An Empirical Analysis of Action Map in Learning Classifier Systems

Masaya Nakata * and Keiki Takadama **

Abstract: An action map is one of the most fundamental options in designing a learning classifier system (LCS), which defines how LCSs cover a state action space in a problem. It still remains unclear which action map can be adequate to solve which type of problem effectively, resulting in a lack of basic design methodology of LCS in terms of the action map. This paper attempts to empirically conclude this issue with an intensive analysis comparing different action maps on LCSs. From the analysis on a benchmark classification problem, we identify a fact that an adequate action map can be determined depending on a type of problem difficulty such as class imbalance, more generally, a complexity of classification or decision boundary of problem. We also conduct an experiment on a human activity recognition task as a real world classification problem, and then confirm that a suggested adequate action map from the analysis enables an LCS to improve on the performance. Those results claim that the action map should be selected adequately in designing LCSs in order to improve their potential performance.

Key Words: learning classifier system, action map, performance analysis, evolutionary computation, classification.

1. Introduction

A learning classifier system dubbed LCS [1] is a paradigm of an evolutionary rule-based machine learning approach [2], where we can flexibly select or customize their rule-representation, machine learning or evolutionary computation [3] techniques in developing proper LCSs for specific problems such as data-mining and on-line control [4]. More than eighty specific versions of LCSs now have been proposed on their own aims with their different ways [5]. Such a trend of LCS research leads to expand the applicable problem domains of machine learning as an efficient evolutionary approach.

While diversifying LCS versions, however, there still are lacks of a basic design methodology of LCSs which indicates what should be considered in designing any specific version of LCS. How should common options configuring any LCS be selected for solving problems effectively? A rule fitness is a good example. The rule fitness is a fundamental option which defines what a good rule is, and it is commonly used in evolving rules on all versions of LCS. Then, the design or definition of rule fitness had been a long-standing issue. Traditionally, LCSs employ a strength-based fitness which defines a good rule as one that predicts the highest-reward when a rule’s action is executed'. In 1995, Wilson introduced a reinforcement learning based LCS called extended LCS (XCS) [6] with an accuracy-based fitness relying on a principle: a good rule is to accurately predict class or reward. Through an analysis comparing different types of rule fitness [7], it has been a well-known fact that the accuracy based fitness is a necessary option of rule fitness in order to improve the LCS performance. In fact, many versions of LCS, e.g., a supervised learning based LCS (called supervised LCS (UCS) [8]), inherit the accuracy based fitness.

An action map, which is a main focus of this paper, is also a fundamental option of LCS. The action map defines how LCSs cover a state-action space in a problem and there are two types of action map: a complete action map [6] which covers all actions at each state; and a best action map [9] which in contrast only covers the highest-return action named a best action at each state (see Section 2.1 for more detail). However, the action map seemingly has not been considered enough as an important option due to the following reasons. First, the action map may not be an independent option, that is, a type of action map can be correspondingly determined depending on other options. For instance, the accuracy based fitness of XCS is originally defined with the complete action map [6]; a supervised learning technique of UCS is designed with the best action map. Second, most versions of LCS inherit existing LCS models especially XCS and UCS. Those versions can be roughly classified as a work of developing extensions or applications but not as a revisit of the options including the action map.

In recent years, some works have attempted to clarify effects of action map, inspired from an interesting fact which is probably caused by a difference of action maps. Specifically, Bernadó-Mansilla et al. compared XCS with UCS in classification with a set of real world datasets [8], and they showed UCS significantly outperforms XCS on many datasets. This is because that UCS’s supervised learning works more effectively than XCS’s reinforcement learning on the classification task. However, for some datasets, XCS significantly outperforms UCS. In other words, UCS may not always perform well even on supervised learning tasks. This paper is motivated to understand why XCS sometimes outperforms UCS. Here, our supposition is that this interesting fact is caused by the difference of action maps (i.e., the XCS’s complete action map and the UCS’s best action map), rather than the difference of their machine learning techniques.

* Graduate School of Engineering, Yokohama National University, 79-5 Tokiwadai, Hodogaya-ku, Yokohama, Japan
** Graduate School of Informatics, The University of Electro-Communications, 1-5-1, Chofugakoua, Chofu, Tokyo, Japan
E-mail: nakata-masaya-tb@ynu.ac.jp, keiki@inf.uec.ac.jp
(Received May 31, 2016)
(Revised October 2, 2017)
Our previous work confirmed its supposition in [10], [11]. We introduced a basic LCS model called the XCS with adaptive action mapping (XCSAM) classifier system [12], which employs the same reinforcement learning technique of XCS but uses the option of the best action map. On classification with the real world datasets, our experiment showed a similar fact to the revealed results in [8]: XCSAM performs well on many datasets while XCS still significantly outperforms XCSAM for some datasets [10]. Hence, as we supposed, it is empirically confirmed that the option of action map affects the LCS performance since an adequate action map can be determined depending on datasets.

According to those recent studies, the action map can be an important option in designing LCSs as well as the rule-fitness. However, it still remains unclear which action map can be adequate to solve which type of problem effectively. Accordingly, in this paper, we attempt to conclude this issue and describe a guideline of action map that suggests which action map should be used for which type of problems. Consequently, this paper claims that LCSs have to employ the adequate action map depending on types of problem in order to improve on the potential performance of LCS. This can be a basic design methodology of LCS in terms of the action map.

This paper hypothesizes that the adequate action map is determined depending on a specific type of problem difficulty which could generally occur in a problem such as missing attributes, class-imbalance, and search space size. Our hypothesis is inspired from the fact that our previous work identified a large scale problem as having a problem difficulty suitable for XCSAM [10] and the fact that some machine learning methods (e.g., a fuzzy genetic based machine learning method and a nearest neighbor classifier method) perform well or badly depending on the problem difficulties (or the data complexity) [13]–[15].

In Section 2, we describe the mechanism of XCS and XCSAM. In Section 3, in order to validate our hypothesis, we conduct an intensive analysis comparing different action maps (i.e., XCS with XCSAM) on a benchmark classification problem with specific problem difficulties. From analysis, we draw the guideline of the adequate action map. In Section 4, we validate our guideline on a real world classification task and then explain why XCS sometimes outperforms UCS or XCSAM for some datasets. In Section 5, on a human activity daily living recognition task [16], we show that an LCS application with the adequate action map effectively improves on the potential performance. Section 6 revisits our guideline, and then we show the adequate action map can be eventually determined depending on the complexity of classification boundary. Finally, our conclusion is given in Section 7.

