A Zeroth-Level Classifier System for Real Time Strategy Games

Conference Paper · August 2011

DOI: 10.1109/WI-IAT.2011.177 · Source: DBLP

3 authors, including:

Kyriakos Chatzidimitriou
Aristotle University of Thessaloniki
25 PUBLICATIONS **118** CITATIONS
SEE PROFILE

Pericles A. Mitkas
Aristotle University of Thessaloniki
272 PUBLICATIONS **1,634** CITATIONS
SEE PROFILE

A Zeroth-Level Classifier System for Real Time Strategy Games

Michalis T. Tsapanos
Department of Electrical and Computer Engineering
Aristotle University of Thessaloniki
Thessaloniki, Greece, 54124
Email: michael.tsapanos@gmail.com

Kyriakos C. Chatzidimitriou and Pericles A. Mitkas
Department of Electrical and Computer Engineering,
Aristotle University of Thessaloniki
and
Informatics and Telematics Institute,
Centre for Research and Technology Hellas
Email: kyrcha@issel.ee.auth.gr, mitkas@auth.gr

***Abstract*—Real Time Strategy games (RTS) provide an interesting test bed for agents that use Reinforcement Learning (RL) algorithms. From an agent's point of view, RTS games constitute a Markovian, partially observable and dynamic environment with a huge state space. In this paper, we present an agent that uses a Zeroth-level Classifier System (ZCS) in order to construct winning policies for this type of games. We also combine ZCS with the replacing traces method in an attempt to improve the behaviour of our agent. We tested the learning abilities of our agent against a static opponent. For the evaluation of our agent, we compare its results with those of a random-acting agent and an agent that uses the SARSA RL algorithm. Results are encouraging since, our ZCS agent managed to outperform the SARSA agent. On the other hand, applying replacing traces to ZCS did not yield the expected results.**

I. Introduction

Video games of today are becoming increasingly realistic, especially in terms of the graphical representation of the virtual world in which the game is situated. Although both gaming industry practitioners and academics [1], [2] agree on the importance of game Artificial Intelligence (AI), most game companies consider graphics and storytelling their highest priorities and allocate less resources to the implementation of the game AI [3]. As a result, even in state-of-the art games, game AI is generally of inferior quality [4].

In most games, the actions of computer controlled units are selected by applying a rule base or a related technique [5]. Strategies created by these techniques tend to be static, and their strategic weaknesses can be exploited by a human opponent [4]. To increase the challenge and the quality of a game we need a game AI that can be adaptive and capable of automatically changing its strategy against the human player.

Falke and Rose [6] applied Zeroth-level Classifier Systems (ZCS) on Real Time Strategy (RTS) games using the game engine Auran Jet. Their scenarios were limited to twenty-on-twenty combat. Their experiments showed that Learning Classifier Systems (LCS) can produce winning policies by using a simple environment modeling. Ponsen et al. [7] used an evolutionary algorithm to generate tactics automatically. The newly formed tactics performed well against many different static opponents, even unseen ones.

In this paper, we have created an agent capable of playing a RTS game competitively by learning a winning policy against a hand-crafted opponent, using a ZCS. Our agent tries to optimize its financial and military policy, while acting in a partially observable and dynamic environment. In an attempt to further improve the behavior of our *ZCS agent* we also used a combination of ZCS with the RL method of *replacing traces*. To evaluate our agent, we test it against a static opponent and compare its results to those of an agent that implements the SARSA algorithm.

II. Real Time Strategy Games

RTS is a category of strategy games that focus on military combat. The game is situated in a virtual representation of a world called the map. To win, a player has to use his units to achieve financial and military supremacy over his opponent. To grow financially, the player has to collect resources and distribute them appropriately in order to create new units and do research for technological upgrades. These actions will enable the player to create a large army and use it to defeat his opponents. In RTS games, the AI component determines the actions of the units controlled by the computer. From a high-level perspective, these include: a) resource gathering, b) the creation of new units and c) the tactics of the armies in combat. Executing these tasks simultaneously in a real time and partially observable environment is what makes game AI development for RTS games particularly challenging.

