2014 IEEE Congress on Evolutionary Computation (CEC)
July 6-11, 2014, Beijing, China

Generalized Classifier System: Evolving Classifiers with Cyclic Conditions

Xianneng Li, Wen He, and Kotaro Hirasawa
Graduate School of Information, Production and Systems, Waseda University, Japan
Email: {sennou@asagi., hewen@toki., and hirasawa@}waseda.jp

***Abstract*—Accuracy-based XCS classifier system has been shown to evolve classifiers with accurate and maximally general characteristics. XCS generally represents its classifiers with binary conditions encoded in a ternary alphabet, i.e., $\{0, 1, \#\}$, where $\#$ is a "don't care" symbol, which can match with 0 and 1 in inputs. This provides one of the foundations to make XCS evolve an optimal population of classifiers, where each classifier has the possibility to cover a set of perceptions. However, when performing XCS to solve the multi-step problems, i.e., maze control problems, the classifiers only allow the agent to perceive its surrounding environments without the direction information, which are contrary to our human perception. This paper develops an extension of XCS by introducing cyclic conditions to represent the classifiers. The proposed system, named generalized XCS classifier system (GXCS), is dedicated to modify the forms of the classifiers from *chains* to *cycles*, which allows them to match with more adjacent environments perceived by the agent from different directions. Accordingly, a more compact population of classifiers can be evolved to perform the generalization feature of GXCS. As a first step of this research, GXCS has been tested on the benchmark maze control problems in which the agent can perceive its 8 surrounding cells. It is confirmed that GXCS can evolve the classifiers with cyclic conditions to successfully solve the problems as XCS, but with much smaller population size.**

I. Introduction

Accuracy-based XCS classifier system [1], [2], [3], [4] has been shown to evolve classifiers with accurate and maximally general characteristics. Different from strength-based learning classifier systems (LCSs) [5], [6] whose shortcomings appear due to the definition of classifier fitness via its estimated payoff which sometimes does not distinguish between accurate and over-general classifiers, XCS escapes the dilemma by developing an accuracy-based fitness scheme to adapt the evolution pressure towards accurate generalized classifiers based on reinforcement learning (RL) [7] concept. A number of machine learning problems, i.e., single-step classification problems [8], [9], [10], function approximation [11], [12] and multi-step RL problems [13], [14], have been successfully solved by XCS. In these problems, XCS is capable of achieving the optimal performance with a minimal representation of the optimal solutions evolved.

Despite many other extensions to improve the expression ability of the classifiers [15], [16], [17], [18], in order to perform its *generalization* ability, XCS follows the other LCSs [5], [6], [9], [10] to represent its classifiers with binary conditions encoded in a ternary alphabet, i.e., $\{0, 1, \#\}$, where $\#$ is a wildcard representing "don't care", which can match with 0 and 1 in inputs. This provides one of the foundations to make XCS evolve the optimal population of classifiers, where each classifier has the possibility to cover a set of perceptions.

However, when applying XCS to solve the multi-step problems, i.e., maze control problems (the details of the maze problems are described in section IV), the ternary conditions of the classifiers are generally matched with the real perceived environments in a form of *chains*. For example, in a maze problem that the agent can judge its 8 adjacent cells as the perceived environments, if an environment $e_1, e_2, ..., e_8$ is perceived, a classifier with condition $c_1, c_2, ..., c_8$ is considered to match with the environment only if each specific attribute matches with the corresponding cell, i.e., $c_1 = e_1$ (north cell), $c_2 = e_2$ (northeast cell), and so on. In other words, the condition of each classifier is treated as a chain. However, this form of configurations is contrary to our human perception, since it does not consider the direction of the agent.

With the directional information, the classifiers do not need to match with the perceived environment using the one-to-one correspondence of each cell/attribute. Instead, each classifier is capable of matching with more environments based on different directions of the agent. This would potentially increase the generalization ability comparing with standard XCS only performing with the ternary alphabet, since the population size of the system can be reduced.

Based on this concept, this paper proposes an extension of XCS by introducing cyclic conditions to represent the classifiers, named generalized XCS classifier system (GXCS). To consider the directional information, GXCS modifies the forms of the classifiers from *chains* to *cycles*, which allows them to match with more adjacent environments perceived by the agent from different directions. Accordingly, with a single classifier of GXCS, higher expression ability is guaranteed comparing with standard XCS.

The rest of the paper is organized as follows: In the next section, XCS is briefly described. Section III describes the implementation of GXCS in details. In section IV, the experimental study is presented under several benchmark maze problems. Finally, the conclusions are presented.

II. XCS in Brief

Formally, XCS treats the classifiers as the individuals of genetic algorithm (GA), where each classifier is interpreted as a decision-making rule as

$$\overbrace{C_1\ C_2\ ...\ C_L}^{\text{condition}} \rightarrow \overbrace{A}^{\text{action}} . \quad (1)$$

$\mathbb{C} = [C_1\ C_2\ ...\ C_L]$ is the *condition* with a set of L attributes specified by a given problem. Each attribute C_i's value c_i is selected from the ternary alphabet $\{0, 1, \#\}$. The *action* A is denoted by a numeric value a from the action space $\mathbb{A}$, which is the recommended consequence based on the condition.

XCS maintains a population of classifiers, denoted by $[P]$, to represent the solution. 4 main parameters are associated with each classifier cl:

- Prediction $cl.p$: the estimated payoff when cl is used.
- Prediction error $cl.\varepsilon$: the estimated error of $cl.p$.
- Fitness $cl.F$: the fitness of cl.
- Numerosity $cl.num$: the number of copies of cl in $[P]$.

XCS mainly consists of three components for decision making: (1) performance; (2) reinforcement; (3) discovery.

