TR-NIS-0001

Analyzing Parameter Sensitivity and Classifier Representations for Real-valued XCS

Atsushi WADA , Keiki TAKADAMA, Katsunori SHIMOHARA, Osamu KATAI

2004.4.7

国際電気通信基礎技術研究所　ネットワーク情報学研究所
〒619-0288 「けいはんな学研都市」光台二丁目2番地2
Tel: 0774-95-2641　Fax: 0774-95-2647

Advanced Telecommunications Research Institute International (ATR)
Network Informatics Laboratories
2-2-2, Hikaridai, "Keihanna Science City", Kyoto 619-0288, Japan
Tel: +81-774-95-1111　Fax: +81-774-95-2647

Analyzing Parameter Sensitivity and Classifier Representations for Real-valued XCS

Atsushi Wada[1,2], Keiki Takadama[1,3],
Katsunori Shimohara[1,2], and Osamu Katai[2]

[1] ATR Human Information Science Laboratories,
Hikaridai, "Keihanna Science City" Kyoto 619-0288 Japan,
{wada,katsu}@atr.co.jp
[2] Kyoto University, Graduate School of Informatics,
Yoshida-Honmachi, Sakyo-ku, Kyoto 606-8501 Japan,
katai@i.kyoto-u.ac.jp
[3] Tokyo Institute of Technology,
Interdisciplinary Graduate School of Science and Engineering,
4259 Nagatsuta-cho, Midori-ku, Yokohama, Kanagawa 226-8502 Japan,
keiki@dis.titech.ac.jp

Abstract. To evaluate a real-valued XCS classifier system, we present a validation of Wilson's XCSR from two points of view. These are: (1) sensitivity of real-valued XCS specific parameters on performance and (2) the design of classifier representation with classifier operators such as mutation and covering. We also propose model with another classifier representation (LU-Model) to compare it with a model with the original XCSR classifier representation (CS-Model.) We did comprehensive experiments by applying a 6-dimensional real-valued multiplexor problem to both models. This revealed the following: (1) there are critical threshold on covering operation parameter (r_0), which must be considered in setting parameters to avoid serious decreases in performance; and (2) the LU-Model has an advantage in smaller classifier population size within the same performance level over the CS-Model, which reveals the superiority of alternative classifier representation for real-valued XCS.

1 Introduction

XCS [5] is a learning classifier system which has the potential to evolve accurate, maximally general classifiers to cover the state space for each action [3, 6]. XCS takes *bit string* inputs, the same as traditional learning classifier systems [2] (LCS). To facilitate XCS and broaden the range of applicable problem representation while keeping its generalization abilities, XCSR [7] was proposed by Wilson to deal with *real-valued* problems, and he found that XCSR could learn appropriately on the real-valued 6-multiplexor problem.

Although Wilson analyzed the potential of XCSR, its validity was insufficient in two respects. Firstly, the parameter settings used for the experiment seemed to be set ad hoc, especially for the two newly introduced parameters m_0 and

r_0 that were used in the real-valued classifier operations of *mutation* and *covering*. Secondly, the reason he adopted proposed classifier representation is not discussed, despite the possibility of other classifier representations.

Therefore, what we focus in this paper are (1) an analysis of the settings of real-valued XCS specific parameters to evaluate the model; and (2) an analysis of classifier representation with classifier operators such as covering and mutation. To achieve the latter, we propose an opponent model that presents another real-valued classifier representation that was inspired by Wilson's other model XCSI to deal with integer-valued input [8]. Although the requirement of extending XCS to integer-valued input is basically similar to that of extending XCS to real-valued input, XCSI adopts a different design concept over classifier representation. This concept can easily be applied to design another real-valued classifier representation, which we propose and adopt in the opponent model. For convenience, we have called this opponent the LU-Model and the original model the CS-Model, names which originate from the attributes used in each classifier condition that will be described later.

The rest of the paper is organized as follows. Section 2 describes both the CS and LU-Models by revealing the part extended from the XCS to achieve real-valued input. Section 3 describes the real-valued 6-multiplexor problem. Section 4 presents some simulation experiments that were done by applying both the CS and LU-Models to the real-valued 6-multiplexor problem. Section 5 has discussions based on the experimental results to validate real-valued XCS. Section 6 is the conclusion.

2 Extensions to XCS for Real-valued Input

Both CS and LU-Models are based on XCS but differ in their classifier representation. This section presents the CS-Model, which adopts XCSR classifier representation and the LU-Model, where classifier representation is inspired by XCSI. It is done by describing their classifier representations in detail, which are the extended parts from XCS[4].

2.1 XCSR-based Classifier Representation (CS-Model)

This section explains the CS-Model regarding its difference from XCS, which is equivalent to describing XCSR classifier representation with classifier operators such as covering, mutation, and crossover. To catch up with recent developments in XCS called the classifier *subsumption* mechanism, the "is-more-general" operator has been additionally defined which checks whether the classifier can subsume the other target classifier.

[4] The implementation of the XCS part of the CS and LU-Models is based on Butz and Wilson [1].

Representation of classifier conditions: The representation of the classifier in the CS-Model differs from the original XCSR in the condition part, which replaces the bit string with a set of attributes named *interval predicates* by Wilson. The interval predicate is composed of two real values (c_i, s_i) where suffix i denotes the position in the condition part. Each interval predicate represents an interval $[c_i - s_i, c_i + s_i]$ on the real number line, and if the corresponding element of the input (which is a real-valued vector) is included in the interval, matching succeeds. If, and only if, all elements match the corresponding interval predicates in the classifier condition, can matching be considered a success. The domain of attributes c_i and s_i are both set between 0 and 1, which inherit the setting of XCSR in the CS-Model, but is not a necessary requirement for this representation.

