The XCS classifier system is an evolutionary rule-based learning technique powered by a Q-learning like learning mechanism. It employs a global deletion scheme to delete rules from all rules covering all state-action pairs. However, the optimality of this scheme remains unclear owing to the lack of intensive analysis. We here introduce two deletion schemes: 1) local deletion, which can be applied to a subset of rules covering each state (a match set), and 2) stronger local deletion, which can be applied to a more specific subset covering each state-action pair (an action set). The aim of this paper is to reveal how the above three deletion schemes affect the performance of XCS. Our analysis shows that the local deletion schemes promote the elimination of inaccurate rules compared with the global deletion scheme. However, the stronger local deletion scheme occasionally deletes a good rule. We further show that the two local deletion schemes greatly improve the performance of XCS on a set of noisy maze problems. Although the localization strength of the proposed deletion schemes may require consideration, they can be adequate for XCS rather than the original global deletion scheme.

Keywords: learning classifier system, evolutionary reinforcement learning, deletion scheme

1. Introduction

Learning Classifier Systems (LCSs) [1] are an evolutionary rule (classifier)-based reinforcement learning technique [2] that aims to produce a set of generalized rules as a solution to a problem. Since the generalized rules represent a common pattern of inputs with simple if-then rule representation, LCSs are useful evolutionary mining tools for extracting understandable knowledge from a problem. LCSs are very actively applied to a wide range of data-mining tasks [3, 4]. In addition, they can also be used as a knowledge discovery system in the context of reinforcement learning problems such as online control task [5].

In LCS research a long standing challenge has been to design an LCS that learns accurately generalized rules. In 1995, Wilson proposed an XCS classifier system [6], that could learn maximally accurate and maximally generalized rules [7, 8]. Technically, XCS employs accuracy-based fitness as a criterion to evaluate the accuracy of generalized rules. Moreover, XCS generates accurately generalized rules with high fitness and deletes inaccurately generalized rules with low fitness. XCS has been the most popular and basic LCS algorithm and recent works show that XCS-based systems can successfully solve complex machine learning problems [9–12].

The XCS mechanism is, however, very complex because it is a combined system of a Q-learning like technique and the genetic algorithm (GA). For instance, many XCS parameter settings have not been optimized with theoretical explanations as we claimed in [13]. However, many works have attempted to understand the working of XCS and have provided the useful insights into the XCS mechanism in terms of both GA and the Q-learning like learning, more generally, the method of combining the above two techniques [3, 14].

The role of GA in XCS has been analyzed intensively as summarized in [15] because GA greatly affects the performance of XCS in producing accurately generalized rules. Previous analyses of GA, for example, [16, 17] have often focused on the rule selection scheme (i.e., parent selection) because the selection scheme directly decides whether XCS can generate accurately generalized rules. For rule deletion in GA, which is the focus of this paper, Kovacs studied the adequate selection probability of the XCS deletion scheme. However, a few XCS deletion schemes were not discussed adequately to conclude empirically that they are adequate for XCS. We are motivated to give insights into the deletion schemes.

The original XCS deletion scheme (called the population-based deletion) was designed to be a global deletion scheme that deletes rules from a population consisting of all rules in XCS. One of principles of this scheme is to delete inaccurate rules with relatively low fitness compared to the average fitness of all rules. While the global deletion scheme seems to be a good strategy because all types of inaccurately generalized rules can be deleted averagely, the optimality of this scheme has not been discussed owing to the lack of intensive analysis. Moreover, it is unclear whether the global deletion scheme is more adequate than a local deletion scheme.
which deletes rules from a subset of the population (i.e.,
deletes from fewer types of rules as opposed to considering
all types of rules).