A. Performance Component

XCS interacts with the environment as a RL agent. At each step t, the system perceives the current environment, denoted by the current state s_t. With the perceived environment, XCS forms *match set* $[M]$ that consists of a set of classifiers whose conditions completely match with s_t. If some possible actions are not presented in $[M]$, *covering* mechanism takes place to generate classifiers with conditions matching with s_t and the missing possible actions. In the covering classifier, each attribute in its condition has probability $P_\#$ to be set to $\#$. Covering mechanism ensures the evolution of XCS towards a complete mapping in any state to predict the effect of every possible action.

After determining the match set $[M]$, XCS computes the system prediction $P(a)$ for each possible action a by the fitness-weighted average over all classifiers whose action is a. Let $[M]_a$ be a subset of $[M]$ that contains classifiers advocating action a. $P(a)$ can be calculated by:

$$P(a) = \frac{\sum_{cl \in [M]_a} cl.p \times cl.F}{\sum_{cl \in [M]_a} cl.F}. \tag{2}$$

For all possible actions $\mathbb{A}$, $\{P(a) | a \in \mathbb{A}\}$ form a *prediction array*.

Based on the prediction array, XCS selects an action a^* to perform using some strategies, i.e., greedy selection ($a^* = \arg\max_{a \in \mathbb{A}} P(a)$), roulette-wheel selection ($a^* = a$ with probability $P(a) / \sum_{a' \in \mathbb{A}} P(a')$) or ϵ-greedy policy (with probability ϵ to select a random action and probability $1 - \epsilon$ to select the greedy action).

The selected action a^* is performed in the environment and possibly a reward r can be obtained. The classifiers in $[M]$ that advocate a^* form the *action set* $[A]$.

After one cycle of the above performance component, the step is incremented, that is, $t = t + 1$.

B. Reinforcement Component

After one step t, the prediction array is sent back to the action set $[A]_{t-1}$ of one step before, i.e., $t - 1$. An estimated payoff P of step $t - 1$ is calculated as follows:

$$P = r_{t-1} + \gamma \max_{a \in \mathbb{A}} P(a), \tag{3}$$

where r_{t-1} is the reward obtained by action set $[A]_{t-1}$, γ is the discount factor, and $P(a)$ is the prediction array of step t.

Afterwards, three parameters $cl.p$, $cl.\varepsilon$, and $cl.F$ of classifiers $cl \in [A]_{t-1}$ are updated. For the classifier prediction, it is updated by:

$$cl.p \leftarrow cl.p + \beta\,(P - cl.p)\,, \tag{4}$$

where β is the learning rate. The prediction error $cl.\varepsilon$ is updated by:

$$cl.\varepsilon \leftarrow cl.\varepsilon + \beta\,(|P - cl.p| - cl.\varepsilon)\,. \tag{5}$$

The fitness $cl.F$ is calculated in a relatively complicated way. First, the classifier *accuracy* $cl.\kappa$ is calculated by:

$$cl.\kappa = \begin{cases} \alpha \left(\frac{cl.\varepsilon}{\varepsilon_0}\right)^{-\nu} & \text{if } cl.\varepsilon > \varepsilon_0, \\ 1 & \text{otherwise,} \end{cases} \tag{6}$$

where ε_0 are the threshold of acceptable prediction error; α and ν are the decline parameters of κ when $cl.\varepsilon > \varepsilon_0$. The *relative accuracy* $cl.\kappa'$ is then assigned with respect to $[A]_{t-1}$:

$$cl.\kappa' = \frac{cl.\kappa}{\sum_{cl' \in [A]_{t-1}} cl'.\kappa}. \tag{7}$$

Finally, the classifier fitness $cl.F$ is updated towards its $cl.\kappa'$:

$$cl.F \leftarrow cl.F + \beta(cl.\kappa' - cl.F). \tag{8}$$

The basic concept of the reinforcement component is derived from RL [7], where the Q-Learning like method is applied to update the three learning parameters.

C. Discovery Component

Niche GA is applied to evolve the action set $[A]$ when its classifiers, i.e., $cl \in [A]$, have averagely remained in the population with more than θ_{GA} learning instances since last GA performs on each of them. In other words, only after the classifiers in $[A]$ have been sufficiently learned by RL, GA will be activated to evolve them.

GA selects two classifiers as the parents with probability proportional to their fitness. Two offspring are generated by crossover and mutation. Different from canonical GA which replaces parents by offspring in the new population, XCS allows the parents to stay in the population competing with their offspring.

D. Summary of XCS

Besides the above three main components, 2 additional techniques contribute to the performance of XCS, including *macroclassifier* and *classifier deletion*.

In XCS, if a newly generated classifier is found to be identical to an existing classifier cl in population $[P]$, it is not inserted into $[P]$. Instead, the *numerosity* parameter $cl.num$ of cl is incremented by one, that is, $cl.num = cl.num + 1$. This

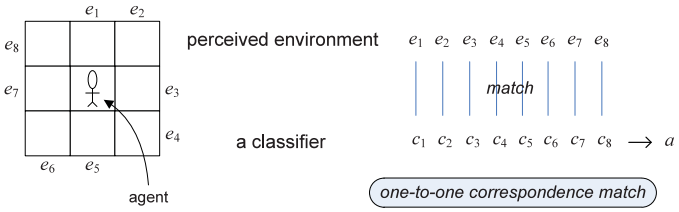

Fig. 1: Perception and matching of XCS in the maze problems

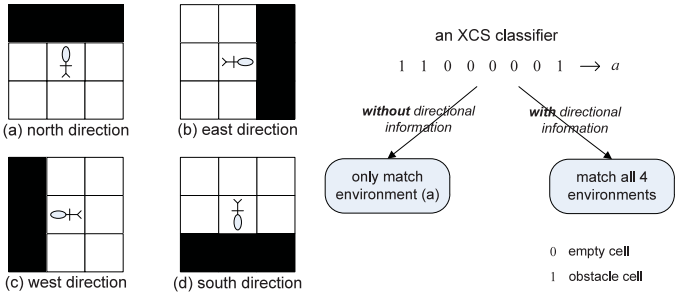

Fig. 2: Perception and matching with and without directions

technique, called macroclassifier, is used to speed up processing and provide a more perspicuous view of XCS to evolve accurate, and maximally general classifiers. The summation of $cl.num$ for all classifiers is equal to the population size N.

