Revisit of Rule-Deletion Strategy for XCSAM Classifier System on Classification*

Masaya Nakata† and Tomoki Hamagami†

The XCSAM classifier system is an approach of evolutionary rule-based machine learning, which evolves rules advocating the highest-return actions at state, resulting in best classification. This paper starts with claiming a limitation that XCSAM still fails to evolutionary generate adequate rules advocating the highest-return actions. Then, under our hypothesis that this limitation is caused from the rule-deletion mechanism of XCSAM, we revisit its rule-deletion strategy in order to promote the evolution of adequate rules. Different from the existing deletion strategy which deletes two rules for each rule-evolution, our deletion strategy is modified to delete more than two rules as necessary. Experimental results on a benchmark classification task validate our modification powerfully works to evolve adequate rules, improving the performance of XCSAM. We further show our modification robustly enables XCSAM to perform well on real world classification task.

1. Introduction

Learning Classifier System (LCS)[6] is an evolutionary rule-based machine learning method which combines evolutionary computation[5] with a rule-based machine learning such as reinforcement learning[18]. LCS was originally introduced as a rule-based cognitive system inspired from a biological adaptive process to environment. A rule (called a classifier) is simply formed as IF-THEN rules. Then, through a rule-generalization process, LCS acquires rules indicating general patterns within a problem; LCS evolves rules to be general, and learns parameters of rules to identify accurately generalized rules. Accordingly, LCSs are capable of producing readable and simplified rules, resulting in human-understandable knowledge. LCSs are actively customized to be applicable to a wide range of knowledge discovery problems[12,2], such as breast cancer mining[9], human-activity recognition[13] and image processing[7], where we want to collect knowledge in order to design a solution.

On LCS field, a main challenge was to design LCS which produces maximally accurate and maximally generalized rules. In 1995, Wilson introduced an accuracy based LCS called XCS[20]. XCS evolves rules that compute the expected payoffs of all the available actions (or classes) in every possible state (or input), resulting in a complete action map. The complete action map is a type of learning strategy which defines how LCS covers a state-action space (see Fig.1-b). Then, the XCS’s complete action map ensures that XCS can always evolve the maximally accurate and maximally generalized rules to solve a given problem[11,10]. This is because that all the possible state-action pairs can be explored in every possible state, as required by reinforcement learning theory[18]. Relying on the XCS’s reliable rule-generalization, many modern LCSs employ the XCS mechanism as their basic LCS model[19]. For instance, the UCS classifier system[1] is a powerful XCS-like system customized for the supervised learning task such as classification task. UCS employs a supervised learning mechanism instead of the Q-learning mechanism of XCS.

While increasing XCS-based systems, we can claim that the XCS’s complete action map potentially requires many rules to cover all the state-action pairs and thus more iterations to reach optimality. To avoid the complete action map, our previous work intro-
duced an extension of XCS called the XCSAM classifier system[14]. Different from XCS, XCSAM employs a learning strategy of best action map which only covers the highest-return action at each state (see Fig. 1-a). We showed XCSAM outperforms XCS in a simple benchmark classification task with binary class (the multiplexer problem[20]), while successfully producing the maximally accurate and maximally general rules as well as XCS[14,15].

However, as we will show experimental evidences in Section 6, XCSAM does not derive any clear advantage compared with XCS on classification task with real world datasets. This is an unexpected result against our previous experiment on the benchmark classification task[14,15]. This paper is motivated to understand why XCSAM sometimes fails to outperform XCS. Specifically, considering XCSAM fails to derive a high classification accuracy, we suspect that XCSAM may be still limited to adequately generate rules advocating the best action map (i.e., best classification). Our hypothesis is that, this problem is caused by XCSAM’s mechanism: since most parts of the XCSAM mechanism are heavily inherited from XCS, those mechanisms are originally designed to build the complete action map. Consequently, XCSAM may fail to build the best action map, which may be a reason that the XCSAM performance degrades in some classification task.

Accordingly, this paper conducts an analysis and identifies an algorithmic limitation of XCSAM. Then, we modify the XCSAM mechanism to overcome the limitation, which results in improving the performance of XCSAM. For this purpose, we take the following steps. Firstly, this paper starts with demonstrating an experimental example that XCSAM fails to optimally solve a problem. Then, through analysis, we confirm a fact that XCSAM still fails to evolutionary generate adequate rules advocating the highest-return actions, i.e., XCSAM’s best action map does not form correctly as it is. From our analysis, as an algorithmic limitation of XCSAM, we suspect that the XCSAM’s rule-deletion strategy poorly perform to enable XCSAM to evolve adequate rules. Accordingly, Section 5 proposes a new rule-deletion mechanism for XCSAM and then our second test show that our mechanism enhances XCSAM to successfully build the best action map in the benchmark classification task. In Section 6, we further conduct an experiment on multi-class classification with real world datasets from UCI repository[17]. In Section 7, we further analyze a complexity of obtained rules (i.e., the number of rules in the population) by our rule-deletion mechanism on the real world classification task. Finally, Section 8 gives conclusion by summarizing contributions of this paper.

