HAVE 2008 – IEEE International Workshop on
Haptic Audio Visual Environments and their Applications
Ottawa – Canada, 18-19 October 2008

Generation of Rule-based Adaptive Strategies for a Collaborative Virtual Simulation Environment

Alejandro Lujan, Richard Werner, Azzedine Boukerche
PARADISE Research Laboratory
University of Ottawa
Email: {aluja023, rwerner, boukerch}@site.uottawa.ca

***Abstract*—Real Time Strategy Games (RTSG) are a strong test bed for AI research, particularly on the subject of Unsupervised Learning. They offer a challenging, dynamic environment with complex problems that often have no perfect solutions. Learning Classifier Systems are rule-based machine learning techniques that rely on a Genetic Algorithm to discover a knowledge map used to classify an input space into a set of actions.**

This paper focuses on the use of Accuracy-based Learning Classifier System (XCS) as the learning mechanism for generating adaptive strategies in a Real Time Strategy Game. The performance and adaptability of the developed strategies with the XCS is analyzed by facing these against scripted opponents on an open source game called Wargus. Results show that the XCS module is able to learn adaptive strategies effectively and achieve the objectives of each training scenario. [1]

I. Introduction

The relationship between Artificial Intelligence and multimedia systems is twofold. On one hand, the creation of a realistic experience in multimedia systems is often an objective where AI can be of aid. On the other hand, AI researchers can find in these systems — particularly large simulations and games — a robust test-bed for their work [1] [2] [3] [4]. It is the belief of many, and mine in particular, that the advancements of AI will shape in great deal the use of technology in decades to come. Games are no exception.

Tipically, the behavior of artificial game agents is expressed through linear scripts written by game developers, such as:

1) Build a town centre
2) Train 5 soldiers
3) Attack enemy town

The obvious disadvantage with such scripts is that they can be deciphered and countered by an intelligent player, which drastically diminishes the entertaining value of the game. While some degree of randomization can mask the linear nature of the agent, it is possible to develop much richer solutions. By including AI techniques to create agents with *intelligent-like* behaviors, it is possible to extend the validity of the opponents in time while generating more enjoyable interactions for the human players.

A wide array of AI techniques have been used in games, such as $A\star$ for navigation, Finite State Machines for behaviour modeling, and Neural Networks for pattern matching. Learning Classifier Systems, although successful in a number of domains, is a technique barely explored in these environments.

A. Motivation

The motivation behind this work lies in the importance of adaptability when simulating intelligent behavior. An environment that seems static or scripted is much less interesting and immersing than one that reacts and adapts to the actions of the player. Accuracy-based Classifier Systems have shown very positive results in other fields, and their ability to generalize allows for representation of high-level concepts. These, combined with the possibility of mapping game states and actions to rulesets, point to a promising combination of classification theory and Game AI which might result in strong, adaptive agents. Our review of the literature showed little evidence of works using XCS for strategy generation in complex games. This research looks to seize this opportunity.

B. Contributions

We can identify the followings contributions of this work to the areas of Classifier Systems and Games:

1) We applied Accuracy-based Classifier Systems to a full scale, non-markovian environment: a Real Time Strategy Game. We report on its behavior and performance. Most of the literature uses simpler environments, and studying XCS in such simulations is of considerable importance.
2) We study the behavior of the system in two variations of the problem: immediate rewards and delayed rewards, and we report on the implications and challenges each case presents.
3) We report on the importance of adaptability in the area of Games as a crucial characteristic of intelligent behavior
4) This work also shows how adaptability can be achieved with the proper design and selection of techniques, which can be used to add realism to game agents and substantially improve the user experience.
5) Lastly, we identify what challenges Machine Learning faces when applied to a problem with the characteristics of many common commercial games.

The rest of this paper is structured as follows: Section II explores related work on the areas of Game AI and Learning Classifier Systems. Section III presents the proposed system.

[1]This work is partially supported by Canada Research Chairs Program, NSERC, ORF, and MRI/Ontario Early Research Award, and Ontario Distinguished Researcher Award

Section IV explains the design of the experiments, and the results are presented in Section V. Finally, conclusions are drawn in Section VI, along with ideas for future work.