When GA or covering mechanism is performed, the population $[P]$ may exceed threshold N. Classifier deletion is carried out to delete excess classifiers. The basic concept of classifier deletion is to delete one classifier if it is sufficiently learned by RL and its fitness is significantly lower than the average fitness of $[P]$. Actually, when a classifier cl is selected to be deleted, its numerosity is decremented by one, i.e., $cl.num = cl.num - 1$, where it is deleted from $[P]$ only if its $cl.num = 0$.

III. Generalized XCS Classifier System (GXCS)

A. Problem Description

When applying XCS to solve the multi-step problems, i.e., maze control problems, each attribute generally denotes a sensory function for the agent to perceive a specific location of its surrounding environments. More precisely, the conditions of the XCS classifiers are represented by 8 attributes in association to its 8 adjacent cells in the grid worlds of maze, where the first attribute C_1 denotes the north cell, and the others are named clockwise. It is notable that when applying XCS to solve this problem, the classifiers are matched with the perceived environments in an one-to-one correspondence way. For instance, if the current perceived environment is $e_1, e_2, ..., e_8$, a classifier with condition $c_1, c_2, ..., c_8$ is considered to match with the environment only if each specific attribute matches with the corresponding cell, i.e., $c_1 = e_1$, $c_2 = e_2$, and so on, as an example shown in Fig. 1.

However, this form implies that the direction of the agent is not considered when performing the matching procedure. It is contrary to our human perception, as well as many real-world applications, that the agent perceives the partial environments with directional information. In other words, the knowledge of the agent, represented as classifiers in XCS, should be more realistically designed to have higher expression ability to reflect the environments.

If we consider the directional information, the classifiers do not need to match with the perceived environment using the one-to-one correspondence of each cell/attribute. But instead, each classifier can match with more environments based on different directions of the agent. As an example shown in Fig. 2, we suppose that there is an XCS classifier denoted as $1\ 1\ 0\ 0\ 0\ 0\ 0\ 1 \rightarrow a$ [1], where 0 denotes that the cell is empty and 1 represents that the cell has obstacle. Originally, XCS can only allow this classifier to match with the perceived environment (a) in Fig. 2, since only environment (a) can match with the condition of this classifier in a one-to-one correspondence way. However, we can easily understand that if we consider the direction of the agent, actually the classifier should also match with environment (b), (c) and (d). This is consistent with our human cognition. Therefore, if we consider the directional information, the required population size of the classifiers could be significantly reduced, which will potentially benefit the generalization of XCS. If 4 directions are considered, i.e., north, east, south and west, the upper bound of the population size could be reduced by 4 times comparing with standard XCS.

This inspires the work of our paper to introduce an extension of XCS that evolves classifiers with directional information, named generalized XCS classifier system (GXCS).

B. Outline of GXCS

GXCS works basically as standard XCS. The most important difference arises from the *knowledge representation* of GXCS, i.e., the structure of the classifiers, where each classifier is encoded with a cyclic condition, which can efficiently adapt to the environments with different directions.

On the other hand, in order to efficiently learn and evolve the classifiers, some modifications and extensions of standard XCS are proposed in GXCS, which mainly include the *matching procedure*, *reinforcement component*, *genetic operation* and *direction subsumption*. The details will be described in the following parts of this section.

C. Knowledge Representation

Differing from standard XCS with chain conditions (as shown in Fig. 1), GXCS encodes the classifiers with cyclic conditions. The cyclic conditions, as shown in Fig. 3, can efficiently suit the agent considering the directional information (refer to the details in the next section about matching procedure).

When performing the classifiers with a specific direction, the actions should also be slightly modified in order to

[1]It is a simplified example of the simulated maze problem for better understanding, where in fact each cell is encoded with 2 or 3 bits rather than 1 bit.

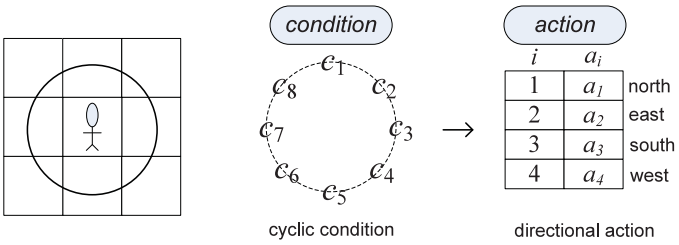

Fig. 3: GXCS classifier with cyclic conditions

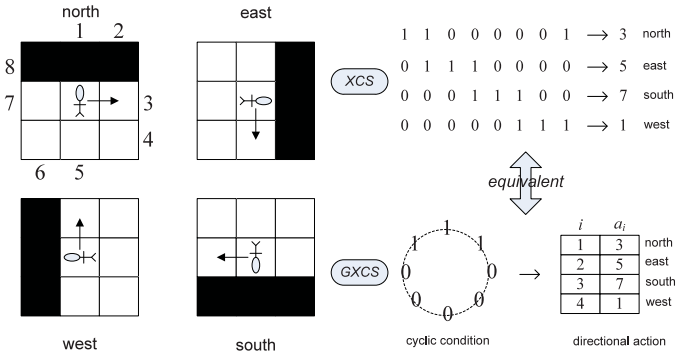

Fig. 4: Relation between the classifiers of XCS and GXCS

match with the performed direction. The reason is obvious that the classifier should be identical in different directions. Accordingly, each classifier of GXCS has 4 directional actions corresponding to the 4 directions, as shown in Fig. 3. However, we actually only need to store the action for the north direction, i.e., a_1 in Fig. 3, where the hereafter directional actions can be computed by just rotating 2 cells. We suppose i is the direction, and $i = 1, 2, 3$ and 4 for the north, east, south and west direction, respectively. Let a_i denote the specific action of direction i, we have

$$a_i = \Big[a_1 + 2(i-1)\Big]\%8, \tag{9}$$

where a_1 is the action for the north direction.