2. XCS Classifier System

Rule format. In XCS the rules consist of a condition, an action, and four main parameters[20]: (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 $p$; and (iii) the fitness $F$, which estimates the accuracy of the payoff prediction given by $p$; and finally (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. As shown in Fig. 2, 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[4], covering is activated when match set contains less than $\theta_{mon}$ actions; however, $\theta_{mon}$ is always set to the number of available actions $|A|$ 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 $s_t$. The system prediction is computed as the fitness weighted average of the predictions of rules in $[M]$, $cl \in [M]$, which advocate action $a_i$:

$$P(a_i) = \frac{\sum_{cl \in [M], a_i} cl.p \cdot cl.F}{\sum_{cl \in [M], a_i} cl.F}$$

(1)

where $[M], a_i$ represents the subset of rules of $[M]$ with action $a_i$, $cl.p$ identifies the prediction of rule $cl$, and $cl.F$ identifies the fitness of rule $cl$. 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 $r_t$ is returned to XCS.
Reinforcement Component. Next, the parameters of rules in $[A]$ are updated in the following order\[4\]: prediction $p$, prediction error $\epsilon$ and fitness $F$. Prediction $p$, Prediction error $\epsilon$ of classifier $cl$ are updated by eq. (2) and eq. (3) where $\beta$ is the learning rate ($0 < \beta < 1$).

\[
cl.p \leftarrow cl.p + \beta (r_t - cl.p)
\]  
(2)

\[
cl.\epsilon \leftarrow cl.\epsilon + \beta (|r_t - cl.p| - cl.\epsilon)
\]  
(3)

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

\[
cl.\kappa = \begin{cases} 
1, & \text{if } cl.\epsilon < \epsilon_0 \\
\alpha \left( \frac{cl.\epsilon}{\epsilon_0} \right)^{-\nu}, & \text{otherwise.}
\end{cases}
\]  
(4)

The accuracy $cl.\kappa$ means that a rule is considered to be accurate if its prediction error $cl.\epsilon$ is smaller than the threshold $cl.\epsilon_0$; a rule that is accurate has an accuracy $cl.\kappa$ equal to 1. A rule is considered to be inaccurate if its prediction error $cl.\epsilon$ is larger than $cl.\epsilon_0$; the accuracy $cl.\kappa$ of an inaccurate rule is computed as a potential descending slope with $\alpha(cl.\epsilon/\epsilon_0)^{-\nu}$. The parameter $\nu$ controls a decreasing slope which is decided depending on a problem. Then the fitness $cl.F$ of each rule in $[A]$ is updated with a relative accuracy $cl.\kappa'$ as follows,

\[
cl.\kappa' = \frac{cl.\kappa \cdot cl.num}{\sum_{a \in [A]} c.\kappa \cdot c.num}.
\]  
(5)

\[
cl.F \leftarrow cl.F + \beta (cl.\kappa' - cl.F).
\]  
(6)

Discovery Component. On a regular basis depending on the parameter $\theta_{mna}[4]$, the steady-state genetic algorithm is applied to rules in $[A]$. It selects two rules, copies them, and with probability $\chi$ 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.

3. XCSAM Classifier System

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 $cl$ is updated according to eq. (7), where $r_t \geq \zeta \max P(s_t,a)$ 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

\[
cl.op \leftarrow \begin{cases} 
cl.op + \beta (1 - cl.op), & \text{if } r_t \geq \zeta \max P(s_t,a) \\
cl.op + \beta (|cl.op| - cl.op), & \text{otherwise.}
\end{cases}
\]  
(7)

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 $\theta_{mna}$ based on the $op$ parameters. Initially, $\theta_{mna}$ 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 $\theta_{mna}$ as the average of $op$ of the rules in $[M]$ having the highest-return action $a^* = \arg\max_a P(a)$. The covering operator takes place if the number of different actions in $[M]$ (denoted by $|A_{[M]}|$) is less than $\theta_{mna}$. Accordingly, if $\theta_{mna}$ 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 GA as in XCS (i.e., generates two offspring and delete two rules). XCSAM generates $\max P(s_t,a)$ ideally converges to the highest reward at $s_t$, the best action can be identified by comparing the given reward with $\max P(s_t,a)[14].$
both the action set $[A]$ and also the not action set $[\bar{A}]$ with rules in $[M]$ having unselected actions (see Fig. 3). As shown in Fig. 3-a, when the selected action is identified as the best action with the condition $r_i \geq \xi_{max} P(s_i, a)$ (as used in 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, two rules are deleted depending on $\theta_{\text{mna}}$. Specifically, XCSAM deletes two rules from $[A]$ if a condition $|A| > \theta_{\text{mna}}$ is satisfied; otherwise, two the deleted rules selected from $[P]$. For instance, if $\theta_{\text{mna}}$ is 1 (the best action is already identified), rules in $[\bar{A}]$ can be deleted since they are unlikely to form the best action map.

When the selected action is not identified as the best action (Fig. 3-b), the parents are selected from $[A]$ to explore the rules having the best action, and the two rules are deleted from $[P]$ as in XCS.

4. Test on Benchmark Classification

Here, we attempt to confirm a limitation that XCSAM fails to evolutionary generate adequate rules advocating the best action map. Specifically, we conduct an experiment on a benchmark classification task (the concatenated multiplexer problem[3]) to evaluate the learning performance of XCSAM.

4.1 Benchmark Classification Problem

The $n \times m$ concatenated multiplexer problem ($n \times m$ CMP)[3] is a benchmark classification task, which is a kind of multiplexer problem[20]. For the detailed problem definition, we firstly explain the definition of the multiplexer problem and then CMP task.

The $l$-bit multiplexer problem is a boolean classification task with binary classes. An instance is defined over a binary string of $l = k + 2^k$ bits; the first $k$ bits represent an address pointing to the rest of $2^k$ 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 3-multiplexer function ($k = 1$), for an instance 101 the answer can be 1 as the bit at 2 (=1+1) with the address = 1 from the first bit ("1").

