Classifier Systems with Native Fuzzy Logic Control Operation

Nugroho Fredivianus
University of Kassel
Wilhelmshöher Allee 73
nfr@vs.uni-kassel.de

Kurt Geihs
University of Kassel
Wilhelmshöher Allee 73
geihs@uni-kassel.de

ABSTRACT

Classifier systems with an efficient learning process are capable of producing a compact set of accurate rules, and therefore have the potential to play the role as a knowledge collector for other systems. In this paper, the accuracy-based Learning Classifier System (XCS) is guided to build the membership function and the rule set commonly owned by a fuzzy logic controller (FLC). Initialized with empty knowledge, an agent responds to real-valued inputs and learns to compose a knowledge using XCS. Results obtained from extensive experiments show that after a number of learning cycles, classifier systems can actually compose a set of rules similar to the mapping mechanism designed by humans for FLC's operation.

KEYWORDS

Classifier systems, XCS, fuzzy logic control

ACM Reference format:
Nugroho Fredivianus and Kurt Geihs. 2017. Classifier Systems with Native Fuzzy Logic Control Operation. In *Proceedings of GECCO '17 Companion, Berlin, Germany, July 15-19, 2017,* 8 pages.
DOI: http://dx.doi.org/10.1145/3067695.3082487

1 INTRODUCTION

In the area of Machine Learning (ML), the classifier systems are one of the classical methods for finding a suitable response of a learning system to incoming inputs. Since being introduced by Holland in 1976, various modifications and improvements were published, with two major sub-groups namely Pittsburgh and Michigan styles. The main difference between them is that the former considers collections of rules as individuals, while the latter sees each classifier (this term is interchangeably used with "rule") as an entity. In 1995, Wilson introduced a Michigan-style variant called the accuracy-based Learning Classifier System (XCS) [17], which has received a lot of attention due to its simplicity and applicability. Improved from the early version that only handles binaries, in 2000 the algorithm was developed into tackling continuous (real) inputs [19].

XCS has been applied to various cases where an agent conducts continuous interaction with the environment, e. g., multi-agent systems (as in [12, 13]), traffic light controller (in [14]), robotics (in [16]). Not only contributions focusing on the practical aspect, several theoretical developments have been investigated, including improvement in handling real values [15], generalization concept [6] and state representation [4]. These advances, from one point of view, enrich the learning system in guiding an agent to behave well in the environment. However, XCS's complex calculations often cause a slow learning rate. As a response to that, a novel method called Rule Combining (RC) was introduced in 2010 [11].

After a series of improvements in [10] and [9], the results obtained by XCS with Rule Combining (known as XCS-RC) opens up the possibility for several directions of development. It successfully provides a superior learning speed and a compact population of rules, in comparison to the classical version of XCS. Therefore, as a stepping stone for further contribution in the applied ML area, we propose an implementation of XCS-RC that learns to perform an FLC operation. This is a role that can only be played by a classifier system capable of building a set of accurate rules in a timely manner, where the classical XCS (or XCSR) requires a considerably longer time in reaching a similar level of performance.

Investigations on the capability of (classical) classifier systems in performing fuzzy operations have been published in a number of works. In [1], Bonarini proposes a so-called Learning Fuzzy Classifier Systems (LFCS) that processes *fuzzified* real input based on the provided membership function and sends an output using defuzzified values. This work was carried on in [7] by employing a mobile robot, and also its accuracy-based variant called FIXCS [2]. Different approaches were also introduced, e. g., using subset of output candidates as in [8], and also in [3] that equips fuzzified XCS with neural functions. From those investigations, one can argue that all of them have made advancement on modified classifier systems without any real attempt to integrate the functions of FLC into the algorithm. Instead, extra components were added, causing even a higher complexity. This does not only make the system more difficult to handle, but also increases inefficiency.

Therefore, our paper aims to introduce a native capability of XCS-RC in performing FLC operation. Here, the term native is used to emphasize the absence of additional component to the system. Continuous values fed to the system go through a set of normal learning steps, before executing one of the available XCS outputs. By equipping it with a properly designed reward mechanism, the system is able to solve FLC's task in guiding an agent to make adequate decisions. The rest of this paper is structured as follows. Section 2 describes XCS briefly, followed by an explanation of XCS-RC in Section 3. Then, the scenario used for guiding the learning system is provided in Section 4, while Section 5 provides the results of the experiments. Finally, Section 6 concludes the investigation along with some outlooks.

GECCO '17 Companion, Berlin, Germany

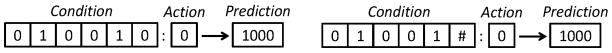
DOI: http://dx.doi.org/10.1145/3067695.3082487

2 XCS

The accuracy-based Learning Classifier System (XCS) is a genetic-based ML algorithm that stores its knowledge in a population of "IF-THEN" rules. Each classifier (symbolized by *cl*) comprises of an input-output pair along with a number of other attributes. The "IF" part of a classifier *cl* is named *Condition* (*cl*.*C*), expressing one or more input states that may cause the rule to be fired. When it occurs, the system executes the "THEN" part using the value of the attribute *Action* (*cl*.*A*) as a response to the obtained input.

