The Minority Game: the emergence of cooperation from selfishness

Giovanni Varutti

February 7, 2024

Contents

1 Introduction to the problem

The aim of this work is to analyze, through simulations, the minority game (MG) model, that is a generic model of competing adaptive agents in an idealized situation. To do so, this model is viewed as a statistical system of many agents, in order to study possible collective and competitive phenomena.

The classical minority game model consists of N (odd) agents. At each time step of the game, each of the N agents takes a binary action $a_i(t) \in \{\pm 1\}$ with $i = 1, \ldots, N$. Drawing a parallelism with the financial world, an agent can either buy goods in the market ($a_i(t) = 1$) or sell them ($a_i(t) =$ -1). The agents who take the minority action win the game, whereas the majority looses. After each time step, the total action of the population is calculated with

$$A(t) = \sum_{i=1}^{N} a_i(t) \qquad , \tag{1}$$

and a payoff $-a_i(t)g[A(t)]$ is given to each agent, with g an odd function of $A(t)$. The simplest choice is $g(x) = \text{sgn}(x)$, with which each winning agent gets one point, whereas the others loose one point. However, in this work, other functions have been used, for example the linear function $g(x) = x$. The way agents choose their action $a_i(t)$ is by inductive reasoning. In particular, agents have a limited analyzing power (a limited memory), and they can only retain the last m winning groups (m is then the memory of each agent) and base their decision on these last m bits of system's history. Agents also have a set of $s \geq 1$ strategies. A strategy is just a mapping from the sequence of the last m history bits to the action of the agent, i.e. a table which tells the agent what to do as a function of the input signal (the last m winning groups). In the code implementation, a strategy has been represented by a 2^m-dimensional array r_i^p, whose components are the outputs of the strategy p of agent i. The prediction of a strategy at certain time step t is given by the $\mu(t) \in \{1, ..., 2^m\}$ component, where $\mu(t)$ is the binary representation of the last m winning groups of the system's history. Then, $\mu(t)$ can be seen as the state of an agent, at certain time step t, depending on his memory. Since there are 2^m possible inputs for each strategy, but only 2 possible outputs, the total number of possible strategies for a given value m of the memory is 2^{2^m}. At the beginning of the game each agent is given a set of s strategies, randomly drawn from the total 2^{2^m} possible strategies. All the s strategies of an agent collect **virtual** payoff depending if they would have won the game or not, given the m past bits, and the actual outcome of the game. This payoff records the merit of a strategy as if it were used each time. Among all the available strategies, agents use the one having the highest score, so the highest ccumulated payoff, for its action. Agents get the real payoff only if the strategy used happens to win the game. After each game step, the system's signal is updated adding the last winning group, that is $W(t) = -\text{sgn}[A(t)]$, to the sequence of history bits, and this information is released to all the agents. Then the next state can be computed with the formula:

$$\mu(t+1) = [2\mu(t) + (W(t) - 1)/2] \mod P \qquad , \tag{2}$$

with $P = 2^m$.

2 Technical details of code implementation

The whole software is written using both *FORTRAN90* and *python*. In particular, there are three different implementations:

- A *python* object-oriented implementation
- A *FORTRAN90* procedural implementation, composed by a single module with all the needed subroutines, and the *main.f90* file to perform the simulations. In this case there are no objects, and all the strategies (for all agents) are stored in a unique 3-dimensional array of size $(N, s, 2**m)$ (mapping the last m bits of the system's signal into the variable μ). Also the strategies' scores are stored in a 2-dimensional array of size (N, s).
- A *FORTRAN90* object-oriented implementation, composed by several modules, one for each object class. This last implementation is the most suitable one, because it is really general, and allows to simulate also eterogeneous configurations, without changing the structure of the code. Moreover, it offers the higer speed of a compiled language, with respect to the *python* implementation, and also an easy way to integrate the code with *python*, through the *f2py* library. Of course, this is the implementation which required more programming effort.

It is also present a *python* notebook file, used to make plots and analyze data. Moreover, *openMP* was used for parallelizing some loops and other parallel tasks. The code and all the related material can be found inside this *GitHub* repository.