For our experiments, we used a simplified version of ORTS game [1], the ORTS-lite, that was used in the 2008 RL-Competition [2]. There are three types of units in the game: marines, workers, and bases. The source of income in this game is the mineral patches. Although simplified, the game retains the main elements of a RTS game. Thus, an agent must be able to distribute its resources correctly and determine the tactics used by the marines in order to win.

III. Zeroth-Level Classifier Systems

A Zeroth-level Classifier System [8] is a learning system that uses RL algorithms along with a Genetic Algorithm (GA) in order to discover an optimal solution for a given problem. A ZCS employs a set, or a population, of rules $\mathbf{R} = \{R_1, R_2 \dots R_N\}$. Each rule, called classifier, consists of a condition and an action part. The condition part represents a state of the environment in which the ZCS acts and is

[1] http://skatgame.net/mburo/orts/
[2] http://2008.rl-competition.org/

encoded over the ternary alphabet 0,1 and #. The # symbol is translated as *"do not care"* and allows for generalization in the condition part as both conditions 11 and 10 are matched by the condition 1#. The # symbol is not used in the action part, as the action chosen by the system must be unique. Each classifier is paired with a scalar variable called $strength$ which expresses the estimated reward of the classifier. Each activation cycle of the ZCS consists of three components: the performance component, the reinforcement component, and the rule discovery component.

A. Performance Component

At each activation cycle of the ZCS, the system receives a binary encoded vector V_t, which represents the current state of the environment. The population of classifiers is scanned and the classifiers whose condition part best matches the input vector V_t form the *match set* $\mathbf{M}$. An active classifier is then selected and the action α that this classifier advocates, is sent to the effector interface of the system. The selection process of the classifier that will advocate the action α can be either deterministic ($detAS = true$), with the selected classifier being the strongest classifier in $\mathbf{M}$, or stochastic ($detAS = false$), in which case each classifier in $\mathbf{M}$ has a selection probability proportional to its $strength$. Finally, the *action set* $\mathbf{A}$ is formed, which contains all the classifiers in $\mathbf{M}$ advocating action α.

B. Reinforcement Component

At each cycle of the ZCS, a portion $\beta \cdot strength$ is subtracted by every classifier in $\mathbf{A}$. The sum S of these portions is distributed as reward to the classifiers of the previous action set $\mathbf{A^{-1}}$, with each classifier receiving $\gamma \cdot S/|\mathbf{A^{-1}}|$. Accordingly, each classifier in $\mathbf{A}$ increases its $strength$ by $\gamma \cdot S^{+1}/|\mathbf{A}|$ where S^{+1} is the reward given by the next step action set, $\mathbf{A^{+1}}$. This mechanism allows the ZCS to back-propagate information about the current estimated reward to past states. If the action α selected by the ZCS leads to a successful completion of its task, an *immediate reward* $\mathcal{R}$ is awarded by the environment to the system. This reward increases the $strength$ of each classifier in $\mathbf{A}$ by $\mathcal{R}/|\mathbf{A}|$. Finally the classifiers in $\mathbf{M} - \mathbf{A}$ are penalized and a portion $\tau \cdot strength$ is subtracted by their strengths.

C. Rule Discovery Component

ZCS utilizes two mechanisms for rule discovery: (i) a *covering operator* and (ii) a *steady state genetic algorithm (GA)*. The *covering operator* is employed when the *match set* $\mathbf{M}$ created in a cycle is empty, or when the $strength$ of each classifier in $\mathbf{M}$ is smaller than a fraction ϕ of the mean $strength$ of the classifier population. In that case a new classifier is created. Its condition part matches the input vector V_t and generalizes by inserting the # symbol at every bit with a probability g, while its action part is selected randomly. The new classifier is inserted in the population. In order to maintain the size of the population, one classifier is also selected to be deleted. The selection is made using probabilities proportional to the inverse strengths of the classifiers.