A classifier's condition contains an array of information, similar to a chromosome in Genetic Algorithm (GA). Each element of the array is independent from the others. In the early versions of XCS, *cl*.*C* can only work with binaries. Later afterward, its capability is extended to covering various types of data, e. g., real values. When a classifier's condition matches the current environmental state, XCS considers the rule's action as the system output. Commonly, *cl*.*A* is expressed using an integer, representing a specific output of the system (e. g., "0" represents "TURN OFF", "1" means "TURN ON"). The third attribute is called *Prediction* (*cl*.*P*), which keeps the judgment value of the IF-THEN pair. This means, that a suitable pair of *cl*.*C* and *cl*.*A* would be denoted by a high value of *cl*.*P*.

A classifier is commonly represented using "$cl.C : cl.A \rightarrow cl.P$" which is similar to the illustration depicted in Figure 1a. The relation among its three main attributes can be described using a sentence *"IF the input state matches cl.C AND the chosen output is cl.A, THEN the predicted payoff is cl.P"*.



(a) Specific **(b) Generalized, 1 wildcard**

Figure 1: Classifier in XCS

In performing its tasks, XCS runs a series of processes that is derived from the common Reinforcement Learning (RL) cycles. Every time the system receives an input x, the state is compared with all existing *cl*.*C* in the population [*P*]. Then, matching rules are collected in a so-called *Match Set* [*M*]. Afterward, XCS selects one of the available actions owned by the classifiers in [*M*]. Rules advocating the winning action are stored into a so-called *Action Set* [*A*]. After executing *cl*.*A* by using it as the system output, a payoff will be obtained from the environment. The received feedback is used to update the rules in [*A*], and then those classifiers are stored back to [*P*].

The number of learning cycles indicates the "lessons" gained by XCS. As a consequence, the number of rules grows over time and an issue of overpopulation arises at some point. To reduce the effect of such problem, XCS employs the concept of *wildcard* (symbolized by "#"), that is capable of matching a wider range of input. For instance, in systems with binary input, a "#" may act as both a "0" and a "1". So, a rule with a generalized condition *cl*.*C* = "01001#" (as in Figure 1b) is capable of matching the inputs "010010" and "010011".

Using wildcards, a huge amount of learned knowledge can be stored into a compact set of generalized rules. However, the issue shifts from overpopulation to another matter: the adequacy of wildcard placement. Increasing the generality of a rule adds the risk

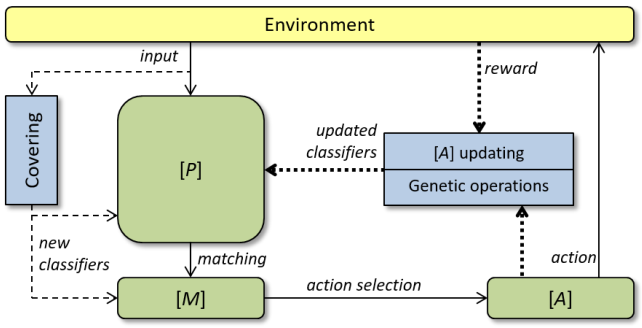
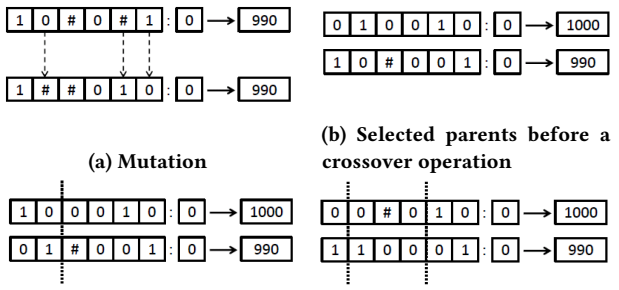


Figure 2: A full learning cycle of XCS

of making a classifier inaccurate, especially when the information contained by a significant bit is disregarded. In response to this matter, XCS is equipped with two mechanisms in placing the wildcards: *covering* and *Darwinian genetic operators*. Figure 2 depicts a full learning cycle of XCS with those two processes.

Covering occurs at the early phase of a learning cycle, i. e., directly after an input x is received. When no classifiers in [*P*] can match x (including when the population is empty), additional rules are created and directly chosen to enter [*M*]. Wildcards are inserted in the new classifiers' conditions based on a random process, leading to a result that always matches x. Such randomized decision making it possible to produce a new classifier having a specific condition $cl.C = x$, or at the contrary, creating a rule whose condition only consists of wildcards. For instance, an input x = "010010" has the possibility to be covered with a wide variation ranging from "010010", "01001#", "0100#0", "0100##" to an extreme result like "######". However, determining a proper probability value for the random process is difficult (e. g., in [6]). This means, that there is a significant possibility for the wildcards to be misplaced and the rule becomes inaccurate.



(a) Mutation **(b) Selected parents before a crossover operation**

(c) After a one-point crossover **(d) After a two-point crossover**

Figure 3: Mutation and crossovers

XCS attempts to minimize such misplacement by employing Darwinian genetic operators, i. e., crossovers and mutations. They are applied to classifiers in [*A*] when some user-defined criteria are fulfilled (e. g., crossover period). This process starts by selecting one or a pair of parent classifiers using either fitness-proportionate [17, 18] or tournament [5] selection. As shown in Figure 3, offspring

are created based on the value of the parents. By modifying the genetic information, XCS is able to reduce the negative impact of the random process, with an expectation of getting better results.