Accordingly, the aim of this paper is to reveal how the
local and global deletion schemes affect the performance
of XCS. To this end, we compare three different deletion
schemes: 1) population-based deletion, which is the
global deletion scheme; 2) match set-based deletion, a lo-
cal deletion scheme that selects rules from a subset of the
population (called a match set); and 3) action set-based
deletion, a stronger local deletion scheme than match set-
based deletion, which is applied to a subset of the match
set (called an action set). Please note that the XCS mecha-

In Section 2, we describe the mechanism of XCS. In
Section 3, we test the three deletion schemes on a bench-
mark reinforcement learning problem (i.e., the maze prob-
lem) to find how the performance of XCS can differ de-
pending on the deletion scheme. In Section 4, we perform
an analysis to explain the differences in XCS performance
revealed in Section 3. In Section 5, under the assump-
tion that the different deletion schemes greatly affect XCS
performance in noisy problems, we conduct an additional
experiment on a noisy maze problem to validate the as-
sumption. Finally, in Section 6, we provide a summary of
this paper.

2. XCS

XCS [6] is a reinforcement learning method in which
generalization is obtained through the evolution of a pop-
ulation of condition-action-prediction rules (called clas-
sifiers). A detailed algorithmic description can be found
in [18].

Rule. In XCS, the rules consist of a condition, an ac-
tion, and five main parameters: (i) prediction \( p \), which es-
imates the average payoff that the system expects when
the rule is used; (ii) prediction error \( \varepsilon \), which estimates the
average absolute error of the prediction \( p \); (iii) fitness \( F \),
which estimates the average relative accuracy of the pay-
off prediction given by \( p \); (iv) action set size \( as \), which es-
imates the average size of the action sets to which this
rule belongs; and finally (v) numerosity \( num \), which indi-
cates the number of copies of rules with the same condi-
tion and the same action present in the population.

Performance Component. In each time step, XCS builds
a match set \([M]\) containing the rules in the population
\([P]\) whose condition matches the current sensory inputs;
if \([M]\) does not contain all the possible actions covering
operator is activated and creates a set of rules that match
and cover all missing actions. The covering operator is
activated when the match set contains less than \( \theta_{num} \) ac-
tions; however, \( \theta_{num} \) is usually set to the number of avail-

Reinforcement Component. When the reward \( r_t \) is re-
ceived and the match set \([M]\) with respect to the resulting
sensory input is formed, the parameters of the rules in \([A]\)
are updated in the following order [18]: prediction, pre-
diction error, action set size, and fitness.

The prediction \( cl.p \) of each rule \( cl \) in \([A]\) is updated with learning rate \( \beta \) \((0 < \beta \leq 1)\) and discount rate \( \gamma \)
\((0 < \gamma \leq 1)\). If the system solves a supervised classifi-
cation (single-step) problem or a termination criterion is
met in a reinforcement learning (multi-step) problem, the pre-
diction \( cl.p \) of each classifier in \([A]\) is updated with
the current reward \( r_t \). Otherwise it updates the rules in the
previous action set \([A]_{-1}\), which is the action set from one
step ago with a previous reward \( r_{t-1} \), as follows,

\[
P = \begin{cases} 
    r_t & \text{if termination criterion is met,} \\ 
    r_{t-1} + \gamma \times \max_{a} P(s_t, a) & \text{otherwise.}
\end{cases}
\]

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

Here, Butz introduced a prediction update weighted by a
fitness gradient to improve on the performance of XCS
in reinforcement learning problems [19, 20] by using the
following equation:

\[
cl.p \leftarrow cl.p + \beta (P - cl.p) \times \frac{cl.F}{\sum_{c \in [A]} c.F}
\]

Then, the prediction error \( cl.\varepsilon \) and the action set size
\( cl.as \) of each rule \( cl \) are updated as follows,

\[
cl.\varepsilon \leftarrow cl.\varepsilon + \beta \left( |P - cl.p| - cl.\varepsilon \right)
\]

\[
cl.as \leftarrow cl.as + \beta \left( \sum_{c \in [A]} c.num - cl.as \right)
\]