In the CMP task, an instance is a $n \times m$-bit binary string where $n$ strings whose length is $m$ are concatenated. Each $m$-bit string is an instance of the $m$-bit multiplexer problem. Then, the answer can be decided as follows. Firstly, an answer of each $m$-bit string (forming the instance) is calculated as solving the original $m$-bit multiplexer problem; then, the $n$ answers are concatenated as the final answer, forming an $n$-bit binary string. For instance, on $3 \times 3$ CMP, the final answer of an instance 000 001 001 is 1001, since answers of 3-bit instances can be determined as 0, 0 and 1 respectively. Thus, the length of instance $L$ and the number of available actions $|A|$ can be $n \times m$ and $2^n$ respectively. In this paper, we test XCSAM on three CMP tasks having different state-action spaces: we employ $2 \times 3$ CMP, $3 \times 3$ CMP, $4 \times 3$ CMP whose $L = 6$, 9 and 12; $|A| = 4$, 8 and 16 respectively.
Problem with fewer iterations than XCS; the classification accuracy of XCS reaches 100% after 180,000 iterations; while XCSAM reaches optimally after the 100,000 iterations. This is an expected fact that our previous works showed XCSAM solves a simple classification task with fewer iterations than that XCS takes[14,15]. However, when the state-action space is further increasing (i.e., 4×3 CMP), the classification accuracy of XCSAM unexpectedly degrades to about 45%; while XCS outperforms XCSAM with the higher classification accuracy about 62%.

The experimental results show that XCSAM outperforms XCS with a relatively small state-action space, while it is still limited to solve a problem with a larger scale problem (i.e., 4×3 CMP). This result captures an example that XCSAM is still limited to solve a classification problem optimally.

4.4 Analysis
To clarify why XCSAM failed to solve 4×3 CMP, we analyze the rules in the population acquired by XCSAM. Fig. 6 shows the optimality of action $op$ (the vertical axis) and the prediction error $\epsilon$ (the horizontal axis) of rules in $[P]$ over iterations. Note that the figure shows the rules acquired through one experiment. The $op$ indicates how much the rule’s action is likely to form the best action; the rule’s action can be the best action with the $op$ close to 1 while the not-best action with the $op$ close to $|A|$. The $\epsilon$ indicates the accuracy of rule; the rules can be identified as accurate when its $\epsilon$ is near to 0. Hence, XCSAM should have rules located at the bottom-left corner in Fig. 6, indicating the best action map is accurately built. Specifically, the best action map should consist of accurate rules whose action receives the highest-return reward at each state. In the other words, it should not include redundant rules which have an action receiving not the highest-return reward at each state. For instance, in Figure 1, the rules having action $a_1$ or $a_3$ are the redundant rules for XCSAM since those two actions will receive the reward $r=500,0$ (not the maximum reward 1000) respectively. In addition, inaccurate rules having a large prediction error $\epsilon$ are also the redundant rules for XCSAM as XCSAM is designed to learn the accurate rules having a small value of $\epsilon$ as XCS.

Fig. 6-a shows the rules in $[P]$ on 2×3CMP where XCSAM successfully derived the maximum classification accuracy. At the beginning of the experiment (1,000th iteration), many redundant rules (with a large $op$ and a large $\epsilon$) remain in the population. This is an initial rule-evolution process to explore accurate rules having the best action. After 10,000 iterations, XCSAM generates accurate rules having the best action while keeping many redundant rules. When the iteration is further increasing to 50,000, XCSAM gradually focuses on the best action map by deleting most redundant rules; however, accurate rules with $op=4$...
still remain in the final population (500,000th iteration). On 3×3CMP (Fig. 6-b), similar to 2×3CMP, XCSAM generates many rules at the beginning of the experiment (10,000th iteration); and then it gradually finds good rules advocating the best action map after 30,000th iterations. However, at the end of experiment (500,000th iteration), the final population consists of more redundant rules than the case of 2×3CMP. From Fig. 6-c, when the state-action space is further increasing (i.e., 4×3CMP), XCSACM completely fails to build the best action map (i.e., fail to identify the correct class for an input) and thus the classification accuracy would greatly degrade.

Those tendencies are clearly highlighted in the $\theta_{mna}$ value of XCSAM. Again, XCSAM adaptively calculates $\theta_{mna}$ from the parameter $op$; $\theta_{mna}$ should converge close to 1.0 with $op = 1.0$ when XCSAM has the rules having the best action at a given state; it converge to more than 1.0 when the population contains the redundant rules not having the best action (see Section 3). Fig. 7 shows the average $\theta_{mna}$ value of XCSAM calculated for a CMP instance on test problems; as in the experimental design, the $\theta_{mna}$ is reported as the moving average over the last 5000 test problems. All the plots are averages over 30 experiments. From the figure, the $\theta_{mna}$ value of XCSAM does not fully converge to 1.0 on 3×3 CMP and 4×3 CMP, because XCSAM wrongly maintains the redundant rules in the population (on 3×3 CMP) and fails to identify the good rules likely to be the best action map (on 4×3 CMP). This indicates that XCSAM fails to build the best action map when the state-action space is increasing.

Our analysis indicates that, while XCSAM can potentially build the best action map, the redundant rules wrongly remain in the final population with increasing the state action space. This results in degrading an efficiency of rule-evolution toward the best action map with accurate rules.

5. Revisit of Rule-deletion Strategy

This section presents a deletion strategy for XCSAM in order to overcome the limitation of XCSAM suggested in the previous section. Then, we test XCSAM with the new deletion strategy called XCSAM-DS on the concatenated multiplexer problem.

