and reward are not necessarily correlated and that one metric cannot substitute for both. Thus, we believe that when possible, future studies should report both statistics.

3.4.2. *Exact Convergence to a Nash Equilibrium*

We now investigate the extent to which our algorithms converged exactly (within machine precision) to Nash equilibria.

CLAIM 13. *Our algorithms often converged to Nash equilibria, but also often failed to converge.*

Results are shown for the percentage of interactions in which an algorithm and its opponent converged to a Nash equilibrium. In the 2×2 set of games, the majority of algorithms converged more than half the time, while in the 10×10 set of generators no algorithm did so. In the literature, convergence to an equilibrium strategy is often considered for self play on a single game. Here we consider each algorithm in self play and against six opponents, across our full sets of 2×2 and 10×10 generators.

Figure 11 displays the empirical probability of each of the algorithms converging with their opponent to a Nash equilibrium. The algorithms are sorted according to decreasing overall probability of convergence, averaged across both sets of games.

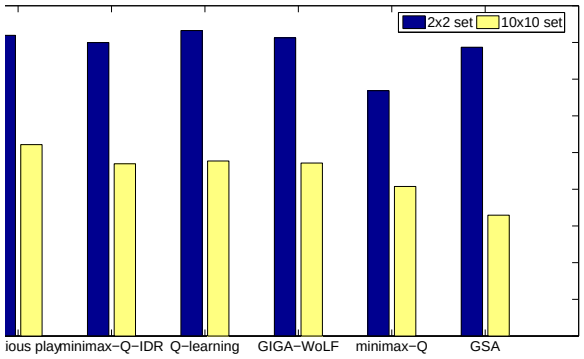

Figure 12. Percentage of repeated games where an algorithm and its opponent converge on average to an ℓ_1 distance of within 0.005 of an equilibrium.

Fictitious play had the highest success in both sets of games, converging more than two thirds of the time in the 2×2 set. Surprisingly, it was followed by minimax-Q-IDR, perhaps an indication of how often the 2×2 games could be iteratively reduced to a Nash equilibrium. Equally surprisingly is the fact that GIGA-WoLF was not as successful and was in fact beaten by Q-learning in both families of generators. Q-learning regularly converged, despite the facts that it always plays a pure strategy and that it ignores its opponent's ability to adapt. GSA almost never converged to an equilibrium, because its property of jumping out of local minima prevented it from converging at all.

For all of the algorithms, we observed a higher percentage of convergence in the 2×2 set of games than in the 10×10 set of generators. One possible reason for this is that pure strategy equilibria were more prevalent in our 2×2 games. Coordination problems may also have been simpler to solve in the smaller games.

3.4.3. *Joint ℓ_1 Measure of Convergence to a Nash Equilibrium*

We can now ask the question of whether our algorithms converge to a strategy that is *close* to a Nash equilibrium, regardless of whether they converge exactly. To answer this question, we use the ℓ_1 measure of convergence to a Nash equilibrium, and ask how often the algorithms converged to strategies that were less than 0.005 from a Nash equilibrium.

CLAIM 14. *Our algorithms often converged to strategies that are close to a Nash equilibrium.*

Figure 12 displays the percentage of repeated games in which the average joint ℓ_1 distance to the closest Nash equilibrium was less than 0.005. The

algorithms are listed in the same order as Figure 11. The results show that in the 2×2 and 10×10 sets every algorithm had a substantially higher chance of being "near" an equilibrium than of converging exactly to it. The percentage increase in convergence rates for the 2×2 games was a minimum of 13%, a maximum of 74% and a median of 16%. For the 10×10 games it was somewhat smaller, with a minimum of 7%, a maximum of 33% and a median of 10%. The biggest increase was for the GSA algorithm, where the values increased by 74% and 33% in the 2×2 and 10×10 set of generators respectively. This supports our earlier claim that the "jumping" step in the GSA algorithm caused it not to converge exactly to Nash equilibria, even when it got very close. Our algorithms were still more likely to nearly converge in the 2×2 set than the 10×10 set of generators. In the 2×2 set of games, Q-learning got close to an equilibrium marginally more often than fictitious play. In the 10×10 set of generators the situation was reversed, and fictitious play reached the window more than half the time.