Covering operator: The covering operator creates a new classifier that matches a specified input. When a real-valued vector is denoted as $(x_1, ..., x_i, ..., x_n)$, where n is the dimension of input, each interval predicate of covered classifier condition $(c_1, s_1)...(c_i, s_i)...(c_n, s_n)$ is set as follows.

$$\begin{cases} c_i = x_i \\ s_i = rand(r_0). \end{cases} \tag{1}$$

Here, r_0 is a parameter used to decide the distribution range of the spread of the covering interval, where $rand(x)$ is a function that returns a random value distributed in the interval $0 \leq rand(x) \leq x$. The value of r_0 is set below 1 to maintain the s_i within its domain of $[0, 1]$ inherited from XCSR, but is not a necessary requirement for this operation.

Mutation operator: The mutation operator mutates the classifier condition by adding delta values Δc_i and Δs_i to interval predicate variables c_i and s_i at the constant possibility of mutation parameter μ at each interval predicate. Each delta value for attributes c_i and s_i are calculated as follows.

$$\begin{cases} \Delta c_i = \pm rand(m_0) \\ \Delta s_i = \pm rand(m_0). \end{cases} \tag{2}$$

Here, m_0 is a parameter used to decide the distribution range of both Δc_i and Δs_i, where $\pm rand(x)$ is a function that returns a random value distributed in interval $0 \leq rand(x) \leq x$ with the sign chosen uniform randomly. If the mutated value exceeds the domain of $[0, 1]$, the value is adjusted to 0 or 1. The setting for this domain is inherited from XCSR, but is not a necessary requirement for this operation.

Crossover operator: The crossover operator works the same as the crossover in XCS, except that the crossover point is not set between the condition bits but between the interval predicates.

Is-more-general operator: The is-more-general operator judges whether a classifier condition is more general than another classifier condition. The basic idea of generality is the inclusion of the set of classifier condition's possible matching inputs. If the possible matching inputs of classifier condition X completely include and are larger than the possible matching inputs of classifier condition Y, X is more general than Y. This idea can be realized for real-valued classifier representation by comparing the inclusion of the interval on the real number line for each corresponding interval predicate. For two classifier conditions $X : (c_1, s_1)...(c_i, s_i)...(c_n, s_n)$ and $Y : (c'_1, s'_1)...(c'_i, s'_i)...(c'_n, s'_n)$, if $(c_i - s_i) \leq (c'_i - s'_i)$ and $(c'_i + s'_i) \leq (c_i + s_i)$ for all i except where all attributes are equal, X is more general than Y.

2.2 XCSI-inspired Classifier Representation (LU-Model)

This subsection proposes the LU-Model with another real-valued classifier representation inspired by XCSI, which is an XCS extended model to deal with integer-valued inputs. XCSI adopts a different design concept over classifier representation, as it specifies the interval by using the value for the lower and upper bounds. This concept can easily be applied to designing real-valued classifier representation that differs from the CS-Model. The details are described below.

Representation of classifier condition: The representation of classifier condition in the LU-Model seems to be like that in the CS-Model as its interval predicate is composed of two real values (l_i, u_i), where suffix i denotes the position in the condition part. However, the denoting interval on the real number line differs from the CS-Model. The ith interval predicate simply denotes an interval $[l_i, u_i]$. If the corresponding element of input is included in the interval, matching between the element and the interval predicate succeeds. If, and only if, all elements match the corresponding interval predicates in the classifier condition, can matching be considered a success. The domain of attributes is restricted to $0 \leq l_i \leq u_i \leq 1$. This setting for domain inherits the concept of XCSI, but is not a necessary requirement for this representation.

Covering operator: The covering operator creates a new classifier that matches a specified input. When a real-valued vector is denoted as $(x_1, ..., x_i, ..., x_n)$, where n is the dimension of the input, each interval predicate of the covered classifier condition $(l_1, u_1)...(l_i, u_i)...(l_n, u_n)$ is set as follows.

$$\begin{cases} l_i = x_i - rand(r_0) \\ u_i = x_i + rand(r_0). \end{cases} \tag{3}$$

Here, r_0 is a parameter used to decide the distribution range of the distance from input value x_i to l_i and u_i, where $rand(x)$ is a function that returns a random value distributed in the interval $0 \leq rand(x) \leq x$. If the covering value

exceeds the domain of $0 \leq l_i \leq u_i \leq 1$, l_i and u_i are set to be kept within their domains as the follows: if l_i is smaller than 0, l_i is set to 0; and if u_i exceeds 1, u_i is set to 1.

Mutation operator: The mutation operator mutates the classifier condition by adding delta values Δl_i and Δu_i to l_i and u_i at the constant possibility of mutation parameter μ at each interval predicate. Each delta value for attributes l_i and u_i is calculated as follows.

$$\begin{cases} \Delta l_i = \pm rand(m_0) \\ \Delta u_i = \pm rand(m_0). \end{cases} \tag{4}$$

Here, m_0 is a parameter used to decide the distribution range of both Δl_i and Δu_i, where $\pm rand(x)$ is a function that returns a random value distributed in the interval $0 \leq rand(x) \leq x$ with the sign chosen uniform randomly. If the mutated value exceeds the domain of $0 \leq l_i \leq u_i \leq 1$, l_i is set to 0 or u_i, and u_i is set to l_i or 1 depending on the following: (1) if the mutated l_i is smaller than 0, l_i is set to 0; (2) if the mutated l_i exceeds u_i, l_i is set to u_i; (3) if the mutated u_i exceeds 1, u_i is set to 1; and (4) if the mutated u_i is smaller than l_i, u_i is set to l_i.

