EPMAS: EVOLUTIONARY PROGRAMMING MULTI-AGENT SYSTEMS

Ana M. Peleteiro, Juan C. Burguillo
Telematics Engineering Department
University of Vigo
Campus de Lagoas-Marcosende, 36310 Vigo, Spain
Email: {apeleteiro, J.C.Burguillo}@det.uvigo.es

Zuzana Oplatkova, Ivan Zelinka
Faculty of Applied Informatics
Tomas Bata University in Zlin
Nad Stranemi 4511, 76001 Zlin, Czech Republic
Email: {oplatkova, zelinka}@fai.utb.cz

KEYWORDS

Evolutionary Programming, Grammatical Evolution, Multi-agent Systems

ABSTRACT

Evolutionary Programming (EP) seems a promising methodology to automatically find programs to solve new computing challenges. The Evolutionary Programming techniques use classical genetic operators (selection, crossover and mutation) to automatically generate programs targeted to solve computing problems or specifications. Among the methodologies related with Evolutionary Programming we can find Genetic Programming, Analytic Programming and Grammatical Evolution. In this paper we present the Evolutionary Programming Multi-agent Systems (EPMAS) framework based on Grammatical Evolution (GE) to evolutionary generate Multi-agent systems (MAS) ad-hoc. We also present two case studies in MAS scenarios for applying our EPMAS framework: the predator-prey problem and the Iterative Prisoner's Dilemma.

INTRODUCTION

In our evolving society, the computing and engineering problems we have to solve are becoming more difficult and intractable every day. Future computers and software systems should be able to automatically deal with these new challenges. This is a central topic in Artificial Intelligence discipline (Russell and Norvig, 2003).

Evolutionary algorithms and multiagent systems seem two promising methodologies to automatically find solutions for these new challenges. These algorithms use the classical genetic operators (selection, crossover and mutation) to find an optimal solution, and they have been successfully applied in the automatic management of isolated problems, i.e., evolutionary methods can lead to the realization of artificial intelligent systems (Fogel et al., 1966). Among the methodologies related with Evolutionary Programming we can find Genetic Programming (Koza, 1994), Analytic Programming (Zelinka and Oplatkova, 2003) and Grammatical Evolution (O'Neill and Ryan, 2003). The first methodology (GP) is based on the LISP language and its tree syntax, while the last two ones have the advantage of being language independent. There are several approaches in the literature applying GP (Haynes and Sen, 1996; Calderoni and Marcenac, 1998) to try to solve problems in Multi-agent Systems. However, to our knowledge, this is the first approach using a GE framework for solving general problems in MAS.

In this paper we present an analysis of the conditions needed to apply Evolutionary Programming techniques to create a Multi-agent System able to solve a given problem. Then we propose the use of Grammatical Evolution (GE) to provide 'good' solutions in MAS scenarios. Finally, we present two case studies based on two typical MAS scenarios. The first one is the well known predator-prey problem, in which some predators aim at surrounding the prey. The second one is the Iterative Prisoner's Dilemma, a classical Game Theory scenario, where we aim at finding an optimal solution for several configurations of the famous Axelrod's tournament (Axelrod, 1984).

The rest of the paper is organized as follows: in section GRAMMATICAL EVOLUTION we give a brief introduction to this evolutionary technique. In section A FRAMEWORK FOR EVOLUTIONARY PROGRAMMING MULTI-AGENT SYSTEMS we present our theoretical framework for evolving Multi-agent Systems. Section CASE STUDIES presents two experiments where we test our framework with classical MAS scenarios. Finally, we present the conclusions and our future work.

GRAMMATICAL EVOLUTION (GE)

Grammatical Evolution (GE) (O'Neill and Ryan, 2003) is an evolutionary computation technique based on Genetic Programming (GP) (Koza, 1994). GP provides a systematic method to allow computers to automatically solve a given problem from a high-level statement. The idea is to use evolutionary or genetic operations (selection, crossover and mutation) to generate new populations of computer programs, measuring the performance (i.e., the fitness) for each one, and obtaining the one with the best fitness as the result.