3 Statistical analysis of the classical Minority Game

Firstly, we can show the time behavior of the total action $A(t)$, for a certain number of time steps, and with a payoff function $g(x) = \text{sgn}(x)$.

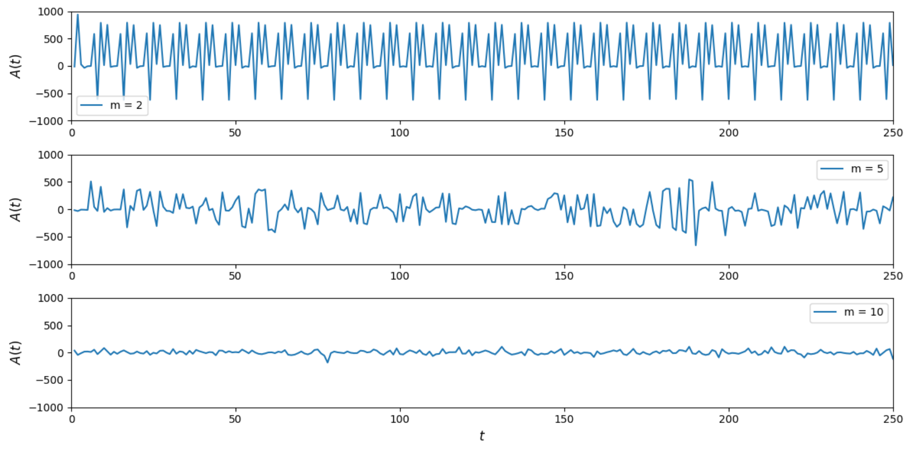

Figure 1: Temporal behaviour of $A(t)$, for a system with $N = 1001$ agents, $s = 5$ strategies, and different memory values.

As one may expect, since the game is symmetrical in buying and selling, the temporal signal of $A(t)$ fluctuates around zero, and we can verify that $\langle A(t) \rangle = 0$. Relevant statements can already be made looking at this plot. We can notice that large fluctuations imply small winning minorities, and so a large waste of the available resources (the winning places), since still more players could have taken the winning side. On the other hand, smaller fluctuations imply larger winning minorities, and a more efficient usage of available resources. In general, this would require coordination and cooperation among agents, which are not built-in features of the population, since agents are selfish by construction. We can see that agents with larger brains (i.e. larger m) cope with each other better. The fluctuations are indeed in decreasing order for ever increasingly intelligent populations (i.e. with $m = 2, 5, 10$). Remarkable is that, for small values of m (such as $m = 2$), a periodic behavior is observed. This behavior can be understood looking at the formula $\mu(t+1) = [2\mu(t) + (W(t) - 1)/2] \mod P$, used to update the variable μ. Smaller values of m imply smaller values of P, and then smaller periods. Of course, as already stated, the game is symmetrical for the actions of buying and selling. This can be experimentally observed plotting the histogram of all values of the total action A:

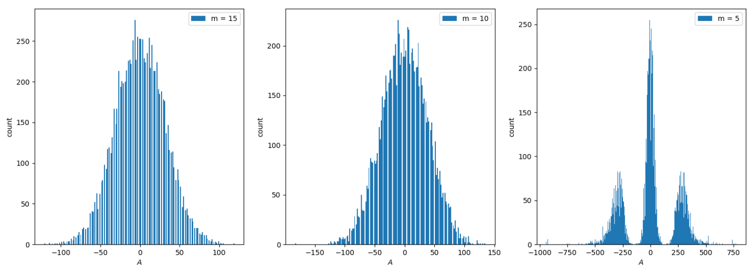

Figure 2: Histogram of A, for a system with $N = 1001$ agents, $s = 5$ strategies, and different memory values.