The most important conclusion to draw is that all of our algorithms converged to near-equilibrium strategies much more often than they converged exactly. Researchers conducting future experimental work should therefore consider reporting convergence to strategies that are near equilibria as well as to exact equilibria.

3.4.4. *Convergence in Self Play*

We now differentiate between self and non-self play to see where our algorithms are more likely to converge to equilibria. Arguably, the property of convergence to a Nash equilibrium in self play is more important than the property of convergence to a Nash equilibrium against an arbitrary opponent, for two reasons. First, we pointed out above that convergence to a Nash equilibrium is a metric that measures both algorithms playing the game; however, in self play this metric really does describe only a single algorithm. Second, if an algorithm is successful then it could be widely adopted, meaning that it *should* expect to encounter itself in self play.

CLAIM 15. *Our algorithms converged more often to an exact Nash equilibrium in self play than in non-self play.*

The majority of theoretical results for convergence in the literature are for self play. We are not aware of previous empirical comparisons between self and non-self play convergence rates. Figure 13 shows the percentage of repeated games in which an algorithm converged exactly to a Nash equilibrium. The results are shown for self and non-self play in the two generator sets and are sorted according to the self play probabilities averaged across all games.

For all algorithms except GSA, the convergence probabilities dropped when moving from self to non-self play; GSA almost never converged in

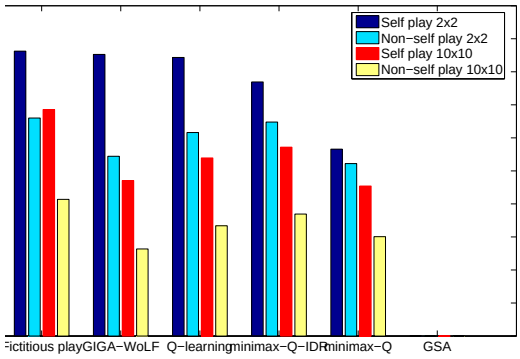

Figure 13. Percentage of repeated games where an algorithm converged on average to an exact Nash equilibrium in self and non-self play.

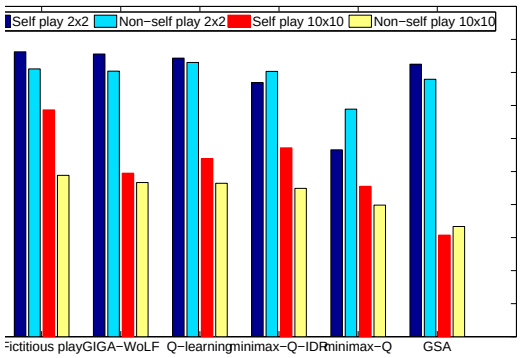

Figure 14. Percentage of repeated games where an algorithm converged on average to an ℓ_1 distance of within 0.005 of an equilibrium, in self and non-self play.

either self or non-self play. Interestingly the difference in convergence rate for minimax-Q between self play (57%) and non-self play (52%) in our 2×2 games was quite small. Q-learning, which ignores what the opponent is doing, was very successful. Since Q-learning only plays pure strategies, this indicates that many of our 2×2 games had pure strategy equilibria.

CLAIM 16. *The difference in convergence rates to strategies close to a Nash equilibria between self and non-self play is smaller than the difference in convergence rates for self and non-self play for exact convergence rates.*

Figure 14 presents the rates for algorithms converging to a strategy close to a Nash equilibrium in self and non-self play in the 2×2 and 10×10 generator

sets. In the 2×2 set, all algorithms except for minimax-Q had a difference of less than 5% between self play and non-self play convergence rates. This is different from the results displayed in Figure 13, where minimax-Q was the only algorithm with a small difference between self and non-self play convergence rates in the 2×2 set of games.