Crossover operator: The crossover operator works the same as the crossover in the CS-Model, except for the difference between the format of interval predicates, which is of no concern in this operation.

Is-more-general operator: The is-more-general operator judges whether a classifier condition is more general than another classifier condition. This is achieved for LU-Model classifier representation as follows. For two classifier conditions $X : (l_1, u_1)...(l_i, u_i)...(l_n, u_n)$ and $Y : (l'_1, u'_1)...(l'_i, u'_i)...(l'_n, u'_n)$, if $l_i \leq l'_i$ and $u'_i \leq u_i$ for all i except for where all attributes are equal, X is more general than Y.

2.3 Real-valued XCS Specific Parameters

While extending XCS to deal with real-valued inputs, new parameters m_0 and r_0 are introduced to XCS for both CS and LU-Models. Although the processes for how m_0 and r_0 are used in each model are different, roughly m_0 controls the upper limit for the random distribution of delta values used in the mutation operator, while r_0 is concerned with the distribution of the spread of covering condition intervals. Originally, the corresponding parameters in XCSR were labeled m and s_0. However, to avoid confusion caused by differences in discussing both models, we unified the names of these corresponding parameters to m_0 and r_0. The correspondence in the names of these parameters are in Table 2.3, where XCSR's parameters have been renamed m_0 and r_0 in the CS-Model to match the others.

Table 1. Correspondence of names of real-valued XCS specific parameters

XCSR	XCSI	CS-Model	LU-Model
m	m_0	m_0	m_0
s_0	r_0	r_0	r_0

3 Real-Valued 6-Multiplexor Problem

The real-valued 6-multiplexor problem is a sample problem presented by Wilson to validate XCSR, which is a real-valued version of the Boolean 6-multiplexor problem. We also employed this problem for two reasons. The first was that the simplicity of the problem made analysis of the experimental results easier while low computational costs allowed comprehensive experiments to analyze parameter dependence in the model that required a huge number of simulations. The second reason was that it would enable us to refer to Wilson's preceding experiment on XCSR, and further discuss the validity of XCSR, and the CS and LU-Models under the same conditions.

The Boolean 6-multiplexor function took a six-bit string as input and output a truth value of 1 or 0. The function was designed as output that would have a value of $(n+2)$th bits where n was calculated by interpreting the two leftmost bits as a binary formatted number. For example, the first two bits of the input string "011010" were "01", which denotes the decimal 1 when interpreted as a binary formatted number, so the output value is the third bit of the string, in this case, 1. Alternatively, in disjunctive normal form, the Boolean 6-multiplexor function F_6 is given as follows where b_i stands for the i-th bit of the strings, the over-line negates the bit, and "+" takes a logical sum.

$$F_6 = \overline{b_0}\overline{b_1}b_2 + \overline{b_0}b_1b_3 + b_0\overline{b_1}b_4 + b_0b_1b_5. \tag{5}$$

To modify the Boolean 6-multiplexor problem to the real-valued 6-multiplexor problem, a parameter vector $\theta = (\theta_0, ..., \theta_5)$ is introduced. For each element in the real-valued input vector $x = (x_0, ..., x_5)$, x_i is converted to 0 if $x_i < \theta_i$; otherwise it is 1. In each learning step of simulation, a randomly chosen value from the domain [0,1] is set to each element of vector x and given for input. If the returned output for x is correct, which has the same value as $F_6(x)$, reward r_{imm} is given, where this is a parameter denoting "immediate reward."

4 Experiment

Simulation experiments were done to validate real-valued XCS by applying the CS and LU-Models to the real-valued 6-multiplexor problem. Common conditions for all experiments can be described as follows[5]: N(max population size)=

[5] Refer to Wilson [7] for meaning of these parameters.

800, β(learning rate)= 0.2, ϵ_0(error threshold to calculate classifier fitness)= 0.2, ν(power parameter to calculate classifier fitness)= 5, θ_{GA}(threshold to invoke GA)= 12, χ(possibility invoking crossover)= 0.8, μ(possibility invoking mutation) = 0.04. The threshold parameter vector $\theta_i (i = 1, ..., 6)$ for the real-valued 6-multiplexor problem was set as $(0.5, 0.5, 0.5, 0.5, 0.5, 0.5)$. In all simulations, the initial classifier population was set to empty. These settings to evaluate XCSR were the same as those in Wilson experiments.

The simulations for all experiments were evaluated by the average reward and the size of the classifier population. The classifier system was expected to acquire a population as small as possible to attain high average reward. For both values, the moving average of 50 previous iterations were calculated to check the temporal change. Here, iteration denotes the number of explored problems in the real-valued 6-multiplexor. Ten simulations were done on each case to obtain average variations.