Finally, rule fitness is updated in two steps: first, the ac-

An Analysis of Rule Deletion Scheme in XCS
curacy $cl.\kappa$ of the rules in $[A]$ is computed as follows,

$$cl.\kappa = \begin{cases} 1 & \text{if } cl.\epsilon < \epsilon_0 \\ \alpha \left( \frac{cl.\epsilon}{\epsilon_0} \right)^{-\nu} & \text{otherwise.} \end{cases} \ldots \ldots (7)$$

The accuracy $cl.\kappa$ means that a rule is considered to be accurate if its prediction error $cl.\epsilon$ is smaller than the threshold $cl.\epsilon_0$; for an accurate classifier, $cl.\kappa = 1$. Note that $\nu$ is a constant XCS parameter ($\nu > 0$) that controls the rate of decline in accuracy $\kappa$; the accuracy $\kappa$ decreases with $-\nu$ if its prediction error $\epsilon$ is larger than $\epsilon_0$. That is, an inaccurate rule with large $\epsilon$ will have low accuracy $\kappa$.

A classifier is considered 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 given by $\alpha(cl.\epsilon/\epsilon_0)^{-\nu}$. Then the fitness $cl.F$ of each rule $cl$ in $[A]$ is updated with a relative accuracy $cl.\kappa'$ as follows,

$$cl.\kappa' = \sum_{c \in [A]} c.\kappa \times c.num, \ldots \ldots \ldots (8)$$

$$cl.F \leftarrow cl.F + \beta (cl.\kappa' - cl.F). \ldots \ldots \ldots (9)$$

**Discovery Component.** On a regular basis depending on the parameter $\theta_{GA}$, a genetic algorithm is applied to the rules in $[A]$ or a previous action set $[A]_{-1}$. XCS selects two rules based on the fitness of rules $[A]$, copies them, and performs crossover and mutation on the copies with probability $\chi$ and $\mu$, respectively. The resulting offspring are inserted into the population and two rules are deleted if the number of rules in the population $[P]$ is larger than a population size limit $N$ to keep the population size constant. If the rule in $[P]$ has the same condition and the same action of offspring, the numerosity $num$ increases by one.

### 3. Experiment

Here, we describe an experiment we conducted on a set of maze problems [6] to understand how the performance of XCS can differ depending on deletion schemes.

#### 3.1. Three Deletion Schemes

We compared the following three deletion schemes for XCS: 1) population-based deletion, 2) match set-based deletion and 3) action set-based deletion, of which the latter two have been introduced newly for this analysis. The match set-based deletion and the action set-based deletion schemes are designed to delete rules from the match set $[M]$ and the action set $[A]$, respectively; if the match set or the action set is empty, the deleted rules are selected from the population as well as the population-based deletion (however, this case occurs almost never).

Because the match set-based deletion and the action set-based deletion schemes delete rules from subsets of the population, we can say that these two deletion schemes are niche (or local) deletion strategies, while the population-based deletion scheme is a global one. In detail, different from population-based deletion, match-set based deletion is guaranteed to delete rules which match the current state; and action set-based deletion deletes those rules that match the current state and have the executed action.

In short, the algorithmic difference among the population-based, match set-based and action set-based deletion schemes is that they consider deleting rules covering all state-action pairs, each state pair, and each state-action pair respectively. Our analysis aims to understand how this algorithmic difference affects the XCS performance and pick the most suitable deletion scheme for XCS.

#### 3.2. Maze Problems and Experimental Design

**Maze problem.** Maze problems are grid-like environments in which each position can be either empty, an obstacle “T” or food “F.” A learner perceives eight surrounding cells: an empty position is coded by “00,” an obstacle by “01,” and food by “11.” There are eight actions that lead to the eight surrounding cells. In each experiment, the learner is randomly placed in one of the empty positions and is required to reach the food position. Upon reaching the food position, the learner receives a 1000 reward and the iteration ends. The maximum number of steps is 50, which forces the system to restart when it has performed more than 50 actions [15]. This can improve the performance of XCS by guaranteeing that the search space is explored uniformly and that the system does not get stuck [19].