5.1 Design of Rule-deletion Strategy

As shown in our analysis, many redundant rules remain in the population, in other words, XCSAM continuously generates the redundant rules. This suggests that the redundant rules are selected as parents by GA. Accordingly, in order to build the best action map effectively, XCSAM should promote the evolution of rules likely to have the best actions. One possible solution is that, XCSAM removes most rules advocating not best actions so that those rules are not considered to be candidate of parents. In XCSAM, when $\theta_{mna} = 1$ (which means XCSAM correctly identified the best action), some rules will be redundant if $[M]$ has more than one type of action. However, the steady-state GA in XCSAM is designed to delete two rules for each generation (even when $[\bar{A}]$ consists of more than two redundant rules). For instance, on the case of Fig. 3-a, one redundant rule would remain in $[\bar{A}]$ after the rule-deletion is applied with $\theta_{mna} = 1$.

We modify the rule-deletion mechanism of XCSAM in order to delete more than two rules as necessary. We employ the condition $|A_{[M]}| > \theta_{mna}$ to decide whether the deleted rules can be selected from $[\bar{A}]$. Specifically, as shown in Fig. 8, our 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 $a$; and then it identifies an action $a_{mop}$ unlikely to be the best action compared with other actions, i.e., $a_{mop} = \arg \max_a OP(a)$. Next,
Nakata and Hamagami: Revisit of Rule-Deletion Strategy for XCSAM Classifier System on Classification

5.2 Test on Benchmark Classification

We conduct an experiment on the concatenated multiplexer problem in order to evaluate the performance of XCSAM with the new deletion mechanism called XCSAM-DS. We use the same problems (i.e., 2×3 CMP, 3×3 CMP, 4×3 CMP) with the same experimental settings.

Fig. 9 shows the classification accuracies of XCSAM-DS on CMPs. On 2×3 CMP XCSAM-DS derives the maximum classification accuracy as well as XCS and XCSAM (see Fig. 4 and Fig. 5). On 3×3 CMP, XCSAM-DS also reaches optimally slightly faster than XCSAM; XCSAM-DS derives the maximum classification accuracy after 80,000 iterations, but 100,000 iterations for XCSAM. When the state-action space is further increasing (i.e., 4×3 CMP), XCSAM-DS successfully solves the problem, while XCSAM and XCS fail to solve the problem optimally.

Next, we validate the effect of our rule-deletion to understand whether XCSAM-DS successfully builds the best action map. Fig. 10 shows the optimality of action $op$ (the vertical axis) and the prediction error $\epsilon$ (the horizontal axis) of rules in $[P]$ on 4×3CMP. Note that the figure shows the rules acquired through one experiment. From the figure, at the beginning of the experiment (50,000th iteration) XCSAM-DS generates many rules as the initial rule-evolution process. After 250,000 iterations, XCSAM-DS gradually shifts to the best action map with rules having a small $op$ and a small $\epsilon$. When the iteration is further increasing to 400,000, XCSAM-DS almost builds the best action map while few redundant rules remain in the population. However, the final population (500,000th iteration) consists of the accurate rules having the best actions, which indicates XCSAM-DS correctly builds the best action map. Fig. 11 shows the average $\theta_{\text{mna}}$ value of XCSAM-DS calculated for a CMP instance on test problems. From the figure, the $\theta_{\text{mna}}$ value of XCSAM-DS almost fully converge to 1.0 on 3×3 CMP and 4×3 CMP in which $\theta_{\text{mna}}$ value of XCSAM does not fully converge to 1.0. This indicates that, XCSAM-DS successfully builds the best action map with very fewer redundant rules than XCSAM.

The experiment results suggest that our deletion mechanism promotes to build the best action map. Consequently, XCSAM-DS outperforms XCSAM since the XCSAM-DS can classify the inputs to their correct classes with a guidance of the best action map. In summary, our deletion strategy, i.e., deleting more than two redundant rules as necessary, can be an adequate strategy for the rule-deletion in evolving the best action map.

6. Test on Real World Classification

Finally, we apply four LCSs (XCS, XCSAM, UCS XCSAM-DS) to the classification task with real world datasets. In this section, different from the previous section, we compare XCSAM-DS with UCS which is a customized LCS for a supervised learning task. As described in Section 1, UCS employs a supervised learning instead of the reinforcement learning as in XCS and XCSAM. We do not describe the mechanism of UCS but it can be found in[1]; in short, as an advantage of UCS, UCS receives an instance with its correct class (i.e., the best action is always given to UCS) while XCS-based systems identify the best action through a try and error process.

6.1 Experimental Design

This experiment uses 30 datasets available in the UCI repository[17] as summarized in Table 1-a. Here, in the table, $L$ and $|A|$ represent the length of input
Table 1 The dataset configurations, classification results and $\theta_{max}$ values on the real world classification task. $L$ and $|A|$ indicate the instance length and the number of available actions respectively.

