Heterogeneous Populations of Learning Agents in Minority Games

David Catteeuw
Computational Modeling Lab
Vrije Universiteit Brussel
Pleinlaan 2
1050 Brussels, Belgium
dcatteeu@vub.ac.be

Bernard Manderick
Computational Modeling Lab
Vrije Universiteit Brussel
Pleinlaan 2
1050 Brussels, Belgium
bmanderi@vub.ac.be

ABSTRACT
We study the combination of co-evolution and individual learning in minority games (MGs). Minority games are simple models of distributed resource allocation. They are repeated conflicting interest games involving a large number of agents. In most of the literature, learning algorithms and parameters are evaluated under self-play. In this article, we want to explore by means of an evolutionary algorithm (EA) whether agents that are free to choose or change their learning parameters can improve their individual welfare. Furthermore, we are interested to see whether an increase of the agents’ strategy space is beneficial to the entire population. I.e., can such agents use the available resources more efficiently or will the price of anarchy increase? Experiments show that the heterogeneous setting can achieve outcomes which are good from the viewpoint of the system, as well as for the individual users. The average of the evolved learning parameters are mostly reasonable values for the homogeneous setting. More importantly we show that algorithms which achieve better results in a homogeneous setting may be outcompeted when confronting other algorithms directly in a heterogeneous setting.

Categories and Subject Descriptors
I.2.6 [Artificial Intelligence]: Learning; I.2.11 [Artificial Intelligence]: Distributed Artificial Intelligence—Multiagent Systems; I.2.8 [Artificial Intelligence]: Problem Solving, Control Methods, and Search

General Terms
Algorithms, Performance, Economics

Keywords
minority game, congestion game, adaptation, reinforcement learning, co-evolution, evolutionary algorithm

1. INTRODUCTION
In the Minority Game [4], a large number of agents repeatedly and simultaneously choose one of two sides. Only agents on the minority side win. Hence, this is an anti-coordination game: an individual must do the opposite of the others in order to be successful. From the viewpoint of the group, the more agents on the minority side, the better.

The MG is a simple model of the stock market where you want to sell at the point where everyone else is buying, and vice versa: the best time to buy shares is when everyone is selling theirs. The MG also models other distributed resource allocation problems, such as grid computing [6] and network routing. In grid computing jobs are submitted to computational resources, without knowing the exact load of each computer, or the time needed to process the jobs already in its queue. Obviously, you benefit if you can submit your job to a machine which is less used. In network routing, long paths must be allocated to preserve a minimum bandwidth for streaming video. These decisions are made not knowing what actions other agents (routers and servers in the case of network routing) are currently taking or will take.

A slightly more general class of games are Congestion Games [11]. Just as in Congestion Games, the cost of using a resource depends on the number of players using that resource at the same time. Usually the cost increases according to some simple function. In this article we will concentrate on MGs with two resources and consider three different cost functions, see Section 2. The main difference between MGs and CGs is that in Congestion Games, the actions of agents correspond to subsets of resources instead of single resources.

Apart from its usefulness as a model of many practical distributed resource allocation problems, the MG is also interesting as a test-case or benchmark game for testing learning algorithms, for several reasons. First, the MG is a repeated game, allowing the application of online update rules which include Learning Automata [9] and Q-Learning [13], both reinforcement learning algorithms [12]. Second, the MG is hard: it has no trivial solution. There is no best action, and the outcome of each interaction depends fully on the aggregate action of all players. Third, the number of players is high. Minority Games with hundreds of players are no exception. This is in contrast to most popular benchmark games in multiagent learning which are typically two-player games: prisoners’ dilemma, matching pennies, battle of the sexes, etc. Finally, many simple extensions have already been proposed in literature: different payoff functions [4], multiple resources [3, 7], agents exchanging information [10], etc. These MG variations model more realistic distributed resource allocation problems and allow to test different aspects of learning in a multiagent context. The details of the MGs we study here are discussed in Section 2.

Previously, people have mostly been concerned with homogeneous populations where all agents, competing in the same Minority Game, use the same learning algorithm and
parameters \([7, 6, 8, 3]\). Separate experiments are done for different values of the learning parameters and different learning algorithms. In the end, group welfare (also called ‘volatility’ in MG literature) is compared.