We use the three maze fields of Maze5 Maze6 and Woods14 (see Fig. 1), in which their optimum steps are 4.61, 5.19, and 9.5, respectively [15]. Maze6 is similar to Maze5, but it is a more difficult problem as it requires a longer step to reach the food position; the reward position is more hidden, resulting in a longer random walk [19].

**Experimental design.** Each experiment consists of multiple iterations indicating a problem that XCS must solve. One iteration consists of one learning problem and one test problem. Each learning problem is either an exploration problem or an exploitation problem [6]. During exploitation problems, the system selects actions randomly from those represented in the match set. During exploration problems, the system always selects the action with highest expected return. The reinforcement component is turned on during both exploration but it is turned off during exploitation problems. The discovery component is enabled only during exploration problems, and it is turned off during exploitation problems. Before solving the learning problem, a learner decides to solve either the exploration problem or the exploitation problem with the probability 0.5. In addition, to evaluate the performance of XCS over learning problems, XCS additionally solves a test problem. In detail, after solving the learning problem, XCS must solve the test problem to sample LCS performance; as the test problem is being solved, the learner
always selects the action with highest expected return as in the exploitation problem but both the reinforcement and the discovery components are turned to ensure that only the performance of XCS is sampled.

**Evaluation criterion.** We calculate the average number of steps to reach the food position as a well-used criterion to evaluate the performance of XCS. The average number of steps to reach the food position indicates how correctly XCS solves a problem that must converge to the optimum step of the mazes. The average number of steps (sampled from the test problem) is reported as the moving average over 500 iterations. All reported plots are averages over 30 experiments.

**XCS parameter setting.** We use the same XCS parameter settings as in [19]. For Maze5, \(N = 3000, \varepsilon_0 = 5, \gamma = 0.7, P_b = 0.3, \chi = 0.8, \mu = 0.01, \beta = 0.2, \alpha = 0.1, \delta = 0.1, \nu = 5, \theta_{GA} = 25, \theta_{del} = 20, \text{ and } \theta_{sub} = 20 \) and the GA and action set subsumptions are turned off. The maximum number of iteration is 10000. For Maze6, we use the settings used for Maze5, except for \(\varepsilon_0 = 1\) and \(\theta_{GA} = 100\); for Woods14, \(\varepsilon_0 = 0.05, \theta_{GA} = 400\) and the maximum number of iteration is 20000. Please note that the parameter settings of \(\varepsilon_0\) and \(\theta_{GA}\) are set to different values depending on the maze problem. Although the setting of those two parameters is not our focus in this paper, we give discussion of this settings in Appendix.

3.3. Results

Figure 2 reports the average number of steps to reach the food position of XCSs with the three deletion schemes on the three maze fields (i.e., Maze5, Maze6 and Woods14). In the figure, population, match set and action set indicate the population-based deletion, the match set-based deletion and the action set-based deletion respectively. From the figure, in the three mazes, all three deletion schemes solve the problem optimally; their average numbers of steps converge to the optimum steps 4.61, 5.19, and 9.5 respectively. In addition, in Maze5 and Maze6, the average number of steps of the match set-based and the action set-based deletion schemes converge slightly faster than those of the population-based deletion scheme, although we cannot say whether the differences are significant.1

The results suggest that the local deletion schemes, namely, the match set-based and action set-based deletion, may enable XCS to perform optimally with fewer learning problems than the global deletion scheme (i.e., the population-based deletion). Next, we analyze why those differences occur depending on the deletion schemes.