| a) Dataset              | b) Classification accuracy | c) $\theta_{max}$ |
|-------------------------|-----------------------------|-------------------|
|                         | XCS  | XCSAM | UCS  | XCSAM-DS | XCSAM  | XCSAM-DS   |
| Annealing               | 38   | 6     | 0.925| 0.906 | 0.881  | 0.915  | 1.390  | 1.366  |
| Audiology               | 69   | 24    | 0.460| 0.588 | 0.579  | 0.569  | 7.410  | 6.358  |
| Aus.Creditcard          | 14   | 2     | 0.845| 0.836 | 0.855  | 0.841  | 1.022  | 1.026  |
| Balance Scale           | 3    | 3     | 0.829| 0.803 | 0.878  | 0.854  | 1.038  | 1.030  |
| Breast Cancer W         | 9    | 2     | 0.957| 0.967 | 0.958  | 0.969  | 1.013  | 1.008  |
| Bupa                    | 6    | 2     | 0.687| 0.672 | 0.676  | 0.667  | 1.024  | 1.028  |
| Cardiotocography        | 23   | 10    | 0.725| 0.732 | 0.763  | 0.739  | 2.956  | 2.679  |
| Cmc                     | 9    | 3     | 0.528| 0.507 | 0.504  | 0.517  | 1.629  | 1.499  |
| Dermatology             | 33   | 6     | 0.164| 0.183 | 0.172  | 0.205  | 5.455  | 5.358  |
| Glass                   | 9    | 6     | 0.737| 0.696 | 0.710  | 0.691  | 1.204  | 1.405  |
| Heart-c                 | 13   | 5     | 0.482| 0.512 | 0.528  | 0.492  | 1.722  | 1.713  |
| Heart-h                 | 13   | 5     | 0.644| 0.680 | 0.660  | 0.614  | 1.248  | 1.174  |
| Hepatitis               | 19   | 2     | 0.884| 0.832 | 0.820  | 0.826  | 1.018  | 1.039  |
| Iris                    | 4    | 3     | 0.960| 0.940 | 0.940  | 0.953  | 1.001  | 1.000  |
| Labor                   | 16   | 2     | 0.797| 0.783 | 0.823  | 0.837  | 1.294  | 1.228  |
| LED 10                  | 7    | 10    | 0.693| 0.677 | 0.710  | 0.706  | 4.324  | 3.541  |
| Libras                  | 91   | 15    | 0.147| 0.172 | 0.242  | 0.222  | 13.385 | 13.159 |
| Ozone                   | 73   | 2     | 0.791| 0.800 | 0.767  | 0.832  | 1.383  | 1.237  |
| Pima Indians            | 8    | 2     | 0.577| 0.577 | 0.563  | 0.602  | 1.042  | 1.054  |
| Primary-tumor           | 17   | 21    | 0.398| 0.348 | 0.363  | 0.375  | 9.700  | 6.243  |
| Segment                 | 19   | 7     | 0.942| 0.948 | 0.944  | 0.953  | 1.177  | 1.116  |
| Sonar                   | 60   | 2     | 0.453| 0.472 | 0.525  | 0.490  | 1.990  | 1.990  |
| Soybean                 | 35   | 19    | 0.214| 0.452 | 0.483  | 0.401  | 10.983 | 12.469 |
| Teaching Assistant      | 5    | 3     | 0.650| 0.637 | 0.622  | 0.676  | 1.111  | 1.075  |
| Vowel                   | 13   | 11    | 0.620| 0.625 | 0.796  | 0.845  | 4.497  | 1.453  |
| Wine                    | 13   | 3     | 0.956| 0.955 | 0.956  | 0.956  | 1.036  | 1.045  |
| Wine Quality Red        | 12   | 11    | 0.579| 0.579 | 0.598  | 0.595  | 4.349  | 3.119  |
| Wine Quality White      | 12   | 11    | 0.496| 0.471 | 0.435  | 0.488  | 5.320  | 5.509  |
| Yeast                   | 8    | 10    | 0.536| 0.476 | 0.537  | 0.558  | 5.152  | 4.243  |
| Zoo                     | 17   | 7     | 0.950| 0.921 | 0.960  | 0.951  | 1.002  | 1.001  |

and the number of possible actions respectively.

As in a previous work[1], if the attributes are binary, the data is coded in the ternary alphabet (0, 1 and #). If the attributes are real, the data is coded as an interval range. For simplicity, all real attributes are normalized to the range [0,1]. Nominal attributes are translated into numeric values, and so considered as real attributes. The datasets are run on a stratified ten-fold cross-validation test and we calculate the classification accuracy of each LCS on the test dataset. The parameter settings are the same as the previous experiment except for $N = 6400$. For UCS, as in[1], we use $\nu=10$ and $acc_0 = 0.99$ and other parameters of UCS are the same of XCS. The maximum iteration is 100,000.

We also statistically compare the classification results on overall datasets[8]. In detail, we firstly apply the Friedman test to the overall classification results; and we further apply the Holm method as a post-hoc test if the Friedman test reject the null hypothesis (i.e., there is a significant different on the overall classification results).

### 6.2 Results

Table 1-b shows the classification accuracies of three LCSs. From the overall results, XCSAM, XCSAM-DS, UCS seemingly derives the better classification accuracies than XCS on datasets having relatively large instance length $L$ and the number of available actions $|A|$, e.g., Audiology, Libras and Soybean. Based on those classification results, we now apply the statistical test. Table 2-a shows the average rank of the four LCSs which are calculated from the overall classification results in Table 2-b. From those average rank, we get the Friedman statistic value $\chi^2_F = 7.830$ and $F_P = 2.763$ for the F distribution with 3 and 87 degrees of freedom (see[8] for the detailed calculations of $\chi^2_F$ and $F_P$). Finally, with $F_P$, the $p$ value can be calculated as 0.047. Hence, the overall results of the four LCSs likely have a significant difference as $p = 0.047$ is smaller than 0.05. Let us continue to apply the Holm method as the post-hoc test. We first test each pair of LCSs with the Wilcoxon signed rank test; we calculate the $p$ value of Wilcoxon signed rank test, denoted by $p_W$. Then, with Holm method’s
procedure, we further calculate an adjusted \( p \) value denoted by \( p_{adj} = \frac{p_W}{i} \), and finally we compare \( p_{adj} \) with the standard condition \( p_{adj} < 0.05 \). For all possible pairs of LCSs, the \( p_W \) and \( p_{adj} \) values are summarized in Table 2-b. From the table, we can see that XCSAM-DS significantly outperforms XCSAM as \( p_{adj} = 0.042 < 0.05 \). Our statistical test suggests that, our rule-deletion mechanism (i.e., XCSAM-DS) likely improved the performance of XCSAM on the real world classification task.