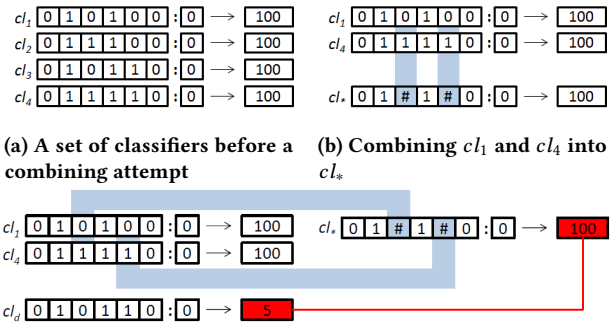
3 RULE COMBINING

Since proposed by Wilson in 1995, a number of XCS variants were published where most of them used the "conventional" crossover and mutation for discovering generalized accurate rules. In 2010, Fredivianus et al.introduced a novel mechanism namely Rule Combining (RC)[11], aiming to place wildcards properly using the induction concept, which in the end leading to a faster learning rate. The variant, called XCS with Rule Combining (XCS-RC), does not insert any wildcards while performing covering, and no Darwinian genetic operators are involved. Instead, generalizations are made based on the existing knowledge. The following subsection explains the concept using XCS-RC with binary input.

3.1 Binary Input

In RC, wildcards are inserted after comparing experienced rules and making conclusions (generalizations) accordingly. Based on a number of classifiers that is considered as existing knowledge, the system seeks any existing patterns available and then creates generalized offspring. There are three prerequisites of performing the generalization: a pair of experienced rules, similar values of $cl.P$ owned by the parents, and no disproving classifiers. An instance of a combining process is provided in Figure 4.

Firstly, Figure 4a depicts a set of four rules where none of them currently has any wildcards. Then, the system attempts to perform induction by picking a pair of parents. Suppose cl_1 and cl_4 are picked to be combined, the system seeks a pattern by comparing their values. Figure 4b shows the result, i. e., a so-called *child candidate* symbolized with cl_*. A number of wildcards are placed in some elements of $cl_*.C$ where the parents have different values. The identical ones are simply inherited by the offspring.



(a) A set of classifiers before a combining attempt

(b) Combining cl_1 and cl_4 into cl_*

(c) The child candidate cl_* is disproved by cl_d

Figure 4: Combining

The result of the combining process is a rule with two wildcards, making the offspring cover a larger state space than any other classifiers that currently exist. In other words, the coverage of each parent (in this case cl_1 and cl_4) is always a subset of the child's. Later when cl_* enters the population, it becomes a full classifier and no longer be regarded as a candidate. At the same time, since the parents' coverage are already handled by the child, they are subsumed and no longer exist in the population.

However, cl_* is not always accepted as a full classifier, e. g., in a different situation illustrated by Figure 4c. Suppose another rule (i. e., cl_d) exists in the population, it will become a disproval to the candidate. After creating cl_*, an examination is executed, comparing it all member of $[P]$ including cl_d. At this point, the system finds out that both cl_d and cl_* are able to cover a similar input "010110". Since their values of $cl.P$ is significantly different (i. e., 100 to 5), accepting cl_* in the population has a risk of unclarity to the system. It would be "confusing" whether responding to an input x = "010110" with the action "0" deserves a high payoff as suggested by cl_* or a low one as in cl_d's prediction. In such case, the term "disproval" is used to explain the role of cl_d in the combining process. Therefore, every generalization should satisfy all the system's existing knowledge, otherwise no offspring is created. Another issue regarding the disproving mechanism is the possibility that a candidate is being examined when the actual disproval has not been learned yet. To tackle the possibility, a set of knowledge correction mechanism is provided in XCS-RC (see [11] for more details).

3.2 Real-Valued Input

Performing a machine learning algorithm using binary inputs can be considered artificial. Therefore, an improvement of XCS-RC that is capable of handling real-valued input is introduced in [10] and improved in [9]. This advanced work opens up several possibilities to develop XCS in general into performing various classification operation, including FLC that will be discussed in Section 4.

(a) Binary input **(b) Real input**

Figure 5: Input for each element of a $cl.C$

The first issue that distinguishes between binary and real inputs is the coverage. As depicted in Figure 5, binaries are easier to handle since the values are either "0" or "1" while wildcard is basically a concept that covers both values (see Figure 5a). Different from that, handling a continuous input means that the system should be able to deal with varying representations, ranging from the lower and upper limit (e. g., from 0.000 to 1.000). So, the concept of generalization should also be changed, making it possible to operate a so-called *partial wildcard* as depicted in Figure 5b. Such coverage tackles more than one input value, but it is not a *full wildcard* that handles the whole range of possible input. Partial wildcards do not exist in binaries, but play a major role in handling real-valued inputs.

Like the way it learns binary inputs, XCS-RC does not create wildcards in covering continuous values. It covers all inputs specifically at the received value, i. e., using a dot. Since the state space of the input can be extremely high, such covering has a greater

risk of overpopulation compared to the binaries. Therefore, generalizations are performed by XCS-RC using the same paradigm of induction. At the very first combining attempt, the system only owns classifiers with specific coverages. This is performed using a mechanism illustrated in Figure 6, where a child candidate cl_* is created from a pair of rules. Each parent covers a dot in their jth element of the condition ($cl.C[j]$). In short, the smaller value becomes the lower limit, while the other is taken as the upper.