GSA and GIGA-WoLF are the only algorithms that have any significant difference in self play percentages between convergence to the exact equilibrium and convergence to a strategy close to an equilibrium. This indicates an important difference between self-play and non-self-play convergence: when most algorithms converge in self play, they tend to converge to the exact Nash equilibrium. This was true for both the 2×2 and 10×10 generators (compare Figures 13 and 14). In the 10×10 set, the only algorithms that got close to an equilibrium more often than they converged to the exact equilibrium were the three tracking algorithms. In non-self play, this was not the case and there were large differences in the convergence rates when comparing exact convergence to convergence to strategies close to an equilibrium.

3.5. Observations from Empirical Play

We now present a series of high level observations that we have made from our empirical study. We list these to provide the reader with an overview of our results and to single out some noteworthy points.

OBSERVATION 1. *No algorithm in our test set dominated.*

In our sets of 2×2 and 10×10 generators there was no single algorithm that beat every other algorithm on all games. While an algorithm might have performed well according to one metric (e.g., GIGA-WoLF on regret), there is no algorithm that dominated on both reward and Nash convergence metrics.

OBSERVATION 2. *An algorithm needs to be tested on a variety of generators in order to obtain accurate performance information.*

We feel that this is one of our more important observations. We observed a great deal of variance in our experimental results as we varied the game generators and their payoff structures. We can therefore conclude that it is dangerous to claim anything about an algorithm's general performance based on an experiment involving a single instance of one game. Instead, testing needs to be done over a wide variety of games.

OBSERVATION 3. *We observed no relationship between the reward an algorithm obtained and the number of actions in a game. Thus it is a good idea to test on games with different numbers of actions.*

We ran a small additional experiment to check for a relationship between the reward that algorithms obtain and the number of actions in a game. We used two generators, the *Covariant game* and *Grab the dollar*, generating twenty random instances of each game from two to ten actions. A linear regression test showed that there was no linear relationship between the reward that an agent obtained and the number of actions in these game generators. Subjectively, we could see no evidence of a nonlinear relationship either. When claims are made about algorithm performance it should be with reference to the specific game size tested, since the performance of an algorithm on other game sizes may not always be similar. While it may be possible that a trend exists for other generators, at the least our experiment shows that such a trend does not exist for *all* generators.[9]

OBSERVATION 4. *Having information about the games and opponents being played can provide insight into which algorithm to use in a given environment.*

This argument follows from the observation that algorithms perform differently on different generators and against different opponents. We note that a learning algorithm that makes use of such prior information was proposed by Powers and Shoham (2004).

OBSERVATION 5. *The choice of which algorithm to use in an environment should depend on which metrics of performance are considered important; success on one metric often fails to transfer to success on another.*

We mentioned in our discussion of Claim 12 that success in converging to a Nash equilibrium and success in obtaining reward were not equivalent. When deciding what algorithm to use in a given scenario or when designing a new algorithm, our results suggest that one should choose a specific metric that will be used for evaluation. It may then be easier to identify an algorithm that would achieve the desired objective; e.g., GIGA-WoLF was observed to be very effective at minimizing regret.

OBSERVATION 6. *Fictitious play was best at converging to a Nash equilibrium.*

Fictitious play obtained the highest rates of convergence to a Nash equilibrium, as we showed in Section 3.4.[10] This was especially true for the larger 10×10 set of generators.

[9] Freund and Schapire (1996) presents work on large games with regret matching algorithms.

[10] One of our anonymous reviewers suggested that the algorithms in (Foster and Vohra, 1997) may be expected to converge to a Nash equilibrium even more reliably than fictitious play.

OBSERVATION 7. *If reward is important, then GSA or Q-learning are good options.*

Figure 4 showed that GSA and Q-learning obtained the highest reward averaged over all opponents and generators. Q-learning had the highest median reward and GSA had the highest mean reward. These algorithms both scored relatively well against each of the other opponent algorithms.