4.1 Preliminary Experiment 1: Parameter dependency on CS-Model

To examine the dependence of real-valued classifier representation specific parameters m_0 and r_0, we did a preliminary experiment by applying the CS-Model to the real-valued 6-multiplexor problem using four sample combinations of (m_0, r_0) pairs: (a) $(0.1, 1.0)$, (b) $(0.1, 0.5)$, (c) $(0.1, 0.25)$ and (d) (0.5, 0.5). Here, the setting for (a) represents the same conditions as Wilson's experiment. The settings for (b) and (c) were selected to check the effect of change on r_0 parameter compared with (a). The setting for (d) was selected to check the effect of the m_0 parameter.

The results we obtained from the experiments are Fig. 1, which shows the relation between average reward on the vertical axis and iterations on the horizontal axis. Figure 2 shows the relation between population size on the vertical axis and iterations on the horizontal axis. Focusing on the r_0 parameter by comparing (a), (b) and (c), there is a significant difference between (c) and the others. For (c), there is no improvement in average reward as Fig. 1 shows, and the population size remains at a maximum limit of 800 throughout the simulation period in Fig. 2. This implies the existence of a threshold on parameter r_0, which is roughly between the 0.5 used in (b) and the 0.25 used in (c) causing a serious decrease in performance . Focusing on the m_0 parameter by comparing (a) and (d), there are no noticeable differences in average reward as shown in Fig. 1. In terms of population size, (d) converges smaller than (a) as shown in Fig. 2, where the difference is far smaller than that of the previous effect of r_0 between (a) and (c).

4.2 Preliminary Experiment 2: CS-Model vs. LU-Model

To evaluate the differences between the CS and LU-Models, we did another preliminary experiment by applying the LU-Model to the real-valued 6-multiplexor

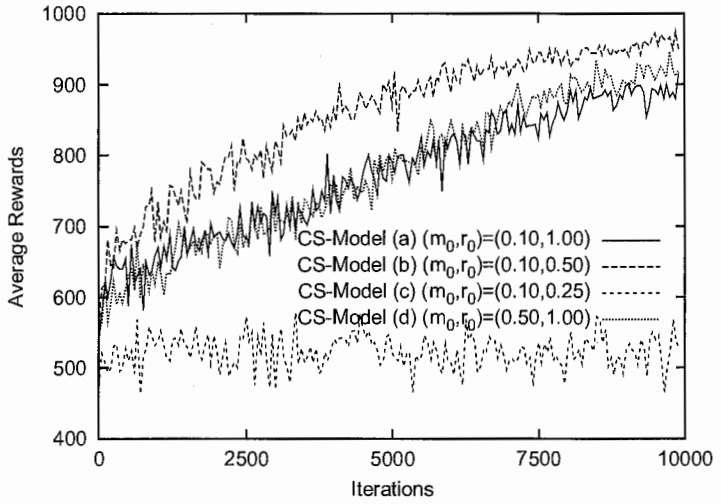

Fig. 1. Experimental results for (a) to (d) on CS-Model: relation between average reward and iterations.

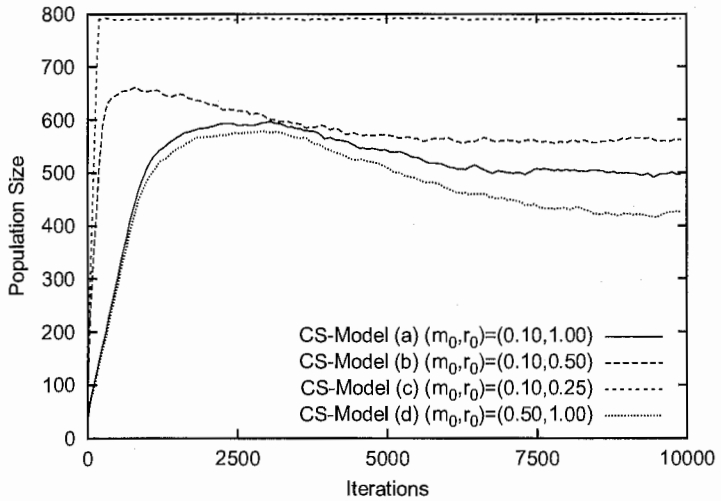

Fig. 2. Experimental results for (a) to (d) on CS-Model: relation between average performance and iterations.

problem. We used 2 settings of (a) $(0.1, 1.0)$ and (c) $(0.1, 0.25)$ for parameters (m_0, r_0). (a) had the same setting as Wilson's original experiment, and (c) yielded a distinctive result in the previous experiment on the CS-Model.

The experimental results for the LU-Model compared with the results of the previous experiment on the CS-Model are in Fig. 3, which reveals the relation between average reward on the vertical axis and iterations on the horizontal axis. Figure 4 has the relation between population size on the vertical axis and iterations on the horizontal axis. By comparing the CS and LU-Models, for (a), there is a difference where the LU-Model records a higher average reward than the CS-Model throughout the period of simulation as Fig. 3 shows, while the population size converges to less than that in the CS-Model in Fig. 4. In the (c) of the LU-Model, a similar phenomenon with that of the CS-Model can be observed where there are no improvements in average reward in Fig. 3. and the population size remains at a maximum limit of 800 throughout the simulation period in Fig. 4. From a comparison of both cases, there seems to be a similar performance dependence on the r_0 setting between the CS and LU-Models, which indicates that learning proceeds during a large r_0 value but fails with a small r_0 value.