It is explicit that with the cyclic conditions and directional actions, 1 single GXCS classifier is equivalent to 4 XCS classifiers, as an example shown in Fig. 4. Accordingly, GXCS has potential to evolve a more compact population of classifiers comparing with standard XCS, where a smaller number of population size can be expected.

D. Matching Procedure

Comparing with the one-to-one correspondence matching of XCS, GXCS employs a slightly different matching procedure. If an environment $e_1, e_2, ..., e_8$ is perceived, each GXCS classifier is matched with the environment 4 times based on the perception of 4 different directions. In other words, the GXCS classifiers have more chance to match with the environment comparing with the XCS classifiers.

However, with the cyclic conditions, the direction-based matching can be carried out quite efficiently, where we only need to specify the first attribute to be compared with the perceived environment $e_1, e_2, ..., e_8$ in a clockwise way. After

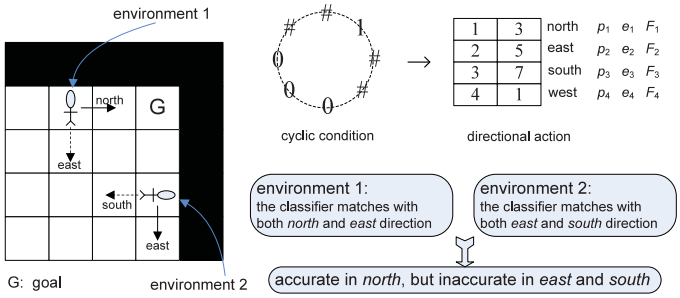

Fig. 5: Perceptual aliasing problem of GXCS classifiers

the matching of one direction is finished, the cyclic conditions are just rotated 2 attributes to define a new first attribute for the next direction. For example, the first attribute is set at c_1 when matching with the environment in the north direction, while it is modified to c_3 when matching with the east direction.

E. Reinforcement Component

Since each classifier can match with the environment with 4 directions, it might be possible that a classifier can match with the environment using more than one direction for a perceived environment. More importantly, when matching with different directions, the performance of the classifier could be different, where it can provide an accurate action in one direction, but tends to be inaccurate in another direction. As an example shown in Fig. 5, a classifier can match with environment 1 in both north and east directions, and with environment 2 in both east and south directions. However, it is only accurate in environment 1 in north direction, but tends to be inaccurate in the other two directions. This circumstance can be explained by the *perceptual aliasing problem* [19] in non-Markovian situations, where the different environments can be perceived as the same by the GXCS classifiers.

In order to solve this problem, we modify the standard XCS by giving the specific prediction p, prediction error ε and fitness F to each direction of each classifier. In other words, associated with each GXCS classifier, there are 4 groups of learning parameters corresponding to each direction, rather than a single group in the XCS classifier (as shown in Fig. 5). Accordingly, RL is capable of automatically determining which directional matching should be preferred for a particular environment, which will be directly shown up in the three learning parameters. From this point of view, the directions in the directional action, i.e., 1 (north), 2 (east), 3 (south) and 4 (west) shown in Fig. 4, work in a similar way as the internal memory register which was proposed in XCS with internal memory (XCSM) [19], [20].

The learning procedure of GXCS to update these parameters is the same as XCS.

F. Genetic Operation

As standard XCS, GXCS applies niche GA in its action set $[A]$ to generate new offspring. Two classifiers, i.e., cl_1 and cl_2, are selected as the parents with the probability proportional to their fitness. However, since each classifier has 4 fitness values

corresponding to different directions, we use the particular directional fitness in which it is performed in the action set. That is, for a particular classifier cl, if it matches with the current environment from the north direction in $[A]$, its fitness $cl.F_1$ corresponding to the north direction is used for selection. cl_1 and cl_2 are crossed with crossover rate χ, and mutation with rate μ is applied to the two generated crossover offspring, denoted by $cl_1^{'}$ and $cl_2^{'}$.

However, differing from XCS which assigns the average p, ε and F of the two parents to the two generated offspring, GXCS prepares the initial p, ε and F of the offspring based on the similarity between them and their parents. The Hamming distance between offspring $cl_1^{'}$ and the two parents cl_1 and cl_2 are first computed, denoted as D_1 and D_2, respectively. After that, its initial prediction $cl_1^{'}.p$ is assigned by

$$cl_1^{'}.p = \frac{D_2}{D_1 + D_2} cl1.p + \frac{D_1}{D_1 + D_2} cl2.p. \tag{10}$$

The initial ε and F are computed similarly,

$$cl_1^{'}.\varepsilon = \frac{D_2}{D_1 + D_2} cl1.\varepsilon + \frac{D_1}{D_1 + D_2} cl2.\varepsilon. \tag{11}$$

$$cl_1^{'}.F = \frac{D_2}{D_1 + D_2} cl1.F + \frac{D_1}{D_1 + D_2} cl2.F. \tag{12}$$

Based on the empirical studies, it is found that the similarity-based parameter assignment can provide slightly faster convergence speed than the original average assignment of XCS in the GXCS scenario.

G. Direction Subsumption

In standard XCS, two classifiers are considered to be identical if and only if both of them have the same conditions and the same action. If two identical classifiers are found in XCS population, only one copy is remained, but its numerosity num is incremented by 1. On the other hand, GXCS is capable of representing the classifiers with directional information. Therefore, two classifiers are compared in a different way with more chance to be identical to each other. That is, two classifiers are compared with each other from 4 different directions. Accordingly, GXCS classifiers have 4 times higher possibility to find the identical classifiers than XCS. If two classifiers are found to be identical, even with different directions, still only one copy remains in the population and the numerosity num is incremented by 1. This procedure, which is called *direction subsumption*, allows GXCS to potentially generate compact population with a smaller number of classifiers.