4. Analysis

The tendency, that is, the convergence of the average number of steps to reach the food position, is strongly re-
related to a problematic over-generalization issue in XCS. Let us first explain this issue. As the principle of XCS is to produce accurately generalized rules having high fitness \( F \), XCS can solve the problem optimally when those accurately-generalized rules are produced. In the other words, if the population consists of inaccurately generalized rules, the performance of XCS degrades. Here, such inaccurately generalized rules are often called over-generalized rules. Those over-generalized rules have low fitness and large prediction error (see Eqs. (7)–(9)). This can be explained as follows; because the over-generalized rules have a generalized condition \( C \) including a relatively large number of don’t cares “#,” they match some states to which different \( \text{maxPA}(s_t, a) \) values are assigned. Consequently, to improve the performance of XCS (or to solve a problem with fewer learning problems), XCS should eliminate the over-generalized rules.

Accordingly, we analyze how the three different deletion schemes affect the deletion of the over-generalized rules. To this end, we investigate the rules in the population in terms of the following two measurements: the fitness of rule and a specificity of a rule. Here, the specificity \( sp \) of a rule can be one measurement to quantify the generality of rules. The specificity measures how many specific symbols (i.e., “0” or “1”) a rule condition \( C \) consists of, and it is calculated as a number of specific symbols divided by the condition length. For instance, the rule condition \( C = "0011##" \) has a specificity of 0.667 given by 4/6. The low specificity indicates that its rule condition includes a relatively large number of don’t care symbols.

Figure 3 shows the relationship between the fitness and the specificity of rules in the population; each plot denoted by a circle indicates one rule. Please note that we plot the rules acquired by the previous experiment; and that the fitness is plotted on the log scale. In terms of fitness, we can identify the maximally accurate rules if their fitness is the maximum 1.0, and so we can also identify the rules with fitness less than the maximum as candidates of over-generalized rules. However, in terms of the specificity, it is difficult to discuss or decide the optimal specificity that the maximally accurate rules must have. This is because, the generalized rules match a different combination of state, and then, computing the optimal specificity for each possible combination would be very complex. However, we can still discuss the over-generalized rules if they have relatively low specificity. Here, we only show the rules in the populations at 500 iterations; the populations acquired for the initial test problems are better from the viewpoint of analyzing the over-generalized rules because they will be eliminated as the number of iterations increases. Again, rules with lower fitness and lower specificity can be considered over-generalized rules.

From the figure, the population-based deletion scheme derives more over general rules that have low fitness (\( F < 1.0 \)) and relatively low specificity (\( sp < 0.5 \)) than the match set-based and the action set-based deletion schemes. This result was obtained in only one experiment of the total 30 experiments. However, this tendency can be confirmed as a significant difference; we calculated the average of the number of rules having \( F < 1.0 \) and \( sp < 0.5 \) for the 30 populations at 500 iterations. Then, we obtained the average values of the population-based, match set-based and action set-based deletion schemes as 611.2, 305.6 and 374.9, respectively, where for all pairs of deletion schemes we found a significant differences with \( p < 0.01 \).

Accordingly, our analysis suggests that the local deletion schemes (i.e., the match set-based and the action set-based deletion schemes) promote the deletion of the over-generalized rules compared with the global deletion scheme (i.e., the population-based deletion). The global deletion scheme considers all types of rules to-

\[ \text{specificity} = \frac{\text{number of specific symbols}}{\text{condition length}} \]

\[ \text{maxPA}(s_t, a) = \text{maximum of PA}(s_t, a) \]

\[ \text{fitness} = \text{sum of accuracy} \]

\[ \text{prediction error} = \text{sum of prediction error} \]

\[ \text{fitness} = \text{fitness} \]

\[ \text{sp} = \text{specificity} \]

\[ \text{PA}(s_t, a) = \text{probability of action} \]

\[ \text{specificity} < 0.5 \]

\[ \text{fitness} < 1.0 \]

\[ \text{prediction error} < 0.5 \]

\[ \text{fitness} \leq 1.0 \]

\[ \text{specificity} < 0.5 \]
gether to select rules to be deleted; this is seemingly a good strategy because all types of over-generalized rules can be deleted averagely. However, with this strategy, very weak opportunities for deletion will be assigned to each over-generalized rule. In contrast, a rule-subset to which the local deletion scheme is applied must consist of smaller types of over-generalized rules than the population. This means stronger opportunities for deletion will be assigned to the over-generalized rules than the global deletion scheme. This is why local deletion schemes promote the deletion of the over-generalized rules.