OBSERVATION 8. *Iterated removal of dominated actions may benefit algorithms other than minimax-Q.*

As discussed in Claim 10, minimax-Q-IDR consistently achieved higher reward than minimax-Q and generally outperformed it. These algorithms play the same strategy except for the iterative removal of dominated strategies. As far as we know, the combination of IDR with the minimax-Q strategy is new to this paper; the approach of removing dominated strategies could be beneficial to other algorithms. We mention this as an open question in the conclusion.

OBSERVATION 9. *Our platform made it easier to run large experiments.*

Our platform proved extremely useful for this research. It was a straightforward process to set up the parameters for each experiment. The platform then automatically split up the experiment into individual jobs for our cluster and these jobs were run in parallel. This system ran for seven weeks with an approximate total CPU time of two years. We were forced to rerun some portions of the experiments due to small cluster outages; nevertheless, our platform made it easy to combine all of our experimental results once all of our data had been collected.

The platform also served us well in the analysis phase of our experimental work. The GUI greatly speeded the selection of parameters of interest, and probably meant that we ran many more iterations of analysis (visualization, formation of a hypothesis, testing of the hypothesis through a new visualization, and so on) than we otherwise would have done. There was rarely a need to run new experiments when testing a new hypothesis as all of the data combinations were already present. When a smaller sub-experiment was necessary to test a hypothesis, this was set up by loading the configuration file from the main experiment, ensuring consistency with the parameters in the main experiment.

4. Conclusion

This article has described a platform for conducting experiments with multiagent reinforcement learning algorithms and the results of an experimental investigation conducted with this platform. One of the main lessons we learned is that it can be misleading to use a single metric to judge an algorithm's performance: each of our metrics described a different facet of an algorithm's performance, and performance as measured with different metrics was often only weakly correlated.

We end with some questions that were raised by our work, and that could serve as interesting directions for future study:

1. *What sorts of algorithms will dominate Q-learning?*

   Q-learning proved to be surprising resilient for an algorithm that ignores the existence of other agents in the environment. Why was it able to obtain such impressive results? While of course algorithms could be specifically designed to target Q-learning, what sorts of general approaches would perform well against Q-learning and also perform well in other settings? What can we learn from the fact that Q-learning performs so well?

2. *Could other algorithms be improved by iteratively removing dominated actions and then applying the algorithm's strategy to the remaining payoff space?*

   Minimax-Q-IDR produced better results over minimax-Q by following this strategy. Would this technique be useful to other algorithms?

3. *Why were our results on our set of 10x10 games qualitatively different from our results on our set of 2x2 games?*

   We conducted an exploratory experiment with two generators, looking for a relationship between algorithm performance and game size. The results suggest that there was no clear relationship between reward performance and the number of actions in the game for any of the agents. However, we obtained consistently different results for the two generator sets. (For example, Figure 11 on page 28 showed that algorithms converge to an equilibrium less often in the set of generators with ten actions.) Was this due to different game structures, were the results random, or is there another explanation?

References

Axelrod, R.: 1987, *Genetic Algorithms and Simulated Annealing*, Chapt. The Evolution of Strategies in the Iterated Prisoner's Dilemma, pp. 32–41. Morgan Kaufman, Los Altos, CA.

Bowling, M.: 2000, 'Convergence Problems of General-sum Multiagent Reinforcement Learning'. In: *ICML 7*. pp. 89 – 94.

Bowling, M.: 2004, 'Convergence and no-regret in multiagent learning'. In: *NIPS 17*.

Bowling, M. and M. Veloso: 2001a, 'Convergence of Gradient Dynamics with a Variable Learning Rate'. In: *ICML 18*.

Bowling, M. and M. Veloso: 2001b, 'Rational and Convergent Learning in Stochastic Games'. In: *IJCAI 17*.

Bowling, M. and M. Veloso: 2002, 'Scalable Learning in Stochastic Games'. In: *AAAI Workshop on Game Theoretic and Decision Theoretic Agents*.