2. Two LCS Models

2.1 Action Map

In LCS, the action map defines how LCSs cover a state-action space in a problem. Two action maps, a complete action map [6] and a best action map [9], have been proposed, and all LCSs can be roughly classified to having either one of the two action maps. Note that XCS and XCSAM perform with the complete action map and the best action map, respectively.

The best action map covers only the highest-return action (called the best action) in every state. Here, the highest-return action is a necessary action to solve a problem, which receives the highest reward at each state. For instance, in classification problems, the highest-return action indicates a correct action to classify a state to a correct class. For instance, as shown in Fig. 1-a), the best action map covers only one state-action pair \( s_1 - a_2 \) which receives the maximum reward \( r = 1000 \) at the state \( s_1 \). Thus, an LCS with the best action map attempts to learn only necessary classifiers advocating the best actions (called optimal classifiers) to solve a problem.

The complete action map covers the whole state-action space (i.e., all possible state-action pairs). As shown in Fig. 1-b), the complete action map covers all possible state-action pairs \( (s_1 - a_1, s_1 - a_2, \text{ and } s_1 - a_3) \) at the state \( s_1 \). An LCS with the complete action map attempts to learn classifiers advocating the best action but also the non-best actions which do not receive the maximum reward at each state. For instance, in classification problems, the complete action map covers all classifiers advocating the correct class and the incorrect classes to a state.

2.2 XCS

Rule format. In XCS the rules consist of a condition, an action, and four main parameters [6]: (i) the prediction \( p \), which estimates the relative payoff that the system expects when the rule is used; (ii) the prediction error \( \varepsilon \), which estimates the error of the prediction \( p \); (iii) the fitness \( F \), which estimates the accuracy of the payoff prediction given by \( p \); and (iv) the numerosity \( num \), which indicates how many copies of rules with the same condition and the same action are present in the population. On a binary classification task, the rule is simply coded by 0, 1, and don’t care # which matches any symbol.

Performance Component. At time \( t \), XCS builds a match set \([M]\) containing the rules in the population \([P]\) whose condition matches the current sensory input \( s_t \); if \([M]\) does not contain all the feasible actions, covering takes place and creates a set of rules that matches \( s_t \) and cover all the missing actions. This process ensures that XCS can evolve the complete action map so that in any state it can predict the effect of every possible action in terms of expected returns. In the algorithmic description [17], covering is activated when match set contains less than \( \theta_{num} \) actions; however, \( \theta_{num} \) is always set to the number of available actions \([P]\) so that the match set covers all the actions. For each possible action \( a_i \in [M] \), XCS computes the system prediction \( P(a_i) \) which estimates the payoff that XCS expects
if action \( a_i \) is performed at the current state \( s_i \). The system prediction is computed as the fitness weighted average of the predictions of rules in \([M]\), \( c.l [M] \), which advocate action \( a_i \):

\[
P(a_i) = \frac{\sum_{c.l[M]_a} c.l.p \times c.l.F}{\sum_{c.l[M]_a} c.l.F},
\]

where \([M]_a\) represents the subset of rules of \([M]\) with action \( a_i \), \( c.l.p \) identifies the prediction of rule \( c.l \), and \( c.l.F \) identifies the fitness of rule \( c.l \). Then XCS selects an action to perform; the rules in \([M]\) which advocate the selected action form the current action set \([A]\). The selected action \( a_i \) is performed, and a scalar reward \( r_i \) is returned to XCS.

**Reinforcement Component.** Next, the parameters of rules in \([A]\) are updated in the following order [17]: prediction \( p \), prediction error \( e \), and fitness \( F \). Prediction \( p \) and prediction error \( e \) of classifier \( c.l \) are updated by Eqs. (2) and (3) where \( \beta \) is the learning rate (\( 0 \leq \beta \leq 1 \)).

\[
\begin{align*}
cl.p &\leftarrow c.l.p + \beta (r_i - cl.p), \\
cl.e &\leftarrow c.l.e + \beta (r_i - c.l.p - |c.l.e|).
\end{align*}
\]

Finally, the rule fitness \( F \) is updated in two steps: first, the accuracy \( cl.k \) of the rule in \([A]\) is computed as follows:

\[
cl.k = \begin{cases} 1 & \text{if } c.l.e < e_0, \\ \frac{a(c.l.e/e_0)^{\nu}} {\text{num}} & \text{otherwise}. \end{cases}
\]

The accuracy \( cl.k \) means that a rule is considered to be accurate if its prediction error \( c.l.e \) is smaller than the threshold \( e_0 \); a rule that has an accurate has an accuracy \( cl.k \) equal to 1. A rule is considered to be inaccurate if its prediction error \( c.l.e \) is larger than \( e_0 \); the accuracy \( cl.k \) of an inaccurate rule is computed as a potential descending slope given by \( a(c.l.e/e_0)^{\nu} \). Then the fitness \( c.l.F \) of each rule \( c.l \) in \([A]\) is updated with a relative accuracy \( cl.k' \) as follows:

\[
\begin{align*}
cl.k' &= \frac{cl.k \times c.l.num}{\sum_{cl[A]} c.l.k \times c.l.num}, \\
cl.F &= cl.F + \beta (cl.k' - cl.F).
\end{align*}
\]

**Discovery Component.** On a regular basis depending on the parameter \( \theta_{\text{OA}} \) [17], the steady-state genetic algorithm is applied to rules in \([A]\). It selects two rules, copies them, and with probability \( \nu \) performs crossover on the copies; then, with probability \( \mu \) it mutates each allele. The offspring rules are inserted into the population and two rules are deleted from the population \([P]\) if the number of rules in the population \([P]\) is larger than a population size limit \( N \) to keep the population size constant.

### 2.3 XCSAM

XCSAM extends the original XCS by (i) including a mechanism to identify rules having the best actions and by (ii) getting rid of redundant actions unlikely to be the best actions.