IV. Experimental Study

In order to testify the performance, GXCS is applied to the benchmark maze control problems [1], [13].

A. Maze Problems

In maze problems, the artificial animal (named *animat*) is located in a grid world with cells. The animat is designed with 8 sensor attributes to recognize its 8 surrounding adjacent cells, and can select 8 actions to move to each of its adjacent cells.

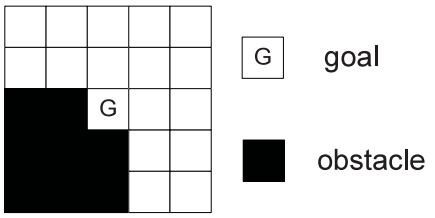

Fig. 6: `Woods1` world

In the grid world, each cell can be occupied with obstacles, floors or a goal. The animat can be randomly placed in the grid world, and it must learn to find the optimal route to reach the goal cell. The target of the classifier systems to solve this problem is to evolve and learn the optimal population of classifiers that can control the animat to reach the goal cell with the fewest number of steps no matter where it is initially placed. Therefore, the performance of this problem can be represented by the number of steps to reach the goal cell (*steps to goal*).

B. Experimental Configurations

When performing each experimental run, exploration and exploitation strategies are used. That is, the system interactively carries out the exploration instance and exploitation instance. In the exploration instance, the animat is randomly placed in the maze world, where the system applies the population $[P]$ of classifiers to control the movement of the animat where the action is selected to be executed by ϵ-greedy policy with the probability of $\epsilon = 0.3$ at each step. Reinforcement component and GA are activated to learn and evolve $[P]$. An exploration instance is finished if the animat reaches the goal cell or a predefined maximal steps, i.e., 2000 in this paper, are reached. After one exploration instance, the exploitation instance is executed. In the exploitation instance, the animat is re-placed in a random start cell, but learning and evolution are blocked, where the population $[P]$ is applied to control the animat using the greedy selection for the action determination.

The exploitation instances are dedicated to evaluate the performance of the classifier systems, where the results shown in this paper is the moving average of the number of steps to the goal cell in the last 50 exploitation instances.

In order to evaluate the performance of GXCS, 3 benchmark maze worlds are considered in this paper: `Woods1`, `Woods2` and `Maze5`, which have been widely studied in the previous XCS research [1], [6], [13], [21]. The evaluation for more complex worlds and systems, i.e., Tileworld testbed [22], [23] remains as our future work.

The comparison between GXCS and standard XCS are presented. All the experimental results presented in this paper are averaged over 30 independent runs in order to remove the random bias of the evolution.

C. `Woods1` World

`Woods1` world is a fairly simple problem that consists of 5×5 grid of cells as shown in Fig. 6. `Woods1` does not consist

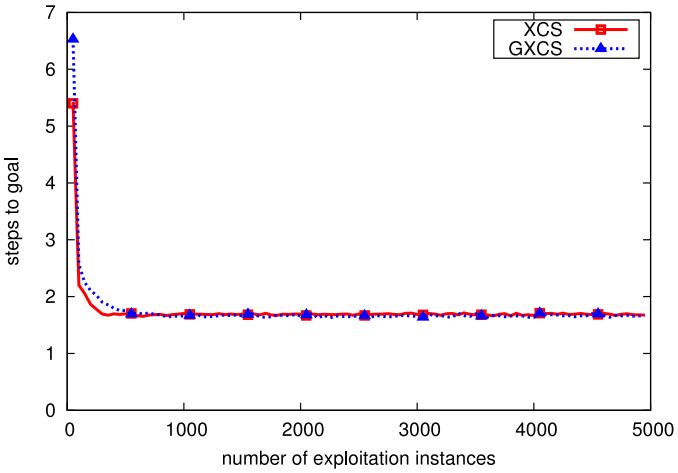

Fig. 7: Performance of XCS and GXCS in `Woods1` world

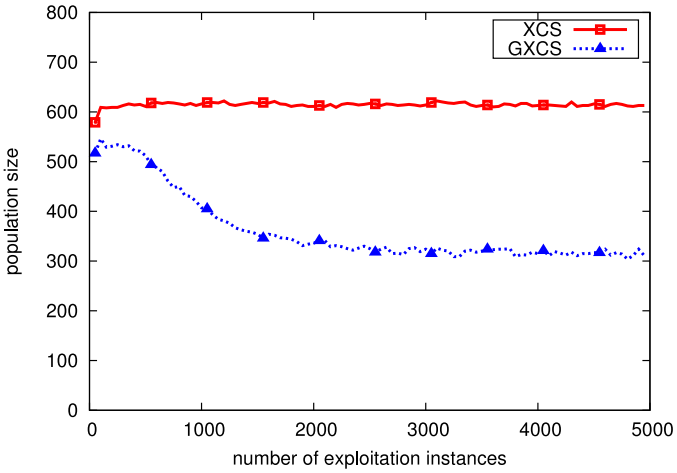

Fig. 8: Population size of XCS and GXCS in `Woods1` world

of boundary, and therefore, when the animat is positioned in the edge cells, it perceives its surrounding environment by adding the cells from the opposite edge as its adjacent cells, like re-entering the world from the opposite edge. It is observed that the *optimal steps* to find the goal cell is 1.7 steps in this problem.

In `Woods1`, each cell is encoded with 2 bits, where "00" denotes empty cell, "01" represents obstacle cell and "11" is goal cell. Therefore, the length of the conditions in each classifier is $2 \times 8 = 16$.

The parameter settings are carried out based on the suggestions of the previous studies [4], [24], [25]. That is, the maximum population size $N = 800$ classifiers (summation of numerosity of each classifier); the learning rate $\beta = 0.2$; the discount factor $\gamma = 0.7$; the error threshold $\varepsilon_0 = 10$; the decline parameters of the classifier accuracy $\alpha = 0.1$, $\nu = 5$; $P_{\#} = 0.6$ in covering mechanism; the crossover rate $\chi = 0.8$ and the mutation rate $\mu = 0.04$; the GA threshold $\theta_{GA} = 25$. All these parameters of XCS and GXCS are the same when performing the experiments. Each experimental run is finished when 5000 exploitation instances are executed.