The *GA* is responsible for the evolution of the population and its purpose is to discover new rules that will provide a better solution to the given problem. The *GA* is invoked at a rate ρ. Upon invocation, two parent classifiers are selected from the population with selection probability proportional to their $strength$. The classifiers are cloned with crossover and mutation operators applied to produce the two off-spring. The $strength$ of the newly created classifiers is set to the mean strength of the parental classifiers. Two classifiers are deleted from the population, their deletion probability being proportional to their inverse strength, and are replaced by the newly created classifiers.

IV. ZCS AGENT IN ORTS-LITE

In this section we will detail the main elements of the ORTS-lite and the implementation of *ZCS agent* .

A. Main Elements of ORTS-Lite

There are three types of units that an agent utilizes to win a game:

1) The *base*, which is the only type of building in the game. Its main purpose is to create new units and to do so it consumes the resources gathered by the workers.
2) The *workers*, whose main task is to gather resources for the agent so as more units can be created by the base.
3) The *marines*, the military units of the game, that defend from and attack the opponent in order to win the game. They can shoot an enemy unit while moving and they will shoot automatically every enemy unit that happens to be inside their attack range unless they are told otherwise.

The game takes place in a 560×560 cell map. In the beginning of every episode each agent begins with a single worker unit, whose primary objective is the construction of the base, placed in a random position and 1000 worth of minerals. There are also ten, randomly placed, mineral patches (the resources of the game). An episode ends when the base of an agent is destroyed. The reward for the winning agent is $\mathcal{R} = 100 - 15t/t_{max}$ where t is current time step and t_{max} is the maximum number of time steps per episode (10000). A tie is possible, in which case the reward points are divided between the opponents.

B. Learning Policies

Our agent features two separate classifier systems. The first system controls the base unit of our agent and is in charge of the unit production policy, while the second assigns tasks to our marine units and is in command of our military tactics.

1) Base ZCS: To learn an optimal policy for unit production, our agent utilizes a ZCS, which is invoked every time the base wants to produce a new unit. The base is never in an idle state but decides its next action as soon as its last action has been completed. If the resources for the chosen action are not available, the base waits until the proper amount is collected and then executes the action. The

TABLE I
BASE ZCS RULE REPRESENTATION.

	Base Classifier Condition Part			
2 bits	00	01	10	11
Marines	≤ 5	6 to 15	16 to 25	> 26
Workers	0	1 to 9	10 to 19	> 20
Enemy Marines	≤ 5	6 to 15	16 to 25	> 26
Enemy Workers	≤ 5	6 to 15	16 to 25	> 26
1 bit	0	1		
Base Threat	No	Yes		
Min Patch Distance	Close	Distant		
	Base Classifier Action Part			
1 bit	0	1		
Action	Create Wor	Create Mar		

TABLE II
MARINE ZCS RULE REPRESENTATION.

	Marine Classifier Condition Part			
2 bits	00	01	10	11
Marines Def Base	≤ 4	5 to 9	10 to 14	> 15
Marines Def Minerals	≤ 4	5 to 9	10 to 14	> 15
Marines Attacking	≤ 4	5 to 9	10 to 14	> 15
Marines Exploring	≤ 4	5 to 9	10 to 14	> 15
Workers	0	1 to 9	10 to 19	> 20
Enemy Marines	≤ 5	6 to 15	16 to 25	> 26
Enemy Workers	≤ 5	6 to 15	16 to 25	> 26
1 bit	0	1		
Base Threat	No	Yes		
Battle Won	No	Yes		
	Marine Classifier Action Part			
2 bits	00	01	10	11
Action	DB	DM	ATT	EXP

environment modeling used by the Base ZCS is shown in Table I.

Some key elements of our environment representation are:

- The representation of the number of own workers differs from that of other units. Having zero workers is a critical state when playing a RTS game, so our agent should learn to create workers when in that state. Otherwise its income will drop to zero and eventually lose the game.
- Our agent perceives the number of enemy units as the maximum number of units observed simultaneously. If an enemy unit is destroyed, the estimated number is decreased by one.
- Informing the agent about the random position of the mineral patches is crucial to the outcome of an episode. A mineral patch is considered to be distant if its distance from the base is larger than 1.5 times the base sight range.