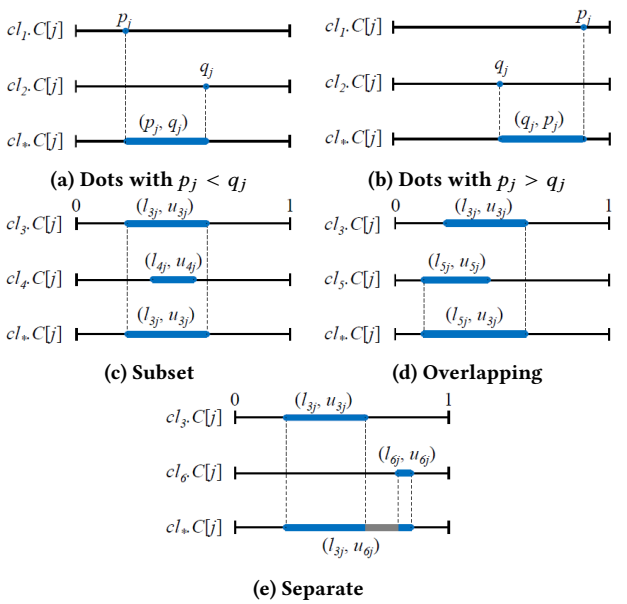


Figure 6: Combining in XCS-RC

In situations illustrated by Figure 6a, the value of p_j is picked as the lower bound of the interval, while q_j is taken as the upper limit. Like in handling binaries, the candidate will be examined before entering the population as a full classifier. So, when no disproval is found for cl_*, the interval covered by $cl_*.C[j]$ will match the jth element of the input within the range of $[p_j; q_j]$. Figure 6b illustrates the situation when the value of q_j is taken as the lower limit while p_j becomes the upper.

After some time, XCS-RC may have an attempt to combine a pair of generalized classifiers, leading to other situations. For instance, the process may involve values which is actually a subset of the other party like in Figure 6c. There is also a possibility that a pair with overlapping coverages is being processed, as depicted in Figure 6d. While those cases have a lower probability of becoming an issue, the third example illustrated by Figure 6e should be handled carefully. Both intervals have no shared area, meaning that there is a chance of producing an over-general rule which may lead to inaccuracy. Similar to combining a pair of two dots, the examination process plays an important role to detect any sign of inadequate generalization and to prevent misleading knowledge, as described in the following subsection.

3.3 Combining Steps

Before entering $[P]$, all child candidates are checked whether any disprovals exist. A disproving rule is a classifier with a shared input coverage with cl_* but having a significantly different prediction (the flag of significance is user-defined). Figure 7 depicts the whole process of combining, where the examination is performed in Step 1. Previously in the preparative Step 0, all rules are grouped in a so-called *CombiningSet* (symbolized by $[C]$) based on their actions (denoted by the color in the illustration). The aim is to simplify further processes, since rules with different $cl.A$ would not affect the running process at all.

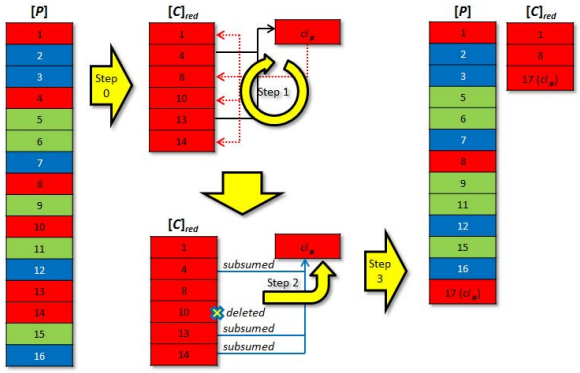


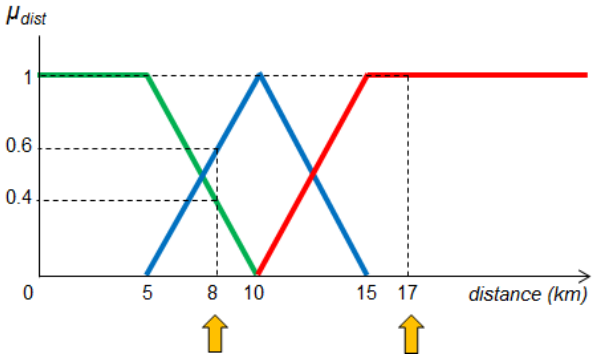
Figure 7: Combining Steps

After passing the examination, the attributes of the offspring are calculated in a way that it would operate like a normal classifier, which is created via covering and being executed several times (a comprehensive overview is presented in [9]). Also, this Step 2 lets the offspring to recruit all subsumable classifiers which are experienced, and remove the inexperienced ones. The attribute values owned by the subsumed rules are used to calculate the offspring's. Finally, in Step 3, cl_* enters the population while at the same time being inserted to the corresponding $[C]$.

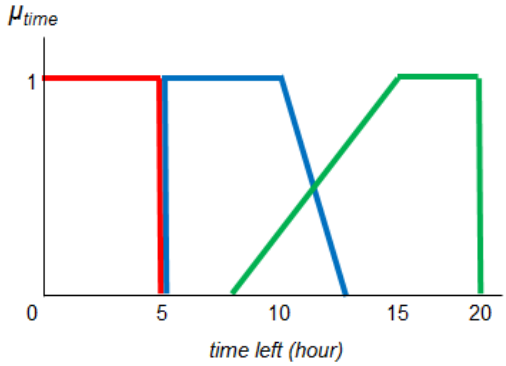
The process of producing offspring continues until no more combining is possible. These steps let XCS-RC compose a compact and accurate set of rules, which is also significantly efficient compared to the classical version of XCS that uses Darwinian genetic operators to place wildcards (comparisons are available in [9]).