We argue it makes more sense to study heterogeneous populations and see whether group welfare can be maintained. A heterogeneous environment is a more natural setting. For example in the stock market the only limits on the traders’ strategies are their time, information and computational constraints. So, any selfish agent – as we assume stock traders and resource users normally are – will change his strategy if he believes he would benefit.

Note that, in this article, we do not study the effects of heterogeneous information as in \([5]\) where agents have different information available. Instead, agents vary in learning strategy. In this article we look at two simple learning schemes: Q-learning combined with \(\epsilon\)-greedy action selection and Learning Automata (LA). Both are discussed in Section 3. We choose these strategies for their simplicity and applicability to many online learning problems. Conveniently, both algorithms make use of two learning parameters in the range \([0, 1]\). We will vary these parameters between agents to create heterogeneous populations.

The experiments are divided into two parts. In the first part (Section 4.3) all agents use the same learning algorithm but may use different learning parameters. After each MG, agents go through a selection and reproduction process. Selection is based on individual fitness and ensures only well performing agents remain in the population. This evolutionary process is discussed in detail in Section 4.2. The experiments will show that good system performance is still achieved. We compare these results to some homogeneous settings. In the second part (Section 4.4), we mix agents with different learning algorithms. Here we show that the algorithm which achieved better results in homogeneous settings can still be outcompeted by a supposedly ‘weaker’ algorithm for MGs.

Before discussing these experiments, we must define the MG, more especially the variations we use and the learning strategies applied.

### 2. MINORITY GAME

When Challet and Zhang, inspired by the El Farol Bar problem \([1]\), defined the MG \([4]\), they also included a basic learning strategy. In this explanation, we leave that out. In Section 3 we will discuss the strategies we apply here.

The MG is played with a large but odd number of agents \(N = 2k + 1\) for some positive integer \(k\). Each agent \(i\) can choose between two possible actions \(a_i\), represented by 0 and 1. All agents select an action simultaneously without any explicit information on the others’ choice. The number of agents choosing action 0 at time \(t\) is denoted by \(#_0(t)\).

Contrary to the original definition of Challet and Zhang \([4]\) the agents do not have access to a public list of previous outcomes – the so-called history. The only information they can use are the payoffs they receive after each action. In this article we study three different payoff functions. For convenience, all payoff functions return values between \(-1\) and \(+1\). The binary payoff function awards \(r = +1\) to each agent in the minority and \(r = -1\) to those in the majority (Equation 1).

\[
r_i(t) = -\text{sgn}(\#_{a_i}(t) - N/2) \quad (1)
\]

Equation 2 is the linear equivalent of the one above. This reward function gives higher payoffs if the minority is smaller. This function gives the individuals more information about the achieved outcome.

\[
r_i(t) = -\frac{2}{N} \left(\#_{a_i}(t) - N/2\right) \quad (2)
\]

The third payoff function 3 embodies what seems socially fair: socially better outcomes are awarded higher payoffs and everyone always receives the same payoff. This payoff scheme should push the group as a whole towards achieving globally good results.

\[
r_i(t) = 1 - \frac{4}{N} \left|\#_{a_i}(t) - N/2\right| \quad (3)
\]

The system performs efficiently when at every round as many agents as possible are on the minority side. Optimally, \(k\) agents choose action 0 and \(k + 1\) choose action 1, or vice versa, as the game is symmetric in the actions. Any reasonable agent strategy should result in a system where \(#_0(t)\) and \(#_1(t)\) fluctuate around \(N/2\) after some time.

The volatility \(V\) measures the system's performance as the variance of \(#_1(t)\) around the mean \(N/2\) normalized to the number of agents (see Equation 4).

\[
V = \frac{1}{TN} \sum_{t=1}^{T} \left(#_1(t) - N/2\right)^2 \quad (4)
\]

A good system performance is one that is lower than 1/4, which is what ‘non-learning’ or ‘randomly choosing’ agents achieve, see Table 1. The same table shows optimal bounds for \(V\), \(W\) and \(F\). These bounds are achieved when, at each round of the MG, the maximum number of agents are in the minority – \(k\) out of \(N = 2k + 1\) – and all agents perform equally well. Results in Table 1 hold for the binary payoff function. The optimal bound for \(V\) and the \(V\) of the ‘non-learning’ agents are independent of any payoff function.