**Identification of Rules having Best actions.**

XCSAM adds a parameter \( op \) to rules, or optimality of action, that for rule \( c.l \) is updated according to Eq. (7), where \( r_i \geq \zeta \max P(s_i, a) \) is a condition to identify the executed action as the best action with a permit value \( \zeta \). Note that the parameter \( op \) of rules in \([A]\) is updated on the reinforcement component. When the selected action is identified as the best action (i.e., the rules in \([A]\) have the best action), the \( op \) value of rules converges to 1; otherwise, \(|A|\). Therefore, rules with an \( op \) close to 1 are good candidates to represent the final best action map while rules with an \( op \) close to \(|A|\) are less likely to be maintained as they are advocating actions with lower expected return.

\[
cl.op \left\{ \begin{array}{ll}
cl.op + \beta (1 - cl.op) & \text{if } r_i \geq \zeta \max P(s_i, a), \\
cl.op + \beta (|A| - cl.op) & \text{otherwise}. 
\end{array} \right.
\]

**Evolution on Best Actions.**

To build the best action map, XCSAM acts on the covering operator to prevent the generation of rules that are not likely to form the best action map. In particular, XCSAM tunes the activation threshold of \( \theta_{\text{max}} \) based on the \( op \) parameters. Initially, \( \theta_{\text{max}} \) is set to \(|A|\) (the same value used in XCS). When \([M]\) is generated, XCSAM computes the prediction array before covering is applied (whereas XCS computes it only after covering). Then, XCSAM computes the \( \theta_{\text{max}} \) as the average of \( op \) of the rules in \([M]\) having the highest-return action \( a^\ast = \arg \max_a P(a) \). The covering operator takes place if the number of different actions in \([M]\) (denoted by \(|A[M]|\)) is less than \( \theta_{\text{max}} \).

Accordingly, if \( \theta_{\text{max}} \) is 1, XCSAM has identified the best action, so it does not need to generate rules having other actions.

After action selection is performed, XCSAM runs the steady-state genetic algorithm (GA) as in XCS (i.e., generates two offspring and delete two rules). XCSAM generates both the action set \([A]\) and also the not action set \([\bar{A}]\) with rules in \([M]\) having unselected actions. When the selected action is identified as the best action by Eq. (7), the parents are selected from \([A]\) to promote the evolution of rules that are likely to be in the final best action map. For the rule-deletion, we use the modified rule-deletion mechanism presented in [11]. Specifically, the rule-deletion mechanism firstly computes the average of \( op \) (denoted by \( OP(a) \)) for each action \( a \) from rules in \([A]\) which have the action \( a \); then it identifies an action \( a_{\text{max}} \) unlikely to be the best action compared with other actions, i.e., \( a_{\text{max}} = \arg \max_a OP(a) \). Next, XCSAM builds a rule set \([OP]\) consisting of rules which have the action \( a_{\text{max}} \); then it deletes all rules in \([OP]\). Those procedures are repeated while \(|A[M]| \geq \theta_{\text{max}} \) is satisfied. Note that the modified deletion mechanism is called when the selected action is identified as the best action; otherwise, the two rules are deleted from \([P]\) as in XCS.

### 3. Comparison of Action Maps

We test two LCS models (XCS and XCSAM) on a benchmark classification task (the multiplexer problem [6]) with the three different problem difficulties; overlapping instances, missing attributes, and class-imbalance. Note that, for a problem difficulty, large search space, our previous work showed XCSAM significantly outperforms XCS on a benchmark classification task with increasing the search space size [10],[11]. However, we discuss the results (presented in this paper) with the previous work together in Section 3.5.

#### 3.1 The Multiplexer Problem

The \( l \)-bit boolean multiplexer function is defined over a binary string of \( l = k + 2^i \) bits; the first \( k \) bits represent an address pointing to the rest of \( 2^i \) bits. The answer is the bit at
\( k + \text{address} \) where \( \text{address} \) is a decimal number which is converted from the address bits. For example, the 6-multiplexer function \((k = 2)\), for an instance 110001 the answer is 1; since \( k = 2 \) it computes the \( \text{address} 3 \) from the first two bits (“11”); the answer is the bit at 5 \((5 = 2 + 3)\). This paper employs the 11-bit multiplexer problem denoted by 11-MUX. When a system executes the correct answer, it receives a positive reward 1000; otherwise 0.

### 3.2 Dataset and Three Problem Difficulties

**Dataset.** We here generate a dataset composing of the 11-MUX instances so that we consider the multiplexer problem as an off-line classification problem. For each experiment, we randomly generate a set of 5,000 11-MUX instances with its correct classes (the instance is randomly generated as an 11-bit binary string). After building the dataset, each problem difficulty is added to the dataset (see below).

**Overlapping instances.** This problem difficulty indicates that the problem has overlapping instances which have the same input but different classes. To simulate the overlapping instances, we add alternating noise to the correct class of instance in the dataset. With an overlapping rate \( P_{\text{an}} \), each instance is decided whether including the alternating noise to its correct class. When adding the noise, the correct class is changed to a different class, i.e., the class “1” is changed to “0” or “0” to “1”. The value of \( P_{\text{an}} \) is set to 0.10, 0.20, and 0.30.

**Missing attributes.** This problem difficulty indicates the instance includes some unknown attributes. The missing attributes can be simulated as the following procedure. With a probability of missing rate \( P_{\text{mr}} \), each attribute of instance in the dataset is replaced with “?”, representing the missing attribute. For instance, an instance on 6-bit multiplexer problem \( \text{“110001”} \) where the correct class is 1 may be converted as \( \text{“??0001”} \) with a probability \( P_{\text{mr}} \). Here, we define the missing attributes are a symbol matching any attribute of rule-condition (i.e., 0, 1 or #). We set \( P_{\text{mr}} \) to 0.05, 0.10, and 0.30.

**Class-imbalance.** This problem difficulty can be observed in many real world data, which indicates that instances with a majority class exist more than instances with a minority class. We use the 11-bit imbalanced multiplexer problem [18]. Different from the above two problem difficulties, the dataset is regenerated in order to include the class imbalance. Firstly, an instance of 11-MUX is randomly generated. If the correct class of the generated instance belongs to the minority class 0, then its instance is added to the dataset with probability \( 1/2^k \), where \( k \) is the imbalance level of the dataset. If the correct class belongs to the majority class 1, its instance is always added to the dataset. This procedure is repeated until generating 5,000 instances in the dataset. We set \( k \) to 7, 8, and 9.

### 3.3 Experimental Design

We use the learning/test problem scheme and one iteration is composed of one learning problem and one test problem [6]. For the learning problem, an 11-bit binary string whose correct class is determined under the 11-MUX’s definition, is randomly selected from the generated dataset, and it is sent to an LCS. For the test problem, an 11-MUX instance is randomly generated without any problem difficulty. Hence, one learning or test problem indicates one instance of 11-MUX. The learning problem is used to learn classifiers, while the test problem is used for only evaluating the LCS performance. Accordingly, for one iteration, the LCS will receive two instances for learning and test problem respectively; it firstly learns classifier based on the instance of learning problem, and then the system performance is immediately evaluated with the instance of test problem. The LCS repeats the iteration until the maximum number of iteration is reached.

For learning problems, the system selects actions randomly from those represented in the match set. For test problems, the system always selects the action with highest expected return and no update is performed. Note that, given instances during test problems are generated without any problem difficulty. The genetic algorithm is enabled only during learning problems, and it is turned off during test problems. The covering operator is always enabled but operates only if needed. Learning problems and test problems are alternate.