However, we should claim that a too strong local deletion scheme, that is, applying deletion to a very specific subset, may not be an adequate deletion strategy for XCS. For instance, the action set \([A]\) is a more specific subset than the match set \([M]\) because the action set consists of rules for each state-action pair while the match set consists of rules matching each state. While the subset includes fewer over-generalized rules than the population, there may possibly be a case that a very specific subset includes no over-generalized rules. Even in this case the local deletion scheme is constrained to delete rules that may prefer to remain in the subset, and thus, over-generalized rules sometimes are not candidates for rules to be deleted. Our claim is highlighted consistently in our analysis, that is, the action set-based deletion scheme has 374.9 over-generalized rules more than the 305.6 rules acquired by the match set-based deletion scheme. Consequently, the match set-based deletion scheme, which is a combined deletion strategy of the local and global deletion schemes, can be suitable.

5. Additional Experiment

While our analysis provides insights about the three deletion schemes, we did not find their useful effects that improves the performance of XCS. As shown in Fig. 2, the previous experiment showed that the three deletion schemes derive the optimal number of steps with almost the same number of iterations. Here, we present an example in which the local deletion schemes greatly improve the performance of XCS. In detail, our final experiment tests the three deletion schemes on noisy maze problems with slip noise [15] to validate our hypothesis: the local deletion schemes enable XCS to work robustly on noisy problems because they can promote the deletion of the over-generalized rules.

5.1. Maze Problems and Experimental Design

Maze problem with slip noise. We use the same maze problem as in the previous experiment. The only difference is that we add slip noise when a learner moves state. A learner occasionally moves to a neighbor state where the executed action does not lead to. The slip noise simulates such movement. In detail, with a probability \(P_{sr}\), a moved state is selected randomly from the seven neighbor states (i.e., around the current state excepting for the state which the executed action leads to).

With the slip noise, because the learner will unexpectedly move to a random state whose \(\maxPA(s, a)\) (similar to the state-action value in typical reinforcement learning) can be different from the expected \(\maxPA(s, a)\), the rules will have low fitness with large prediction error, even if they are potentially the accurate rules for the maze problem without noise. Therefore, for this difficulty, XCS may wrongly identify the accurate rules as the over-generalized rules. We set \(P_{sr}\) to 0.3, 0.4, and 0.5.

Experimental design. We use the same experimental design as in the previous experiment. Again, one iteration consists of the learning problem and the test problem. We add slip noise only to the learning problem (i.e., exploration and exploitation problems) and we do not add it to the test problem to evaluate how robustly XCS can learn the optimum number of steps. Hence, for the three maze fields, the optimum steps are the same as those in the previous experiment (i.e., 4.61, 5.19 and 9.5).

We evaluate the performance of XCS in terms of the average number of steps to reach the food position. The average number of steps (sampled from the test problem) is reported as the moving average over 500 iterations. All reported plots were averaged over 30 experiments. The XCS parameter settings are the same as those in the previous experiment.

5.2. Results

Figures 4–6 show the average number of steps to reach the food position of XCSs with the three deletion schemes for the maze fields with slip noise. Again, in all figures, population, match set and action set indicate the population-based deletion, match set-based deletion and action set-based deletion schemes, respectively. On Maze5, with \(P_{sr} = 0.3\), the three deletion schemes eventually derive the average number of steps that converge near the optimum average of steps 4.61, and the average number of steps of the match set-based and the action set-based deletion converge with fewer iterations than that of the population-based deletion scheme. When the slip rate further increases to 0.4, those two local deletion schemes greatly improve the performance of XCS with the population-based deletion scheme (i.e., the original XCS); the average number of steps of population-based deletion improves slightly over iterations, but it almost does not solve the problem. By contrast, the match set-based and the action set-based deletion schemes converge to about 7 steps. With \(P_{sr} = 0.5\), the population-based deletion scheme fails to solve the problem, while the other two local deletion schemes perform robustly, attaining convergence in about 10 steps.