4 FUZZY TEST SCENARIO

Fuzzy Logic Controller (FLC) is a well-known and widely-used application for automatic control strategy, especially in the industrial area. The system classifies its inputs using a so-called *membership function* based on the fuzzy logic approach. An example is provided in Figure 8a, where a distance function is depicted using a diagram representation. The green, blue and red lines represent the linguistic variable using the terms "Near", "Medium" and "Far", respectively. For instance, a distance of 17 km would be classified into "Far" by the function, while another input value at 8 km is considered as "Medium-Near" with a proportion of 60:40. Figure 8b shows another instance that classifies the time left to reach the destination, i. e., "Short", "Medium" and "Long" for the lines in red, blue and green, respectively.



(a) Distance function



(b) Time function

SPEED		Distance						
		N	NM	MN	M	MF	FM	F
Time	S	M	M	M	F	F	F	F
	M	M	M	M	M	F	F	F
	ML	S	S	M	M	M	F	F
	LM	S	S	S	S	M	M	M
	L	S	S	S	S	S	M	M

(c) Rule set for vehicle speed

Figure 8: Fuzzy membership functions and the corresponding rule set

After determining the proper membership for each input, FLC will decide the action to be executed using a decision making logic, commonly called as the *rule set*. This component contains a number of "IF-THEN" rules that use the linguistic variable like "Near", "Medium" and "Far" in determining the output. An instance of a rule set is provided in Figure 8c, where a pair of inputs coming from both membership functions is used to decide the speed of vehicle, i. e., "Slow", "Medium" and "Fast". The table can be interpreted using a total of 35 rules, each for one output. For instance, a statement *"IF Distance is F and Time is L then Speed is M"* describes the output at the cell on the bottom right. Another example, a rule *"IF Distance is MN and Time is LM then Speed is S"* represents the cell with an underlined decision.

4.1 Learning Scenario

After being proven to be superior from the classical XCS (and XCSR) in handling the multiplexer and checkerboard cases (see [9]), another test is performed to XCS-RC in this investigation. The use of the induction concept extends the capability of the classifier system by opening up the possibility to perform an FLC task that is more challenging. As the initial test, a simple obstacle avoidance scenario called "Fuzzy Bug" is chosen due to its simplicity and applicability in representing the problem.

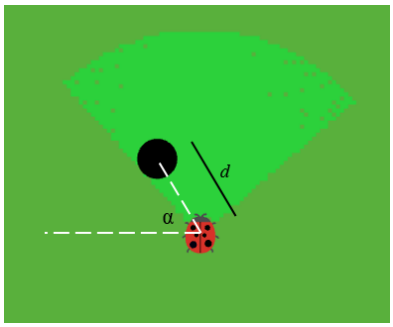
Figure 9a depicts an example of situation that occurs during the investigation. The red bug (beetle) should be guided to avoid an obstacle by performing a rotation further from it. When the black circle is within the bug's limited vicinity (depicted by the light green color), it records the angular position α and distance d, and then transform them using the functions illustrated in Figure 9b and 9c.

To control the behavior of the bug, a rule set with 18 combinations of distance and angular values depicted in Figure 9d is applied. The values for the distances are "Near", "Medium", "Far" while the combinations of "Small", "Medium", "Large" and "Left", "Right" are used to classify the angular position of the obstacle. When a specific pair of input occurs, the expected output will be a combination of rotating either 2, 6 or 10 degree, counterclockwise ("Left") or clockwise ("Right"). For instance, a pair of linguistic values ($d = N$, $\alpha = SL$) will control the bug to rotate 10° to the right (10R). The task of guiding the bug in making appropriate decisions can be easily made by human using FLC, but not necessarily for a learning system. Therefore, a proper reward mechanism is prepared as follows.

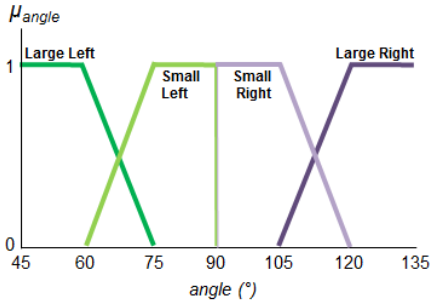
4.2 Reward Mechanism

Since the suitability judgment of each response to a state is determined by the received reward value, composing an adequate reward map for XCS-RC that performs FLC is an important matter. The feedback should be able to represent the provided fuzzy rule set while also sufficiently simple in terms of calculation. In other words, employing XCS to perform FLC operation should be easier than designing a suitable system that comprises of membership functions and rule set. The roles of two membership functions will be played by the pairing elements in the input, respectively representing the angle and the distance. Then, every input is mapped and the system responds to it with an available action advocated by the chosen rules. In short, the learning result should let XCS-RC to perform similarly to FLC.

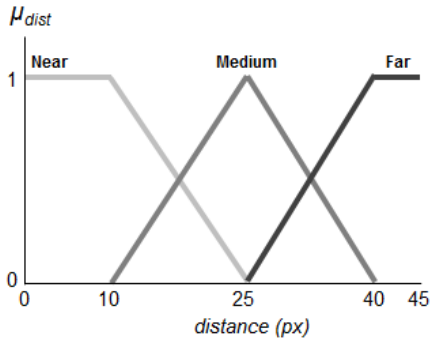
Following the Fuzzy Bug's rotation rule set, the reward map employed in this investigation is designed based on two aspects: avoidance and adequacy. The former is fulfilled when the bug makes a rotation away from the obstacle, while the latter considers the distance (as indicated in Figure 9d) in making decision. This means that a maximal rotation of 10° is considered less suitable when the obstacle is in a relatively larger distance, but preferable for the smaller ones. Figure 10 illustrates the pairs of angle and distance



(a) A situation, angle (α) and distance (d)