The classification accuracy (i.e., the system performance) is reported as the moving average over the last 50,000 (for the class imbalance) or 5,000 (for other problem difficulties) test problems. The classification accuracy is averages over 30 experiments. Throughout this section, if not stated differently, we use the following parameter settings [18] except for \( N = 1600; \ v_0 = 1, \mu = 0.04, P_{\text{an}} = 0.6, \chi = 0.8, \beta = 0.2, \alpha = 0.1, \gamma = 5, \theta_{\text{GA}} = 25 \); in XCSAM \( \zeta = 0.99 \). The maximum iteration is 200,000. Note that we applied a statistical test to each experimental case to find the significant difference of the performances (i.e., XCS and XCSAM); in this section, we found the significant differences for all experimental cases (see Appendix B for more detailed configurations of applied statistical test).

### 3.4 Results

**Overlapping instances.** Figure 2 (a) shows the classification accuracies of XCS and XCSAM on 11-MUX with the different overlapping rates \( P_{\text{an}} = 0.10, 0.20, 0.30 \).

Note that XCS and XCSAM can derive 100% classification accuracy on 11-MUX without any problem difficulty. From the figure, with \( P_{\text{an}} = 0.10 \), XCS significantly outperforms XCSAM \( (p < 0.05) \); the XCS performance reaches to 0.99 while XCSAM derives 0.94 classification accuracy. When \( P_{\text{an}} = 0.20 \), XCS still outperforms XCSAM. However, when \( P_{\text{an}} \) is increased to 0.30, XCS fails to solve the 11-MUX while XCSAM still robustly performs with 0.72 classification accuracy. These results suggest that the complete action map (i.e., XCS) is a more effective action map than the best action map (i.e., XCSAM) when the overlapping rate is small. In contrast, with the large overlapping rate, the best action map can be a robust action map.

**Missing attributes.** Figure 2 (b) shows the classification accuracies of XCS and XCSAM with the missing rates \( P_{\text{mr}} = 0.05, 0.10, 0.30 \).

With \( P_{\text{mr}} = 0.05 \), the XCS performance reaches to 0.97 while the XCSAM performance significantly degrades to 0.89 \( (p < 0.05) \). However, with \( P_{\text{mr}} \) further increased to 0.30, in contrast, XCSAM outperforms XCS \( (p < 0.05) \); XCS fail to solve the 11-MUX with 0.55 classification accuracy; XCSAM still robustly performs with 0.64 classification accuracy. These
results show a very similar tendency to the case of the overlapping instances.

**Class imbalance.** For the class imbalance, we use specific parameter settings suggested in [18]: \( ir = 7, 8, 9, \beta = 0.02, 0.01, 0.005, \theta_{GA} = 400, 800, 1600 \) respectively. We use a tournament selection in GA [19]. The maximum iteration is set to 5,000,000 as in [18]. Figure 2 (c) reports the classification accuracies of XCS and XCSAM with the different levels of class imbalance.

With \( ir = 7 \), the XCSAM performance reaches to 0.91 while the XCS performance converges to 0.87. When the imbalance level \( ir \) is further increased to 8 and 9, XCSAM still slightly outperforms XCS \( (p < 0.05) \). Different from the previous experiments, the results suggest the best action map (XCSAM) can be the adequate action map for all imbalance levels.

### 3.5 Summary and Guideline

Our analysis clarified that the performance of LCS degrades or improves depending on the type of problem difficulty, as we hypothesized. Hence, we can confirm that the potential performance of LCS would improve when LCS employs the adequate action map suitable for a problem difficulty. As a guideline, Table 1 summarizes the adequate action map for each problem difficulty. For the class imbalance, the best action map always contributes to more effectively improve the XCSAM performance compared with the XCS’s complete action map. In addition, our previous work showed XCSAM also outperformed XCS on a problem with a large search space, i.e., a large number of actions and a long length of inputs [11], which is a very similar result to the case of class imbalance. Accordingly the best action map is the adequate action map on a problem including class-imbalance or a large search space. However, for the overlapping instances and the missing attributes, the adequate action map can be changed with the strength of those difficulties. For low strength, the complete action map is adequate while the best action map is more effective for high strength. Note that, the guideline is based on our experiments and thus it is still not confirmed to be available in a wide range of problems. However, we showed an example that the adequate action map can be determined with the type of problem difficulty or strength.

| Problem Difficulty | Strength | Adequate Action Map |
|--------------------|----------|---------------------|
| 1. overlapping instances | low | complete action map |
|                     | high    | best action map     |
| 2. missing attribute | low     | complete action map |
|                     | high    | best action map     |
| 3. class imbalance   | low/high| best action map     |
| 4. state-action space| small/large | best action map    |

### 4. Validation of Guideline

We validate whether the dependency of action map can also occur in real world problems or other LCS models. Here, we refer to two previous results on the real world classification problem. The first results are referred from our previous work, comparing XCS with XCSAM [10] on the classification task with datasets available in UCI repository [20]. As explained in Section 1, although XCSAM outperforms XCS for many datasets, XCS outperforms XCSAM for some datasets. Then, our validation aims at explaining why XCS sometimes outperforms XCSAM; in more detail, the dataset where XCS performs well includes a problem difficulty suitable for XCS’s complete action map. The second results are conducted in [8]. In [8], they compared XCS with UCS on the classification task with 30 datasets from UCI repository [20]. Note that, similar to XCSAM, UCS employs the best action map; but it is powered by a supervised learning technique different from XCS and XCSAM. Hence, the second results can be used to validate whether our guideline also supports the different LCS model (i.e., UCS) from XCS.

Here, for both cases (i.e., our results and the results revealed in [8]), the same experimental settings are employed [8]. In detail, to calculate the classification accuracy, the 10-fold cross validation is applied; after generating 10 subsets of instances from the whole dataset, the test dataset is set to one subset of those; and the other nine subsets are set to the learning dataset. Here, one iteration is to solve one randomly-selected instance from the learning dataset. And then, the LCS repeated iterations until the maximum 100,000 iterations; after that, each instance of the test dataset is sent to the LCS to evaluate the classification accuracy. Hence, the total problem is corresponding to 100,000 iterations (the LCS will receive 100,000 instances from the learning dataset).

We quantify the four problem difficulties within datasets as:

- **Overlapping instances.** The overlapping rate \( P_{on} \) is calculated as the number of overlapping instances divided by the number of all instances. With a large value of \( P_{on} \), a dataset has many overlapping instances.