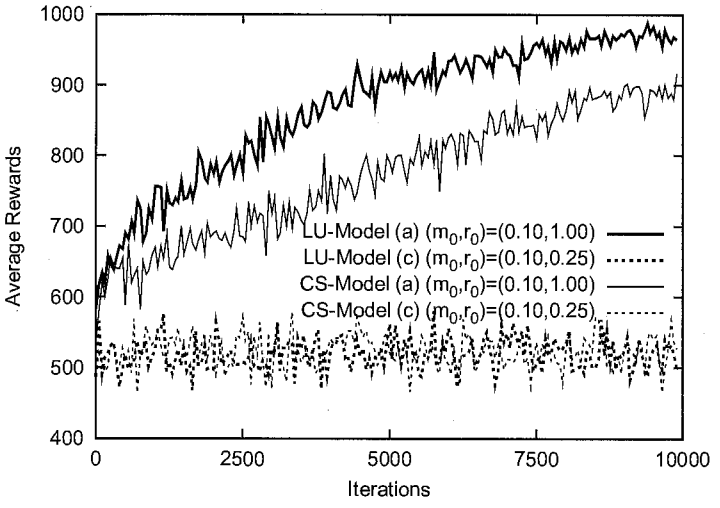

Fig. 3. Comparison of the experiments between CS and LU-Models for (a) and (c): relation between average performance and iterations.

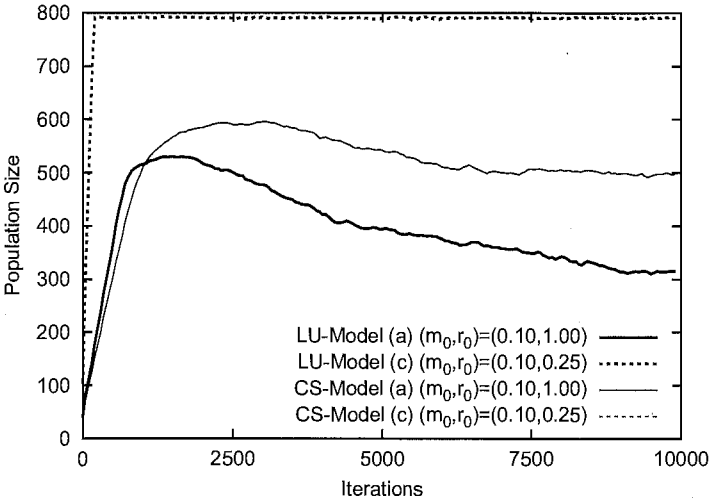

Fig. 4. Comparison of the experiments between CS and LU-Models for (a) and (c): relation between average performance and iterations.

4.3 Comprehensive Experiment

The results of the two preliminary experiments described in Section 4.1 and Section 4.2 indicate that there seems to be a significant dependence of both average reward and population size on the combination of parameters m_0 and r_0. To reveal the big picture on the landscape for the m_0-r_0 plane, we did a comprehensive experiment on both the CS and LU-Models. In the experiment, 400 combinations of parameters m_0 and r_0 were examined, which covered the m_0-r_0 plane with a series of grid points defined in the following matrix. The grid size was set to 0.05, so that the grid points included four sample combinations (a) $(0.1, 1.0)$, (b) $(0.1, 0.5)$, (c) $(0.1, 0.25)$ and (d) (0.5, 0.5), which were used in the preliminary experiments.

$$(m_0, r_0) = \begin{bmatrix} (0.05, 0.05) & (0.05, 0.10) & \dots & (0.05, 1.00) \\ (0.10, 0.05) & (0.10, 0.10) & \dots & (0.10, 1.00) \\ \vdots & \vdots & \ddots & \vdots \\ (1.00, 0.05) & (0.10, 0.10) & \dots & (1.00, 1.00) \end{bmatrix} \tag{6}$$

The results are as follows. Figures 5, 6, 7 and 8 show the average reward for the CS-Model, its population size, and the average reward for the LU-Model, and its population size. In all figures, the x-axis denotes m_0, and the y-axis denotes r_0. Converged values at iterations of 10000 were used to describe the surface for the z-axis. Each grid points' height from the m_0-r_0 plane denotes the

corresponding values. The viewpoint of Figs. 6 and 8 differs from that of Figs. 5 and 7 presenting the whole surface without hidden regions. The labels (a) to (d) each denote the four sample conditions used in the preliminary experiments.

The simulations on the CS-Model revealed the dependence of performance on the m_0-r_0 plane, which can clearly be seen in Figs. 7 and 8. There is an r_0 threshold where the r_0 value is between 0.2 to 0.5, which is plotted as a sharp drop on each surface. In the middle of this drop and, in the area where r_0 is larger, the average reward is quite high being over 900 and the classifier population size converge towards 300. In the other area where r_0 is smaller, the average reward remains low at 500 and the population size nearly remains at the maximum limit of 800. From the simulations on the LU-Model described in Figs. 5 and 6, we can see that the dependence of performance on the m_0-r_0 plane is quite similar to that of the CS-Model, which can be explained by the similarity in the shapes of their landscapes. However, comparing their corresponding values, population size of the LU-Model at less than 200 is below the CS-Model's of above 300 when r_0 is large.