For our purpose, we also define the welfare or fitness. The welfare \(w_i\) of agent \(i\) is the sum of his payoffs \(r_i(t)\) he collected over time. We denote the average fitness of a population by \(W\). Note that \(W\) is certainly negative for the binary and linear payoff function (Equation 1 and 2 respectively). At most \(k\) out of \(2k + 1\) agents can get a positive payoff. Note the close relationship of average fitness \(W\) and volatility \(V\). High volatility will coincide with low average fitness and vice versa.

Finally, we define some notion of fairness \(F\). For our purpose, we would like high fairness to indicate that all agents have very similar fitness. Low fairness should indicate there is many difference between the welfare of the individuals. We choose to define the fairness \(F\) as the inverse of the standard deviation of the individual fitness \(w_i\): 

\[
F = \sqrt{\frac{N}{\sum_{i=1}^{N} (w_i - W)^2}} \quad (5)
\]

This is clearly related to the spread in fitness between individuals. The inverse makes sure that higher values correspond to fairer outcomes. By no means, we claim that this is the only possible definition of fairness, but this one will suffice our purpose here.

Next, we discuss the two reinforcement learning schemes we apply here.
3. REINFORCEMENT LEARNING

Reinforcement Learning (RL) agents solve problems using trial and error. Unlike in supervised learning, agents are not told which actions or decisions are best. Instead, agents receive a reward – which is scalar – after each action taken, as an indication of the quality of their choice. They also have (some) information about the state of their environment. The goal of the agent is to find an optimal policy. A policy is a mapping from each state of the environment to a probability distribution over the possible actions in that state. The agent maximizes his total expected reward if he takes all actions according to an optimal policy.

Since agents are not told which action to take, they should balance exploration and exploitation. While the agent wants to exploit his knowledge of the environment most of the time in order to maximize his rewards, once in a while the agent should explore a new action in order to discover actions that are better than the best one found so far.

See the book of Sutton and Barto [12] for an overview of RL. The two popular reinforcement learning algorithms which we apply here are Q-learning and Learning Automata.

3.1 Q-Learning

One well-known RL-algorithm is Q-learning [13]. It maps each state-action pair \((s, a)\) to the total expected reward if the agent applies action \(a\) in state \(s\). One can prove that Q-learning will find the optimal policy, i.e. the Q-values converge to the true total expected reward provided that the environment is stationary [13]. Unfortunately, in a multiagent setting the environment is non-stationary due to the presence of other agents who also learn and hence change their behavior. Whereas in a stationary environment exploration can be ignored once enough information has been collected, in a non-stationary environment the agent has to continue exploring in order to track changes in the environment.

In order to apply Q-learning to a MG, the rewards, state- and action-space must be defined. The actions simply refer to the resources of the game (0 and 1). The rewards \(r \in [-1, +1]\) are defined by any of the payoff functions in Section 2. Agents receive high rewards when they make good choices, lower rewards result from bad choices. The agents do not receive any extra information from the environment apart from their rewards. This means agents cannot discriminate between different states of the environment. As a consequence, agents have a Q-value for each available action and use the ‘stateless’ Q-learning update rule whenever they receive a reward \(r\) after taking action \(a\):

\[
Q_a \leftarrow Q_a + \alpha (r - Q_a)
\]

In the above update rule, \(\alpha \in [0, 1]\) represents the learning rate. All Q-values are initially set to 0.

The update rule in Equation 6 only tells the agent how to exploit but not how to explore. Therefore we need an exploration strategy or action-selection rule like \(\epsilon\)-greedy or softmax. Here we choose the \(\epsilon\)-greedy action-selection strategy (where \(\epsilon\) is small). The agent selects with probability \(\epsilon\) an action at random according to a uniform distribution and with probability \(1 - \epsilon\) he selects the action with the highest Q-value (braking ties randomly).

1In the extreme case of \(\alpha = 0\), nothing is ever learned. In the other extreme case where \(\alpha = 1\), Q-values only reflect the last reward for the corresponding action.