The performance of XCS and GXCS is reported in Fig. 7. Since this world is very simple, both of XCS and GXCS can easily find the optimal performance around 1.7 steps. However, it is found that XCS is actually slightly faster than GXCS. The reason is that GXCS actually requires to maintain more learning parameters than XCS, since each GXCS classifier includes 4 groups of p, ε and F corresponding to 4 different directions. Therefore, a larger number of learning instances are required to achieve the convergence. Even though, the experimental results confirm that GXCS can converge fast to the optimal performance by learning and evolving the classifiers with cyclic conditions and directional actions.

However, when comparing the population size, i.e., the number of unique classifiers in the population $[P]$, it is found that GXCS can significantly reduce the population size when comparing with XCS, as shown in Fig. 8. The results confirm that GXCS classifiers can ensure higher expression ability to match with more environments using the cyclic conditions and directional actions which make the population $[P]$ not require too much classifiers. As discussed previously, the theoretical upper bound of the population size could be reduced by 4 times in GXCS comparing with XCS, since each GXCS classifier is equivalent to 4 different unique XCS classifiers. Though this upper bound will not be achieved practically, we do find that GXCS requires only around half population size than XCS in `Woods1` to achieve the optimal performance.

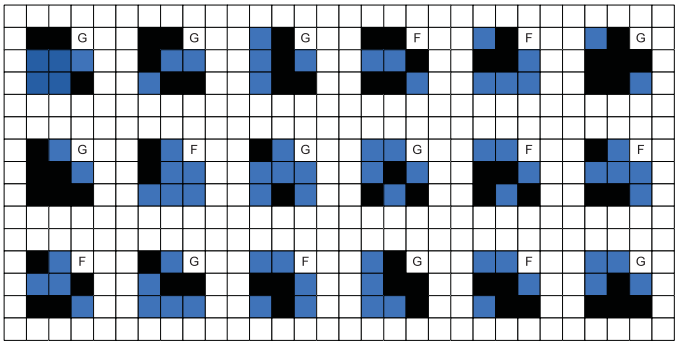

Fig. 9: `Woods2` world

D. *Woods2 World*

`Woods2` world is an extended version of `Woods1`, which consists of 15×30 grid of cells as shown in Fig. 9. It was originally proposed in Wilson's paper [1] to evaluate the performance of XCS. Different from `Woods1`, `Woods2` has two kinds of goal cells and two kinds of obstacles, comparing with only a single kind of goal cell and obstacle cell in `Woods1`. The two kinds of goal cells are denoted as "G" and "F", respectively, which are encoded with 3 bits "110" and "111". The two kinds of obstacles are represented by black cells and blue cells, with encoding "010" and "011", respectively. The empty cells have bits "000". Accordingly, the length of the conditions in each classifier is $3 \times 8 = 24$.

`Woods2` is considered to be more challenging than `Woods1` since the search space of the classifiers is larger

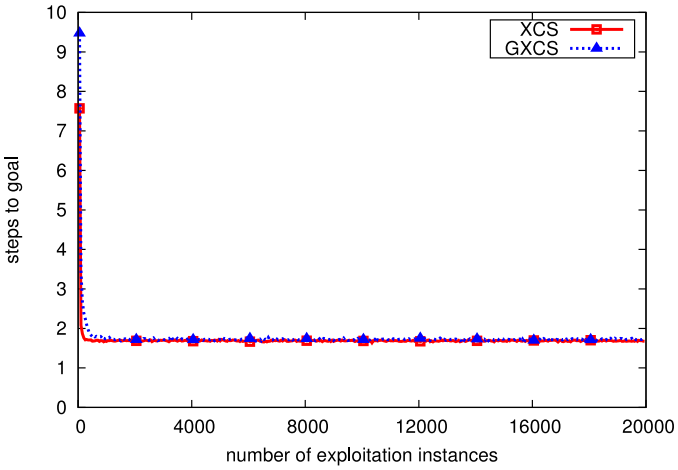

Fig. 10: Performance of XCS and GXCS in `Woods2` world

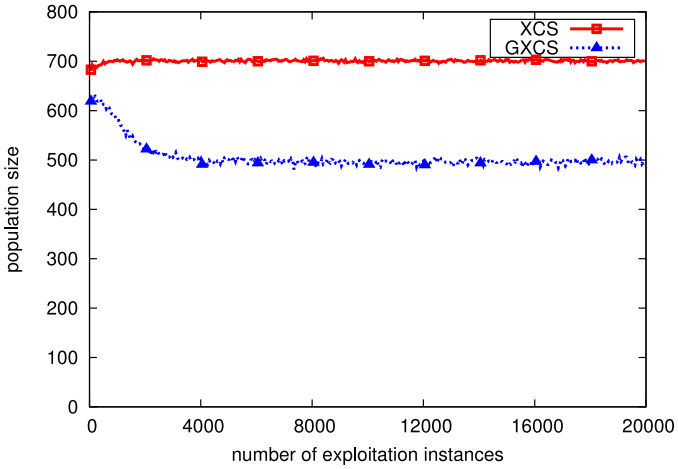

Fig. 11: Population size of XCS and GXCS in `Woods2` world

in `Woods2`. However, if we look back to the relatively large world, `Woods2` is still quite simple, since there are many goal cells located inside the world. Therefore, similar to `Woods1`, the optimal steps to find one of the two kinds of goal cells in `Woods2` is 1.7.