(b) Angle function for FCL



(c) Distance function for FCL

Far	2R	2R	6R	6L	2L	2L
Medium	2R	6R	10R	10L	6L	2L
Near	6R	10R	10R	10L	10L	6L
ROTATION	LL	ML	SL	SR	MR	LR

(d) Rotation rule set

Figure 9: Fuzzy Bug scenario

and its relation to the prepared reward. The horizontal axis is the obstacle's angle (α) within the bug's vicinity that spreads from 45° to 135°, while the vertical value represents distance in pixels. The red and blue shades can be directly related to the fuzzy rotation rule

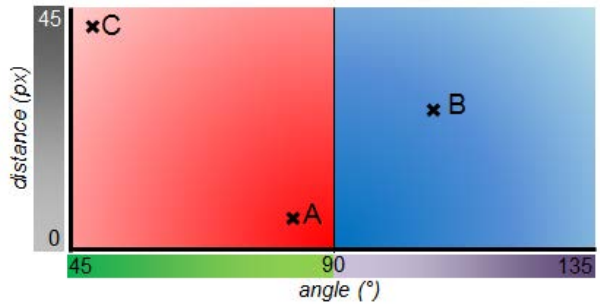


Figure 10: Fuzzy Bug reward map for XCS-RC

set in Figure 9d where areas with darker color are more suitable to be responded with a bigger rotation.

In the practice, performing a "10R" rotation when the obstacle is located at the cross "A" gains a high reward. Similar to that, responding to inputs located at the crosses "B" and "C" with the decisions "6L" and "2R", respectively, will also earn a maximal feedback. Contrary to that, rotations that make the obstacle's angular position smaller then the previous one are judged with a low feedback. The adequacy of the output is determined using the euclidean distance of the input point to ($\alpha = 90, d = 0$). It is calculated in a way so that responding an input ($\alpha = 45, d = 45$) with "2R" receives a reward 1000 but "10R" gets a 500. On the other hand, the feedback for the outputs "10R" and "10L" after a state ($\alpha = 90, d = 0$) is 1000 while the actions "2R" or "2L" gains only 500. The maximal feedback for "6R" and "6L" lays somewhere in the middle of the area. In the following, the experimental results of our investigation are discussed, using the described feedback mechanism.

5 EXPERIMENTAL RESULTS

Discussing a population of classifiers that resembles FLC's membership functions and rule set requires a thorough examination on the classifiers. Therefore, to simplify the explanation, only a set of decisive rules is presented in this paper. In order to maintain a balance between learning new lessons and strengthening good rules, the investigation uses an "old regime" of action selection mechanism with alternating explore-exploit mode. The bug is placed in a world with an obstacle (see Figure 9a) and it should learn to move away from it. For efficiency purposes, the obstacle is intentionally always placed in front of the bug at some random position, making it observable by the bug during the whole runs. The results presented here are an average from 20 simulations, where each run consists of 25 000 explorative and 25 000 exploitative learning cycles.

Table 1 shows that the average number of classifiers owned by the bug by the end of the simulations is around 593.70, including 180.40 (around 30.385%) experienced ones. Also, 23.85 rules were removed from the knowledge due to inadequate combining processes. Despite losing some learning result, these occurrences verify that the knowledge correction mechanism does its assignment in preventing the system from keeping inaccurate rules (more details in [9]).

Table 1: Average Population in Fuzzy Bug

Trial	Population Size	Experienced	Deletion
0	0	0	0
5000	571.50	171.85	4.75
10000	574.30	178.10	8.10
15000	580.80	181.65	11.20
20000	580.00	181.20	14.00
25000	583.75	179.35	15.80
30000	584.40	178.70	17.60
35000	582.55	179.60	19.90
40000	587.55	181.00	21.15
45000	589.10	181.70	22.45
50000	593.70	180.40	23.85

To get a deeper impression on the results, we pick an example from one of the simulations to be discussed here, as seen in Table 2. Among a total of 587 rules, 186 of them are experienced and only 21 individuals are actually required for always obtaining the maximal reward. These rules can be considered as a representation of the whole learning result after 50 000 cycles, covering (almost) the whole input space. In other words, these 21 rules are sufficient for the purpose of making comparison to FLC. All other rules do not provide a higher reward, and therefore less significant in this matter.

The classifiers are grouped into six categories following their $cl.A$, each with different level of red and blue colors. The number of rules advocating the 10° rotation is less than the others, while "6L" and "6R" have the greater share. Rule with the lowest reward is cl_6 with 875.330, which can be considered very high from the minimum value at 0. The highest reward value is 973.352, owned by cl_2. No classifiers reach the maximal feedback at 1000 since each of them is a result from a combining process that requires taking an average value from the involved rules.

Before comparing those 21 rules in the bug's population with FLC rule set, they should be firstly transformed into a decision map shown in Figure 11. Here, the angular position and the distance of the obstacle are mapped into the decision with highest possible reward. For instance, when the bug sees an obstacle at 60.0° and 10.0px away, responding it with the action "6R" will receive the best feedback possible, which is 926.820 provided by cl_{17}. Another example can be given using an input (120.0, 40.0), where the system has learned that "2L" is the most suitable output (cl_7 with 923.549). The other 165 classifiers are not shown here, since they are less desired due to lower payoffs.