<table>
<thead>
<tr>
<th>experiment</th>
<th>(V)</th>
<th>(W)</th>
<th>(F)</th>
</tr>
</thead>
<tbody>
<tr>
<td>optimal bound</td>
<td>(0.5^*/N)</td>
<td>(-1/N)</td>
<td>(1/0)</td>
</tr>
<tr>
<td>non-learning/random</td>
<td>(0.25)</td>
<td>(-0.046)</td>
<td>(31.74)</td>
</tr>
<tr>
<td>QL, (\alpha = 0.0)</td>
<td>(0.0011)</td>
<td>(0.00012)</td>
<td>(0.23)</td>
</tr>
<tr>
<td>QL, (\alpha = 0.1), (\epsilon = 0.01)</td>
<td>(0.01)</td>
<td>(-0.0084)</td>
<td>(45.5)</td>
</tr>
</tbody>
</table>

Table 1: Some benchmarks with averages over 100 samples and standard deviations (preceded by \(\pm\)). For welfare \(W\) and fairness \(F\) higher is better, for volatility \(V\) lower is better. Moreover, we consider any volatility lower than 0.25 (see non-learning agents) as good.

3.2 Learning Automata

Learning automata [9] directly manipulate their policy, which is a probability distribution over the actions. This is contrary to Q-learning which updates Q-values and uses those in combination with an action-selection strategy to determine a policy.

Each time the agent takes an action \(a\) and receives a corresponding reward \(r\), he updates the probability distribution \(p\) over all actions according to following rules:

\[
p_a \leftarrow p_a + \alpha (1 - p_a) - \beta (1 - r) p_a, \tag{7}
\]

\[
p_j \leftarrow p_j - \alpha p_j + \beta (1 - r) \left(\frac{1}{n-1} - p_j\right) \quad \forall j \neq a. \tag{8}
\]

Here, \(n\) is the number of actions. The parameters \(\alpha\) and \(\beta\) (both in the range \([0, 1]\)) are the reward and penalty learning rate. In the literature, the values of \(\alpha\) and \(\beta\) are mostly restricted. For example, the scheme where \(\beta = 0\) is called Linear Reward Inaction and is often used. Here we allow any combination of \(\alpha\) and \(\beta\). The reward \(r\) is assumed to be in the range \([0, 1]\). Since our payoff functions return values in the range \([-1, +1]\), learning automata will first rescale the payoffs to \([0, 1]\).

4. EXPERIMENTS AND RESULTS

In the reported experiments all minority games are played with \(N = 301\) agents. They are run for 10000 rounds. Fitness, volatility and other performance measures are taken over the last 1000 rounds. The first 9000 rounds allow the system to stabilize regardless of initial conditions.

4.1 Incentive to deviate

In a first experiment we check the hypothesis that an individual agent can improve his fitness by applying a strategy different from the other agents.

As an example: when all Q-learning agents use \(\alpha = 0.1\) and \(\epsilon = 0.01\) in a MG with the binary payoff function, system and individual performance is already very good. Both volatility \(V\) and average fitness \(W\) of these agents are much better than those for non-learning agents, see Table 1 for the figures. If one agent from that population uses a different \(\alpha\) (0.0 < \(\alpha\) < 0.1), he can achieve a fitness which is even better than the average of the population. We see similar results when varying the exploration rate \(\epsilon\) of one agent. For \(\epsilon\) smaller than what the group uses (\(\epsilon < 0.01\)), an individual can get better fitness than the group’s average.

Clearly there exists an incentive for an individual to change
its learning parameters and we would like to know what happens if all agents are given the freedom to change. Therefore we set up a simple evolutionary algorithm (EA). Our hypothesis is that selfishness will steer the agents toward learning parameters which not only yield good individual payoff but also good system performance such that the entire population can benefit.

4.2 Evolutionary Algorithm

An evolutionary algorithm is inspired by natural selection in biology. Species which are more fit (i.e., more adapted to the environment) are able to reproduce faster. Less fit species will decrease in numbers and may eventually go extinct. The fitness of an individual is largely determined by its genes. And it is the information in these genes that an individual passes on to its offspring. The overall effect of natural selection is that genes with positive effects on individuals will pass on to the next generation, genes with negative effects may not, since they cause the individual to reproduce less or even die before reproducing. The primary source of diversity in genes of a population is mutation. Any mutation is random and its effect depends on the environment. It is natural selection that will steer the evolution in a particular direction.