- **Missing attributes.** The missing rate \( P_{m} \) is measured as an average of number of missing attributes per one attribute of instance. With a large value of \( P_{m} \), an instance of dataset has many missing attributes.
• **Class imbalance.** The class imbalance is measured as a standard deviation of proportions of instances belonging to each class denoted by $\sigma_{cls}$ (see Appendix A for detail calculation). The datasets are highly imbalanced when $\sigma_{cls}$ is a large value. Note that, as an example to understand how much strength of class imbalance our experiment in Section 4 used, the class imbalance level $ir = 7, 8$, and $9$ are converted to $\sigma_{cls} = 0.702, 0.704$, and $0.706$, respectively.

• **Large search space.** This problem difficulty is simply measured as the input length $L$ and the number of possible classes $|\mathcal{A}|$. The dataset has a large search space with large values of $L$ and $|\mathcal{A}|$.

### 4.1 XCS vs. XCSAM

Table 2 shows quantified values for each problem difficulty on the datasets and the classification accuracies of XCS and XCSAM [10]. Note that we pick up eight datasets for which a significant difference between XCS and XCSAM was found in [10]; on the classification accuracy, the bold numbers represent the significant difference ($p < 0.05$). As we suggested, XCSAM significantly outperforms XCS for many datasets, Audiology, Balance, Libras, Segment, Soybean, and Vowel. However, XCS still outperforms XCSAM on Hepatitis and Vote. Specifically, Vote includes about 6% missing rate. This is a consistent result to our guideline suggested in the previous section, which indicates the XCS’s complete action map can be adequate for a small value of missing rate. Hepatitis includes about 6% missing rate with the class imbalance $\sigma_{cls} = 0.52$; since we do not conduct experiments with more than one problem difficulty, the adequate action map is not empirically confirmed for a combinational effect of different problem difficulties. However, we can suspect that the missing attributes may affect the LCS performance more than the class imbalance. As we will give a detail discussion in Section 6, different from the class imbalance, some instances with missing attributes may not be determinately classified to its correct class. Then, XCSAM (i.e., evolving the best action map) wrongly deletes good classifiers having the correct class due to noise, resulting in a low classification accuracy (see Section 6 for more detail).

For the datasets where XCSAM outperforms XCS, Balance has a problem difficulty of class imbalance which is suitable for the best action map; for Audiology, Libras, Segment, and Vowel, those datasets have a large state action space with a high dimensional input and the large number of actions. For Soybean, it has a small rate of missing attribute about $P_m = 10\%$; however, Soybean can be classified as having a large search space as suggested in [8]. Those results suggest XCSAM’s best action map is suitable for datasets including imbalanced classes or larger search spaces, as expected by our guideline.

### 4.2 XCS vs. UCS

Table 3 shows the classification accuracies of XCS and UCS referred from [8]. Again, we pick up nine datasets for which a significant difference between XCS and UCS was found in [8]. Since the datasets Tao and Thyroid are not available in UCI repository, their overlapping rate, the missing rate, and class-imbalance distribution could not be quantified. From the table, we can see UCS significantly outperforms XCS for six datasets, Audiology, Hepatitis, Segment, Soybean, Thyroid, and Vowel. For these datasets except for Hepatitis, they have a high-dimensional input or a large number of classes; they include the problem difficulty of large state-action space. Hence, we can explain that UCS outperforms XCS because UCS employs the best action map suitable for the large state-action space. For the dataset Hepatitis, since XCS outperforms XCSAM on Hepatitis, we can suppose that XCS also outperforms UCS. However, UCS still outperforms XCS. We have not found any clear reason to explain this inconsistent fact, but we can suspect this is caused from the difference of learning techniques both LCSs use. In fact, we showed XCS significantly outperforms XCSAM on the dataset Hepatitis in [10], which is a consistent result with our supposition (see Section 4.1).

For the datasets where XCS significantly outperforms UCS, we can find three datasets, Cmc, Sick, and Tao. Cmc has a low overlapping rate $P_m = 0.10$; Cmc has the problem difficulties suitable for the complete action map. Sick has a low missing rate $P_m = 0.06$ and the class imbalance $\sigma_{cls} = 0.66$. As we suspected in Section 4.1, the missing attribute may greatly affect the LCS performance than the class imbalance; we can suspect that XCS performs well since the complete action map is suitable for the low strength of missing attributes.

In summary, we showed that the dependency of action map against the problem difficulties also occurs even on the other LCS model (UCS) and the real world classification task, and then its dependency can be a reason to explain why XCS (or UCS, XCSAM) derives good performance on a specific dataset.

### 5. Test on LCS Application

We conduct an experiment on the human activity of daily living recognition task (ADL) as a real world sequence labeling task [16]. Different from the experiment and validation in the previous sections, our experiment here aims to validate that the performance of an LCS application could potentially
improve when it employs the adequate action map for a given problem. Consequently, we can claim that any LCS should first employ the adequate action map in order to enhance the potential performance, before designing extensions or additional mechanisms. We first explain the definition of the ADL recognition task and identify the problem difficulties within the ADL datasets. Next, we introduce an application of LCS with the adequate action map and then conduct experiments.

5.1 Human Activity Dairy Living Recognition

The ADL recognition task is a kind of sequential labeling problem, where a correct class of the current input may be determined depending on previous inputs. This task aims at predicting human-activities from a sequence of inputs which represent a set of sensor. The ADL dataset can be represented as a sequence of \((time, input: class)\) (see Fig. 3). The input "kitchen" is placed at different time stamps "1pm" and "7pm," but it has different classes "lunch" or "dinner," respectively. Note that we do not use the time stamps except to order the inputs, i.e., a rule cannot be represented as "IF time is 7pm THEN dinner." Hence, the input "kitchen" does not unambiguously identify the current class, i.e., the input is perceptually aliased. However, when a learner refers back to the previous input, it can successfully classify it when it considers current and previous inputs. For instance, a rule-condition for correctly predicting the "dinner" class can be \((\text{kitchen}, t_0), (\text{living}, t_1))\).

Here, we use two real world datasets of the ADL recognition task, Ordonez A and Ordonez B; their problem difficulties are summarized in Table 4. Both datasets mainly have very large overlapping rates. Note that the ADL recognition task can be a normal classification task when a system does not consider the sequence of input.

5.2 Design of LCS Application

From Table 4, according to our guideline, we can suppose that an LCS application for the ADL recognition task is recommended to employ the best action map. Accordingly, we use an extension of XCS for sequence labeling called XCS-SL [21], and then we modify it to be XCSAM-SL which employs XCSAM as its basic LCS model instead of XCS. We firstly explain the mechanism of XCS-SL and then XCSAM-SL.