In comparison to the FLC's rotation rule set shown in Figure 9d, the discussion will focus on the similarities and differences. Firstly, FLC and XCS-RC suggest that obstacles on the left hand side should be responded by turning right and vice versa. Secondly, both also advocate that every combination of smaller angular position and smaller distance deserves a greater rotation. From these two indications, one can argue that in general both FLC and XCS-RC are capable of guiding the bug to avoid collisions in a considerably similar way. In short, equipping the bug using only these 21 rules will make it behave like being controlled using FLC.

Table 2: The bug's population of rules

No.	Angle	Distance	Rot	Reward
1.	[90.068;113.626]	[0.014;18.594]	10L	894.724
2.	[109.923;126.559]	[0.193;27.516]	6L	973.352
3.	[109.626;135.815]	[0.124;22.511]	6L	951.490
4.	[90.026;126.901]	[10.772;37.417]	6L	934.550
5.	[90.111;114.555]	[18.358;44.881]	6L	910.560
6.	[126.952;135.851]	[22.828;43.044]	6L	875.330
7.	[111.304;133.684]	[31.187;42.954]	2L	923.549
8.	[110.132;135.889]	[42.601;44.991]	2L	920.612
9.	[130.237;135.924]	[5.914;43.699]	2L	915.376
10.	[45.024;89.955]	[37.465;44.991]	2R	906.131
11.	[45.045;50.599]	[0.182;44.308]	2R	901.240
12.	[46.835;63.961]	[25.537;37.191]	2R	883.947
13.	[53.821;89.989]	[13.185;34.405]	6R	967.265
14.	[54.325;88.567]	[11.827;44.098]	6R	939.546
15.	[66.283;89.837]	[12.625;44.998]	6R	939.059
16.	[45.806;87.117]	[11.827;22.325]	6R	931.090
17.	[45.077;73.002]	[0.164;23.747]	6R	926.820
18.	[45.063;55.322]	[0.09;22.648]	6R	911.394
19.	[50.929;53.78]	[22.489;32.971]	6R	891.439
20.	[89.117;89.977]	[0.499;10.23]	10R	960.967
21.	[65.666;89.117]	[0.043;12.702]	10R	905.096

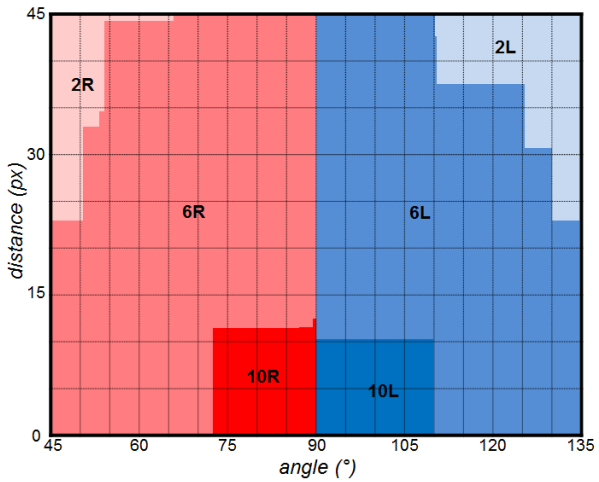


Figure 11: The bug's decision map

The difference between both maps are mainly due to the proportion of areas that are not actually identical. XCS-RC produces a set of knowledge with hundreds of rules where the highest rewards are provided by around 20 to 30 classifiers. From those individuals, using Figure 11 as an instance, the outputs "6L" and "6R" share a greater coverage compared to the others. Such occurrence is caused by at least two factors: rectangular representation of the rules and reward calculation. For the former, an "ideal" learning result is that shown in Figure 10, where the areas are covered in a circular degradation instead of rectangular. The second cause is mainly due to the value of the prediction tolerance parameter (*predTol*) in XCS-RC,

which is a minimal constraint for $cl.P$ owned by two classifiers to be combined. A smaller $predTol$ will rules tend to cover narrower areas which is more accurate, but also making the system requires more individuals to operate in an expected manner.

Another important issue also arises in running the experiments, i. e., finding an adequate setting for the classifier system. Although the number of parameters in XCS-RC is already reduced from that owned by the original XCS, a set of proper values should be handled carefully. In our case, the parameters $predTol$ and prediction error tolerance ($predErrTol$) needs to be adjusted several times. An inadequate setting for the former would cause the rules either very difficult or too easy to be combined, which in turn may cause a high number of deletions particularly when the latter parameter is not set well. After several less adequate results, it is concluded that the values $predTol = 40$ and $predErrTol = 80$ are able to fulfill the requirements.

6 CONCLUSIONS AND OUTLOOK

Among all attempts in modifying classifier systems to behave like FLC, so far XCS-RC is the only variant capable of performing it in a native manner. No additional functions required, internally nor externally. After receiving a "normal" input in real value, the system responds to each state appropriately following the results from a series of learning cycles. With a set of proper parameters, XCS-RC produces a set of knowledge that is capable of playing the roles of membership functions and rule set of a common FLC without human intervention. As other RL algorithms, there are two ways of applying our approach: perform online learning mechanism that lets an agent makes mistakes while collecting knowledge, or conducting a training session to build the population before letting an agent finish its assignment.

Despite some minor issues regarding parameters and the non-identical reward map, the results provided in this paper are promising and certainly useful for further development. Some improvements can be proposed, e. g., using radial coverage instead of rectangular which can be useful for this Fuzzy Bug reward mechanism and similar scenarios. Furthermore, an improvement on the reward calculation might guide the system in building a reward map with a higher resemblance to the one used in FLC. Obviously, investigations involving non-artificial entities (e. g., motors, robots) would significantly contribute to the scientific advances.