Similar to the result of Maze5, with a small slip rate (0.3), all deletion schemes derive almost the same average number of steps; with larger slip rates, the match set-based and the action set-based deletion schemes derive better average number of steps than the population-based deletion scheme. The only clear difference is that the population-based deletion scheme derives a better performance on Maze6 with \(P_{sr} = 0.4\) than that of Maze5.
with $P_{sr} = 0.4$. In detail, on Maze6 with $P_{sr} = 0.4$, the population-based deletion derives a good performance (i.e., the average number of steps) close to the optimum number of steps, while on Maze5 the performance of the population-based deletion greatly degrades. This is an unexpected result because Maze6 is a more difficult maze problem than Maze5 (because the food position of Maze6 is more hidden than Maze5). We have not found any reason to explain this difference, but this will not change our conclusion, that is, the local deletion schemes improve the performance of XCS compared to the original population-based deletion scheme.

Different from the results for Maze5 and Maze6, the match set-based deletion scheme outperforms the two deletion schemes slightly with $P_{sr} = 0.4$ and 0.5, but we cannot find any clear differences in terms of the average number of steps of the three deletion schemes on Woods14. The reason for this result is related to the structure of the maze field of Woods14, rather than any limitation of the local deletion schemes. In detail, as shown in Fig. 1, for each empty state, six of all eight possible actions ideally lead to the same $\max_a P(s, a)$. Hence, the learner can receive almost the same $\max_a P(s, a)$ with a high probability, even when the slip noise is activated. By contrast, in Maze5 and Maze6, each empty state has a smaller number of obstacle neighbors than in Woods14. Hence, slip noise makes those mazes more complex as the $\max_a P(s, a)$ can change unexpectedly with a high probability.

![Fig. 4](image1.png) The average number of steps to reach the food position of XCSs with the three deletion schemes on Maze5 with slip noise; $P_{sr} = 0.3$ (left), 0.4 (center), 0.5 (right).

![Fig. 5](image2.png) The average number of steps to reach the food position of XCSs with the three deletion schemes on Maze6 with slip noise; $P_{sr} = 0.3$ (left), 0.4 (center), 0.5 (right).

![Fig. 6](image3.png) The average number of steps to reach the food position of XCSs with the three deletion schemes on Woods14 with slip noise; $P_{sr} = 0.3$ (left), 0.4 (center), 0.5 (right).
6. Conclusion

This paper attempted to clarify the optimality of the original population-based deletion scheme in XCS. The population-based deletion scheme performs as a global deletion scheme applied to a population including all rules. By contrast, we introduce two local deletion schemes (i.e., match set-based and action-set based deletion schemes) in order to empirically reveal how those three deletion schemes affect the performance of XCS. On maze problems without noise, the three deletion schemes achieve almost the same XCS performance; however, we find very small differences in terms of their convergences of average number of steps. Based on those differences, we analyzed the rules acquired by the three deletion schemes. Then, we revealed that, the local deletion schemes promote the deletion of over-generalized rules, but the stronger local deletion scheme is sometimes forced to delete good rules. From these findings, we suppose that the local deletion scheme improves the performance of XCS on noisy maze problems. The experimental results clearly support our supposition; the local deletion schemes greatly improve the performance of XCS compared to the global deletion scheme. In conclusion, the local deletion scheme can be adequate for XCS rather than the original global deletion scheme, although we need to design the local deletion carefully in terms of localization strength.

This paper uses XCS as a basic algorithm, but local deletion should be applicable to other LCS models such as supervised LCS (UCS) [21]. It might be interesting to see how the three deletion schemes would affect the performance of XCS on single-step problems such as data-mining tasks, but we suppose that multi-step problems are more general and they cover covering single-step problems.