XCS-SL. XCS-SL almost works the same as standard XCS [17], but a rule representation and a mechanism in the the discovery component are modified. In XCS-SL, the rules have a new parameter which is the memory size \(m\) to determine the condition length; the condition is composed of sub-conditions \(C_0, \ldots, C_{m-1}\). Each sub-condition \(C_m\) corresponds to the input at the time stamp \(t_m\). The memory size \(m\) is determined and fixed when the rule is generated, but the maximum memory size for all rules \(M\) is set to a fixed value. After the reinforcement component is performed, XCS-SL evolves rules using GA. In sequence labeling, each input can have its own suitable memory size (i.e., each input may need a different number of previous inputs). Hence, XCS-SL is required to evolve rules which have the suitable memory size. Accordingly, XCS-SL builds subsets \([A(t_m)]\) of the action set each of which consists of rules in \([A]\) whose memory size \(m\) is equal to \(n\). Then XCS-SL selects one subset from among the subsets \([A(t_0)], \ldots, [A(t_n)]\) to perform the GA. Selection is done by a roulette wheel on the average fitness of each subset. After selection, the GA is applied to rules in the selected subset as in XCS. In crossover, each sub-condition is recombined with the corresponding sub-condition of the other offspring. The mutation also changes the memory size \(m\) of a rule to a random value with probability \(\mu\).

From XCS-SL to XCSAM-SL. The above extensions of XCS-SL are simply added to the mechanism of XCSAM described in Section 3. Hence, XCSAM-SL evolves rules as in XCS-SL, but it is based on the best action map unlike XCS-SL.

5.3 Experimental Design

Since the ADL recognition task deals with the sequence datasets here, it is not possible to randomly assign test data like the cross-validation. Thus, the first 70% of the data is set to training data and the last 30% of the data to test data. We calculate the classification accuracy of the test data. The classification accuracy is the average over 30 experiments. We use the following parameter settings: \(N = 3000, \epsilon_0 = 1, \mu = 0.04, \chi = 0.8, \beta = 0.2, \alpha = 0.1, \nu = 5, \theta_{GA} = 25\), and the maximum memory size \(M = 8\). We compare the following four LCSs; XCS-SL, XCSAM-SL, in addition, XCS, and XCSAM both solving the ADL recognition task as a normal classification task.

5.4 Experimental Results

Table 5 shows the classification accuracies of the four LCSs on OrdonezA and OrdonezB. We found significant difference for all combinations of the LCSs \((p < 0.05)\) except for the following two cases: XCSAM and XCS-SL on OrdonezA; XCS and XCS-SL on OrdonezB (see Appendix B for more detail). From the table, since XCS-SL significantly outperforms XCS on both OrdonezA and OrdonezB, the extensions of XCS-SL improves on the classification accuracy. However, it should be noted that, on OrdonezB, XCSAM, which is not customized for the ADL task but employs the adequate action map, significantly outperforms XCS-SL which is customized for it. XCSAM-SL, which employs the adequate action map and the extensions for the ADL task, derives the best performance on both datasets. Our experiments suggested that the adequate action map fundamentally contributes to improve the LCS performance even on the LCS applications. In other words, when an LCS application employs an inadequate action map, its potential performance could degrade.

6. Revisit of Guideline

6.1 Analysis

We analyze the problem difficulties in terms of the classification boundary, and then draw a general guideline of the action
map. To measure the complexity of classification boundary, we calculate a classification probability for each example $P(c|s)$, where $c$ is a class and $s$ is an instance. On the normal 11-MUX the $P(c|s)$ for each instance $s$ with the correct class $c$ would be 1.0 while 0.0 for the incorrect class; that is, each instance can be determinately classified to the correct class.

Figure 4 shows the complexity of classification boundary of the generated dataset of 11-MUX with the different problem difficulties. The vertical axis indicates the classification probability $P(c|s)$ for each instance $s$ and the horizontal axis indicates each instance. Note that, since the dataset is composed of the fixed 5,000 11-MUX instances which some instances have the same attributes, the dataset may not have all type of 11-MUX instances (i.e., 2,048 instances); and thus, the horizontal axis does not reach to 2,048.

From Fig. 4 (a) ($P_{am} = 0.10$), the $P(c|s)$ for each instance is not calculated as 1.0 or 0.0 on 11-MUX with the overlapping instances. This means that each instance cannot be determinately classified to the correct class, and thus the overlapping instances make 11-MUX harder than the normal 11-MUX. When the overlapping rate is further increased to 0.30, the classification boundary becomes more complex (Fig. 4 (b)). Similar to the case of the overlapping instances, the classification boundary becomes more complex with increasing the missing rate (Fig. 4 (c), (d)). In contrast, for the class imbalance, $P(c|s)$ for each instance is calculated as 1.0 or 0.0 as in the normal 11-MUX (Fig. 4 (e), (f)). Note that, with the class imbalance, since the dataset is limited to have various instances which belong to the minority class 0, most instances have the correct class 1. That is, the dataset includes very few minority-class instances but many majority-class instances. Hence, when $ir$ is increased to 8 from 7, the dataset will include a fewer number of minority-class instances with a smaller probability $2^{-ir}$ than the case of $ir = 7$. Hence, each instance can be determinately classified to the correct class. Note that, as the same in the class-imbalance, on the multiplexer problem with the large search space, each instance can also be determinately classified to the correct class.

Accordingly, the four problem difficulties can be classified to two types. First, the overlapping instances and the missing attributes are a difficulty affecting the complexity of classification boundary. On these problem difficulties, the classification boundary becomes more complex with increasing their strength (i.e., the overlapping rate and the missing rate). Second, both the class imbalance and the large search space can be classified to a difficulty not affecting the classification boundary.

Let us revisit the guideline of action map (see Table 1) to draw a more general guideline. For the class imbalance and the large search space, it can be simply rephrased as follows: The best action map is adequate on a problem having no complexity of classification boundary. For the overlapping instances and the missing attributes with low strength, the complete action map is adequate on the problem with a low complexity of classification boundary, while the best action map is adequate for a high complexity. Accordingly, we can draw a more general guideline of action map as summarized in Table 6. Our guideline is still limited to suggesting the adequate action map with a clear degree of high or low complexity. However, it is very hard to show such a degree since the complexity of classification boundary would be changed depending on other factors, e.g., a state-action space.