Let us further analyze the classification results. As we revealed in Section 5, XCSAM-DS outperforms XCS and XCSAM when the state-action space of problem is increasing. This tendency may be observed in the real world classification task. On the 30 datasets, we here simply quantify the state-action space of each LCS better than XCS. However, XCSAM-DS cannot build the best action map compared with XCSAM. Considering that in Section 4 XCS and XCSAM solved 3 \( \times \) 3 CMP whose \( L \times |A| \) is 72 (\( L = 9 \) and \( A = 8 \)) but they failed to optimally solve 4 \( \times \) 3 CMP with \( L \times |A| = 192 \) (\( L = 12 \) and \( A = 16 \)), we can suppose \( L \times |A| \) larger than 72 could be a boundary which makes XCSAM hard to build the best action map. Based on our definition, we selected the following 15 datasets as having a relatively large state action space: \textit{Sonar, Soybean, Vowel, Wine Quality Red, Wine Quality White, Yeast} and \textit{Zoo}.

For those selected datasets, we again apply the Friedman test and the Holm test. Table 3 summarizes the average rank, the \( p_W \) and \( p_{adj} \) values of \( p \) values and \( \theta_{mna} \) of XCSAM and XCSAM-DS. Firstly, from those average rank, we get \( \chi^2_p = 10.220 \) and \( F_p = 4.114 \) for the F distribution with 3 and 42 degrees of freedom; the \( p \) value can be calculated as 0.012 from \( F_p \). So, with \( p < 0.05 \), we further apply the Holm test. From Table 3-b, we can identify that there is a significant difference of classification accuracy between XCS and XCSAM-DS; XCSAM-DS is the only LCS better than XCS. However, XCSAM-DS cannot outperform XCSAM and UCS. In detail, XCSAM-DS does not outperform XCSAM on the datasets \textit{Audiology} and \textit{Soybean} where we can expect our rule-deletion strategy effectively performs since they include a large-state action space. We suspect this is because that, XCSAM-DS may evolve very specific rules to training sets.

Finally, we evaluate how correctly XCSAM-DS builds the best action map compared with XCSAM. Table 3-c summarizes the \( \theta_{mna} \) values of XCSAM and XCSAM-DS. Those values are reported as an average value of \( \theta_{mna} \) calculated for instances on the test datasets (i.e., subsets divided from the 10-fold cross validation). For most datasets, XCSAM-DS performs with a smaller \( \theta_{mna} \) than XCSAM, e.g., on \textit{Vowel}.

Through a statistical test (Wilcoxon signed-rank test), we find a significant difference for overall results (\( p_W \) = 0.035 < 0.05) in which the average ranks of XCSAM and XCSAM-DS are 1.717 and 1.283.

---

Even if a dataset does not have the instances fully covering the state-action space \( L \times |A| \), LCS is permitted to evolutionary explores its space to find better rules.
respectively; the smaller $\theta_{\text{mna}}$ is assigned to the first rank. And so XCSAM-DS averagely performs with a smaller $\theta_{\text{mna}}$ than XCS. Hence, XCSAM-DS builds the best action map with fewer redundant rules than XCSAM, i.e., building the best action map successfully. We further demonstrate an example showing how XCSAM-DS builds the best action map on the dataset Vowel which a clear advantage of XCSAM-DS is found compared with XCSAM; XCSAM-DS derives 84.5% classification accuracy while 62.5% for XCSAM. As we noted in Section 5, we can now suppose on Vowel, XCSAM-DS successfully builds the best action map with the accurate rules on Vowel while in XCSAM the redundant rules still remain in the final population. This can be validated in Fig. 12. Fig. 12 analyzes the optimality of action $op$ (the vertical axis) and the prediction error $\epsilon$ (the horizontal axis) of rules in $[P]$ on Vowel. From the figure, XCSAM generates the redundant rules until the end on experiment (100,000th iteration). In contrast, XCSAM-DS successfully shifts to the best action map with increasing the iterations; it eventually evolves rules forming the best action map at the 100,000th iteration.

In summary, our experiment revealed XCSAM-DS likely outperforms the original XCSAM from overall results. As we are motivated in overcoming the limitation of XCSAM, our experiment supports the effectiveness of our proposal. When focusing the datasets having a large state-action space space, we cannot find a significant difference between XCSAM and XCSAM-DS as XCSAM also performs well on the problem with the large state-action space. Instead, an interesting finding is that, only XCSAM-DS significantly outperforms XCS on the 15 selected datasets; compared with UCS and XCSAM, XCSAM-DS stably derives better classification accuracies than XCS on different datasets. That is, we can suggest that XCSAM-DS can be a stable good solver for a dataset having a large state-action space.

7. Discussion

As the previous sections revealed our rule-deletion mechanism enables XCSAM to build the best action map correctly, which results in improving the XCSAM performance. Here, we additionally evaluate our rule-deletion mechanism in terms of a complexity of obtained rules. In detail, we compare the number of rules in the final population (at the end of iteration) on the real world classification in Section 6; LCSs aims to solve the problem with as small population size as possible.