Brown, G.: 1951, 'Iterative Solution of Games by Ficticious Play'. In: *Activity Analysis of Production and Allocation*. New York.

Claus, C. and C. Boutilier: 1997, 'The Dynamics of Reinforcement Learning in Cooperative Multiagent Systems'. In: *AAAI 4*. pp. 746 – 752.

Foster, D. and R. Vohra: 1997, 'Regret in the On-line Decision problem'. **21**, 40 – 55.

Freund, Y. and Y. Schapire: 1996, 'Game theory, on-line prediction, and boosting'. In: *9th Annual Conference on Computational Learning Theory*. pp. 325 – 332.

Fudenberg, D. and D. Levine: 1999, *The Theory of Learning in Games*. MIT Press.

Fudenberg, D. and J. Tirole: 1991, *Game Theory*. MIT Press.

Greenwald, A. and K. Hall: 2003, ''. In: *ICML 20*.

Hart, S. and A. Mas-Colell: 2000, 'A Simple Adaptive Procedure Leading to Correlated Equilibrium'. *Econometrica* **68**, 1127 – 1150.

Hu, J. and M. Wellman: 1998, 'Multiagent Reinforcement Learning: Theoretical Framework and an Algorithm'. In: *ICML 15*. pp. 242 – 250.

Kaelbling, L., M. Littman, and A. Moore: 1996, 'Reinforcement Learning: A Survey'. *Journal of Artificial Intelligence Research* **4**, 237 – 285.

Lipson, A.: 2005, 'An Empirical Evaluation of Multiagent Learning Algorithms'. Master's thesis, University of British Columbia, Vancouver, Canada.

Littman, M.: 1994, 'Markov Games as a Framework for Multi-agent Reinforcement Learning'. In: *ICML 11*. pp. 157 – 163.

Littman, M.: 2001, 'Friend-or-foe Q-learning in General-Sum Games'. In: *ICML 18*. pp. 322 – 328.

McKelvey, R., A. McLennan, and T. Turocy: 2004, 'Gambit: Software Tools for Game Theory'. Version 0.97.0.6. `http://econweb.tamu.edu/gambit`.

Nudelman, E., J. Wortman, K. Leyton-Brown, and Y. Shoham: 2004, 'Run the GAMUT: A Comprehensive Approach to Evaluating Game-Theoretic Algorithms'. In: *AAMAS 3*.

Osborne, M. and A. Rubinstein: 1994, *A Course in Game Theory*. MIT Press.

Powers, R. and Y. Shoham: 2004, 'New Criteria and a New Algorithm for Learning in Multi-Agent Systems'. In: *NIPS 17*.

Powers, R. and Y. Shoham: 2005, 'Learning Against Opponents with Bounded Memory'. In: *IJCAI*.

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

Shoham, Y., R. Powers, and T. Grenager: 2003, 'Multi-Agent Reinforcement Learning: a critical survey'. Unpublished survey. `http://robotics.stanford.edu/~shoham/`.

Shoham, Y., R. Powers, and T. Grenager: 2004, 'On the Agenda(s) of Research on Multi-Agent Learning'. In: *2004 AAAI Fall Symposium in Artificial Multi-Agent Learning*.

Shoham, Y., R. Powers, and T. Grenager: 2006, 'If Multi-agent Learning is the Answer, What is the Questions?'.
Singh, S., M. Kearns, and Y. Mansour: 2000, 'Nash Convergence of Gradient Dynamics in General-Sum Games'. In: *UAI 16*.
Spall, J. C.: 2003, *Introduction to Stochastic Search and Optimization: Estimation, Simulation and Control*. Hoboken, New Jersey: John Wiley & Sons.
Tesauro, G.: 2003, 'Extending Q-Learning to General Adaptive Multi-Agent Systems'. In: *NIPS 16*.
Vu, T., R. Powers, and Y. Shoham: 2005, 'Learning Against Multiple Opponents'.
Walsh, W., D. Parkes, and R. Das: 2003, 'Choosing Samples to Compute Heuristic–Strategy Nash Equilibrium'. In: *AAMAS '03 Workshop on Agent–Mediated Electronic Commerce*.
Watkins, C. and P. Dayan: 1992, 'Q–Learning: Technical Note'. *Machine Learning* **8**, 279–292.