II. Related Work

A. Research on Game AI

Some research has been done on using different AI techniques on complex, dynamic games[1]. Neural Networks, for example, have been used in [5] and [6] as the decision-making elements for action games. Genetic Algorithms have been used in [7], [8], [8] and [9] for strategy and behavior generation and spatial decisions, whereas [10] and [11] use a technique called Influence Maps for a similar purpose. Spronck et al. proposed a technique called Dynamic Scripting[11][12][13].

Many problems remain open for interesting, practical solutions. Adaptivity is one of them: the challenge of generating behavior highly adaptive to a number of different scenarios and opponent strategies.

B. Research on Learning Classifier Systems

Learning Classifier Systems were proposed in 1986 by Holland [14][15], and quickly became an important area of research. Initially proposed as a general framework for Machine Learning, LCS were composed of a set of rules that represented a knowledge map of a particular problem. Each rule is of the form (condition,action) where each condition matches one or more possible inputs. The system interacts with the environment by sensing a number of parameters and executing actions via effectors. At each timestep, rules whose condition match the inputs compete for activation. The environment then gives some form of feedback that measures the effect of the action executed. The strength of a rule was initially associated directly with the expected feedback of the action in the environment. A Genetic Algorithm serves as the mechanism to search for new, more effective rules.

1) LCS Variations: The most important variation of LCS has been the Accuracy-based Classifier System (XCS) proposed by Stewart Wilson [16], which quickly became the standard in the research community. The main difference of XCS versus LCS is that the strength of the rules were associated not with the predicted feedback, but with the accuracy of the prediction. This modification allowed for a more complete and accurate mapping of the search space. Wilson refined XCS in [17], and others have studied it more formally [18][19][20]. Furthermore, [21] and [22] study the domain of problems where XCS has difficulties. On the topic of non-Markovian problems, Lanzi studies the performance of XCS in [23], proposing an internal memory mechanism. Butz has studied XCS and its mechanisms to develop accurate classifiers for particular problems [24]. XCS has also been compared with other Machine Learning techniques with quite favorable results[25].

Many other variations of LCS have been proposed, such as ZCS[26], a simplification of LCS; UCS[27], a supervised version of XCS; [28], an extension of XCS for integer valued parameters; FCS[29], which uses fuzzy-valued parameters; and ACS (Anticipatory Classifier Systems). Sigaud and Wilson worked in comparing ZCS, LCS and XCS in [30]. This paper concentrates on XCS, the particular flavor selected for this work, given its success among different domains and the readily available libraries.

Among the problems studied within XCS, delayed rewards is of particular importance for this work, and it has been found to be quite challenging. Having immediate rewards is the simplest case for learning, whereas having delayed feedback inserts noise and introduces more complexity in the rewarding schema. Research in Markovian environments has found that the credit assignment technique can be inadequate for some problems[31].

2) Applications of XCS: XCS have been extensively researched and applied to a variety of different domains. They have been studied in the domain of Data Mining[32], economic simulations[33], medical data analysis[34] and personal software agents[35]. In video games domain, [36] applied FXCS in a cooperative game environment, and [37] developed an action selector for a MMORPG.

III. Proposed Approach

In this paper, we implement a mechanism to automatically develop agent behavior using Unsupervised Machine Learning, particularly Accuracy-based Classifier Systems (XCS). The XCS is presented with the challenge of learning successful strategies in a Real Time Strategy Game, within a set of scenarios each with a different objective. The XCS module is used as an offline learning technique.

A. Architecture

The two main components of the system are Wargus[38] and XCSLib[39]. Wargus was selected as the testing platform, because it is an open source, Real Time Strategy Game, with commercial-level complexity. Wargus has been used previously in research projects, such as [40], [13] and [12]. XCSLib was chosen mainly for its integration capabilities with the rest of the system, and its availability as a library (as opposed to a standalone application). The classes implemented to connect them (*middleware* for simplicity) take care of the flow and formatting of information between them. Figure 1 shows a high level view of the components and their interaction.

