Revisiting UCS: Description, Fitness Sharing, and Comparison with XCS

Albert Orriols-Puig and Ester Bernadó-Mansilla

Grup de Recerca en Sistemes Intel·ligents
Enginyeria i Arquitectura La Salle
Universitat Ramon Llull
Quatre Camins 2, 08022 Barcelona (Spain)
{aorriols,esterb}@salle.url.edu

Abstract. This paper provides a deep insight into the learning mechanisms of UCS, a learning classifier system (LCS) derived from XCS that works under a supervised learning scheme. A complete description of the system is given with the aim of being useful as an implementation guide. Besides, we review the fitness computation, based on the individual accuracy of each rule, and introduce a fitness sharing scheme to UCS. We analyze the dynamics of UCS both with fitness sharing and without fitness sharing over five binary-input problems widely used in the LCSs framework. Also XCS is included in the comparison to analyze the differences in behavior between both systems. Results show the benefits of fitness sharing in all the tested problems, specially those with class imbalances. Comparison with XCS highlights the dynamics differences between both systems.

1 Introduction

UCS [1] is a learning classifier system (LCS) derived from XCS [22,23] that works under a supervised learning scheme. UCS inherits the main components and structure of XCS, which are adapted for supervised learning. The main differences between both systems are related to 1) classifier's parameters and their update, and to 2) the lack of a prediction array in UCS. UCS's fitness is based on accuracy, computed as the percentage of correct classifications. This leads UCS to explore the consistently correct classifiers and thus evolve only best action maps.

Previous studies on artificial problems showed that UCS could overcome the fitness dilemma [6] that appeared in some problem categories in XCS, whose effect is a misleading pressure that tends to guide search in the wrong direction. Also evolution of best action maps was proved to be an efficient way of searching the search space, specially in large search spaces. UCS also showed to be competitive in real-world problems with XCS, as well as with some non-evolutionary learners such as C4.5 [19] and SMO [21].

Although UCS's results seem promising, there are still some open issues to be addressed. In [1], UCS's lack of fitness sharing was identified as a potential

J. Bacardit et al. (Eds.): IWLCS 2006/2007, LNAI 4998, pp. 96–116, 2008.

weakness. Fitness sharing is attributed to allow for better resource distribution among the different niches. Without fitness sharing, premature good rules can overtake the population, preventing potential niches from being explored and maintained in the population. Although this effect was not strongly observed in previous analyses, we acknowledge that, as current challenges such as unbalanced class problems are posed to UCS [18], the effect may become important. The literature has also proved that fitness sharing is beneficial for LCSs [3,7]. Other advances of the GA and LCS fields such as tournament selection [4] are also introduced in UCS to further optimize performance.

The aim of this paper is two-fold. Firstly, we want to provide a deeper description of all UCS components, detailed enough to be used as an implementation guide. The description also assembles other operators from XCS such as specify and includes new improvements of LCSs such as tournament selection. Also, a fitness sharing scheme is designed for UCS.

Secondly, the paper makes a comparison of UCS without fitness sharing, UCS with fitness sharing, and XCS. We use a testbed consisting of five binary-input classification problems that gather different complexity factors. Three of them were already used in [1]: a) the parity, b) the decoder, and c) the position problem. They represent respectively two cases of strong specialization with two classes (a) and multiple classes (b), and a case with multiple classes and non-uniform distribution of examples per class (c). We further add the imbalanced multiplexer problem and the multiplexer problem with noise. The designed testbed provides representative conditions of real-world problems for analyzing whether fitness sharing improves performance. By comparing UCS with XCS we update and enhance the previous comparison performed in [1] to the current settings. We seek to understand the dynamics of each approach and analyze conditions where each approach is better suited. The paper aims at enhancing our current comprehension of LCSs, and UCS in particular, and at providing a framework for further UCS system investigations and developments.

The remainder of this paper is organized as follows. Section 2 briefly introduces XCS. Next, UCS is described in detail, including UCS's fitness sharing version. Section 4 analyzes UCS with both fitness sharing and non-sharing, also compared to XCS. Finally, Section 5 summarizes the main differences observed, and Section 6 provides the main conclusions of our work. Appendix A provides a full explanation of all problems used in the comparison.

2 XCS in a Nutshell

This section provides a brief introduction to the XCS classifier system. The reader is referred to [22,23] for a detailed explanation and to [8] for an algorithmic description. Although XCS is applicable to several domains, such as reinforcement learning tasks [15] and function approximations [24], we will restrict our description to XCS as a pure classifier system.

XCS is an accuracy-based learning classifier system introduced by S.W. Wilson [22]. The main difference between XCS and previous strength-based LCSs is that

XCS bases fitness on the accuracy of the reward prediction instead of on the reward itself. This led XCS to avoid the presence of strong overgenerals that appeared in strength-based LCS [11]. The accuracy-based approach makes XCS evolve a complete action map (denoted as [O]) of the environment, evolving not only high-rewarded rules (i.e., consistently correct rules), but also consistently incorrect rules (i.e., rules with zero prediction and low error.).

XCS works as an online learner. For each input example, XCS forms the match set [M] consisting of all classifiers with matching condition. If not enough classes are covered in [M] XCS triggers the covering operator, which creates new classifiers with uncovered classes. Under pure exploration, a class is selected randomly, and all classifiers predicting that class form the action set [A]. The class is sent to the environment and the received reward is used to update the parameters of the classifiers in [A]. Eventually, the genetic algorithm is triggered inside the action set [A], and subsumption may be applied to favor accurate and general classifiers. Under exploit mode, XCS predicts the most voted class for each input example. The class vote is computed as a fitness weighted average of the predictions of all classifiers advocating that class.