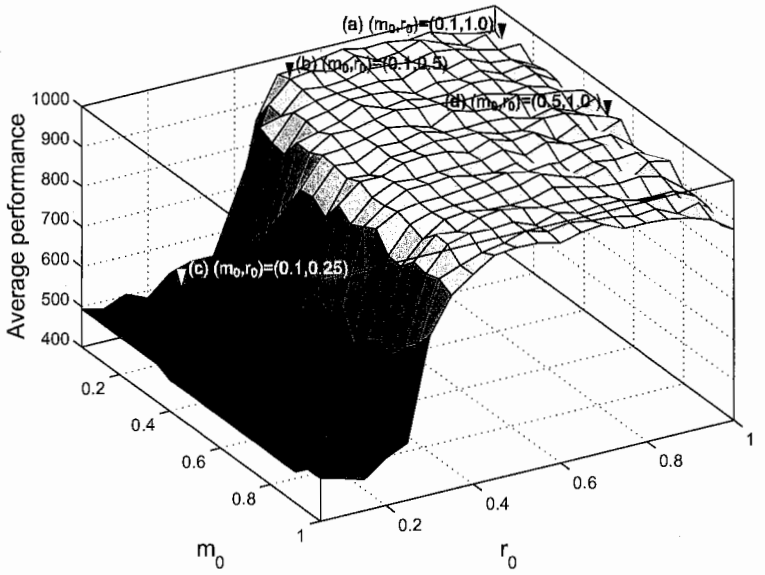

Fig. 5. Relation between average performance and time steps in CS-Model.

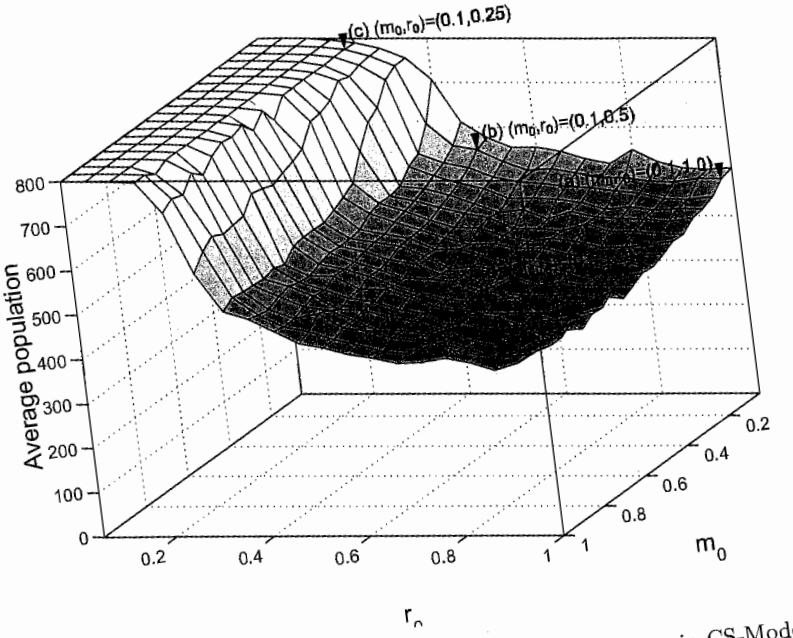

Fig. 6. Relation between average performance and time steps in CS-Model.

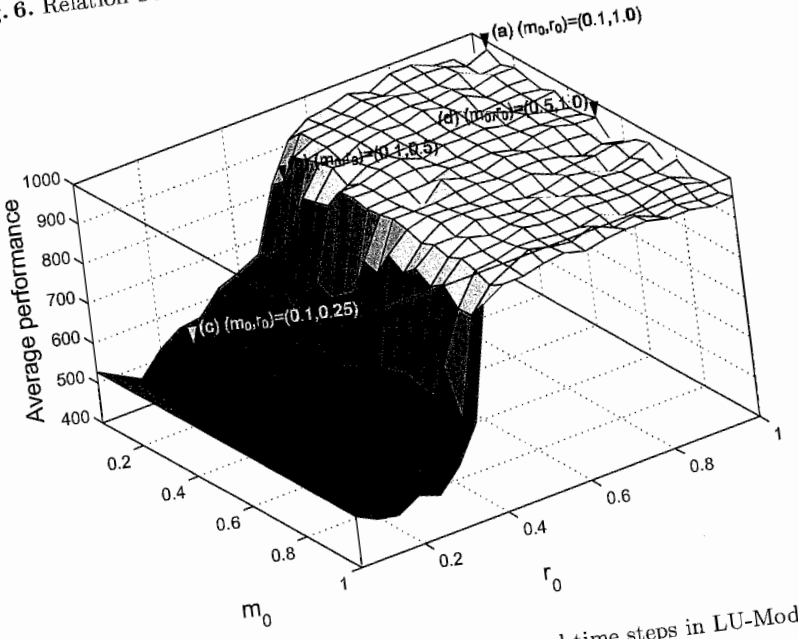

Fig. 7. Relation between average performance and time steps in LU-Model.