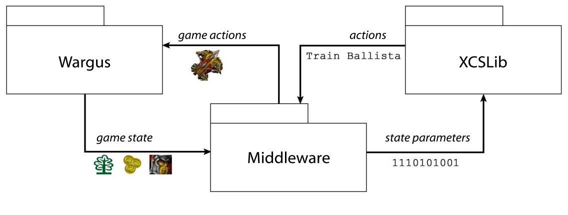

Fig. 1. High level architecture

The game is initialized in learning mode, and at each call of the AI, the XCS module is instructed to take a step. The XCS receives information of the game state, through its sensors, and then formatted into an input string. This string is compared to the condition part of the rules in the XCS population (matching process), and the selected action is then returned to the main script as a result. This is the action that the AI will execute in the game. This XCS step can also trigger the Genetic Algorithm, which creates new rules as the result of combining existing ones.

The performance of the rulesets is monitored by the XCS continuously, alternating between learning and exploitation mode. In learning mode, the actions are selected randomly, providing all rulesets equal opportunities to participate. In exploitation mode, the action proposed in the match set with the largest prediction value is picked. The environment feedback on each of these exploitation mode evaluations is logged for later evaluation.

IV. Experiments

Four training scenarios were designed, each with a particular goal, a map, and a scoring function. A random player was also implemented, allowing us to compare its performance with the trained agents. These are the four scenarios:

1) **Peasants** The agent was rewarded for training peasants. This is the easiest ruleset to learn, for only two actions are relevant: building a town hall and training peasants.
2) **Footmen** The agent was expected to train an army of footmen, for which he needed to build a town hall and barracks first (if these were not present initially). The ruleset is slightly more complicated than the first objective.
3) **Archers** The agent was rewarded for training an army of archers, and three different buildings were relevant for this task, which makes this objective harder than the previous.
4) **Counter-building** The agent was rewarded for different units, all of which are effective against enemy buildings.

The initial conditions (resources, location, available buildings) were randomized at the beggining of each game. This forced the system to explore the search space more thoroughly, instead of just concentrating on one particular case.

The design of the scoring functions is such that if the actions are immediate (meaning their effect is visible on the environment on the next step), the game becomes an *immediate reward* problem. This is a consequence of rewarding every necessary building block, such as creating the needed structures and training the units. Thus, action chains are necessary, but can be individually found by the XCS. Two phases of the experiments were carried out, applying immediate rewards in the first, and delayed rewards in the second phase.

Finally, a series of experiments were ran to measure the overhead of the XCS module in comparison to the overall game performance. These consisted on running the four learning scenarios, and log the duration of an AI cycle and the time spent in an XCS call.

A summary of the most important system parameters used is shown in Table I.

Parameter	Value
Population size	300
Problems	50-200
Crossover method	2-point crossover
Crossover probability	0.8
Mutation probability	0.04
GA Selection policy	Tournament
Don't care probability (P#)	0.75
Learning rate	0.2
θ GA	5

TABLE I
XCS Parameters

V. Results

The experiments were ran in two phases: immediate rewards and delayed rewards. The credit assignment schema implemented in XCSLib does not guarantee success when applied to non-markovian problems, such as the one used in this work. It relies on the causal relationship between state and action, that is, an action's prediction is based on the expected success of its successor states. This relationship is altered in a non-markovian problem, because sequences of states respond not only to a probability distribution, but also to the environment (map, resources, opponents). Thus, the credit assignment mechanism had to be modified, and replaced for a discounted reward policy.

The experiments carried out with delayed rewards were not successful in exhibiting learning. Adjusting the two main parameters of the credit assignment mechanism (window size and discount rate) did not have an impact on the results. For time constraints, it was decided to leave this opportunity open, and concentrate on the immediate reward results, which were showing a good deal of success. Thus, the rest of the results presented here will refer only to the immediate reward mode.

To measure the performance of the agents in a scenario, the XCS runs a number of problems, where each step alternates between learning mode and testing mode. The testing mode gives a measure of the progress of the agent, given that it uses a greedy selection policy for the classifiers, that is, it selects the classifier that offers the highest prediction. As the rulesets evolve, if learning is indeed taking place, the results of these testing-mode actions should improve.