Table 4 compares the final population size (i.e., the number of rules at the end of iterations) of the four LCSs on the 30 datasets. Table 4-b summarizes those average rank in which the smallest population size is assigned to the first rank. From Table 4-b, XCS averagely obtained smaller population sizes than the other three LCSs and UCS seemingly produced larger population sizes than XCS, XCSAM and XCSAM-DS. Through the Friedman test which finds a significant difference with $p=1.116E-07$ (calculated from $\chi^2_F = 29.800$ and $F_F = 15.7356$ for the F distribution with 3 and 87 degrees of freedom), we find significant differences for all pairs except for XCSAM vs XCSAM-DS (see Table 4-c). That is, XCS averagely obtained smaller population sizes XCSAM, UCS and XCSAM-DS; while UCS derives larger population sizes than XCSAM and XCSAM-DS.

This is an unexpected fact that, although XCSAM-DS deletes more than two rules (but other LCSs delete the maximum two rules), XCSAM-DS does not produce the smaller population sizes than those LCSs. Consequently, our rule-deletion mechanism does not really improve to reduce the population size (i.e., the complexity of rules) compared XCS and XCSAM.
Table 4  The final population sizes of the four LCSs on the real-world classification task

| Dataset                | XCS  | XCSAM | UCS   | XCSAM-DS |
|------------------------|------|-------|-------|----------|
| Annealing              | 6310.7 | 6326.1 | 6325.6 | 6332.1   |
| Audiology              | 6305.9 | 6360.5 | 6355.8 | 6356.6   |
| Aus.Creditcard         | 5591.2 | 5758.0 | 5824.2 | 5739.1   |
| Balance Scale          | 3432.0 | 4435.7 | 4731.7 | 4472.6   |
| Breast Cancer W        | 5299.5 | 5557.0 | 5713.7 | 5542.3   |
| Bupa                   | 4413.2 | 5285.9 | 5302.9 | 5309.2   |
| Cardiotocography       | 5906.4 | 6121.5 | 6125.8 | 6093.3   |
| Cmc                    | 5102.1 | 5090.2 | 5377.3 |           |
| Dermatology            | 6136.1 | 6278.5 | 6272.6 | 6275.8   |
| Glass                  | 5502.9 | 5671.7 | 5732.2 | 5713.7   |
| Heart-c                | 5564.2 | 5900.7 | 5930.1 | 5918.8   |
| Heart-h                | 5754.3 | 5852.0 | 5908.1 | 5889.6   |
| Hepatitis              | 5236.0 | 5578.7 | 5681.7 | 5601.5   |
| Iris                   | 2872.7 | 4260.1 | 4715.9 | 4209.1   |
| Labor                  | 5942.5 | 5934.1 | 5926.6 | 5912.4   |
| LED 10                 | 1387.0 | 2650.7 | 540.9  | 1511.4   |
| Libras                 | 6397.2 | 6398.9 | 6399.1 | 6398.3   |
| Ozone                  | 6390.5 | 6386.7 | 6376.5 | 6384.0   |
| Pima Indians           | 5051.4 | 5417.6 | 5577.5 | 5499.1   |
| Primary-tumor          | 4361.3 | 5497.8 | 5635.3 | 5355.4   |
| Segment                | 5985.4 | 6053.1 | 6042.9 | 6013.3   |
| Sonar                  | 6386.4 | 6385.4 | 6389.0 | 6386.9   |
| Soybean                | 6188.9 | 6279.1 | 6227.9 | 6283.8   |
| Teaching Assistant     | 4361.3 | 5497.8 | 5635.3 | 5355.4   |
| Vowel                  | 5758.3 | 5783.0 | 5968.4 | 5877.7   |
| Wine                   | 5890.3 | 5821.7 | 5884.9 | 5812.2   |
| Wine Quality Red       | 5493.9 | 5607.4 | 5747.9 | 5412.3   |
| Wine Quality White     | 4361.3 | 5497.8 | 5635.3 | 5355.4   |
| Yeast                  | 5385.3 | 5417.4 | 5369.3 | 5186.5   |
| Zoo                    | 2036.5 | 4662.2 | 4740.7 | 4623.8   |

is because that, even if XCSAM-DS deletes more than two redundant rules unlike other three LCSs, instead XCSAM-DS fills the population with many types of accurate rules likely to be the best action map. In the other words, our deletion strategy promotes to produce various accurate rules (and thus, specific rules), which results in increasing the diversity of rules. As summarized in Section 6, this effect of our proposal almost contributes to improve the XCSAM performance (as our experiments revealed); but this sometimes wrongly causes the over-fitting problems.

8. Conclusion

This paper empirically claimed the limitation that XCSAM still fails to evolutionary generate adequate rules advocating the highest-return actions. Our analysis showed, XCSAM continually generates the redundant rules and thus the best action map is not correctly built. Then, we identified an algorithmic limitation of XCSAM that XCSAM deletes two rules for each rule-generation; this rule-deletion strategy is well-used in LCS including XCS, however, it may not be adequate for XCSAM. Accordingly, we introduced a rule-deletion mechanism that deletes more than two redundant rules as necessary in order to promote the evolution of rules advocating the best action map. This effect has been validated by our analysis. Our proposal is a very simple modification from the original XCSAM mechanism. However, on a benchmark classification task, the experiments suggest our modification powerfully works to build the best action map in XCSAM. Consequently, XCSAM-DS (XCSAM with the new rule-deletion mechanism) improves the performance on the problems with increasing the state action space. We further showed, on the real world classification task, XCSAM-DS derives the better performance than XCS and XCSAM for some datasets; XCSAM-DS derives a clear advantage compared with XCS.