All the previous plots are indeed symmetrical, and this confirms the fact that $\langle A(t) \rangle = 0$. It is remarkable that, for small values of m, for example $m = 5$, the histogram shows two smaller side peaks. Whereas, in the other two cases, with $m = 10$ and $m = 15$, the distribution is characterized by a single central peak for $A = 0$. Repeting the simulation for different values of m (and always for $N = 1001$ agents), it was found that, for $m \geq 10$ the histogram has only one central peak, and for $1 \leq m < 10$ the distribution has also side peaks. This values of m are not universal, and change with the number of agents N in the population. This suggests the existence of a transition-phase point, which depends on N and m. The reason of this behavior will be clearer with the further analysis.

The advantage of the larger brain (memory) sizes over the smaller ones can be better appreciated performing some simulations with a mixed population of N agents, with different memory values ($m = 1, ..., 10$) and equal number of strategies. We want to force unequally memory-equiped agents to play together, and see that the agents with less memory are exploited by the agents with bigger memoriy size. We can plot the average number of wins per time step (for each agent) after a large number of runs ($\sim 10^4$) of the game.

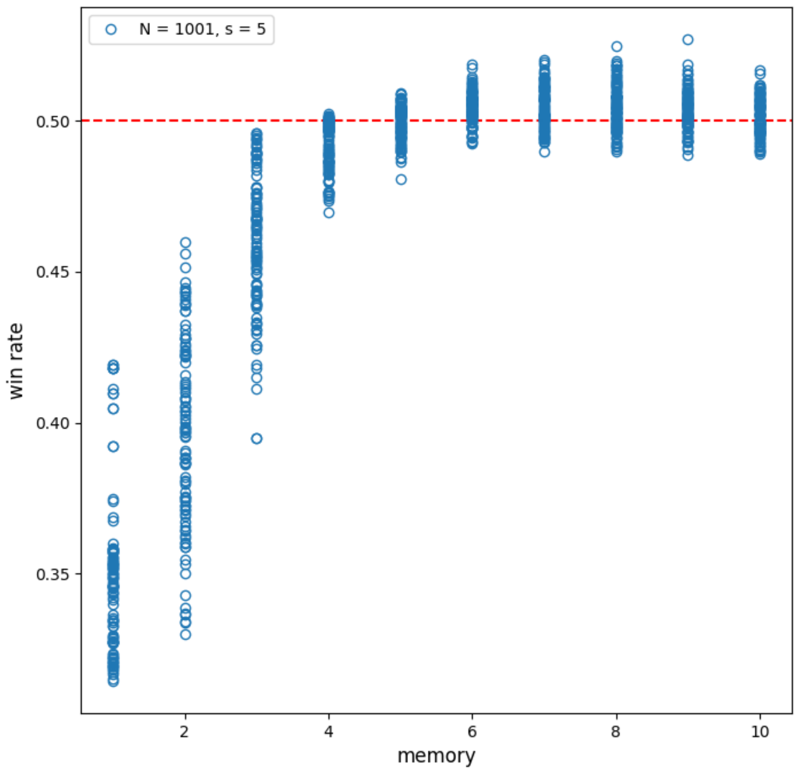

Figure 3: Win rate for each agent in a mixed population of agents with different memory sizes, and $s = 5$ strategies. There are $N = 1001$ agents in total equally divided in the memory categories.

We can see that, as the memory of a group of agents increases, the agents of that group win more, on average, and the spread between the best and the worse agent is smaller. Note that, above a certain size ($m = 6$), the average performance of a population appears to saturate, and further increasing the memory does not seem to improve the win rate. This is due to the limited resources available in the game (the winning places are always $< N/2$). We can also reach the same conclusions plotting the number of winners rate for different populations with the **same** parameters.

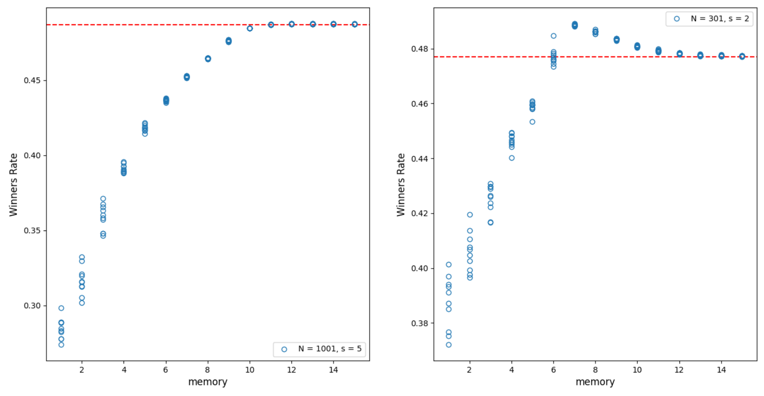

Figure 4: Average rate of winning agents (among all agents), across 10^5 runs, for different populations with the same parameters. On the left: $N = 1001, s = 5$. On the right: $N = 301, s = 2$.

Inspired by the question of the phase-transition point, emerged in the previous analysis, we want to study the fluctuations of $A(t)$ around its mean value. These fluctuations can be obtained computing the variance $\sigma^2 = \overline{\langle[A(t) - \langle A(t)\rangle]^2\rangle}$, where $\langle ... \rangle$ is a time average for long times and $\overline{...}$ is an average over possible realizations of the initial random strategies. As mentioned before, σ is related to the typical size of the winning group, so the smaller σ, the more winners are in the game. Therefore, the variance represents the volatility or the global efficiency. In the following figure, the values $\overline{\langle A(t)\rangle}/N$ (mean value of A per agent) and $\sigma * *2/N$ (variance of A per agent) are plotted in function of the control parameter $\alpha = 2^m/N$. All the averages have been taken over 10^5 time steps, and over 10 different realizations of the population.

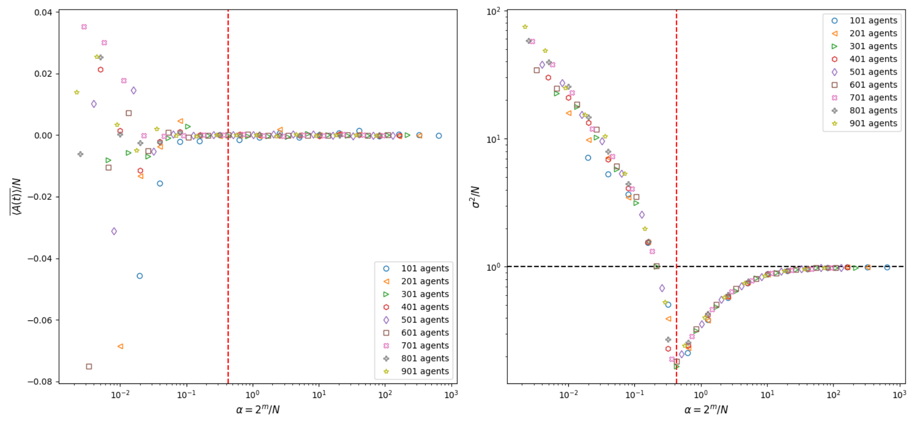

Figure 5: Behaviour of μ and σ^2 of $A(t)$ in function of the control parameter $\alpha = 2^m/N$.

This figure shows a really remarkable behavior, and the clear existence of a transition-phase point. In particular, we can see that, for large values of α (that is the termodynamic limit with $\alpha \to \infty$), σ^2/N approaches the value for the random choice game, $\sigma^2/N = 1$, i.e., the game in which each agent randomly chooses his action with equal probabilities at each time step. At low values of α, the average value of σ^2 is very large, and thus the size of the losing group is much larger than $N/2$, and this means large fluctuations and a waste of global gain. This region is then globally inefficient. At intermediate values of α, the variance is less than the random case, and we can individuate a minimum value for $\alpha = \alpha_c \approx 0.425$. This critical value represents the transition-phase point. In this region, the size of the losing group (and thus of the winning one) is close to $N/2$, and this region is then globally efficient. The fact that σ^2 gets below 1 for a given interval of values suggests that agents cooperate in order to reach a state in which less resources are globally wasted, which is contrary to their selfish intrinsic nature. This behaviour can be understood also thinking that, for $\alpha << 1 \implies N >> 2^m$, the number of agents in the population is grater then the strategies space, and so groups of agents with the same set of similar strategies tend to emerge. Those agents end up using their same best strategy and become a crowd, since they react to the available information in the same way. We can perfom the same statistical analysis for the number of winners variable:

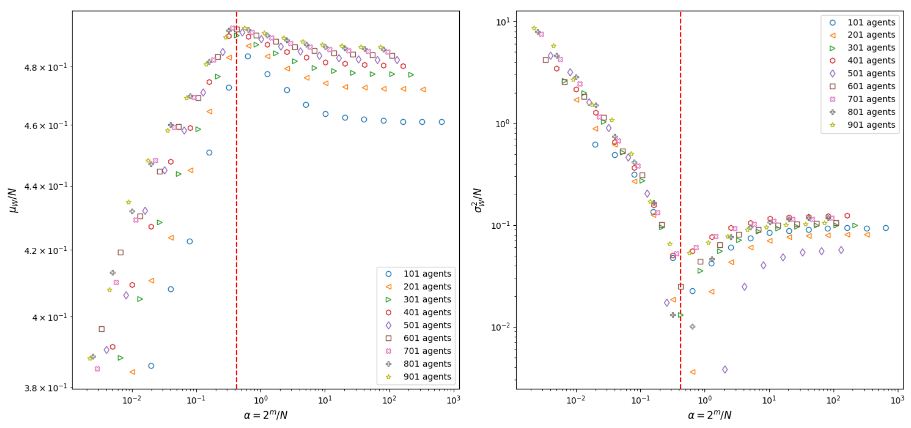

Figure 6: Behaviour of μ and σ^2 of the number of winners, in function of the control parameter $\alpha = 2^m/N$.

While the variance has a similar behavior as before, the mean value of the number of winners reaches a maximum for $\alpha = \alpha_c \approx 0.425$, as expected and discussed. After the last analysis, we can better understand the different behaviors of the histograms viewed before. In fact, for a population of $N = 1001$ agents, we can estimate the critical value of the memory: $\alpha_c = 2^{m_c}/N \implies m_c \approx \log_2(\alpha_c N) = \log_2(0.425 * 1001) \approx 8.73$. Taking the nearest integer we obtain $m_c = 9$, that is exactly the critical transition value emerged from the histograms simulations.

Now we can analyze the behavior of agents in function of the number s of alternative strategies. In the following figure, it is shown the number of winners rate (with time) for various populations with the same parameters, varying each time the number of available strategies.

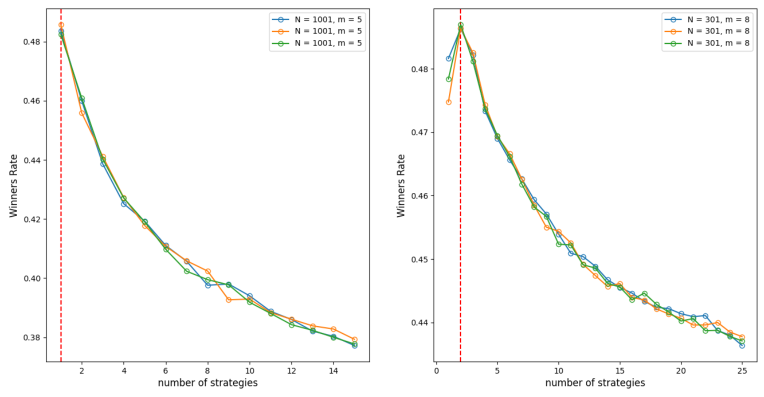

Figure 7: Average rate of winning agents, across 10^5 runs, for different populations with the same parameters. On the left: $N = 1001, m = 5$. On the right: $N = 301, m = 8$.

We can see that, in general, with increasing number of strategies, the agents tend to perform worse. What happens is that the players tend to switch strategies often and are more likely choose some outperforming strategy. At this point one can think that bad players are bad because of their bad strategies. In order to check whether there are really good and bad strategies, we can plot the win rate (and the score) of all the strategies in a certain population, and analyze how the distributions vary with the time steps. In the following figure, we can see three different distributions of the average (time) wins and scores for all the strategies in a population.

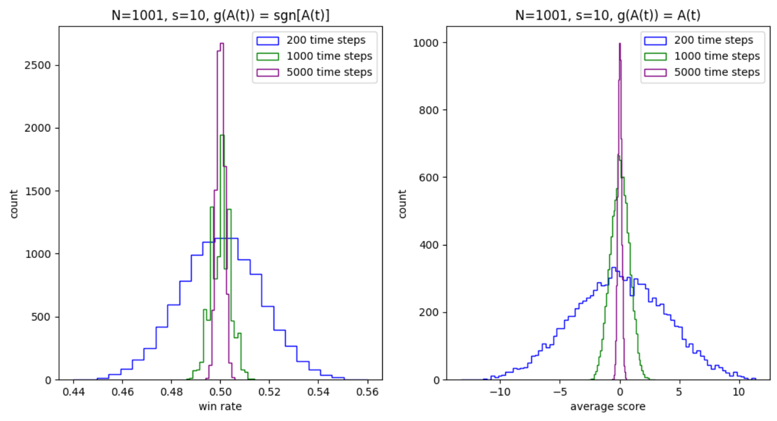

Figure 8: histograms of the win rate and the average score for all the strategies in a population with parameters $N = 1001, s = 10, m = 5$. The distributions are taken after 200, 1000 and 5000 time steps. On the left figure the win rate is displayed, using a payoff function $g(A) = sgn[A]$. On the right figure the average score is displayed, using a payoff function $g(A) = A$.

The longer the simulation time, the more concentrate is the distribution, indicating all the strategies are equivalent to each other, in the limit $t \to \infty$, since the distribution tends to be increasingly peaked at zero. So the bad players are bad because of their specific composition, and also for causal factors (they are unlucky).

4 Analysis of some evolutionary techniques

The aim of this section is to generalize the classical MG model, introducing some genetic approaches, in which the bad agents are regulary discarded from the game, and new agents are introduced to replace the eliminated ones. The first evolutionary technique we can introduce is the so called Darwin-ist selection. The worst agent is replaced by a new one after a finite time steps, and the new agent is a clone of the best agent: it inherits all the strategies but with corresponding virtual scores reset to zero. Also his personal score is reset to zero, of course. To keep a certain diversity in the population, we introduce a mutation probability in the cloning process. We allow one of the strategies of the best player to be replaced by a new random one. We expect this population is capable of learning, since it is reproductive. To see if this population is capable of learning, we can plot the total action $A(t)$ in function of the time step t.

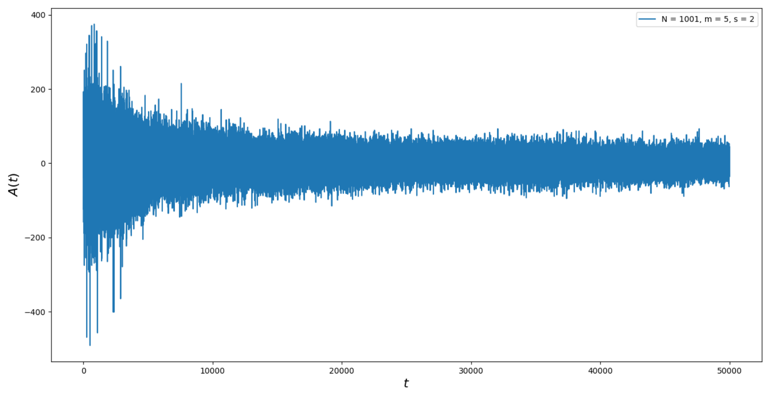

Figure 9: Temporal behaviour of $A(t)$ for a system with $N = 1001$ agents, $m = 5$ and $s = 2$.

As we can see the fluctuations reduce with time steps, and as we have already discussed, this implies a more efficient way to use the available resources, and more winning agents for each time step. The second evolutionary technique consists in letting the new-born agents to have, with a certain probability, one bit of memory more or less, with respect to the discarded agent. Moreover, we have to assure that this structural mutation is strictly neutral, so that the probability to have a new bigger memory is the same to the probability to have a smaller memory. The evolution mechanism has to decide which mutation is better (if there is one). We can simulate this evolutionary mechanism strating with a population with a little memory, $m = 2$, and see if agents with larger values of m are selected by the evolution. In the following figures we can see the results of these simulations:

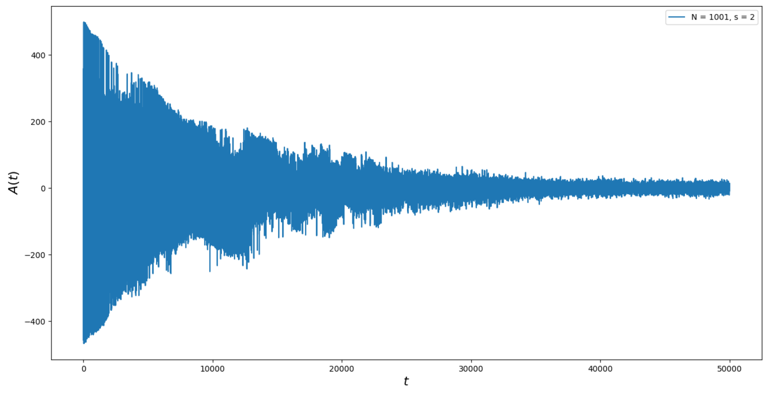

Figure 10: Temporal behaviour of $A(t)$, for a system with $N = 1001$ agents, $s = 2$ strategies.

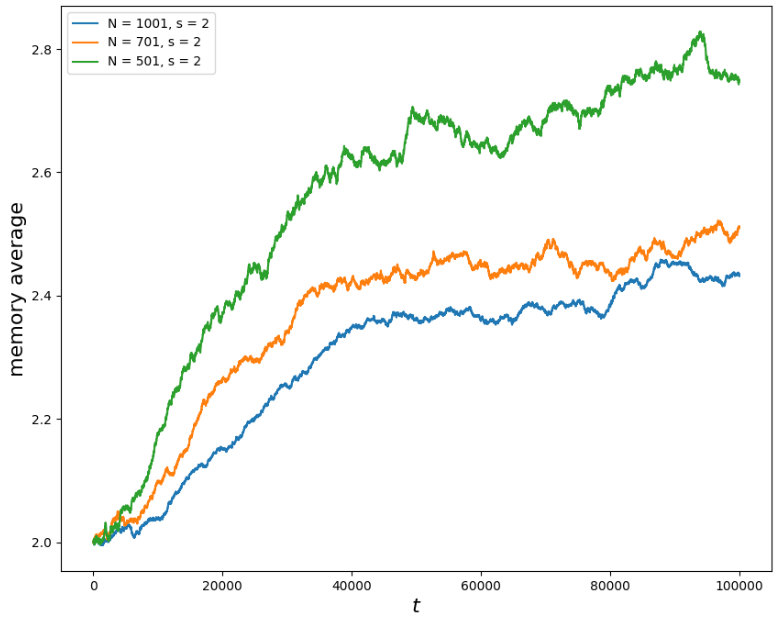

Figure 11: memory average of the population's agents, in function of the time step of the simulation. Three different curves are plotted, for populations with, respectively, $N = 1001, 701, 501$ agents.

The first figure shows a sharp learning behaviour, across the time steps, as discussed before. Even more interesting is the second picture, which shows that the memory average of the population increases with time steps. This means that the evolution selects the agents with higher values of m, and discard agents with little memory. This behavior is in fact expected, according to what we have discussed about the memory behavior in the previous section. However, such an evolution appears to saturate at a given level, that is not the same for all the populations. Moreover, we can notice that larger populations ($N = 1001$) have smaller learning rate with respect to smaller populations ($N = 501$).

In the **DATA** folder and in the **PLOTS** folder of the GitHub repository, all the produced material can be found. The GitHub repository contains also additional materials and analysis, for example the simulation of a more realistic market mechanism with the introduction of the producers and the speculator agents, that are not reported in this report.