A. Quantitative Analysis: Performance

Figures 2 to 5 show the results of the performance measure through time. The x axis represents the problems ran, where each problem is one Wargus game. Each game is a multistep problem, consisting of 50 steps each on average (ranging from 40 to 60). Thus, 100 problems will represent an average of 5000 learning steps. On the y coordinate, the graph shows the reward in learning mode for each problem. These are shown as a moving average of the last 2 problems (5 in the case of **cbuilding**) The graphics also include a second curve, representing the performance of a random player. This is an agent that performs random actions at every step, and is shown here just as a means of comparison with the XCS-based agent.

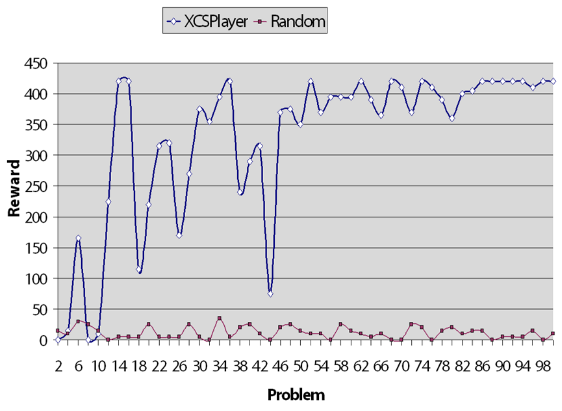

Fig. 2. Peasants Scenario

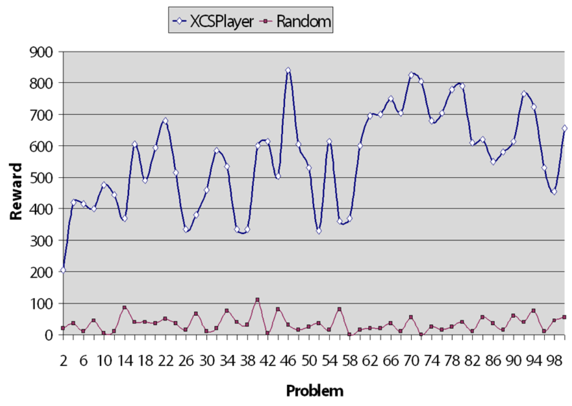

Fig. 4. Archers Scenario

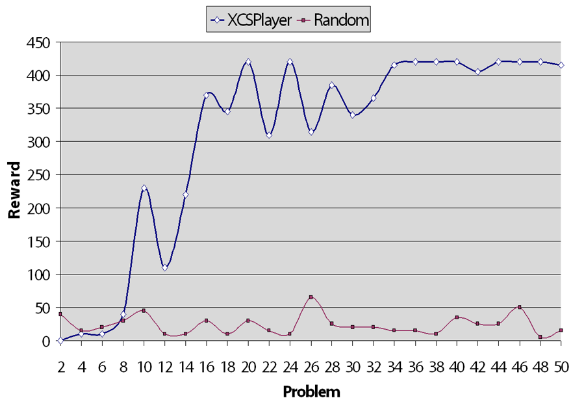

Fig. 3. Footmen Scenario

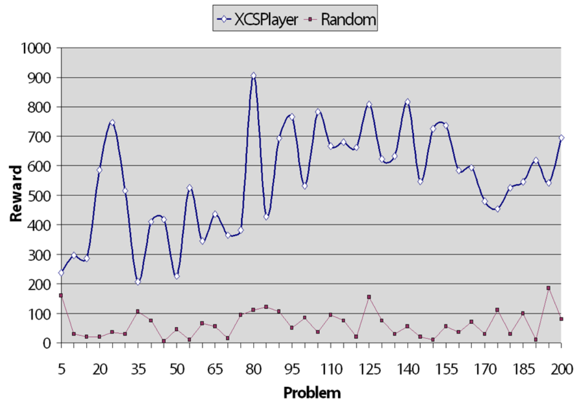

Fig. 5. Counter-building Scenario

All resulting charts show a consistent tendency to improve the agent's rewards. Also, it is quite clear that their performance rapidly differentiates from the random player, which is a first indicative of some degree of learning.