The parameter settings remain the same as the previous studies [4], [24], [25]. That is, $N = 800$; $\beta = 0.2$; $\gamma = 0.7$; $\varepsilon_0 = 10$; $\alpha = 0.1$; $\nu = 5$; $P_\# = 0.5$; $\chi = 0.8$ and $\mu = 0.01$; $\theta_{GA} = 25$. Each experimental run is finished when 20000 exploitation instances are executed.

The performance of XCS and GXCS is reported in Fig. 10. Similarly to the results of `Woods1`, both of XCS and GXCS can easily find the optimal performance around 1.7 steps. Moreover, XCS is also slightly faster than GXCS in this problem. GXCS can converge fast to the optimal performance by learning and evolving the classifiers with cyclic conditions and directional actions.

As for the population size, it is found that GXCS can also reduce the population size when comparing with XCS, as shown in Fig. 11. The reduction of the population size in `Woods2` is relatively smaller than that in `Woods1`.

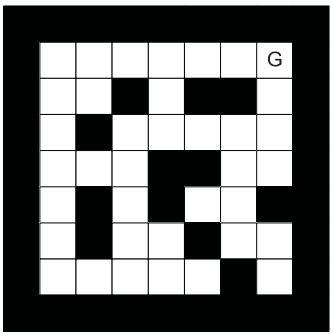

Fig. 12: `Maze5` world

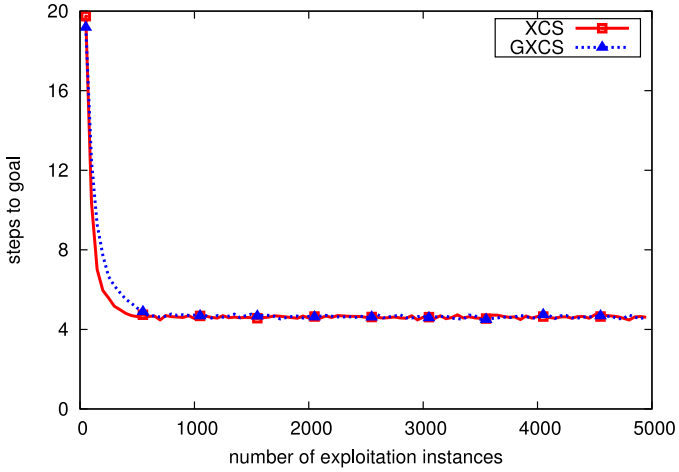

Fig. 13: Performance of XCS and GXCS in `Maze5` world

E. `Maze5` World

`Maze5` world is a much more difficult problem for XCS since much less generalizations are possible [3], [13]. It has 9×9 grid of cells as shown in Fig. 12. Different from `Woods1` and `Woods2` whose left and right edges are connected, as are the top and bottom, `Maze5` fills the 4 edges with obstacles, which results in much more complex scenarios. In this problem, the optimal steps to find the goal cell is 4.6 steps.

The parameter settings are as follows: $N = 3000$; $\beta = 0.2$; $\gamma = 0.7$; $\varepsilon_0 = 5$; $\alpha = 0.1$; $\nu = 5$; $P_\# = 0.3$; $\chi = 0.8$ and $\mu = 0.01$; $\theta_{GA} = 25$. Each experimental run is finished when 5000 exploitation instances are executed.

The performance plotted in Fig. 13 shows that GXCS is capable of finding the optimal performance for `Maze5`. Similarly to the previous two experiments, GXCS requires slightly larger number of learning instances to converge comparing with XCS. However, GXCS can find the optimal population with much smaller number of classifiers than that of XCS, as shown in Fig. 14.

F. Discussions

Considering these 3 experiments as a whole, it is confirmed that GXCS can successfully learn and evolve a population of classifiers with cyclic conditions and directional actions. Though each GXCS classifier is required to maintain more

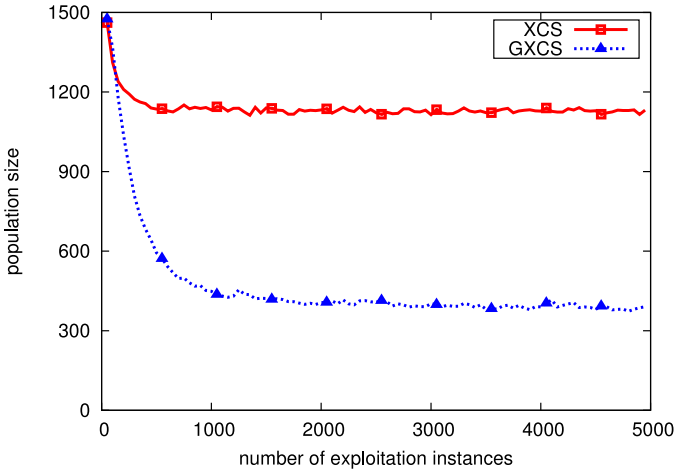

Fig. 14: Population size of XCS and GXCS in `Maze5` world

learning parameters than the XCS classifier, the experimental studies confirm that such kind of representation can have higher expression ability to adapt to more environments. As a consequence, GXCS can achieve the optimal performance as XCS, but with much smaller population size.

However, one of the remaining issues which is not covered here is the computational cost. Since each GXCS classifier should adapt to the environments from different directions, the computational cost for the matching procedure is more than that of XCS. Therefore, one of the future work of this paper is to accelerate the matching procedure of GXCS, as the ones done in XCS [26]. Another issue in the future is to extend this work to more complex problems, especially the real-world scenarios like robotics [27], [28], [29]. In this case, the directional information is rather richer by considering not only the north, east, south and west directions, but also the directions at any angle.

V. Conclusions

This paper develops an extension of XCS by introducing cyclic conditions and directional actions to represent the classifiers. The proposed system, named generalized XCS classifier system (GXCS), is dedicated to modify the forms of the classifiers from chains to cycles, which allows them to match with more adjacent environments perceived by the agent from different directions. Accordingly, a more compact population of classifiers can be evolved to perform the generalization feature of GXCS. Three well-known maze problems are testified in this paper, where the results confirm that GXCS is capable of finding the optimal performance as standard XCS, but with much smaller population sizes.

References

[1] S. W. Wilson, "Classifier fitness based on accuracy," *Evol. Comput.*, vol. 3, no. 2, pp. 149–175, 1995.

[2] T. Kovacs, "XCS classifier system reliably evolves accurate, complete, and minimal representations for boolean functions," in *Soft comput. in eng. design and manufacturing*. Springer, 1998, pp. 59–68.

[3] P. L. Lanzi, "An analysis of generalization in the XCS classifier system," *Evol. Comput.*, vol. 7, no. 2, pp. 125–149, 1999.

[4] M. V. Butz, T. Kovacs, P. L. Lanzi, and S. W. Wilson, "Toward a theory of generalization and learning in XCS," *IEEE Trans. Evol. Comput.*, vol. 8, no. 1, pp. 28–46, 2004.

[5] J. H. Holland and J. S. Reitman, "Cognitive systems based on adaptive algorithms," *SIGART Bull.*, no. 63, pp. 49–49, 1977.

[6] S. W. Wilson, "ZCS: A zeroth level classifier system," *Evol. Comput.*, vol. 2, no. 1, pp. 1–18, 1994.

[7] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[8] S. W. Wilson, "Mining oblique data with XCS," in *Advances in Learning Classifier Systems*, 2001, pp. 158–174.

[9] E. Bernadó, X. Llorà, and J. M. Garrell, "XCS and GALE: A comparative study of two learning classifier systems on data mining," in *Advances in Learning Classifier Systems*, P. Lanzi, W. Stolzmann, and S. W. Wilson, Eds. Springer Berlin Heidelberg, 2002, pp. 115–132.

[10] J. Bacardit and M. Butz, "Data mining in learning classifier systems: Comparing XCS with GAssist," in *Learning Classifier Systems*, 2007, pp. 282–290.

[11] M. Butz, P. Lanzi, and S. Wilson, "Function approximation with XCS: Hyperellipsoidal conditions, recursive least squares, and compaction," *IEEE Trans. Evol. Comput.*, vol. 12, no. 3, pp. 355–376, 2008.

[12] R. Preen and L. Bull, "Dynamical genetic programming in XCSF," *Evol. Comput.*, vol. 21, no. 3, pp. 361–387, 2013.

[13] M. V. Butz, D. E. Goldberg, and P. L. Lanzi, "Gradient descent methods in learning classifier systems: Improving XCS performance in multistep problems," *IEEE Trans. Evol. Comput.*, vol. 9, no. 5, pp. 452–473, 2005.

[14] M. Nakata, P. Lanzi, and K. Takadama, "Enhancing learning capabilities by XCS with best action mapping," in *Proc. of the Parallel Problem Solving from Nature - PPSN XII*, 2012, pp. 256–265.

[15] M. Iqbal, W. N. Browne, and M. Zhang, "Extracting and using building blocks of knowledge in learning classifier systems," in *Proc. of the Genetic and Evol. Comput. Conf.*, 2012, pp. 863–870.

[16] ——, "Extending learning classifier system with cyclic graphs for scalability on complex, large-scale boolean problems," in *Proc. of the Genetic and Evol. Comput. Conf.*, 2013, pp. 1045–1052.

[17] ——, "Reusing building blocks of extracted knowledge to solve complex, large-scale boolean problems," *IEEE Trans. Evol. Comput.*, 2013, (early access).

[18] X. Li and K. Hirasawa, "A learning classifier system based on genetic network programming," in *Proc. of the IEEE Int'l Conf. on Syst., Man, Cybern.*, 2013, pp. 1323–1328.

[19] P. L. Lanzi, "Adding memory to XCS," in *Proc. of the IEEE Congress on Evol. Comput.*, 1998, pp. 609–614.

[20] P. L. Lanzi and S. W. Wilson, "Toward optimal classifier system performance in non-markov environments," *Evol. Comput.*, vol. 8, no. 4, pp. 393–418, 2000.

[21] M. Nakata, P. L. Lanzi, and K. Takadama, "Simple compact genetic algorithm for XCS," in *Proceedings of the IEEE Congress on Evol. Comput.*, 2013, pp. 1718–1723.

[22] X. Li, W. He, and K. Hirasawa, "Genetic network programming with simplified genetic operators," in *Proceedings of the Int'l Conf. on Neural Information Processing*, 2013, pp. 51–58.

[23] X. Li, S. Mabu, and K. Hirasawa, "A novel graph-based estimation of distribution algorithm and its extension using reinforcement learning," *IEEE Trans. Evol. Comput.*, vol. 18, no. 1, pp. 98–113, 2014.

[24] M. V. Butz and S. W. Wilson, "An algorithmic description of XCS," *Soft Computing*, vol. 6, no. 3-4, pp. 144–153, 2002.

[25] M. V. Butz, "XCSJava 1.0: An implementation of the XCS classifier system in Java," Illinois Genetic Algorithms Laboratory, UIUC, Tech. Rep. No. 2000027, 2000.

[26] M. V. Butz, P. L. Lanzi, X. Llora, and D. Loiacono, "An analysis of matching in learning classifier systems," in *Proc. of the Genetic and Evol. Comput.*, 2008, pp. 1349–1356.

[27] X. Li, B. Li, S. Mabu, and K. Hirasawa, "A novel estimation of distribution algorithm using graph-based chromosome representation and reinforcement learning," in *Proc. of the IEEE Congress on Evol. Comput.*, 2011, pp. 37–44.

[28] X. Li, S. Mabu, and K. Hirasawa, "Use of infeasible individuals in probabilistic model building genetic network programming," in *Proc. of the Genetic and Evol. Comput. Conf.*, 2011, pp. 601–608.

[29] ——, "An extended probabilistic model building genetic network programming using both of good and bad individuals," *IEEJ Trans. on Electrical and Electronic Engineering*, vol. 8, no. 4, pp. 339–347, 2013.