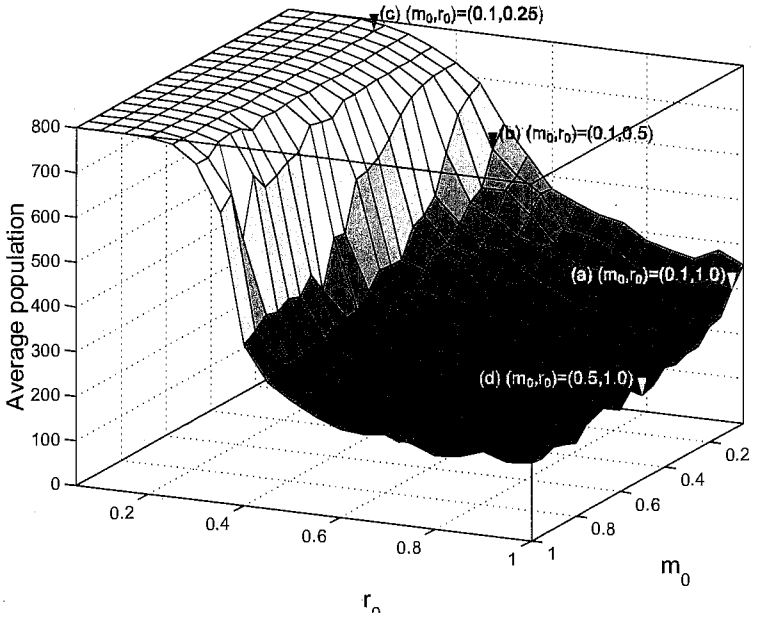

Fig. 8. Relation between average performance and time steps in LU-Model

5 Discussion

5.1 Validating Parameter Sensitivity of XCSR (CS-Model): Analysis of Cover Spread Parameter(r_0) Sensitivity

To validate our evaluation of XCSR, we will discuss the sensitivity of model specific parameters for the CS-Model, especially the serious performance decrease caused by the small r_0 value. We first found this in the first preliminary experiment and verified it through the comprehensive experiment described in Section 4.3. It was not a special case on a specific (m_0, r_0) setting, but a general phenomenon that occurred when r_0 was smaller than a specific threshold that was roughly between 0.2 to 0.5.

As r_0 is a parameter used to decide the distribution range of intervals in covered classifier condition, an assumption concerning the covering process can be made, where the scenario can be described as follows. Consider the entire process of simulation where r_0 was set to be small. In the early stages of simulation, the covering would frequently occur as classifier population was initially set as empty. The population size would soon reach the maximum limit, because the classifiers created by covering could only cover small areas of the input state space as the size of their condition intervals were limited within r_0. Soon, input that was not covered by the present classifier population would arrive, then a new classifiers would be created to cover it while one of the existing classifiers

would be deleted. This cycle would be repeated until the state space was covered through simulation. During this period, classifiers would be replaced one after another before they had become experienced, and learning could not be attained.

This assumption explains results such as (c), where no improvements in the average reward can be seen (Fig. 1) and the population size remains at a maximum limit of 800 throughout the simulation period (Fig. 2.) We can check this assumption by calculating the rate the area is covered by the N classifiers over the state space, where N is the maximum limit for population size. In the real-valued 6-multiplexor problem, the area of the state space is 1.0^6. Here, if we assume that all the intervals in N classifier conditions takes a maximum value of r_0, the total area of the covered space would be $N \times {r_0}^6$ and the coverage rate would be $(N \times {r_0}^6)/1.0^6$ where r_0 must at least be larger to make $(N \times {r_0}^6)/1.0^6$ larger than 1.0 to cover the entire state space. In practice, there are overlapping areas between each classifier condition, which require an extra r_0 value to cover the state space. This can be calculated by simple simulation that examines the rate of coverage of the state space by N covered classifiers. The process of simulation is described as follows.

1. Generate a 6-dimensional input vector by setting a random number within a range of [0,1] for each element and check if the randomly generated input vector is covered by any existing classifiers. Repeat until the uncovered input vector is found.
2. Create a classifier by covering operation for the input vector generated in 1 and insert it into the classifier population.
3. Repeat 1 and 2 until the size of the classifier population reaches N.
4. Calculate the rate of coverage of the state space by generating 1000 sample random inputs. The coverage rate is estimated by dividing the number of covered inputs within the number of total sample inputs.

The results are in Fig. 9, indicating the relation between r_0 and the simulated value for state space coverage rate compared with the value of $(N \times {r_0}^6)/1.0^6$. The horizontal axis denotes r_0 and the vertical axis denotes the coverage rate of state space. When r_0 gets smaller than the threshold roughly between 0.2 and 0.5, the simulated coverage rate quickly decreases to converge to 0. As this curve resembles the curve of the relation between r_0 and average reward, the assumption seems to be valid.

Therefore, the evaluation done by Wilson where XCSR could learn appropriately on the real-valued 6-multiplexor problem should be limited within conditions where the parameter r_0 is set sufficiently large for the covered classifiers conditions to cover the entire input state space. These conditions should be maintained to avoid serious decreases in performance.

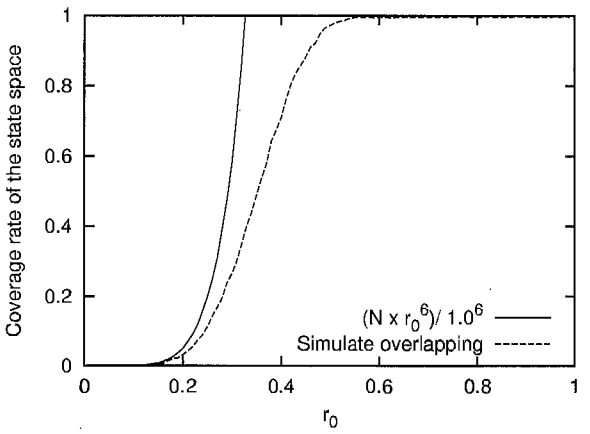

Fig. 9. Relation between coverage rate of the state space and r_0 (N=800)

5.2 Validating Classifier Representation for real-valued XCS: Superiority of LU-Model to CS-Model on Classifier Population Size

Although the CS and LU-Models have a similar tendency towards parameter dependence, by focusing on an absolute value of performance, the LU-model performs well as was found during the second preliminary experiment in Section 4.2, and verified by the comprehensive experiment in Section 4.3 and this is what we will discuss here. The superiority of the LU-Model over the CS-Model is that it requires a smaller classifier population size, while achieving the same average reward where r_0 is large enough to learn, where the threshold is that discussed in the first discussion. This can be explained by analyzing how the difference in the classifier condition expression affects the difficulty of classifier subsumption. The classifier subsumption is a process that suppresses the classifier population size by letting a more general classifier to subsume the other classifier, where the definition of generality is described in Section 2.1 for the CS-Model and in Section 2.2 for the LU-Model.