All charts are drawn on a scale that reflects the optimal performance of each scenario. In the **peasants** and **footmen** scenarios, optimal scripts with ideal initial conditions will achieve average rewards between 42 and 45. Since initial conditions are randomized, the optimal performance in each problem will oscillate, with a variation range of around 10%, according to preliminary experiments. In **archers** and **counter-building**, optimal rewards will vary between 800 and 1000. We can see in the charts that the XCS based agents achieve results very close to optimal values, which indicates that the learning goals have been clearly achieved.

More importantly, because the agents have been trained in randomized conditions, they are able to achieve their objective in a wide array of situations. Scripts are typically restricted to a particular set of conditions, which the players can exploit to weaken the agent.

We can also observe in the charts that the rewards stabilize to a great deal in **peasants** and **footmen**, while the other two still exhibit some degree of variation even on advanced stages of learning. This is mostly due to the randomization of the initial conditions. Resources, available buildings and units are all set randomly at the beginning of each problem, and different conditions allow for different scores after a given timeout. Thus, the total score obtained at the end of each game will vary, even for optimal scripts, which largely explains the observed variation in the rewards. It is also likely that the rulesets reach a near-optimal behaviour, and thus achieve a performance slightly lower than an optimal script. This in itself is not a problem, for the objective of the learning is not to perform better than the scripts, but rather be able to perform consistently well in a wide variety of conditions. The results do show a very good degree of consistency in achieving high rewards on the goals set in each scenario.

B. Qualitative Analysis: Rulesets

Studying the rulesets of an XCS is difficult, specially for large rulesets with many parameters. In our case, the resulting populations had 170 to 250 classifiers. Interpreting these sets of rules is not trivial. Here, we present a small subset of the rules for two scenarios: **footmen** and **archers**. Shown are the fittest rules, which are also the most used in the training experiments.

Figure 6 shows the two most important rules in the **footmen** scenario[2]. The first one takes care of training the footmen, and is the action that is more largely rewarded by the scoring function. We can read this classifier as saying: "if there is a certain amount of gold, and barracks are available, train a footman". This is the most simple rationale that can be developed for this goal, and the amount of gold checked for corresponds with the resources needed to execute the training action. The second rule is quite interesting to interpret. Given the nature of the first rule, and others present in the population, this rule will only be selected if the Gold parameter has a value of 001, which is not enough to carry the action of training a footman. Thus, the XCS chooses, through this classifier, to carry a No-Operation action, which in practice will do nothing but wait until the next step. The effect of this combination of rules, is that when the agent has too little gold, it will wait until its peasants gather enough of it. By then, the Gold parameter will change and the first rule will be selected. Other rules in this ruleset take care of building the necessary structures (Town Hall and Barracks) if they are not present.

```
Gol Woo Arc Bal Ftm Kni Psn Tow Br Bsm LuMi St TwnH Action
#1# ### ### ### ### 0## ### ### 1   #   #   #   #   TRAIN_FOOTMAN
##1 ##1 #1# 0## ### #0# ### 1## #   #   #   #   #   NO_OPERATION
```

Fig. 6. Fittest and most used rules in the footmen scenario

```
Gol Woo Arc Bal Ftm Kni Psn Tow Br Bsm LuMi St TwnH Action
##1 ##1 ### 0## ### 0## ### ### 1  #    #    #   #   TRAIN_ARCHER
1#1 ### ### ### ### ### ### ### 1  #    #    #   #   TRAIN_ARCHER
111 ### ### ### ### ### 0## ### 1  #    1    0   #   TRAIN_ARCHER
#11 ##1 ### ### ### 0## ### ### #  1    #    #   #   TRAIN_ARCHER
##1 ### #1# ### ### 0## 0## ### #  1    #    #   #   TRAIN_ARCHER

#11 ##1 ##1 ### ### 0## ### ### #  #    #    #   #   BUILD_L_MILL
#11 ##1 ### ### ### 0## ### ### #  #    #    #   #   BUILD_L_MILL
#11 ### 0#1 ### ### 0#0 ### ### #  #    #    #   #   BUILD_L_MILL
```

Fig. 7. Fittest and most used rules in the archers scenario

In the case of the **archers** scenario, the top rules developed show a very similar pattern. The main action of this scenario is to train archers, which is taken care of by a set of 9 rules. Other rules in this scenario take care of building the needed structures, as shown in Figure 7.