### Table 5  Classification accuracies of four LCSs on OrdonezA and OrdonezB

|             | OrdonezA | OrdonezB |
|-------------|----------|----------|
| XCS         | 0.66     | XCS      | 0.461    |
| XCSAM       | 0.78     | XCSAM    | 0.537    |
| XCS-SL      | 0.78     | XCS-SL   | 0.493    |
| XCSAM-SL    | 0.82     | XCSAM-SL | 0.575    |

### Table 6  Revisited guideline of adequate action map

| Complexity of class boundary | Adequate action map |
|------------------------------|---------------------|
| No complexity                | Best action map     |
| Low                          | Complete action map |
| High                         | Best action map     |

6.2 Discussion

We here discuss why the adequate action map can be decided depending on the complexity of classification boundary (i.e., the type of problem difficulty or its strength).

**No complexity of classification boundary.** This is an ideal case, in which an instance in the dataset can be determinately classified to the correct class, and our analysis revealed that the best action map is the adequate action map for this case.

The efficiency of classifier evolution is important to improve the LCS performance. Then, the best action map potentially
can be a better strategy than the complete action map. This is because the complete action map requires many classifiers to cover all possible state-action space, while the best action map only requires few classifiers covering the highest return action in each state (i.e., the correct class in each instance). This difference affects the classifier-evolution efficiency; LCSs with the best action map (e.g., XCSAM and UCS) delete classifiers having the incorrect class in order to focus on evolution of classifiers having the correct class; thus, by evolving fewer classifiers focusing on the the correct class, those LCSs can generate good classifiers with fewer generations than the complete action map. This advantage of the best action map is greatly enhanced on a problem including a large search space or a class imbalance both of which make a problem hard to find the good classifiers.

Low complexity of classification boundary. This is a slightly complex case, in which an instance cannot be determinately classified to the correct class, but it can be almost successfully classified with a high classification probability $P(c|x)$. In this case, our analysis showed that the complete action map is the adequate action map.

Different from the case of no complexity of classification boundary, in this case, the best action map does not always contribute to correctly identify the good classifiers. Typically, LCSs with the best action map evolve the candidate of classifier likely to be the best action map; in other words, they often are designed to remove classifiers unlikely to be the best action map in order to hold only the correct class. Accordingly, when good classifiers (leading to the correct class) are wrongly identified as having an incorrect class due to noise, they will be removed from the classifier population. Hence, LCS with the best action map would forget the good classifiers which classify instances to the correct class, resulting in a low classification accuracy.

The complete action map can sidestep this limitation of the best action map. LCSs with the complete action map are designed to cover all types of action in GA; and thus, even if the good classifiers are wrongly identified as having the incorrect class (i.e., receiving the negative reward 0), they do not immediately remove those classifiers. This allows LCS to robustly perform with holding the good classifiers.

High complexity of classification boundary. This is a complex case, in which it is hard to determinately classified to the correct class with a low classification probability $P(c|x)$. In this case, our analysis showed that the best action map is the adequate action map.

In this case, different from the low complexity of classification boundary, it is very hard for LCSs to find good classifiers. Technically, LCSs decide an executed action with classifiers having high fitness (which is the common classifier parameter in LCSs); the good classifiers are identified when they have high fitness. Note that, the fitness is calculated from an estimated classification accuracy or the receive reward from a problem. However, with the high complexity, the fitness is unreliably estimated, and so LCSs cannot select the correct class without a reliable guidance of classifier fitness. Note that, with the low complexity, although the fitness is still limited to be accurately estimated, it is useful to identify good classifiers because good classifiers can classify instances to the correct class with a high probability.

In short, LCSs rely on identifying the good classifiers by learning classifier fitness in order to execute the correct class. However, LCSs fail to accurately estimate classifier fitness with a high complexity of classification boundary.

For this point, the evolution of rules that cover the best action map contributes to identify the good classifiers. LCSs attempt to remove classifiers having the incorrect class. Hence, LCSs with the best action map roughly identify good classifiers by deleting redundant classifiers. Even when LCSs wrongly delete good classifiers (as discussed above), we can suggest that this effect contributes to improve the LCS performance. This is because as shown in Fig. 2, with $P_{on} = 0.3$, $P_{on} = 0.3$, XCSAM outperforms XCS. In contrast, evolving the complete action map does not work to identify the good classifiers. Hence, LCS holds all types of actions for a given instance, but it might not select a good candidate of correct class without a reliable guidance of classifier fitness.

Summary. In summary, in terms of the efficiency of classifier evolution, the best action map is potentially the better strategy than the complete action map. Hence, it can be the adequate action map on a problem with no complexity of classification boundary. With the low complexity where the instance can be almost correctly classified (i.e., the fitness is still useful to identify the good classifiers), since evolving the best action map may wrongly delete classifiers having the correct class, the complete action map is the adequate action map with keeping all types of actions. When the classification boundary is further being complex (i.e., it is difficult to identify the good classifiers with the fitness), the best action map is adequate to robustly improve the XCS performance, since the evolution of the best action map acts on identifying the good classifiers.

7. Conclusion

We empirically confirmed that the adequate action map is determined depending on a type of problem difficulty or its strength. Our guideline is still derived from our limited experiments and validations. However, we showed an interesting fact that such a dependency of action map against the problem difficulties can be eventually caused from a complexity of classification boundary of the problem. Overall results claim that, before designing LCSs including their applications, the action map would have to be adequately selected at first because their performance degrades if they employ an inadequate action map. Our claims help to design, improve or extend LCS to be more powerful as a basic design methodology of LCS.

As future work, we should extend our guideline to be available in a problem including more than one problem difficulty. For this purpose, rather than conducting various experiments with a different combination of problem difficulties, we next attempt to theoretically analyze an effect of classification boundary to the type of action map. This would result in a reliable and general guideline of action map.

References

[1] J.H. Holland: Escaping brittleness: The possibilities of general purpose learning algorithms applied to parallel rule-based system, Machine Learning, Vol. 2, pp. 593–623, 1986.
[2] J. Holmes, P. Lanzi, W. Stolzmann, and S. Wilson: Learning
classifier systems: New models, successful applications, *Information Processing Letters*, Vol. 82, No. 1, pp. 23–30, 2002.

[3] D.E. Goldberg: Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley, 1989.

[4] P.L. Lanzi: Learning classifier systems: Then and now, *Evolutionary Intelligence*, Vol. 1, pp. 63–82, 2008.

[5] R. Urbanowicz and J. Moore: Learning classifier systems: A complete introduction, review, and roadmap, *Journal of Artificial Evolution and Applications*, Vol. 2009, 736398, 2009.

[6] S.W. Wilson: Classifier fitness based on accuracy, *Evolutionary Computation*, Vol. 3, No. 2, pp. 149–175, 1995.

[7] T. Kovacs: Strength or Accuracy: Credit Assignment in Learning Classifier Systems, Springer-Verlag London, 2004.