Intervals expressed by interval predicates (c_i, s_i) are allowed to take ranges which exceed [0,1] in the CS-Model, because the bounds of both c_i and s_i are between 0 and 1. For example, the interval predicate $(c_i, s_i) = (0.1, 0.3)$ denotes [-0.2,0.4] as indicated by (b) in Fig. 10, where c_i and s_i values are both within [0,1] but the denoting interval exceeds [0,1]. Here, the Fig. 10 has three intervals (a) to (c) on the real number line, where the horizontal axis denotes the real number line and the vertical axis is used to distinguish the three intervals. This excess of matching, covering and mutation operations causes no problems, as only the sub part within [0,1] is used for the classifier operations, which is [0,0.4] in this case, described as interval (c) in Fig. 10. However, there is a problem

with subsumption. For example, although interval (a) in Fig. 10 denoting [0,0.6] is more general than interval (b) in the effective range, which is equal to (c), interval (a) cannot subsume interval (b). This could occur, in general, to an interval that exceeds the range of [0,1].

However, the LU-Model does not suffer from this problem, as the interval expressed by its interval predicate (l_i, u_i) is limited within the effective range of $0 \leq l_i \leq u_i \leq 1$. For this reason, we found that the LU-Model could successfully subsume not general classifiers which could be alternated by more general classifiers, and this resulted in a smaller classifier population size than in the CS-Model. This presents the possibility of an alternative classifier representation for real-valued XCS, which was adopted and validated in the LU-Model.

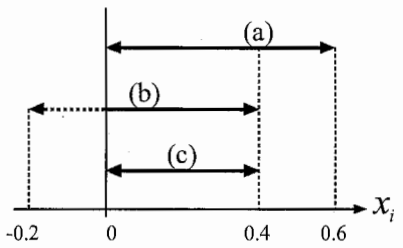

Fig. 10. Interval Graph to Describe Subsumption Difficulty in CS-Model

6 Conclusion

In this paper, we discussed a validation of XCSR in Wilson's experiment in two respects: (1) we analyzed the settings of real-valued XCS specific parameters to evaluate the model; and (2) we analyzed classifier representation with classifier operators such as covering and mutation. To achieve the latter, we proposed an opponent – the LU-Model and compared it with the original – the CS-Model. We conducted comprehensive experiments by applying the 6-dimensional real-valued multiplexor problem to both models, which revealed the following: (1) there is a critical threshold on covering operation parameter (r_0), which must be considered in setting parameters to avoid a serious decrease in performance; and (2) the LU-Model has an advantage with smaller classifier population size within the same rate of performance as the CS-Model, which demonstrated alternative classifier representation for real-valued XCS.

In future work, we intend to do an intensive analysis on GA operations to validate the discovery of a general and accurate real-valued classifier set. Other classes of problems should then be applied to make the discussion general, as all the results are based on the real-valued 6-multiplexor problem in this paper.

Acknowledgement

The research reported here was supported in part by a contract with the Telecommunications Advancement Organization of Japan entitled, 'Research on Human Communication'.

References

1. M. V. Butz and S. W. Wilson, "An algorithmic description of XCS," in Soft Computing, Vol. 6, pp. 144-153, 2002.
2. J. H. Holland, "Escaping brittleness: the possibilities of general-purpose learning algorithms applied to parallel rule-based systems," in Machine learning, an artificial intelligence approach, Volume II, 1986.
3. T. Kovacs, "XCS classifier system reliably evolves accurate, complete, and minimal representations for boolean functions," in Soft Computing in Engineering Design and Manufacturing, pp. 59-68, 1997.
4. R. S. Sutton and A. G. Barto, Reinforcement Learning: an Introduction, MIT Press, Cambridge MA, 1998.
5. S. W. Wilson, "Classifier fitness based on accuracy," in Evolutionary Computation, Vol. 3, No. 2, pp. 149-175, 1995.
6. S. W. Wilson, "Generalization in the XCS classifier system," in Genetic Programming 1998: Proc. of the Third Annual Conference, pp. 665-674, 1998.
7. S. W. Wilson, "Get real! XCS with continuous-valued inputs," in Learning Classifier Systems: from Foundations to Applications, Lecture Notes in Artificial Intelligence (LNAI-1813). Berlin: Springer-Verlag, 2000.
8. S. W. Wilson, "Mining oblique data with XCS," in Lanzi, P. L., Stolzmann, W., and S. W. Wilson (Eds.), Advances in Learning Classifier Systems. Third International Workshop (IWLCS-2000), Lecture Notes in Artificial Intelligence (LNAI-1996). Berlin: Springer-Verlag, 2001.