One of the main advantages of GE compared to GP is that it can evolve complete programs in any arbitrary programming language (O'Neill and Ryan, 2003), which provides flexibility to the system under development.

Proceedings 24th European Conference on Modelling and Simulation ©ECMS Andrzej Bargiela, Sayed Azam Ali David Crowley, Eugène J.H. Kerckhoffs (Editors)
ISBN: 978-0-9564944-0-5 / ISBN: 978-0-9564944-1-2 (CD)

GE is a grammar-based form of GP, and this means that we can apply genetic operators to an integer string, subsequently mapped to a program, through the use of a grammar. The grammar describes the output language, using the Backus Naur Form (BNF) (Garshol, 2003), in the form of production rules, and it governs the creation of the program itself. Changing the grammar we can radically change the behavior of the generation strategy.

GE has been applied in many fields, for instance, in technical trading (predicting corporate bankrupt, bond credit rating) (Brabazon and Oneill, 2004), to find trigonometric identities (Ryan et al., 1998), or to solve symbolic regression problems (Ryan and O'Neill, 1998).

A FRAMEWORK FOR EVOLUTIONARY PROGRAMMING MULTI-AGENT SYSTEMS

In this section we provide a description of our framework for using Evolutionary Programming to automatically generate code for Multi-agent Systems (MAS). In order to do this, we need to guarantee that the properties that characterize a MAS are preserved, that we have an iterative process to successively refine the generated agents and that we are able to test somehow the behavior of the MAS to evaluate its performance.

But, first we need to define our concept of agent. Unfortunately, there is no general agreement in the research community about what an agent is. Therefore we cite a general description (Wooldridge and Jennings, 1995), and according to it, the term agent refers to a hardware or (more usually) software-based computer system characterized by the well-known properties:

- **Autonomy**: *agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state*. To achieve this property, we need to obtain a different code for the different agents of the MAS and include, within the code properties, the capability to decide when and how to act depending on internal states and external perceptions.

- **Social ability**: *agents interact with other agents (and possibly humans) via some kind of agent-communication language*. This can be obtained providing properties for explicit or implicit interaction, or communication, among the agents of the system.

- **Reactivity**: *agents perceive their environment, and respond in a timely fashion to changes that occur in it*. To do this we need to generate code for the agents able to react to the different inputs provided by the environment.

- **Pro-activeness**: *agents do not simply act in response to their environment; they are able to exhibit goal-directed behavior by taking the initiative*. To achieve this property, the agent must include code for deciding how to act depending on internal states or external inputs.

However, there are other desirable, although not mandatory, attributes that can be present: *benevolence, rationality* and *adaptability (or learning)* (Wooldridge, 2002). Now we can define a Multi-agent System as a system consisting of an interacting group of agents.

The simulation of agents and their interactions, usually known as agent-based modeling, is the basis for the iterative approach presented in this section in order to successively refine a MAS and achieve the best possible solution for a given problem or scenario.

We have described a MAS as a set of multiple autonomous entities (agents) that interact, and the right interaction is the key to achieve the desired MAS behavior. We need a framework to model distributed problems, and we must define a procedure to solve them by means of successive iterations that shift the MAS behavior to find alternative solutions, and eventually the optimum.

We consider that we can get closer to this conceptual framework by mixing the conceptual modeling of the problem at one level and its actual performance at another one. The algorithm in Table 1 describes the basic steps needed to generate MAS solutions and to evaluate its performance.

Table 1: Generating Evolutionary Solutions for a MAS

```
1 Generate a new solution for the MAS.
2 Simulate the new solution.
3 Evaluate the results of the simulation, and get the fitness.
4 If a STOP criteria does not hold, go back to (1).
```

This cycle is an iterative process that successively generates, and ideally improves, the solutions for the MAS behavior. To obtain a final solution we need to describe how to manage the three phases that compose this iterative process:

1. **Generate a solution for the MAS**: to do this we need to be able to generate evolutionary code, different for every agent of the MAS. Thus, we need multiple instances of an Evolutionary Programming tool in order to generate different code for the different agents.

2. **Simulate the MAS behavior**: for this we need to simulate the agents generated in the step 1 within the MAS environment, being able to produce some final result (fitness) to be used as input for the evaluation phase.

3. **Evaluate the results of the simulation and get the fitness**: for this we need to define some type of fitness for the different agents as a measure to indicate

how well they have achieved their individual objectives. We may also define a global fitness to describe if the system as a whole has reached the pursued objective. This individual and global fitness must act as a feedback for the whole system in the generation of a new solution, ideally better than the previous ones.

Now, we propose the use of two open source tools, freely available on the Internet, to generate solutions by means of Evolutionary Programming in MAS (EPMAS).

Grammatical Evolution in Java (GEVA)

Grammatical Evolution in Java (GEVA) (O'Neill et al., 2008) is a free and open source environment for using GE, developed at the UCD's Natural Computing Research & Applications group. This software provides a search engine framework to build the functions, as well as a simple GUI and the genotype-phenotype mapper of GE.

Netlogo

NetLogo (Wilensky, 2007) is a free Multi-agent modeling environment for simulating natural and social phenomena. It is particularly well suited for modeling complex systems that evolve. Modelers can give instructions to hundreds of agents operating independently. This makes it possible to explore the connection between the micro-level behavior of individuals, and the macro-level patterns that emerge from the interaction of many individuals.

This environment has been used to carry out simulations in areas such as biology, medicine, physics, chemistry, mathematics, computer science, economics and social psychology.

NetLogo provides a 2-D world made of agents that simultaneously carry out their own activity. We can model the behavior of these agents, individually and also in group. This makes Netlogo a really useful tool for Multi-agent simulations. Besides, with its user-friendly graphical interface, a user may interact with the system and obtain real-time feedback from the simulations.

A Framework for EPMAS

The two previous tools, i.e., GEVA and Netlogo, solve the needs expressed in Table 1 to create a framework for Evolutionary Programming Multi-agent Systems. GEVA environment has been designed to obtain solutions for a particular function using a pre-defined function population. To generate different functions for the agents in the system we need to create multiple instances of GEVA, each one managing the code for every particular agent (or type of agents) in the MAS. Unfortunately, GEVA has not included the possibility to run several instances of the tool in a shared memory, and this means that we need to use external facilities (files at the OS level) to communicate those processes.

Once we have created the code for all the agents in the system with several GEVA instances, we need to simulate the actual solution for the MAS, and test its behavior in the particular problem it is aimed to solve (step 2 in Table 1). This can be done using the Netlogo simulation tool. In this case, the connection between GEVA and Netlogo has been also achieved by means of file interactions. In Fig. 1 we show a schema of the communication between GEVA and Netlogo to implement the model described in Table 1.

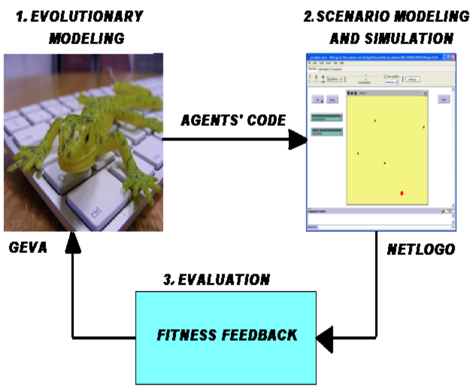


Figure 1: Communication between GEVA and Netlogo.

Finally, we have to define a fitness measurement, dependent from the problem definition, to provide the fitness feedback for the different GEVA instances, in order to generate a new solution for every agent of the MAS.

The algorithm of Table 1 stops when some criterion selected by the system designer has been achieved. This means that the solution complies with certain criteria, for instance, that a number of iterations have been performed, or that a particular set of expected values in the solution has been accomplished.

CASE STUDIES

In this section we describe two example scenarios for MAS, and we present the results of the experiments performed using GE to obtain a solution for each of them. In the following two subsections we describe the experiments, as well as presenting the results obtained.

Predator-prey Scenario

The predator-prey pursuit problem (Benda et al., 1986) is one of the first and well-known testbed for learning strategies in Multi-agent Systems. It consists of a set of agents, named predators, that aim at surrounding another agent, named prey, that must escape from them. This toy problem has been addressed many times by the Artificial Intelligence (AI) Community. Initially, Korf (Korf, 1992) proposed a solution without Multi-agent communication. Since then, a great number of alternatives have emerged usually involving reinforcement learning (Tan, 1997) techniques.

We use Netlogo to simulate the scenario where the prey and the predators perform their chase (Fig. 2). We have four predators (black arrows) that aim at surrounding and catching one prey (a red cicle).

The predators can move to the North (N), South (S), East (E) and West (W), and they can only see the prey if they are closer than a quarter of the maximum distance of the map scenario. Our predators may use a communication function (*broadcast();*) that allows to communicate the position of the prey to the rest of the predators. But if any predator cannot see the prey, then they move randomly. A predator catches the prey if both are located in the same position. The prey behavior is really simple: it randomly moves to the N, S, E or W all over the map (it is a non-toroidal world).

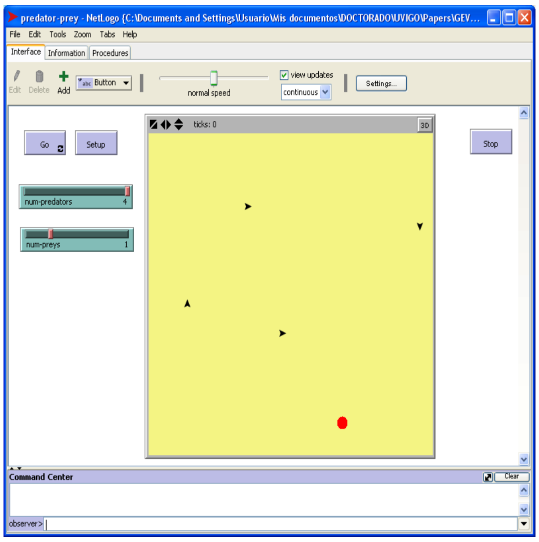

predator-prey - NetLogo {C:\Documents and Settings\Usuario\Mis documentos\DOCTORADO\UVIGO\Papers\GEV...
File Edit Tools Zoom Tabs Help
Interface Information Procedures
Edit Delete Add
Button
normal speed
view updates
continuous
Settings...
ticks: 0
3D
Go
Setup
Stop
num-predators 4
num-preys 1
Command Center
Clear
observer>

Figure 2: Netlogo Scenario where the Chase is Performed. Predators are Depicted as Black Arrows and the Prey as a Red Circle.

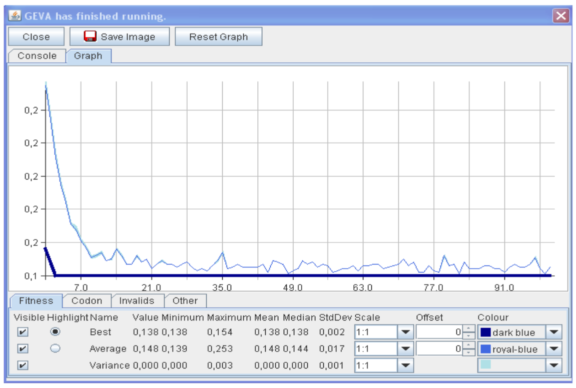
Our predators' program should evolve to find the best function to catch the prey. To do this, we use GE, and we generate the function (from Eq. 1) with GEVA and pass it to Netlogo. Then, Netlogo simulates a run of the MAS where the predators try to catch the prey with the given evolutive programs.

In this experiment, the parameters used in GEVA are a population size of 100 programs, 100 generations to run and a mutation probability equal to 0.01. In Netlogo, we run every simulation ten times, and each run ends when a predator catches the prey, or when the maximum time for catching the prey has finished. After these runs, Netlogo returns to GEVA an average fitness value which indicates how well the generated code has worked. GEVA stores this value with the generated code, and provides a new program (generated from Eq. 1), repeating these steps until the evolutionary process finishes.

To measure the fitness value in Netlogo, we give a positive reward every time the predator gets closer to the prey in the last move it has performed, or we do not give a reward if the predator gets away (we do this valuation before the prey moves). We repeat this process until the end of the run, where we have two possible outcomes: if a predator has caught the prey, then we divide the reward obtained by the total distance it has traveled. If not, we assign the smallest fitness value, and we invert this values (since GEVA finds minimums). Finally, Netlogo passes this value to GEVA, which stores it, generates another function and repeats the whole process again. The algorithm stops when GEVA does not generate more functions, i.e., the evolutionary process stops, and gives as a result the agent program that has the best performance.

$$\begin{aligned} &< opbroad >< op >< op >< op >< op > \\ &< op > = ifPreyNorth\{< dir >\}; \\ &ifPreySouth\{< dir >\}; \\ &ifPreyEast\{< dir >\}; \\ &ifPreyWest\{< dir >\}; \\ &< dir > = \{\ N,\ S,\ E,\ W\ \} \\ &< opbroad > = \{broadcast(); notBroadcast();\} \end{aligned} \tag{1}$$

After executing the chase in our EPMAS framework, we obtained that the program with best performance is the one shown in Eq. 2. It is the reasonable result, since first predators broadcast the prey position to their mates (if they see the prey), and then move according to the prey's position. We can see the evolution of the fitness in Fig. 3.



Visible	Highlight	Name	Value	Minimum	Maximum	Mean	Median	StdDev	Scale	Offset	Colour
✓	●	Best	0,138	0,138	0,154	0,138	0,138	0,002	1:1	0	dark blue
✓	○	Average	0,148	0,139	0,253	0,148	0,144	0,017	1:1	0	royal-blue
✓		Variance	0,000	0,000	0,003	0,000	0,000	0,001	1:1		

Figure 3: Representation of the Fitness Evolution in GEVA for the Predator-prey Model.

$$\begin{aligned} &broadcast(); \\ &ifPreyNorth\{N\}; \\ &ifPreySouth\{S\}; \\ &ifPreyEast\{E\}; \\ &ifPreyWest\{W\}; \end{aligned} \tag{2}$$

Game Theory Scenario

Game Theory (Binmore, 1994) is a branch of applied mathematics that helps to understand the strategies that selfish individuals may follow when competing or collaborating in games and real scenarios (Osborne, 2003).

The concept of cooperation evolution has been successfully studied using theoretical frameworks like the Prisoner's Dilemma (PD) (Axelrod, 1984), which is one of the most well-known strategy games. It models a situation in which two entities have to decide whether cooperate or defect, but without knowing what the other is going to do.

Nowadays, the PD game has been applied to a huge variety of disciplines: economy, biology, artificial intelligence, social sciences, e-commerce, etc. Table 2 shows the general PD form matrix, which represents the rewards an entity obtains depending on its action and the opponent's one. In this matrix, T means the Temptation to defect, R is the Reward for mutual cooperation, P the Punishment for mutual defection and S the Sucker's payoff. To be defined as a PD, the game must accomplish that:

$$\begin{aligned} T > R > P > S \\ 2R > T + S \end{aligned} \tag{3}$$

Table 2: General Prisoner's Dilemma Matrix.

	Player B Cooperates	Player B Defects
Player A Cooperates	R, R	S, T
Player A Defects	T, S	P, P

There is a variant of PD, which is the Iterated Prisoner's Dilemma (IPD) (Axelrod, 1984), in which the game is played repeatedly. In it, the players can punish their opponents for previous non-cooperative behavior, remembering their opponent previous action and adapting their strategy. Game Theory shows that the optimal action if both players know that they are going to play exactly N times is to defect (it is the Nash equilibrium of the game) (Binmore, 1994). But when the players play an indefinite or random number of times, cooperation can emerge as a game equilibrium. The IPD game models transactions between two persons requiring trust and cooperative behavior, and that is why this game has fascinated researchers over the years.

Axelrod organized a tournament in the eighties, where famous game theorists sent their strategies to play the IPD. The winner was Tit-For-Tat (TFT), a simple strategy where the player cooperates on the first move, and in the rest of iterations it reciprocates what the other player did on the previous move. TFT is considered to be the most robust basic strategy. Although, for a certain range of parameters, and in presence of noise, it was found that a strategy, named Pavlov, that beats all the other strategies by giving preferential treatment to co-players which resemble Pavlov (Nowak and Sigmund, 1993).

In our example, we want to emulate Axelrod's tournament by means of evolutionary computation. To do that, we 'organize' a tournament, where every player individually plays against all the rest. Within the set of players, some of them play fixed strategies, and others play as evolutionary agents.

The values we use for the payoff matrix are the classical ones to play the Prisoner's Dilemma, i.e., R=3, S=0, P=1 and T=5. The idea here is to perform experiments in a controlled environment, to see if GE provides 'good' solutions for several well-known configurations. For that we have defined some tournaments with concrete players. The main GEVA parameters we have used are: a population size of 300 programs, 300 generations, and a mutation probability of 0.05. The players play 1000 rounds against every opponent. The strategies considered for the fixed players are:

- **all-D**: the player always defects.
- **all-C**: the player always cooperates.
- **TFT**: the player uses Tit-For-Tat that cooperates at the first iteration and then reciprocates what the opponent did on the previous move.
- **Pavlov (P)**: the player cooperates at the first iteration, and whenever the player and co-player did the same thing at the previous iteration; otherwise, Pavlov defects.
- **Random (R)**: the player cooperates with probability 0.5.
- **ITFT**: the player uses inverse TFT.

Next, we present the results in tables, where *Initial conditions* refers to the initial action taken for the first iteration, *Type of players* are the strategies that the fixed players play, and finally *Results* describe the agent program (denoted as EA, Evolutionary Agent) obtained with the best performance for the tournament configuration.

In Table 3, we can observe that the best behavior against strategies that do not take into account the opponent's last action is to defect, since we obtain maximum gain without being punished.

In Table 4, we present how the evolutionary agent (EA) behaves against more efficient strategies, in fact, against the two strategies with best behavior in Axelrod's tournament.

In the first tournament, we see that TFT is the best strategy if we have players playing all-D, P and TFT, which is normal since the number of defectors is low, the EA defects against them, and TFT works well if playing against TFT and P.

Table 3: Three Tournaments: Players all-D; all-D and all-C and all-D, all-C and rand.

	Tourn. 1	Tourn. 2	Tourn. 3
Initial conditions	D	D, C	D, C, R
Type of players	20 all-D	10 all-D, 10 all-C	7 all-D, 7 all-C, 7 rand
Results	all-D	all-D	all-D

Table 4: Three Tournaments: Players all-D, P, TFT; P, TFT, all-D, all-C, ITFT and P, TFT.

	Tourn. 1	Tourn. 2	Tourn. 3
Initial conditions	D, C, C	C, C, D, C, D	C, C
Type of players	5 all-D, 10 P, 10 TFT	5 P, 5 TFT, 5 all-D, 5 all-C, 5 ITFT	10 P, 10 TFT
Results	TFT	all-D	P, TFT, all-C

In the second tournament the EA plays against all the strategies (except for random). The result is that the best behavior is to always defect. This makes sense since as we have five all-D, to obtain the maximum gain against them the player should defect. Besides, against ITFT, if the player defects the opponent will cooperate the following round, thus defector obtains maximum gain. Against Pavlov, the EA gets the minimum gain in one round, and the maximum one in the other, since initially the Pavlov player cooperates, while the evolutionary agent defects, thus in the following round they both defect, and in the next Pavlov will cooperate again. With TFT, the player gets the minimum gain every time, but still is more than zero.

Finally, in the third tournament of Table 4, the EA plays against the two more efficient strategies, P and TFT, obtaining that the best to do in this case is to use one of those strategies or to be an all-C, i.e., all of them are equivalent.

Table 5 shows the results when two evolutionary agents play a tournament, with other 25 players that use several fixed strategies. As stated before, GEVA does not consider multi-threading, thus the evolutionary players have to communicate via system files, because each of them is an instance of GEVA, and this takes an important amount of time. We perform a tournament in a Pentium(R) Dual Core CPU, E5200 @ 2.50GHz, 3.50GB RAM, lasting two hours for finishing. In Table 5 we observe that we obtain all-D for both evolutionary players, which is a coherent result comparing it with the second tournament of Table 4.

Table 5: Two Evolutionary Player Playing against P, TFT, all-D, all-C and ITFT.

	Tournament
Initial conditions	C, C, D, C, D
Type of players	5 P, 5 TFT, 5 all-D, 5 all-C, 5 ITFT
Results	all-D, all-D

CONCLUSIONS AND FUTURE WORK

In this paper we have presented the use of Evolutionary Programming to evolve Multi-agent problems by means of Grammatical Evolution. The main contribution of the paper is an evolutionary framework to allow the emergence of Multi-agent Systems adapted to the particular conditions and the scenario to model. We present two case studies with classical MAS scenarios to show that the combination of GEVA and Netlogo is a good option to evolve Multi-agent Systems from scratch by combining system generation and simulation.

While some approaches have considering the use of GP in MAS, to the best of our knowledge, this is the first framework combining GE with Multi-agent Systems for solving general scenarios. Among the conclusions obtained we have found that GE can be a good candidate to automatically solve Multi-agent problems. Nevertheless, due to the novelty of this approach, we have found the limitations of the present version of GEVA and Netlogo to simulate complex MAS scenarios. This happens due to lack of support for multi-threading operations, and therefore we had to communicate the different GEVA instances, and the Netlogo simulator, by means of operating system files, which delays a lot the whole process.

As future work, we plan to find a sound and complete formal description of our MAS framework, to create our own evolutionary simulation environment for avoiding the problems found, and finally to validate such new formal framework with more complex examples from different disciplines.

REFERENCES

Axelrod, R. (1984). *The Evolution of Cooperation.* Basic Books.

Benda, M., Jagannathan, V., and Dodhiawala, R. (1986). On optimal cooperation of knowledge sources - an empirical investigation. Technical Report BCS–G2010–28, Boeing Advanced Technology Center, Boeing Computing Services, Seattle, Washington.

Binmore, K. (1994). Game theory and the social contract volume i: Playing fair. *The MIT Press: Cambridge, MA.*

Brabazon, A. and Oneill, M. (2004). Evolving technical trading rules for spot foreign-exchange markets using grammatical evolution. *Computational Management Science*, 1(3-4):311–327.

Calderoni, S. and Marcenac, P. (1998). Genetic progamming for automatic design of self-adaptive robots. In *EuroGP '98: Proceedings of the First European Workshop on Genetic Programming*, pages 163–177, London, UK. Springer-Verlag.

Fogel, L. J., Owens, A. J., and Walsh, M. J. (1966). *Artificial Intelligence through Simulated Evolution*. John Wiley, New York, USA.

Garshol, M. (2003). *BNF and EBNF: What are they and how do they work?*

Haynes, T. and Sen, S. (1996). Evolving Behavioral Strategies in Predators and Prey. In *Adaptation and Learning in Multiagent Systems*, pages 113–126.

Korf, R. (1992). A simple solution to pursuit games. In *Proceedings of the 11th International Workshop on Distributed Artificial Intelligence. Glen Arbor, MI.*

Koza, J. R. (1994). *Genetic programming II: automatic discovery of reusable programs*. MIT Press, Cambridge, MA, USA.

Nowak, M. and Sigmund, K. (1993). A strategy of win-stay, lose-shift that outperforms tit-for-tat in the prisoner's dilemma game. *Nature*, 364(6432):56–58.

O'Neill, M., Hemberg, E., Gilligan, C., Bartley, E., McDermott, J., and Brabazon, A. (2008). Geva: grammatical evolution in java. *SIGEVOlution*, 3(2):17–22.

O'Neill, M. and Ryan, C. (2003). *Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language*. Kluwer Academic Publishers, Norwell, MA, USA.

Osborne, M. J. (2003). *An Introduction to Game Theory*. Oxford University Press, USA.

Russell, S. J. and Norvig, P. (2003). *Articial Intelligence: A Modern Approach, 2nd Ed.* Prentice Hall, Englewoo.

Ryan, C. and O'Neill, M. (1998). Grammatical evolution: A steady state approach. In *In Late Breaking Papers, Genetic Programming*, pages 180–185.

Ryan, C., O'Neill, M., and Collins, J. (1998). Grammatical evolution: Solving trigonometric identities. In *Proceedings of Mendel '98: 4th International Conference on Genetic Algorithms, Optimization Problems, Fuzzy Logic, Neural Networks and Rough Sets*, pages 111–119.

Tan, M. (1997). Multi-agent reinforcement learning: Independent vs. cooperative learning. In Huhns, M. N. and Singh, M. P., editors, *Readings in Agents*, pages 487–494. Morgan Kaufmann, San Francisco, CA, USA.

Wilensky, U. (2007). Netlogo: Center for connected learning and computer-based modeling.

Wooldridge, M. (2002). *Introduction to MultiAgent Systems*. John Wiley & Sons.

Wooldridge, M. and Jennings, N. R. (1995). Intelligent agents: Theory and practice.

Zelinka, I. and Oplatkova, Z. (2003). Analytic programming comparative study. In *CIRAS03, The second International Conference on Computational Intelligence, Robotics, and Autonomous Systems*.

AUTHOR BIOGRAPHIES

ANA PELETEIRO received the M.Sc. degree in Telecommunication Engineering in 2009 (Honor mention) at University of Vigo. She is currently a PhD student in the Department of Telematic Engineering at the same university. Her research interests include intelligent agents and multi-agent systems, self-organization, evolutionary algorithms and game theory. Her email is `apeleteiro@det.uvigo.es` and her personal webpage `http://www-gti.det.uvigo.es/∼apeleteiro`

JUAN C. BURGUILLO received the M.Sc. degree in Telecommunication Engineering in 1995, and the Ph.D. degree in Telematics (cum laude) in 2001; both at the University of Vigo, Spain. He is currently an associate professor at the Department of Telematic Engineering in the University of Vigo. His research interests include intelligent agents and multi-agent systems, evolutionary algorithms, game theory and telematic services. His email is `J.C.Burguillo@det.uvigo.es` and his personal webpage `http://www.det.uvigo.es/∼jrial`

ZUZANA OPLATKOVA was born in Czech Republic, and went to the Tomas Bata University in Zlin, where she studied technical cybernetics and obtained her MSc. degree in 2003 and Ph.D. degree in 2008. She is a lecturer (Artificial Intelligence) at the same university. Her research interests are: evolutionary computing, neural networks, evolutionary programming, metaevolution. Her e-mail address is: `oplatkova@fai.utb.cz`

IVAN ZELINKA was born in Czech Republic, and went to the Technical University of Brno, where he studied technical cybernetics and obtained his degree in 1995. He obtained his Ph.D. degree in technical cybernetics in 2001 at Tomas Bata University in Zlin. He is now professor (artificial intelligence, theory of information) at the Department of Informatics and Artificial Intelligence. His e-mail address is: `zelinka@fai.utb.cz` and his Webpage can be found at `http://www.ivanzelinka.eu`