Appendix

A. Visualization

In this appendix, we describe the visualization interface that we built as part of our platform.

Figure 2 presented a pipeline diagram for running experiments on the platform. The visualization phase was not shown in that diagram, but the process for visualizing results is now given in Figure 15. This process allows the user to visualize the results of metrics over a set of agents, generators, instances and iterations. The user is able to make these selections through the GUI displayed in Figure 16.

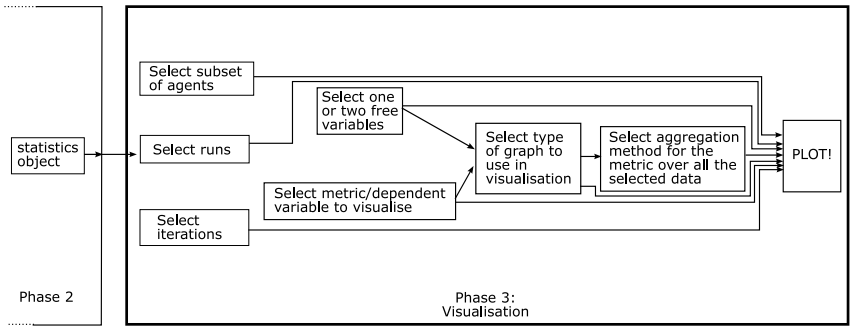

Figure 15. The pipeline for visualizing the results of an experiment

The first step in the visualization process is to select which set of agents we want to visualize. When we select agents A and B, we are pulling the data from A as the row player and B as the column player and vice versa. We can also choose to limit some of the agents to be row players and others

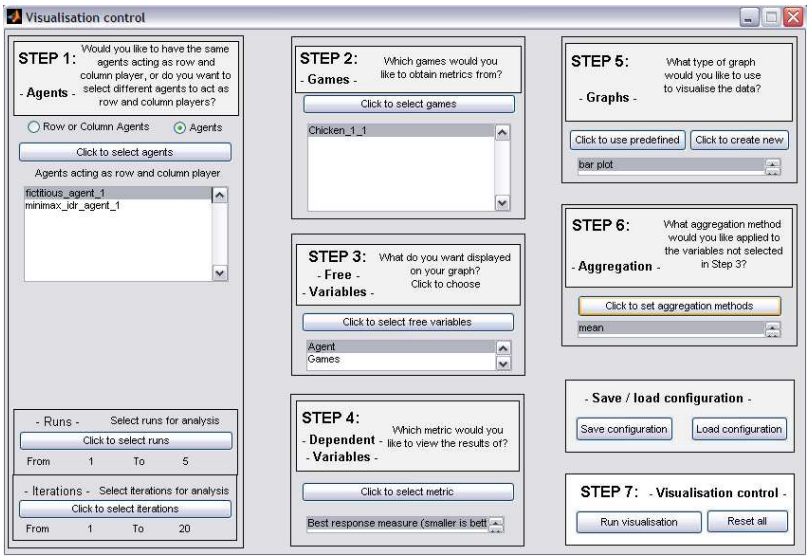

Figure 16. The graphical user interface for controlling and setting the visualization process. This allows a user to specify what data they want to view, which metrics, how they should be combined and how to visualize these statistics.

to be column players. In the upper left block of Figure 16, the fictitious play and minimax-Q-IDR agents have been selected. When a user clicks to select agents, a popup window such as the one shown in Figure 17 is displayed. This lists all of the agents that were used in the experiment and those agents that have been selected for visualization. Users are able to move agents between the two listboxes. Once users are satisfied with their choices, the new selection is shown in the relevant box in the interface shown in Figure 16.