The **peasants** and **counter-building** scenarios show behaviours very similar to the ones just presented in terms of rule-action pairs. In all scenarios, we can see a consistent success in obtaining rewards, and this ability improves with the number of problems used for training.

C. Overhead Study

The measured times for the game AI cycle and XCS calls show that the latest occupy only the 1.6% of the cycle time, which seems like a very reasonable overhead to the overall system.

[2]The parameters shown in the figures have been abbreviated for presentation purposes, and correspond to: Gold, Wood, Archers, Ballistas, Footmen, Knights, Peasants, Towers, Barracks, Blacksmith, Lumber Mill, Stables and Town Hall respectively. The rest of the parameters are omitted for brevity, but are all fully generalized in the resulting rulesets, which is optimal given the learning goals.

VI. Conclusions and Future Work

In this work, we have successfully applied an XCS as the decision making module on a Real Time Strategy Game, which represents a complex, dynamic, non-markovian problem. Four different scenarios were used for the experiments, with different levels of difficulty. In the immediate reward mode, agents trained in these four scenarios developed rulesets that exhibited clear signs of learning. A quantitative analysis shows improvement throughout the training, reaching levels that closely approach those of an optimal script. An examination of the rulesets confirms that the rationale represented by the rule-action pairs makes sense in the context of the learning goals.

On the other hand, the delayed reward mode of the experiment did not show signs of learning in terms of improvement in the agents' rewards. This could be attributed in principle to the various sources of noise in the rewards, an ineffective credit assignment technique, and the arising of parasite rules.

Given that the rulesets were trained in randomized initial conditions, they are forced to explore the search space rather than concentrating on a small set of conditions. Thus, the rulesets are able to adapt to a wide array of conditions and opponent behaviors. A linear game script, the most common representation for artificial agents, is limited in its ability to adapt to its environment. Thus, it is our belief that a decision making schema such as the one applied in this work is of greater value than the existing script solutions.

An implementation such as the one presented here can be expanded for a number of commercial-level games. This would provide a framework to automatically generate strategies without the need to write and tune scripts tailored to each scenario and map. Furthermore, these strategies would be stronger in adapting to a wide array of conditions and user actions.

A. Future Work

The most important line of future work is to continue exploring the problem of delayed rewards. Since games are typically of this nature, it is paramount to handle noise and delayed rewards properly.

- The credit assignment technique needs to be revised, with the goal of performing well with different levels of noise and delayed rewards
- Identify the sources of noise, and make efforts to minimize its effects in the learning process.
- More complex training and testing, where the evaluation functions are simpler and do not need to assign rewards to intermediate states in the action chains.
- Examine other variations of Classifier Systems and compare its performance with XCS. Since parameters in the game are mostly integer by nature, it seems natural to explore XCSI. Seeing the success of FCS in some domains, it is an interesting alternative to examine as well.
- The overhead analisys suggests that online learning would be feasible. This would be an interesting area to explore,

focusing on the performance and stability of the XCS rulesets once the agent starts interacting with the user.

- When implementing larger learning objectives, paralelization could be an interesting proposal. Learning is a processor-intensive phase, and running in a paralelized platform would offer considerable advantages.
- Finally, while using XCSLib, it was clear that its documentation covers only the superficial operations of its main components. There is an steep learning curve when modifying it or even understanding the detailed operation of its components. Improving its documentation would be a task of great value for researchers in the area.

REFERENCES

[1] J. Laird and M. van Lent, "Human-level ai's killer application: Interactive computer games," 2000. [Online]. Available: citeseer.ist.psu.edu/article/laird01humanlevel.html

[2] M. Lewis and J. Jacobson, "Game engines in scientific research - introduction," *Communications of the ACM*, vol. 45, no. 1, pp. 27–31, 2002. [Online]. Available: citeseer.ist.psu.edu/lewis02game.html

[3] A. Nareyek, "Computer games: boon or bane for AI research?" *Knstliche Intelligenz*, pp. 43–44, 2004.

[4] M. Buro, "Real-time strategy games: a new AI research challenge," in *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*. Morgan Kaufmann, 2003, pp. 1534–1535.

[5] R. Miikkulainen, B. D. Bryant, R. Cornelius, I. V. Karpov, K. O. Stanley, and C. H. Yong, "Computational intelligence in games," *Computational Intelligence: Principles and Practice*, 2006.

[6] K. O. Stanley, B. D. Bryant, and R. Miikkulainen, "Real-time neuroevolution in the NERO video game," *IEEE Transactions on Evolutionary Computation*, vol. 9, no. 6, pp. 653– 668, 2005.

[7] N. Cole, "Using a genetic algorithm to tune first-person shooter bots," *Congress on Evolutionary Computation, 2004. CEC2004.*, vol. 1, pp. 139–145, 2004.

[8] T. E. Revello and R. McCartney, "Generating war game strategies using a genetic algorithm," in *CEC '02: Proceedings of the Evolutionary Computation on 2002. CEC '02. Proceedings of the 2002 Congress*. Washington, DC, USA: IEEE Computer Society, 2002, pp. 1086–1091.

[9] J. K. Bassett and K. A. D. Jong, "Evolving behaviors for cooperating agents," in *International Syposium on Methodologies for Intelligent Systems*, 2000, pp. 157–165. [Online]. Available: citeseer.ist.psu.edu/bassett00evolving.html

[10] C. Miles and S. J. Louis, "Towards the co-evolution of influence map tree based strategy game players," *2006 IEEE Symposium on Computational Intelligence and Games*, pp. 75–82, 2006.

[11] P. Spronck, I. Sprinkhuizen-Kuyper, and E. Postma, "Online adaptation of game opponent AI in simulation and in practice," in *Proceedings of the 4th International Conference on Intelligent Games and Simulation*, 2003, pp. 93–100.

[12] M. Ponsen, "Improving adaptive game AI with evolutionary learning," *Proceedings of the International Conference on Computer Games: Artificial Intelligence, Design and Education*, pp. 389–396, 2004.

[13] M. Ponsen, H. Munoz-Ávila, P. Spronck, and D. W. Aha, "Automatically generating game tactics via evolutionary learning," *IAAI-05 : Annual Conference on Innovative Applications of Artificial Intelligence No17*, 2006.

[14] J. Holland, *Adaptation in natural and artificial systems*. University of Michigan Press, 1975.

[15] J. H. Holland, "Escaping brittleness: the possibilities of general-purpose learning algorithms applied to parallel rule-based systems," *Computation & intelligence: collected readings*, pp. 275–304, 1995.

[16] S. W. Wilson, "Classifier fitness based on accuracy," *Evolutionary Computation*, vol. 3, no. 2, pp. 149–175, 1995. [Online]. Available: citeseer.ist.psu.edu/wilson95classifier.html

[17] ——, "Generalization in the XCS classifier system," in *Genetic Programming 1998: Proceedings of the Third Annual Conference*. University of Wisconsin, Madison, Wisconsin, USA: Morgan Kaufmann, 22-25 1998, pp. 665–674. [Online]. Available: citeseer.ist.psu.edu/wilson98generalization.html

[18] M. V. Butz, P. L. Lanzi, X. Llorà, and D. E. Goldberg, "Knowledge extraction and problem structure identification in XCS," *Lecture notes in computer science*, pp. 1051–1060, 2004.

[19] M. V. Butz, D. E. Goldberg, and P. L. Lanzi, "PAC learning in XCS," Illinois Genetic Algorithms Laboratory, Tech. Rep. 2004011, 2004.

[20] P. L. Lanzi, D. Loiacono, S. W. Wilson, and D. E. Goldberg, "Extending xcsf beyond linear approximation," in *GECCO '05: Proceedings of the 2005 conference on genetic and evolutionary computation*. New York, NY, USA: ACM, 2005, pp. 1827–1834.

[21] T. Kovacs and M. Kerber, "Some dimensions of problem complexity for XCS," in *Proceedings of the 2000 Genetic and Evolutionary Computation Conference Workshop Program*, A. S. Wu, Ed., 2000.

[22] ——, "What makes a problem hard for XCS?" *Lecture Notes in Computer Science*, vol. 1996, pp. 80–??, 2001. [Online]. Available: citeseer.ist.psu.edu/518304.html

[23] P. L. Lanzi and S. W. Wilson, "Toward optimal classifier system performance in non-markov environments," *Evolutionary Computation*, vol. 8, no. 4, pp. 393–418, 2000. [Online]. Available: citeseer.ist.psu.edu/lanzi00toward.html

[24] M. V. Butz, T. Kovacs, P. L. Lanzi, and S. W. Wilson, "How XCS evolves accurate classifiers," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, L. Spector, E. D. Goodman, A. Wu, W. B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. H. Garzon, and E. Burke, Eds. San Francisco, California, USA: Morgan Kaufmann, 7-11 2001, pp. 927–934. [Online]. Available: citeseer.ist.psu.edu/article/butz01how.html

[25] E. Bernado, X. Llorà, and J. M. Garrell, "XCS and GALE: a comparative study of two learning classifier systems with six other learning algorithms on classification tasks," *Lecture notes in computer science*, vol. 2321, pp. 115–132, 2001.

[26] S. W. Wilson, "ZCS: A zeroth level classifier system," *Evolutionary Computation*, vol. 2, no. 1, pp. 1–18, 1994. [Online]. Available: citeseer.ist.psu.edu/wilson94zcs.html

[27] A. Orriols and E. Bernadó-Mansilla, "The class imbalance problem in learning classifier systems: a preliminary study," in *GECCO '05: Proceedings of the 2005 workshops on Genetic and evolutionary computation*. New York, NY, USA: ACM, 2005, pp. 74–78.

[28] S. W. Wilson, "Compact rulesets from xcsi," in *IWLCS '01: Revised Papers from the 4th International Workshop on Advances in Learning Classifier Systems*. London, UK: Springer-Verlag, 2002, pp. 197–210.

[29] M. Valenzuela-Rendón, "The fuzzy classifier system: motivations and first results," in *PPSN I: Proceedings of the 1st Workshop on parallel problem solving from nature*. London, UK: Springer-Verlag, 1991, pp. 338–342.

[30] O. Sigaud and S. W. Wilson, "Learning classifier systems: a survey," *Soft Comput.*, vol. 11, no. 11, pp. 1065–1078, 2007.

[31] A. Barry, "Limits in long path learning with xcs," 2003. [Online]. Available: citeseer.ist.psu.edu/barry03limits.html

[32] H. A. Abbass, J. Bacardit, M. V. Butz, and X. Llora, "Online adaption in learning classifier systems: Stream data mining," 2004. [Online]. Available: citeseer.ist.psu.edu/abbass04online.html

[33] S. Y. B. Wong and S. Schulenburg, "Portfolio allocation using xcs experts in technical analysis, market conditions and options market," in *GECCO '07: Proceedings of the 2007 GECCO conference companion on genetic and evolutionary computation*. New York, NY, USA: ACM, 2007, pp. 2965–2972.

[34] J. H. Holmes, "Evolution-assisted discovery of sentinel features in epidemiologic surveillance," Ph.D. dissertation, Drexel University, Philadelphia, PA, USA, 1996.

[35] Z. Zhang, S. Franklin, and D. Dasgupta, "Metacognition in software agents using classifier systems." Menlo Park, CA, USA: American Association for Artificial Intelligence, 1998, pp. 83–88.

[36] O. Pérez, "Aplicación de técnicas de AI emergentes para la generación de estrategias cooperativas en juegos de video," 2004.

[37] G. Robert, P. Portier, and A. Guillot, "Classifier systems as 'animat' architectures for action selection in mmorpg," 2002. [Online]. Available: citeseer.ist.psu.edu/563384.html

[38] T. W. Team, "Wargus," 2002, uRL: http://wargus.sourceforge.net.

[39] P. L. Lanzi, "The XCS library," uRL: http://xcslib.sourceforge.net.

[40] M. Molineaux, D. W. Aha, and M. Ponsen, "Defeating novel opponents in a real-time strategy game," *Reasoning, Representation, and Learning in Computer Games: Papers from the IJCAI Workshop (Technical Report AIC-05-127)*, 2005.