Appendix A. Parameter Settings of $\varepsilon_0$ and $\theta_{GA}$

The setting of parameters $\varepsilon_0$ and $\theta_{GA}$ is an important issue in the XCS classifier system as those parameters greatly affect the performance of XCS. For those two parameters, LCS often employs the standard parameter settings (suggested in [18]) which are empirically determined, or we empirically tune those settings depending on the problem. On the standard parameter settings [18], $\varepsilon_0$ is set to one percent of the maximum reward $r$ the system will receive; $\theta_{GA}$ is usually set to 25-50. However, the optimality of both values is not supported with any theoretical explanations. In fact, on the maze problems which this paper employs, Butz [19] reported those two standard values are not adequate and so he introduced tuned parameter settings of $\varepsilon_0$ and $\theta_{GA}$ in solving the maze problems. Then, we used the same settings of his tuned values.

In his tuning, $\varepsilon_0$ is set considering the maximum optimum step to the food position on the maze. For instance, on the maze field Woods14, the farthest state to the food position requires 18 steps to reach it optimally. Thus, to reach the optimal steps, XCS requires to distinguish the delayed payoff $r \times \gamma^{18}$ and $r \times \gamma^{19}$; the difference of $r \times \gamma^{18}$ and $r \times \gamma^{19}$ is 0.4889, where the reward $r = 1000$ and the discount rate $\gamma = 0.7$. Thus, when a rules is used in the farthest state, its rule must predict the delay reward $r \times \gamma^{18}$ with the prediction error $\varepsilon$ smaller than 0.4889; otherwise its rule wrongly distinguishes the delayed payoff $r \times \gamma^{18}$ and $r \times \gamma^{19}$. However, in his settings, the $\varepsilon_0$ is set to 0.05 on Woods14 much smaller than 0.4889. We suppose that this setting considers a safe case that XCS certainly distinguishes the delayed payoff $r \times \gamma^{18}$ and $r \times \gamma^{19}$. It still unclear how $\varepsilon_0$ can be set to a adequate value on the maze problem. That is, the adequate $\varepsilon_0$ value may not be determined with only the maximum optimum step; even when two maze problems have the same maximum optimum steps each other, $\varepsilon_0$ may be different depending on their problem complexity of mazes. For instance, although Maze5 and Maze6 both have the same maximum optimum step 8, $\varepsilon_0$ is set to 5 and 1 for Maze5 and Maze6 respectively; because the food position in Maze6 is more hidden than Maze5. In short, as far as we know, $\varepsilon_0$ should be set to a small value when the problem complexity of maze (e.g., the maximum optimum step and the maze structure) increases; however, there is no effective methodology in setting the adequate value of $\varepsilon_0$.

For $\theta_{GA}$, which determines a frequency of activation of GA (i.e., it controls how often GA is called to generate new rules), Butz also empirically determined the value of $\theta_{GA}$ on the maze problems. A small value of $\theta_{GA}$ results in a frequent activation of GA while a large value results in a less frequent activation of GA. In [19], Butz explained that, a less frequent activation enables the reinforcement component to develop more accurate parameter estimates which in turn enables a better genetic selection. In the other words, when XCS frequently generates GA with a small $\theta_{GA}$, XCS cannot hold well-updated rules as XCS is forced to delete some rules in generating new rules. Hence, since on a difficult problem XCS is required to learn rules accurately estimate the prediction with a small prediction error, $\theta_{GA}$ should be a large value to update rules enough; $\theta_{GA}$ is set to 25, 100 and 400 for Maze5, Maze6 and Woods14, depending on the difficulty of maze problems. However, similar to the case of $\varepsilon_0$, an adequate value of $\theta_{GA}$ is not discussed enough.

In summary, the existing works provided an insight into the setting of parameters $\varepsilon_0$ and $\theta_{GA}$. However, there still lacks of a universally setup guide for these two parameters, because XCS is a non-deterministic algorithm with GA in which a theoretical analysis of the system behavior would be difficult.

References:


