SIMULATING MINORITY GAME WITH TuCSoN

Enrico Oliva
Mirko Viroli
Andrea Omicini
DEIS, ALMA MATER STUDIORUM—Università di Bologna
via Venezia 52, 47023 Cesena, Italy
E-mail:{enrico.oliva,mirko.viroli,andrea.omicini}@unibo.it

ABSTRACT

Minority Game is receiving an increasing interest because it models emergent properties of complex systems including rational entities, such as for instance the evolution of financial markets. As such, Minority Game provides for a simple yet stimulating scenario for system simulation.

In this paper, we aim at showing new perspectives in agent-based simulation by adopting a novel MAS meta-model based on agents and artifacts, and by applying it to Minority Game simulation. To this end, we adopt the TuCSoN infrastructure for agent coordination, and its logic-based tuple centre abstractions as artifact representatives. By implementing Minority Game over TuCSoN, we show some of the benefits of the artifact model in terms of flexibility and controllability of the simulation.

INTRODUCTION

Minority Game (MG) is a mathematical model that takes inspiration from the "El Farol Bar" problem introduced by Brian Arthur (Arthur 1994). It is based on a simple scenario where at each step a set of agents perform a boolean vote which conceptually splits them in two classes: the agents in the smaller class win. In this game, a rational agent keeps track of previous votes and victories, and has the goal of winning throughout the steps of the game—for which a rational strategy has to be figured out. Several researches showed that, although very simple, this model takes into account crucial aspects of some interesting complex systems coupling rationality with emergence: e.g. bounded rationality, heterogeneity, competition for limited resources, and so on. For instance, MG is a good model to study market fluctuation, as an emergent property resulting from interactions propagating from micro scale (agent interaction) to macro scale (collective behaviour).

As showed by Renz and Sudeikat (2005), a multiagent system (MAS) can be used to realise a MG simulation—there, BDI agents provide for rationality and planning. An agent-based simulation is particularly useful when the simulated systems include autonomous entities that are diverse, thus making it difficult to exploit the traditional framework of mathematical equations.

In this paper we proceed along this direction, and adopt a novel MAS meta-model based on the notion of artifact (Ricci et al. 2006). The notion of artifact is inspired by Activity Theory (Ricci et al. 2003): it represents those abstractions living in the MAS environment that provide a function, which agents can exploit to achieve individual and social goals. The engineering principles promoted by this meta-model makes it possible to flexibly balance the computational burden of the whole system between autonomy of the agents and the designed behaviour of artifacts.

In order to implement MG simulations we adopt the TuCSoN infrastructure for agent coordination (Omicini and Zambonelli 1999), which introduces tuple centres as artifact representatives. A tuple centre is a programmable coordination medium living in the MAS environment, used by agents interacting by exchanging tuples (logic tuples in the case of TuCSoN logic tuple centres). As we are not concerned much with the mere issues of agent intelligence, we rely here on a weak form of rationality, through logic-based agents adopting pre-compiled plans called *operating instructions* (Viroli and Ricci 2004).

By implementing MG over TuCSoN, we can experiment with flexibility and controllability of the artifact model, and see if and how they apply to the simulation – in particular, artifacts allow for a greater level of controllability with respect to agents. To this end, in this paper we show how the model allows some coordination parameters to be changed during the run of a simulation with no need to stop the agents: this can be useful e.g. to change the point of equilibrium, controlling the collective behaviour resulting by interactions propagated from the entities at the micro level.

The remainder of this paper is organised as follows. First, we introduce the general simulation framework based on agents and artifacts. Then, we provide the reader with some relevant details of the Minority Game. Some quantitative results of MG simulation focussing on system dynamics and run-time changes are presented, just before final remarks.