2) Marine ZCS: Learning an optimal policy for assigning tasks to the military units is essential for winning a RTS game. Our agent uses a ZCS to assign tasks to the marines the moment they are created by the base. These tasks are: 1) Defend Base (DB), 2) Defend Minerals (DM), 3) Attack (ATT) and 4) Explore (EXP).

These tasks can be overridden, when a marine spots an unprotected enemy unit, in which case the marine tries to destroy the enemy unit, or when a nearby allied unit is under attack, in which case the marine will assist in its defense. The environment representation used by the Marine ZCS is shown in Table II.

The *Battle Won* bit takes the value 1 when our agent observes that the population of enemy marines is 3 or less, if it has been previously observed to be larger than 10. In essence it indicates that a major battle has been won. When that occurs, the agent reassigns tasks to the marines every 500 time-steps in an attempt to take advantage of the increased number of marines.

C. Replacing Traces

In an attempt to enable ZCS to back-propagate information in classifiers selected in the past more efficiently, we created the *trace-ZCS agent* that uses the notion of *replacing traces* [9].

Every classifier of the population is paired with a scalar variable e, called trace. The trace for each classifier is set to 0 at the beginning of every episode. After an action has been chosen and the action set $\mathbf{A}$ has been formed, the immediate reward $\mathcal{R}$ (if its value is not 0) and the discounted reward from the action set $\mathbf{A}^{-1}$ are summed and divided by the size of $\mathbf{A}$. The resulted value is appointed to the δ variable. Then, for every classifier R_i in $\mathbf{A}$, the value of δ is reduced by $strength(R_i)/|\mathbf{A}|$ and the trace of every classifier is set to 1. Finally, for every classifier in the population, a fraction $e_i \cdot \beta$ of δ is added to every classifier's strength and the trace for every classifier decays by $\gamma \cdot \lambda$.

Replacing traces give ZCS the ability to create chains of cooperative rules, which could prove valuable to achieve high-end behavior in multi-step tasks such as RTS games.

V. EXPERIMENTS

To evaluate our agent we set it against a simple hand-crafted opponent (part of the ORTS-Lite platform) and we compared its results with those of a randomly acting agent, called random agent, and an agent that uses the SARSA algorithm. The SARSA agent, created by Marc Lanctot and modified by Marc G. Bellemare for the RL-Competition, uses two different SARSA algorithms, one controlling the marines and one controlling the base. For a description of the SARSA algorithm see [9].

The *ZCS agent* , the SARSA agent and the random agent, execute the actions they make in the exact same way. This means that a marine ordered to defend his base will defend it in the same way in all three agents. This fact allows us to better compare the performance of our agent.

Each experiment conducted consists of 37.5 million time-steps according to the rules of the 2008 RL-Competition. To evaluate the performance of an agent, we keep track of the

TABLE III
ZCS PARAMETERS THROUGH CONDUCTED EXPERIMENTS.

Parameter	Description	Value
N	Number of Classifiers (Marines-Base)	400-100
$detAS$	Deterministic action selection	True
S	Initial rule strength	20
g	Generalization probability	0.33
β	Learning rate	0.2
γ	Discount factor	0.71
τ	Tax for classifiers in $\mathbf{M} - \mathbf{A}$	0.1
ρ	GA invocation rate	0.0005-0.008
χ	Crossover probability	0.5
μ	Mutation probability	0.002
ϕ	Covering invocation threshold	0.5

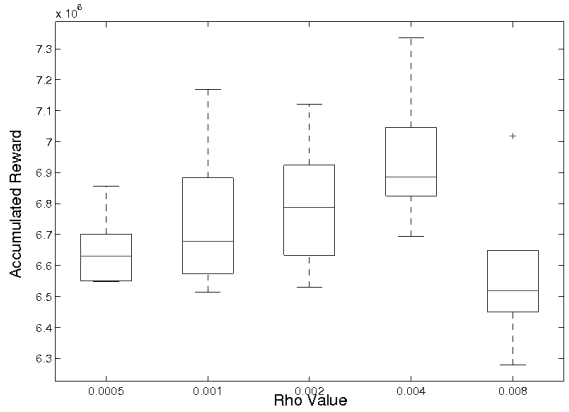

Fig. 1. Boxplot of the Accumulated Rewards of ZCS agent for Different Values of the ρ Parameter.

accumulated award which is the sum of rewards our agent receives from the environment.

A. Achieving Optimal Performance

Table III summarizes the parameter values for our ZCS agent used through our initial experiments.

Our initial experiments focused on finding the optimal value of the ρ parameter, which determines the frequency of the genetic algorithm invocations. The results of these experiments are presented to Fig 1 and indicate that the optimal ρ value is 0.004.

B. Comparing ZCS, SARSA and Random Agents

Having established an optimal ρ value, we evaluated the performance of our agent by comparing it to the performance of the SARSA and the random agent. The results of the experiments are presented in Table IV. The ZCS agent seems to outperform the other agents. It achieves a greater mean reward and small standard deviation, which establishes it as the most stable. The large standard deviation of the other agents is the reason why we conducted more experiments for them.

C. Replacing Traces

For the experiments of our trace-ZCS agent, we kept the parameter values equal to those of the ZCS agent except for the learning rate parameter β, which was set to 0.01. For larger values of β the $strength$ of all the classifiers decays to 0. The trace decay ratio variable λ was set to 0.9.

As illustrated in Table IV, the trace-ZCS agent, stable as it may be, did not manage to outperform the ZCS agent. The lower performance of the trace-ZCS agent is probably due to the greater impact that the randomization of the environment has to its learning process.

TABLE IV
EXPERIMENT RESULTS FOR THE ZCS, SARSA, RANDOM AND TRACE-ZCS AGENTS.

Statistics (presented in thousands)	**Agent**			
	Random	SARSA	ZCS	trace-ZCS
Num. of Experiments	10	10	5	5
Mean Reward	4026	4975	6946	6145
Standard Deviation	961	1660	236	266

VI. CONCLUSIONS AND FUTURE WORK

In conclusion, our ZCS agent appears to outperform the SARSA agent and manages to maintain a stable performance. The adaptation of an initial random set of rules through ZCS and repetitive gameplaying against a static opponent created policies that dominated the static rule-based player. Its performance peaks for $\rho = 0.004$ and has the potential to be improved even further with some fine-tuning of its parameters values. The trace-ZCS agent did not exhibit the expected performance and by observing it while playing we can report that it tends to favour one action over the others. A more in-depth tuning of its parameters will hopefully yield better performance. In the future, besides attempting to maximize the performance of our agent, it would be interesting to test it against different opponents and also to investigate its performance on a more standardized environment. We also plan to have our agent play against itself, where we will hopefully observe an "arms race".

REFERENCES

[1] J. E. Laird, "It knows what you 're going to do: Adding anticipation to a quakebot," in *Proceedings of Fifth International Conference on Autonomous Agents*, 2001, pp. 385–392.

[2] S. Rabin, *AI Game Programming Wisdom 2*. Charles River Media, 2004.

[3] A. Nareyek, "Artificial intelligence in computer games - state of the art and future directions," *ACM Queue*, vol. 1, no. 10, pp. 58–65, 2004.

[4] J. Schaeffer, "A gamut of games," *AI Magazine*, vol. 22, no. 3, pp. 29–46, 2001.

[5] S. Rabin, *AI Game Programming Wisdom*. Charles River Media, 2002.

[6] W. J. Falke and P. Rose, "Dynamic strategies in a real-time strategy game," 2003, pp. 1920–1921.

[7] M. Ponsen, H. Munoz-Avila, P. Spronck, and D. W. Aha, "Automatically generating game tactics through evolutionary learning," *AI Magazine*, vol. 27, no. 3, pp. 75–84, 2006.

[8] S. W. Wilson, "Zcs: A zeroth level classifier system," *Evolutionary Computation*, vol. 2, no. 1, pp. 1–18, 1994.

[9] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press, 1998.