EAs can also serve as a model of social learning. In human or animal societies, individuals may learn by copying the behavior of ‘better performing’ or higher regarded individuals. In such an imitation process very fit behavior will propagate faster through the population and will be used by more individuals as opposed to unfit behavior.

The basic evolution strategy [2] called \((\mu + \lambda)\)-ES is the following:

1. At the start, generate \(N\) individuals at random.
2. Determine the fitness of all \(N\) individuals.
3. Select the \(\mu < N\) best individuals as parents for the next generation.
4. Pick \(\lambda = N - \mu\) individuals from the \(\mu\) parents at random with replacement and weighed by their fitness. Create for each parent one offspring. The offspring is a mutation of the parent. The new generation consists of the \(\mu\) best individuals of the previous generation and the \(\lambda\) offspring.
5. Repeat step 2 to 4 until a maximum number of generations has been reached.

In our case an individual will be fully determined by its learning parameters \(-\alpha\) and \(\varepsilon\) for Q-learning, \(\alpha\) and \(\beta\) for learning automata. Both are real-valued numbers in the range \([0, 1]\). The initial population’s parameters are chosen at random according to a uniform distribution over \([0, 1]\). The fitness of an individual will be determined by a MG among the entire population: \(N = 301\) agents picking one of two actions repeatedly for 10000 rounds. The more an individual chose the minority action during the last 1000 rounds, the higher his fitness.

We create \(\lambda = 7\) offspring at each generation. Each offspring inherits the learning parameters of his parent with some small Gaussian noise added. The noise is drawn at random from a normal distribution with mean \(\mu = 0.0\) and variance \(\sigma^2\) which starts at 0.1\(^2\) and slowly decreases to 0.0 from generation to generation. The noise is drawn independently for both parameters. The parameter space is considered to be torus shaped, i.e., if a mutation creates a value outside \([0, 1]\) it is wrapped to the other side. In another experiment we clamped such mutations to the bounds 0.0 and 1.0, causing no qualitative differences.

At the start of every MG, the Q-values, fitness, etc. are all reset to their initial value. Only the learning parameters are passed on from one generation to the next.

4.3 Evolving Learning Parameters

In these experiments, all agents use the same learning algorithm but have different learning parameters which evolve over time. We look at the 3 different payoff functions and compare the evolutionary setting with the homogeneous one. The learning parameters used in the homogeneous setting are taken from what is evolved on average after 1000 generations.

For the binary payoff function, evolution always reaches a system of very low volatility, both for Q-learning and learning automata (see Table 2). Very similar learning parameters emerge each time and these yield almost equally good results when applied in the homogeneous setting, see Table 3. Only the fairness among learning automata seems to be a concern. We will come back to this in Section 4.4.

For the linear payoff function, only learning automata seem to be able to evolve reasonably low volatility (see Table 2). This is unexpected, the linear payoff function holds more information for the agents, and hence should make it easier to reach good resource usage. Again the application of the average evolved parameters in a homogeneous settings yields similar results. Except for Q-learning, there the results are even worse. For an other combination of parameters (\(\alpha = 0.1\) and \(\varepsilon = 0.01\)), better but far from satisfying results could be achieved.

The ‘social’ payoff function (Equation 3) yields particularly bad results. The learning parameters seem to make random walks while evolving and nothing is ever learned. This makes sense. As all agents perform equally well, natural selection is unable to push the group in any particular direction, even though the payoff function still awards higher payoffs for better distributions. For the homogeneous

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Binary Payoff</th>
<th>Linear Payoff</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\alpha)</td>
<td>(0.050)</td>
<td>(0.076)</td>
</tr>
<tr>
<td>(\varepsilon/\beta)</td>
<td>(0.0080)</td>
<td>(0.013)</td>
</tr>
<tr>
<td>(V)</td>
<td>(0.0045)</td>
<td>(0.013)</td>
</tr>
<tr>
<td>(W)</td>
<td>(-0.0050)</td>
<td>(-0.0041)</td>
</tr>
<tr>
<td>(F)</td>
<td>(13.08)</td>
<td>(504.66)</td>
</tr>
</tbody>
</table>

Table 2: Results after 1000 generation of evolving learning parameters. For each figure, the average over 30 samples is shown and below it the standard deviation preceded by ±. For both payoff functions the best figures are indicated in bold. For more details see the text.
Table 3: Results for a homogeneous setting where learning parameters are fixed to what was evolved during the experiments reported in Section 4.3, see Table 2. For each figure, the average over 30 samples is shown and below it the standard deviation preceded by ±. For both payoff functions the best figures are indicated in bold.