[8] E. Bernadé-Mansilla and J.M. Garrell-Guiu: Accuracy-based learning classifier systems: Models, analysis and applications to classification tasks, *Evolutionary Computation*, Vol. 11, No. 3, pp. 209–238, 2003.

[9] S.W. Wilson: ZCS: A zero level classifier system, *Evolutionary Computation*, Vol. 2, No. 1, pp. 1–18, 1994.

[10] M. Nakata, P.L. Lanzi, and K. Takadama: Complete action map or best action map in accuracy-based reinforcement learning classifier systems, *GECCO’14*, pp. 557–564, 2014.

[11] M. Nakata and T. Hamagami: Revisit of rule-deletion strategy for xcsam classifier system on classification, *Trans. Inst. Syst. Contr. Inf. Eng.*, Vol. 30, No. 7, pp. 273–285, 2017.

[12] M. Nakata, P.L. Lanzi, and K. Takadama: XCS with adaptive action mapping, *SEAL2012*, pp. 138–147, 2012.

[13] J. Luengo and F. Herrera: Domains of competence of fuzzy rule based classification systems with data complexity measures: A case of study using a fuzzy hybrid genetic based machine learning method, *Fuzzy sets and Systems*, Vol. 161, No. 1, pp. 3–19, 2010.

[14] J. Sánchez, R. Mollineda, and J. Sotoca: An analysis of how training data complexity affects the nearest neighbor classifiers, *Pattern Analysis and App.*, Vol. 10, No. 3, pp. 189–201, 2007.

[15] R. Mollineda, J. Sánchez, and J. Sotoca: Data characterization for effective prototype selection, *IbPRIA2017*, pp. 27–34, 2005.

[16] F. Ordoñez, P. Toledo, and A. Sanchis: Activity recognition using hybrid generative/discriminative models on home environments using binary sensors, *Sensors*, Vol. 13, No. 5, pp. 5460–5477, 2013.

[17] M.V. Butz and S.W. Wilson: An algorithmic description of XCS, *Soft Computing*, Vol. 6, No. 3–4, pp. 144–153, 2002.

[18] A. Orriols and E. Bernadé-Mansilla: Bounding XCS’s parameters for unbalanced datasets, *GECCO’06*, pp. 1561–1568, 2006.

[19] M.V. Butz, D.E. Goldberg, and K. Tharakunnel: Analysis and improvement of fitness exploitation in XCS: Bounding models, tournament selection, and bilateral accuracy, *Evolutionary Computation*, Vol. 11, No. 3, pp. 239–277, 2003.

[20] A. Asuncion and D.J. Newman: UCI machine learning repository, http://www.ics.uci.edu/∼mlearn/MLRepository.html.

[21] M. Nakata, T. Kovacs, and K. Takadama: A modified XCS classifier system for sequence labeling, *GECCO’14*, pp. 565–572, 2014.

[22] M. Nakata, T. Kovacs, and K. Takadama: XCS-SL: A rule-based genetic learning system for sequence labeling, *Evolutionary Intelligence*, Vol. 8, No. 2, pp. 133–148, 2015.

**Appendix A Calculation of Class Imbalance Strength**

The standard deviation $\sigma_{cls}$ is calculated as follows. First, in order to measure a pure class distribution which does not depend on the total instances each dataset has, a proportion of instances belonging to each class $c$ is calculated as:

$$ x_c = \frac{D_c}{\max_{c \in \mathcal{A}} D_c}, \quad (A.1) $$

where $D_c$ is the number of instances belong to class $c$, $\max_{c \in \mathcal{A}} D_c$ is the maximum number of instances belong to the class (i.e., the majority class $\mathcal{A}$), and $|\mathcal{A}|$ is the number of possible classes defined in the dataset. With the proportions $x_c$ and their average $\bar{x}$, the standard deviation $\sigma_{cls}$ is calculated as:

$$ \sigma_{cls} = \sqrt{\frac{\sum_{c \in \mathcal{A}} (x_c - \bar{x})^2}{|\mathcal{A}|}}. \quad (A.2) $$

We further explain how to calculate the class imbalance strength for the class imbalance level $ir$. As described in Section 3.2, probabilistically, per one minority-class instance, there are $2^{|\mathcal{A}|}$ majority-class instances in the dataset as defined in [18]. Consequently, $2^{|\mathcal{A}|}$ represents a ratio of minority and majority classes. Then, we use this ratio to calculate the standard deviation $\sigma_{cls}$ of $ir$. That is, we consider that the number of instances belonging to the minority class $D_0$ is 1 and that of instances for the majority class $D_0$ is $2^{|\mathcal{A}|}$; e.g., with $ir = 7$, $D_0$ is $2^7 = 128$ while $D_0$ is always 1. With these values and Equations (A.1) and (A.2), we get, for $ir = 7, 8, 9$, to $\sigma_{cls} = 0.702, 0.704, 0.706$, respectively.

**Appendix B Result of Statistical Test**

For all experimental cases shown in this paper, in order to select an appropriate statistical test, we checked the normality of variance of experimental result (i.e., the performances of 30 runs) through the Shapiro-Wilk test, and then we also checked the homogeneity of variances of two compared experimental results through F test. With $p < 0.05$ of the Shapiro-Wilk test, we should use the Mann-Whitney U test as non-parametric test; with $p \geq 0.05$ of the Shapiro-Wilk test and $p < 0.05$ of F-test, we should use Welch’s t-test for non-homogeneity; and finally, with $p \geq 0.05$ of the Shapiro-Wilk test and F-test, we can use Student’s t-test with the validated normality and homogeneity. Accordingly, through the appropriate statistical test determined by the above discussion, we can find significant difference ($p < 0.05$) for all cases except for two cases: XCSAM-XCS-SL on Ordzone A and XCS-XCS-SL on Ordzone B.

---

**Masaya Nakata (Member)**

He received the PhD in informatics from the University of Electro-Communications in 2015. Since 2016, he is an Assistant Professor of the Graduate School of Engineering, Yokohama National University. His research interests are in evolutionary computation and data mining. He received the IEEE Computational Intelligence Society Japan Chapter Young Researcher Award 2012.

**Keiich Takadama (Member)**

He received the Doctor of Engineering Degree from the University of Tokyo, in 1998. He joined Advanced Telecommunications Research Institute (ATR) International from 1998 to 2002 as a visiting researcher and worked at Tokyo Institute of Technology from 2002 to 2006 as a lecturer. He moved to the University of Electro-Communications as an associate professor in 2006 and is currently a professor from 2011. His research interests include learning classifier system and emergent computation. He is a member of IEEE and major AI- and informatics-related academic societies in Japan.