THE TuCSoN FRAMEWORK FOR SIMULATION

The architecture proposed for MAS simulation is based on TuCSoN (Omicini and Zambonelli 1999), which is an infrastructure for the coordination of MASs. TuCSoN provides agents with an environment made of logic tuple centres, which are logic-based programmable tuple spaces. The language used to program the coordination behaviour of tuple centres is ReSpecT, which specifies how a tuple centre has to react to an observable event (e.g. when a new tuple is inserted) and has to accordingly change the tuple-set state (Omicini and Denti 2001). Tuple centres are a possible incarnation of the coordination artifact notion (Omicini et al. 2004), representing a device that persists independently of agent life-cycle and provides services to let agents participate to social activities.

In our simulation framework we adopt logic-based agents, namely, agents built using a logic programming style, keeping a knowledge base (KB) of facts and acting according to some rule—rules and facts thus forming a logic theory. The implementation is based on tuProlog technology[1] for Java-Prolog integration, and relies on its inference capabilities for agent rationality. Agents roughly follow the BDI architecture, as the KB models agent beliefs while rules model agent intentions.

To coordinate agents we take inspiration from natural systems like ant-colonies, where coordination is achieved through the mediation of the environment: our objective is to have a possibly large and dynamic set of agents which coordinate each other through the environment while bringing about their goals.

Externally, we can observe overall system parameters by inspecting the environment, namely, the tuple centres agents interact with. In this way we can try different system behaviours changing only the coordination behaviour of the environment. Furthermore we can change, during the simulation, some coordination parameters (expressed as tuples in a tuple centre), programming and then observing the transition of the whole system either to a new point of equilibrium or to a divergence.

Three kinds of agents are used in our simulation: player agents, monitor agents and tuning agents (as depicted in Figure 1): all the agents share the same coordination artifact. The agent types differ because of their role and behaviour: player agents play MG, the monitor agent is an observer of interactions which visualises the progress of the system, the tuning agent can change some rules or parameters of coordination, and drives the simulation to new states. Note that the main advantage of allowing a dynamic tuning of parameters instead of running different simulations lays in the possibility of tackling emergent aspects which would not necessarily

[1] http://tuprolog.alice.unibo.it

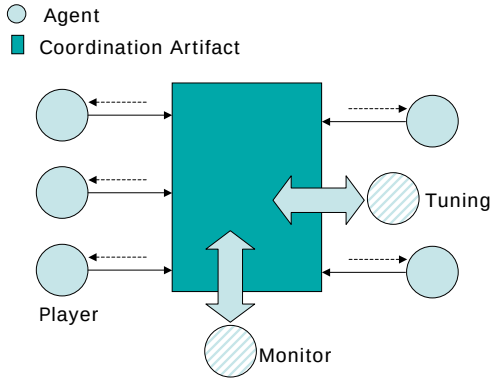

Figure 1: TuCSoN Simulation Framework for MG

appear in new runs.

The main control loop of a player agent is a sequence of actions: observing the world, updating its KB, scheduling next intention, elaborating and executing a plan. To connect agent mental states with interactions we use the concept of action preconditions and perception effects as usual.

MINORITY GAME

MG was introduced and first studied by Challet and Zhang (1997), as a means to evaluate a simple model where agents compete through adaptation for finite resources. MG is a mathematical representation from 'El Farol Bar' problem introduced by Arthur (1994), providing an example of inductive reasoning in scenarios of bounded rationality. The game consists in an odd number N of agents: at each discrete time step t of the game an agent i takes an action $a_i(t)$, either 1 or -1. Agents taking the minority action win, whereas the majority looses. After a round, the total action result is calculated as:

$$A(t) = \sum_{i}^{N} a_i(t)$$

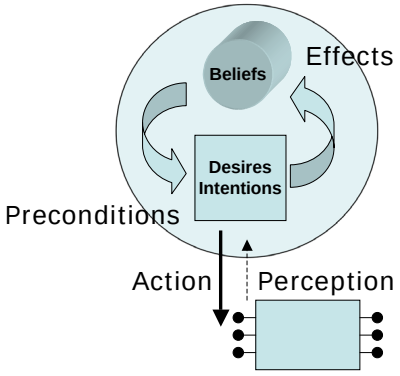

Figure 2: Agent Architecture

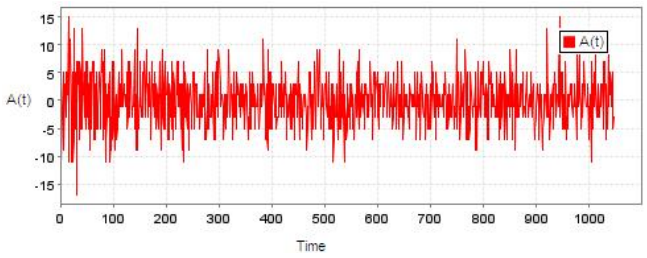

Figure 3: Typical Time evolution of the Original MG with $N = 51$, $m = 5$ and $s = 2$

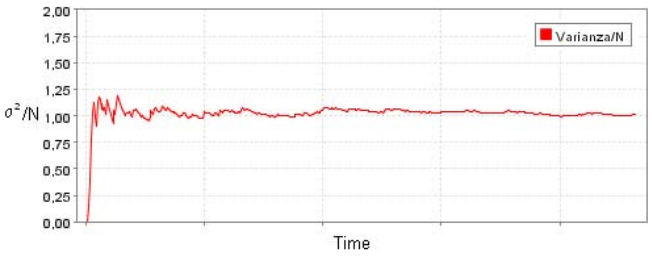

Figure 4: Variance of the Game with 11 Random Agents

In order to take decisions agents adopt strategies. A strategy is a choosing device that takes as input the last m winning results, and provides the action (1 or -1) to perform in the next time step. The parameter m is the size of the memory of the past results (in bits), and 2^m is therefore the potential past history that defines the number of possible entries for a strategy.

The typical strategy implementation is as follows. Each agent carries a sequence of 2^m actions, called a strategy, e.g. $[+1, +1, -1, -1, +1, -1, +1, +1]$. The information on past wins is mapped on a natural number between 0 and $2^m - 1$, which is used as position in the above sequence of the next action to take: for instance, if $[-1, +1, -1]$ is the past winning group, we read it as 010 (that is, 2), and accordingly pick the decision in position 2 inside $[+1, +1, -1, -1, +1, -1, +1, +1]$, that is -1.

Each agent actually carries a number $s \geq 2$ of strategies. During the game the agent evaluates all its strategies according to their success, and hence at each step it decides based on the most successfull strategy so far. Figure 3 shows a typical evolution of the game.

One of the most important applications of MG is in the market models: Challet et al. (2000) use MG as a coarse-grained model for financial markets to study their fluctuation phenomena and statistical properties. Even though the model is coarse-grained and provides an over-simplified micro-scale description, it anyway captures the most relevant features of system interaction, and generates collective properties that are quite similar to those of the real system.

Another point of view, presented e.g. by Parunak et al. (2002), considers the MG as a point in space of a Resource Allocation Game (RAG). In this work a generalisation of MG is presented that relaxes the constraints on the number of resources, studying how the system behaves within a given range.

In a more recent paper, Renz and Sudeikat (2005) observe that MG players could be naturally modelled as agents with a full BDI model, and use a new adaptive stochastic MG with dynamically evolving strategies in the simulation.

MG Performance

In order to track the performance of an MG system, the most interesting quantity is *variance*, defined as $\sigma^2 = \overline{[A(t) - \overline{A(t)}]^2}$: it shows the variability of the bets around the average value $\overline{A(t)}$. In particular, the normalised version of variance $\rho = \sigma^2/N$ is considered.

Generally speaking, variance is the inverse of global efficiency: as variance decreases agent coordination improves, making more agents winning. Variance is interestingly affected by the parameters of the model, such as number of agents (N), memory (m) and number of strategies (s): in particular, the fluctuation of variance is shown to depend only on the ratio $\alpha = 2^m/N$ between agent memory and the number N of agents.

For large values of α—the number of agents is small with respect to the number of possible histories—the outcomes are seemingly random: the reason for this is that the information that agents observe about the past history is too complex for their limited processing analysis.

When new agents are added, fluctuation decreases and agents perform better by choosing randomly, in this case $\rho = 1$ and $\alpha \approx 1/2$, as visible in the results of our simulation in Figure 4—the game enters into a regime where the loosing group is close to $N/2$, hence we might say coordination is performing well.

If the number of agents increase further, fluctuations rapidly increase beyond the level of random agents and the game enters into the crowded regime. With a low value of α the value of σ^2/N is very large: it scales like $\sigma^2/N \approx \alpha^{-1}$.

The results of other observations suggest that the behaviour of MG can be classified in two phases: an information-rich *asymmetric* phase, and an unpredictable or *symmetric* phase. A phase transition is located where σ^2/N attains its minimum ($\alpha_c = 1/2$), and it separates the symmetric phase with $\alpha < \alpha_c$ from an asymmetric phase with $\alpha > \alpha_c$.

All these cases have been observed with the TuCSoN simulation framework described in next section.

THE SIMULATION FRAMEWORK

The construction of MG simulations with MASs is based on the TuCSoN framework and on tuProlog as an inferential engine to program logic agents. The main innovative aspect of this MG simulation is the possibility of studying the evolution of the system with particular and different kinds of agent behaviour at the micro level, imposed as coordination parameters which are changed on-the-fly.

Operating Instructions

Each agent has an internal plan, structured as an algebraic composition of allowed actions (with their preconditions) and perceptions (with their effects), that enables the agent to use the coordination artifact to play the MG. This plan can be seen as Operating Instructions (Viroli and Ricci 2004), a formal description based on Labelled Transition System (LTS) that the agent reads to understand what its step-by-step behaviour should be. Through an inference process, the agent accordingly chooses the next action to execute, thus performing the cycle described in Section 2.

Operating instructions are expressed by the following theory:

```
firststate(agent(first,[])).
definitions([
  def(first,[],...),
  def(main,[S],
       [act(out(play(X)),pre(choice(S,X))),
        per(in(result(Y)),eff(res(Y))),
        agent(main,[S])]
  ),
  ...
]).
```

The first part of operating instructions is expressed by term `first`, where the agent reads the game parameters that are stored in the KB, and randomly creates its own set of strategies.

In the successive part `main`, the agent executes its main cycle. It first puts tuple `play(X)` in the tuple space, where $X = \pm 1$ is agent vote. The precondition of this action `choice(S,X)` is used to bind in the KB X with the value currently chosen by the agent according to strategy S. Then, the agent gets the whole result of the game in tuple `result(Y)` and applies it to its KB. After this perception, the cycle is iterated again.

Tuple Centre Behaviour

The interaction protocol between agents and the coordination artifact is then simply structured as follows. First each agent puts the tuple for its vote. When the tuples for all agents have been received, the tuple centre checks them, computes the result of the game—either 1 or -1 is winning—and prepares a result tuple to be read by agents.

The ReSpecT program for this behaviour is loaded in the tuple centre by a configuration agent at bootstrap, through operation `set_spec()`. The following ReSpecT reaction is fired when an agent inserts tuple `play(X)`, and triggers the whole behaviour:

```
reaction(out(play(X)),(
  in_r(count(Y)),
  Z is Y+1,
  in_r(sum(M)),
  V is M+X,
  out_r(sum(V)),
  out_r(count(Z))
)).
```

This reaction considers the bet (`X`) counts the bets (`Z`) and computes the partial result of the game (`V`). When all the agents have played, the artifact produces the tuple `winner(R,NS,T1,T2,last/more)`, which is the main tuple of MG coordination.

```
reaction(out_r(count(X)),(
  rd_r(numag(Num)),
  X=:=Num,
  in_r(totcount(T)),
  P is T+1,
  rd_r(game(G)),
  in_r(sum(A)),
  out_r(sum(0)),
  rd_r(countsession(CS)),
  in_r(count(Y)),
  out_r(count(0)),
  %%calculate variance
  in_r(qsum(SQ)),
  NSQ is A*A+SQ,
  out_r(qsum(NSQ)),
  %%calculate mean
  in_r(totsum(R)),
  NewS is R+A,
  out_r(totsum(NewS)),
  rd_r(tuning1(T1)),
  rd_r(tuning2(T2)),
  out_r(winner(A,P,CS,T1,T2,G)),
  out_r(totcount(P))
)).
```

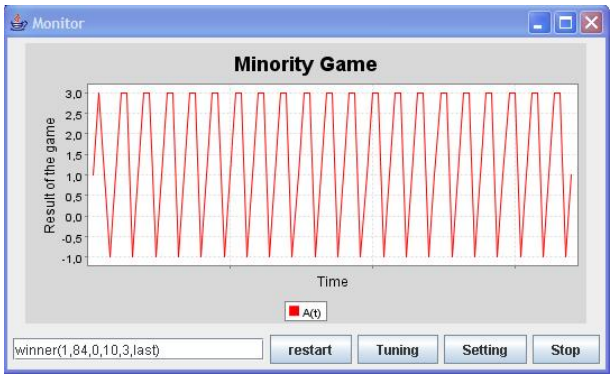

Figure 5: Interface of the Monitor Agent

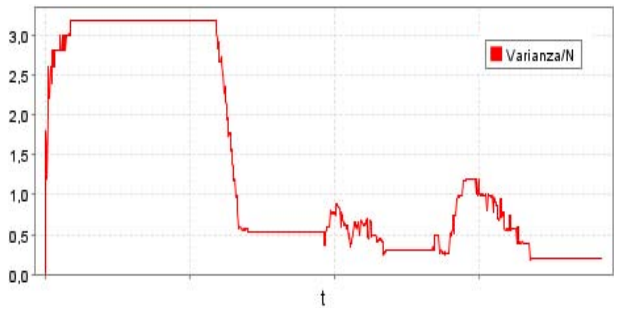

Figure 6: Variance of the System with Initial Parameters $N = 5$ and $m = 3$

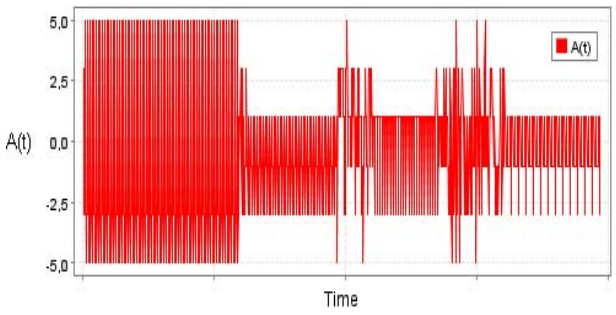

Figure 7: System Evolution of the Variance in Figure 6

The `winner` tuple contain the result of game (`R`), the number of step (`NS`), two tuning parameters (`T1` and `T2`) and one constant to communicate agents whether they have to stop or to play further (`last/more`). Figure 5 reports the graphical interface of the monitor agent that during its life-time reads the tuple `winner` and draws variance.

The simulation architecture built in this way allows for on-the-fly change of some game configuration parameters—such as the dimension of agent memory—with no need to stop the simulation and re-program the agents.

By changing the parameters, the tuning agent can drive the system from an equilibrium state to another, by controlling agent strategies, the dimension of memory, or the number of losses that an agent can accept before discarding a strategy. This agent observes system variance, and decides whether and how to change tuning parameters: reference variance is calculated by first making agents playing the game randomly—see Figure 4. The new value of parameters is stored in tuple centre through tuples `tuning1(T1)` and `tuning2(T2)`, the rules of coordination react and update the information that will be read by the agents.

Simulation Results

The result of the tuned simulation in Figures 6 and 7 shows how the system changes its equilibrium state and achieves a better value of variance.[2] In this simulation the tuning agent is played by a human that observes the evolution of the system and acts through the tuning interface to change the coordination parameters, such as threshold of losses and memory, hopefully finding new and better configurations. The introduction of the threshold of losses in the agent behaviour is useful when the game is played by few agents: these parameters enable system evolution and a better agent cooperative behaviour.

[2]In Figure 6, the first phase of equilibrium is followed by a second one obtained by changing the threshold parameter $S = 5$. Finally, a third phase is obtained changing the dimension of the memory to $m = 5$.

CONCLUSION

In this paper, we aim at introducing new perspectives on agent-based simulation by adopting a novel MAS meta-model based on agents and artifacts, and by applying it to Minority Game simulation. We implement and study MG over the TuCSoN coordination infrastructure, and show some benefits of the artifact model in terms of flexibility and controllability of the simulation. In particular, in this work we focus on the possibility to build a feedback loop on the rules of coordination driving a system to a new and better equilibrium state. Many related agent simulation tools actually exist: as this paper is a starting point, we plan to perform a systematic comparison of their expressiveness and features. In the future, we are interested in constructing an intelligent and adaptive tuning agent with a BDI architecture, substituting the human agent in driving the evolution over time of the system behaviour.

REFERENCES

Arthur, W. B. 1994, May. Inductive reasoning and bounded rationality (the El Farol problem). *American Economic Review* 84 (2): 406–411.

Challet, D., M. Marsili, and Y.-C. Zhang. 2000, February. Modeling market mechanism with minority game. *Physica A: Statistical and Theoretical Physics* 276 (1–2): 284–315.

Challet, D., and Y.-C. Zhang. 1997, December. Emergence of cooperation and organization in an evolutionary game. *Physica A: Statistical and Theoretical Physics* 246 (3–4): 407–418.

Jennings, N. R., C. Sierra, L. Sonenberg, and M. Tambe. (Eds.) 2004, 19–23 July. *3rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*, New York, USA. ACM.

Omicini, A., and E. Denti. 2001, June. Formal ReSpecT. *Electronic Notes in Theoretical Computer Science* 48:179–196.

Omicini, A., A. Ricci, M. Viroli, C. Castelfranchi, and L. Tummolini. 2004, 19–23 July. Coordination artifacts: Environment-based coordination for intelligent agents. See Jennings et al. (2004), 286–293.

Omicini, A., and F. Zambonelli. 1999, September. Coordination for Internet application development. *Autonomous Agents and Multi-Agent Systems* 2 (3): 251–269.

Parunak, H. V. D., S. Brueckner, J. Sauter, and R. Savit. 2002, 15–19 July. Effort profiles in multi-agent resource allocation. In *1st International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2002)*, ed. C. Castelfranchi and W. L. Johnson, 248–255. Bologna, Italy: ACM.

Renz, W., and J. Sudeikat. 2005. Modeling Minority Games with BDI agents – a case study. In *Multiagent System Technologies*, ed. T. Eymann, F. Klügl, W. Lamersdorf, M. Klusch, and M. N. Huhns, Volume 3550 of *LNCS*, 71–81. Springer. 3rd German Conference (MATES 2005), Koblenz, Germany, 11-13 September 2005. Proceedings.

Ricci, A., A. Omicini, and E. Denti. 2003, April. Activity Theory as a framework for MAS coordination. In *Engineering Societies in the Agents World III*, ed. P. Petta, R. Tolksdorf, and F. Zambonelli, Volume 2577 of *LNCS*, 96–110. Springer-Verlag.

Ricci, A., M. Viroli, and A. Omicini. 2006, March. Programming MAS with artifacts. In *Programming Multi-Agent Systems*, ed. R. P. Bordini, M. Dastani, J. Dix, and A. El Fallah Seghrouchni, Volume 3862 of *LNAI*, 206–221. Springer. 3rd International Workshop (PROMAS 2005), AAMAS 2005, Utrecht, The Netherlands, 26 July 2005. Revised and Invited Papers.

Viroli, M., and A. Ricci. 2004, 19–23 July. Instructions-based semantics of agent mediated interaction. See Jennings et al. (2004), 102–109.