However, as shown by the result with the real world classification task, our rule-deletion strategy may lead to learn very specific rules for training datasets, which results in degrading a generality of acquired rules. This may implicitly suggest a disadvantage of the concept of best action map in LCSs. Accordingly, we next should further analyze our rule-deletion strat-
Table 5 The average rank of population size of four LCSs and the adjusted \( p \) value with Holm method on all 30 datasets

| Compared pair            |\( p_W\)   |\( p_{adj} \) |
|--------------------------|-----------|-------------|
| 6 XCS vs XCSAM           | 1.880E-05 | 1.880E-05   |
| 5 XCS vs UCS             | 1.000E-04 | 5.000E-04   |
| 4 XCS vs XCSAM-DS        | 4.400E-04 | 1.760E-03   |
| 3 UCS vs XCSAM-DS        | 7.140E-04 | 2.142E-03   |
| 2 XCSAM vs UCS           | 0.004     | 0.008       |
| 1 XCSAM vs XCSAM-DS      | 0.092     | 0.092       |

Reference to understand which rules should be held in the final population. Then, rather than building the pure best action map, a new concept of learning strategy which combines best action map with complete action map, may be a good solution to overcome the limitations of both existing action maps.

References

[1] E. Bernadó-mansilla and J. M.Garrell-Guiu: Accuracy-based learning classifier systems: models, analysis and applications to classification tasks; *Evolutionary Computation*, Vol. 11, pp. 209–238, MIT press (2003)

[2] L. Bull, E. Bernadó-Mansilla and J. H. Holmes, editors: *Learning Classifier Systems in Data Mining*, volume 125 of *Studies in Computational Intelligence*, Springer (2008)

[3] 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, MIT press (2003)

[4] M. V. Butz and S. W. Wilson: An algorithmic description of XCS; *Journal of Soft Computing*, Vol. 6, Nos. 3–4, pp. 144–153, Springer (2002)

[5] D. E. Goldberg: *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison Wesley (1989)

[6] 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)

[7] M. Iqbal, S. S. Naqvi, W. N. Browne, C. Hollitt and M. Zhang: Salient object detection using learning classifiersystems that compute action mappings; *Proceedings of Conference on Genetic and Evolutionary Computation 2014*, GECCO2014, pp. 525–532, ACM (2014)

[8] J. Demšar: Statistical comparisons of classifiers over multiple data sets; *Journal of Machine Learning Research*, Vol. 7, pp. 1–30 (2006)

[9] F. Kharbat, L. Bull and M. Odeh: Mining breast cancer data with XCS; *Proceedings of Conference on Genetic and Evolutionary Computation 2007*, GECCO 2007, pp. 2066–2073 (2007)

[10] T. Kovacs: Evolving optimal populations with XCS classifier systems; Technical Report CSR-96-17 and CSRP-96-17, School of Computer Science, University of Birmingham, Birmingham, U.K. (1996)

[11] T. Kovacs: Strength or accuracy? Fitness calculation in learning classifier systems; P. L. Lanzi, W. Stolzmann and S. W. Wilson, editors, *Learning Classifier Systems*, Volume 1813 of *Lecture Notes in Computer Science*, pages 143–160, Springer Berlin Heidelberg (2000)

[12] P. L. Lanzi: Learning classifier systems: then and now; *Evolutionary Intelligence*, Vol. 1, pp. 63–82 (2008)

[13] M. Nakata, T. Kovacs and K. Takadama: A modified XCS classifier system for sequence labeling; *Proceedings of Conference on Genetic and Evolutionary Computation 2014*, GECCO 2014, pp. 565–572, ACM (2014)

[14] M. Nakata, P. L. Lanzi and K. Takadama: XCS with adaptive action mapping; *Simulated Evolution and Learning - 9th International Conference*, SEAL 2012, Volume 7673 of *Lecture Notes in Computer Science*, pages 138–147, Springer (2012)

[15] M. Nakata, P. L. Lanzi and K. Takadama: Selection strategy for XCS with adaptive action mapping; *Proceedings of Conference on Genetic and Evolutionary Computation 2013*, GECCO 2013, pp. 1085–1092, ACM (2013)

[16] A. Orriols-Puig and E. Bernadó-Mansilla: Bounding XCS’s parameters for unbalanced datasets; *Proceedings of Conference on Genetic and Evolutionary Computation 2006*, GECCO 2006, pp. 1561–1568, ACM (2006)

[17] UCI Machine Learning Repository; http://archive.ics.uci.edu/ml/

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

[19] R. J. Urbanowicz and J. H. Moore: Learning classifier systems: A complete introduction, review, and roadmap; *Journal of Artificial Evolution and Applications*, 2009:1:1–1:25 (2009)

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

Masaya Nakata (Member)

He received a PhD in informatics from the University of Electro-Communications. Since 2016, he has been an Assistant professor at the Graduate School of Engineering, Yokohama National University. His research interests include evolutionary computation, reinforcement learning, data mining, more specifically, learning classifier systems. He received the best paper award and the IEEE Computational Intelligence Society Japan Chapter Young Researcher Award from the Japanese Symposium of Evolutionary Computation 2012. He was a co-organizer of the International Workshop on Learning Classifier Systems (IWLCS) for 2015-2016.

Tomoki Hamagami

He received a PhD from Graduate School of Science and Technology, Chiba University in 1999, while working at SECOM Co., Ltd. He is currently a professor at the Division of Physics, Electrical and Computer Engineering in Yokohama National University. He has worked on distributed intelligent systems, artificial intelligence and applications of these technologies: social systems, and medical, and welfare support systems. He is a member of IEEJ, IEICE, IPSJ, SICE, and IEEE.