REFERENCES

[1] Andrea Bonarini. 2000. An Introduction to Learning Fuzzy Classifier Systems. In *Learning Classifier Systems, From Foundations to Applications*. Springer-Verlag, London, UK, UK, 83–106. http://dl.acm.org/citation.cfm?id=646371.689035

[2] Andrea Bonarini and Matteo Matteucci. 2007. FIXCS: a Fuzzy Implementation of XCS. In *FUZZ-IEEE 2007, IEEE International Conference on Fuzzy Systems, Imperial College, London, UK, 23-26 July, 2007, Proceedings*. IEEE, 1–6. DOI: http://dx.doi.org/10.1109/FUZZY.2007.4295648

[3] Larry Bull and Toby O'Hara. 2002. Accuracy-based Neuro And Neuro-fuzzy Classifier Systems. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '02)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 905–911. http://dl.acm.org/citation.cfm?id=646205.682486

[4] Martin V. Butz. 2005. Kernel-based, Ellipsoidal Conditions in the Real-valued XCS Classifier System. In *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation (GECCO '05)*. ACM, New York, NY, USA, 1835–1842. DOI: http://dx.doi.org/10.1145/1068009.1068320

[5] Martin V. Butz, David E. Goldberg, and Kurian Tharakunnel. 2003. Analysis and improvement of fitness exploitation in XCS: bounding models, tournament selection, and bilateral accuracy. *Evol. Comput.* 11, 3 (2003), 239–277. DOI: http://dx.doi.org/10.1162/106365603322365298

[6] M. V. Butz, T. Kovacs, P. L. Lanzi, and S. W. Wilson. 2004. Toward a Theory of Generalization and Learning in XCS. *IEEE Transactions on Evolutionary Computation* 8 (2004), 28 – 46.

[7] Brian Carse and Anthony G. Pipe. 2001. X-FCS: a fuzzy classifier system using accuracy based fitness - first results.. In *EUSFLAT Conf.* (2008-11-26), Jonathan M. Garibaldi and Robert Ivor John (Eds.). De Montfort University, Leicester, UK, 195–198.

[8] J. Casillas, B. Carse, and L. Bull. 2007. Fuzzy-XCS: A Michigan Genetic Fuzzy System. *Fuzzy Systems, IEEE Transactions on* 15, 4 (aug. 2007), 536 –550. DOI: http://dx.doi.org/10.1109/TFUZZ.2007.900904

[9] Nugroho Fredivianus. 2015. *Heuristic-based Genetic Operation in Classifier Systems*. Ph.D. Dissertation. http://digbib.ubka.uni-karlsruhe.de/volltexte/1000046880 Karlsruhe, KIT, Diss., 2015.

[10] Nugroho Fredivianus, Kais Kara, and Hartmut Schmeck. 2012. Stay real!: XCS with rule combining for real values. In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference companion (GECCO Companion '12)*. ACM, New York, NY, USA, 1493–1494. DOI: http://dx.doi.org/10.1145/2330784.2331009

[11] Nugroho Fredivianus, Holger Prothmann, and Hartmut Schmeck. 2010. XCS Revisited: A Novel Discovery Component for the eXtended Classifier System. In *Simulated Evolution and Learning*. Lecture Notes in Computer Science, Vol. 6457. Springer Berlin / Heidelberg, 289–298.

[12] Matthew Gershoff and Sonia Schulenburg. 2007. Collective Behavior Based Hierarchical XCS. In *Proceedings of the 9th Annual Conference Companion on Genetic and Evolutionary Computation (GECCO '07)*. ACM, New York, NY, USA, 2695–2700. DOI: http://dx.doi.org/10.1145/1274000.1274064

[13] Hiroyasu Inoue, Keiki Takadama, and Katsunori Shimohara. 2005. Exploring XCS in multiagent environments. (2005), 109–111. DOI: http://dx.doi.org/10.1145/1102256.1102281

[14] Fabian Rochner, Holger Prothmann, Jürgen Branke, Christian Müller-Schloer, and Hartmut Schmeck. 2006. An Organic Architecture for Traffic Light Controllers.. In *GI Jahrestagung (1) (LNI)*, Christian Hochberger and Rüdiger Liskowsky (Eds.), Vol. 93. GI, 120–127.

[15] Christopher Stone and Larry Bull. 2003. For real! XCS with continuous-valued inputs. *Evol. Comput.* 11 (September 2003), 299–336. Issue 3.

[16] Andrew Webb, Emma Hart, Peter Ross, and Alistair Lawson. 2003. Controlling a Simulated Khepera with an XCS Classifier System with Memory. In *Advances in Artificial Life*, Wolfgang Banzhaf, Jens Ziegler, Thomas Christaller, Peter Dittrich, and JanT. Kim (Eds.). Lecture Notes in Computer Science, Vol. 2801. Springer Berlin Heidelberg, 885–892. DOI: http://dx.doi.org/10.1007/978-3-540-39432-7_95

[17] Stewart W. Wilson. 1995. Classifier fitness based on accuracy. *Evol. Comput.* 3, 2 (1995), 149–175. DOI: http://dx.doi.org/10.1162/evco.1995.3.2.149

[18] Stewart W. Wilson. 1998. Generalization in the XCS classifier system. *Genetic Programming 1998: Proceedings of the Third Annual Conference* (1998), 665–674.

[19] Stewart W. Wilson. 2000. Get Real! XCS with Continuous-Valued Inputs. In *Learning Classifier Systems, From Foundations to Applications*. Springer-Verlag, London, UK, 209–222.