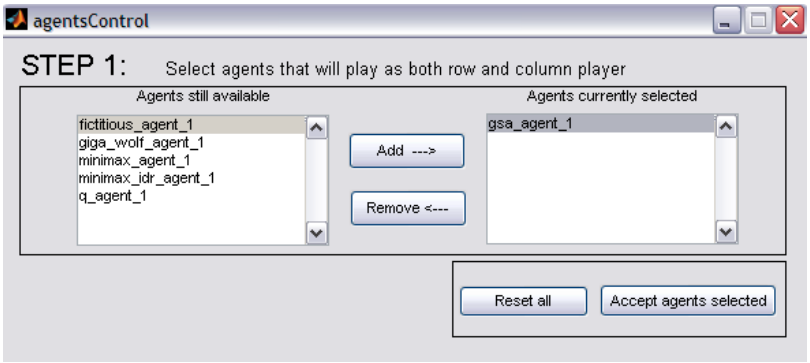

Figure 17. GUI for step 1 of the visualization process, selecting the agents

The next step in the process is to decide from which runs or instances of the game generators we want to view metrics. A user can thus decide to use all instances, or perhaps only one instance per generator. Each instance or run is composed of a number of iterations. We can similarly chose from which iterations of the selected instances we want to view metrics. This provides control down to single stage game interactions between agents.

Once the user has selected the agents, runs and iterations that will be used in the visualization process, the next step is to select the game generators. The generators that are currently selected are displayed in the top middle listbox in the GUI of Figure 16.

The third step is to select free variables. These are the variables whose values will be varied in the visualization. The possible free variables are *agent, agent, games, iterations* and *runs/instances*. *Agent* is repeated twice because it allows metrics to be visualized over either pairs of agents or individual agents. The free variables form a four dimensional table. The dimensions are defined by (*algorithm pairing*, *game*, *run/instance*, *iteration*), which correspond to the possible free variables. Each cell in the table contains the results of all metrics recorded for that pairing of agents playing on a specific instance of a generator in one iteration/stage game. Selecting one or two free variables means that we want to project the table down to one or two dimensions. The dimensions of the table that are kept correspond to the free variables that are selected. The dimensions corresponding to the free variables that are not selected are the dimensions that are aggregated across in the projection. For example, if we wanted to view the performance of specific agents on specific games, then the free variables would be *agent* and *games*. The graph would show results for each agent on each game, but aggregated over all opponents, instances and iterations. If we want to obtain a statistic of how every agent did, but want an aggregated result against all opponents, on all games and all instances and iterations, the free variable would then just be *agent*. To view the performance of each agent against each opponent, the free variables would be *agent* and *agent* and the graph would show results for each combination of agents, aggregated over all the games, instances and iterations.

The fourth step in the process is to select the dependent variable. This corresponds to the metric that we want to visualize in a plot. The metrics that are available are those that were recorded during the experiment that we are visualizing.

The fifth step in the visualization process is to select the visualization routine that will be used to present the metric. Currently, there is a standard line plot, bar plot, box plot and scatter plot implemented.

Once free variables have been selected, the 4D table containing our data is reduced in dimensionality. The dimensions which correspond to the free variables that were not selected are eliminated through an aggregation method. The final step in the visualization process is to select this aggregation method.

Our platform currently supports three methods for achieving this aggregation, *no aggregation*, *averaging* the data and *totalling/summing* the data. The *no aggregation* method is used in the box and scatter plots, where the plot itself provides a view of the entire data set. The line and bar plot use the *averaging* and *totalling* methods.

Putting this all together, if the selected free variable is *agent*, the dependent variable is *reward* and the aggregation method is *averaging*, then a single reward statistic is calculated for each agent. This corresponds to the average reward that each agent achieved against all opponents, on all of the selected game generators. If a user selected *agent* and *agent* as the free variables, *regret* as the dependent variable and *total* as the aggregation method, then the graph would display the total regret that each agent achieved against each opponent, over all game generators.