<table>
<thead>
<tr>
<th></th>
<th>Binary Payoff Function</th>
<th>Linear Payoff Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>α</td>
<td>QL 0.050 LA 0.91</td>
<td>QL 0.076 LA 0.89</td>
</tr>
<tr>
<td>ε/β</td>
<td>0.0021 0.00011</td>
<td>0.00092 0.052</td>
</tr>
<tr>
<td>V</td>
<td>0.0050 0.0025</td>
<td>5.42 0.10</td>
</tr>
<tr>
<td>W</td>
<td>±0.0021 ±0.0012</td>
<td>±4.23 ±0.0047</td>
</tr>
<tr>
<td></td>
<td>±0.0051 ±0.0034</td>
<td>±0.072 ±0.0013</td>
</tr>
<tr>
<td></td>
<td>±0.00053 ±0.000021</td>
<td>±0.056 ±0.000064</td>
</tr>
<tr>
<td>F</td>
<td>21 1.24</td>
<td>94.83 591.69</td>
</tr>
<tr>
<td>±12.26 ±1.52</td>
<td>±89.13 ±61.86</td>
<td></td>
</tr>
</tbody>
</table>

Figure 1: Evolution of the number of Q-learning agents for binary and linear payoff functions.

Figure 2: Evolution of the Q-learning agents’ learning parameters for linear payoff functions.

Using the binary payoff function, the LA are outcompeted from the start and steadily disappear from the population (solid line in Figure 1). The low fairness in the experiments from Section 4.3 already showed that the same LA automata are always on the winning side and others always on the losing side. The parameters to which they evolve (α high and β almost 0) is preventing them to be adaptive. The LA scheme where β = 0.0, also called Linear Reward Inaction, is known for its convergence towards pure strategies (the probability for one particular action becomes 1.0 and that action gets selected all the time from then on).

In the experiments from Section 4.3 with the linear payoff function, LA evolved a higher β (0.052) and thus are expected to be more responsive. Until generation 50 we see that many Q-learning agents get replaced by LA. Later, the remaining Q-learning agents seem to have evolved efficient learning parameters (see Figure 2) and yet again the LA are outcompeted. Once the LA have gone extinct the Q-learning agents evolve again to the same parameters of Section 4.3 and actually, the system performance deteriorates.

5. CONCLUSIONS

We first showed there is an incentive for an agent to individually change strategy and to use a different strategy from that of the group even if the system is already in an efficient regime.

When all agents can change their learning parameters they may still reach efficient resource usage. Moreover the evolved parameters work well in the homogeneous setting. In cases were efficient schemes were not evolved, it was also hard (or impossible) to ‘manually’ discover parameters that give reasonable results for the homogeneous setting.

We note that the payoff function has a huge impact on the efficiency of natural selection. Differentiating between ‘good’ and ‘bad’ choices or behavior is definitely necessary. The ‘social’ payoff function which rewards the entire group for reaching good distributions over the resources, does not distinguish between individuals and hence is useless for natural selection. Surprisingly, giving agents more information...
is not necessarily an advantage for the group. The linear payoff function holds more information for the agents. This results in Q-learning evolving a very low exploration probability ($\varepsilon < 0.001$). These agents are effectively exploiting too much to achieve low volatility.

From the last experiments (Section 4.4) we conclude that comparing learning algorithms under self-play may yield very different results than comparing under direct competition. These kinds of parameter learning and algorithm comparison techniques are quite interesting for selecting learning algorithms and parameters for agents in uncooperative games. We believe these techniques can easily be extended to MGs with more resources and consequently CGs.

6. REFERENCES