3 Description of UCS

3.1 UCS Components

UCS is an accuracy-based learning classifier system introduced in [1]. It inherits the features of XCS, but specializes them for supervised learning tasks. UCS mainly differs from XCS in two respects. Firstly, the performance component is adjusted to a supervised learning scheme. As the class is provided with each new example, UCS only explores the class of the input examples. This implies that UCS only evolves high-rewarded classifiers, that is, the best action map [B]. Secondly, accuracy is computed differently in both systems. UCS computes accuracy as the percentage of correct classifications instead of computing it from the prediction error.

In the following, we give a deeper insight into UCS by explaining each component of the system.

Classifier's Parameters. In UCS, classifier's parameters are the following: a) accuracy *acc*; b) fitness F; c) correct set size *cs*; d) numerosity *num*; and e) experience *exp*. Accuracy and fitness are measures of the quality of the classifier. The correct set size is the estimated average size of all the correct sets where the classifier participates. Numerosity is the number of copies of the classifier, and experience is the number of times a classifier has participated in a match set.

Performance Component. UCS is an online learner that receives a new input example $x = (x_1, ..., x_n)$ at each learning iteration. As it works under a supervised learning scheme, also the class c of the example is provided. Then, the system creates the *match set* [M], which contains all classifiers in the population [P] whose condition matches x. Next, all the classifiers in [M] that predict the

class c form the *correct set* [C]. If [C] is empty, the covering operator is activated, creating a new classifier with a generalized condition matching x, and predicting class c. The remaining classifiers form the incorrect set ![C].

In test mode, a new input example x is provided, and UCS must predict the associated class. To do that, the match set [M] is created. All classifiers in [M] emit a vote, weighted by their fitness, for the class they predict. The vote of young classifiers (i.e., $exp < \theta_{del}$) is decreased by multiplying its vote by exp/θ_{del} to avoid that poorly evaluated classifiers emit a strong vote if more experienced classifiers exist. The most-voted class is chosen. New inference schemes have been proposed in [2]. Under test mode, the population of UCS does not undergo any change. All update and search mechanisms are disabled.

Parameter Updates. Each time a classifier participates in a match set, its experience, accuracy and fitness are updated. Firstly, the experience is increased. Then, the accuracy is computed as the percentage of correct classifications:

$$acc = \frac{number\ of\ correct\ classifications}{experience} \tag{1}$$

Thus, accuracy is a cumulative average of correct classifications over all matches of the classifier. Next, fitness is updated according to the following formula:

$$F_{micro} = (acc)^{\nu} \tag{2}$$

where ν is a constant set by the user that determines the strength pressure toward accurate classifiers (a common value is 10). Thus, fitness is calculated individually for each microclassifier, and it is not shared. The fitness of a macroclassifier F_{macro} is obtained by:

$$F_{macro} = F_{micro} \cdot num \tag{3}$$

Finally, each time the classifier participates in [C], the correct set size cs is updated. cs is computed as the arithmetic average of all sizes of the correct sets in which the classifier has taken part.

Discovery Component. The genetic algorithm (GA) is used as the primary search mechanism to discover new promising rules. The GA is applied to [C], following the same procedure as in XCS. It selects two parents from [C] with a probability that depends on classifier's fitness. The same selection schemes applied in XCS can be used in UCS, such as proportional selection or tournament selection. The two parents are copied, creating two new children, which are recombined and mutated with probabilities χ and μ respectively.

Finally, both children are introduced into the population. First, each offspring is checked for subsumption with its parents. The subsumption mechanism is adapted from XCS as follows: if one of the parents is sufficiently experienced ($exp > \theta_{sub}$), accurate ($acc > acc_0$) and more general than the offspring, then the offspring is not introduced into the population and the numerosity of this parent is increased. If the offspring cannot be subsumed for any of its parents, it is inserted in the population, deleting another classifier if the population is full.

Specify. The specify operator [14] is also adapted from XCS. At each learning iteration, the accuracy of the correct set $acc_{[C]}$ is compared to the average accuracy of the whole population $acc_{[P]}$. If $acc_{[C]} \leq \frac{acc_{[P]}}{2}$ and the average experience of the classifiers in [C] is greater than N_{Sp} ($exp_{[C]} \geq N_{Sp}$), a classifier is randomly selected from [C], with probability inversely proportional to its accuracy. Then, the classifier is copied, and its *don't care* symbols are specified —to the value of the input example— with probability P_{Sp}. Finally, the new classifier is introduced in the population, removing potentially poor classifiers if the population is full.

Deletion. The deletion probability of each rule is calculated as:

$$p_{del} = \begin{cases} \frac{cs \cdot num \cdot F_{[P]}}{F_{micro}} & \text{if } exp > \theta_{del} \text{ and } F_{micro} < \delta F_{[P]} \\ cs \cdot num & \text{otherwise} \end{cases} \quad (4)$$

where δ and θ_{del} are parameters set by the user, and $F_{[P]}$ is the average fitness of the population. In this way, deletion will tend to balance resources among the different correct sets, while removing low-fitness classifiers from the population. As fitness is computed from the percentage of correct classifications of a classifier, classifiers that predict wrong classes are not maintained in the population, and so, only the best action map evolves.

Classifier Parameters Initialization. UCS is very robust to parameter initialization since the initial value of most of the parameters is lost the first time that the classifier participates in a match set. When a classifier is created by covering, its parameters are set to: $exp = 1$, $num = 1$, $cs = 1$, $acc = 1$ and $F = 1$. If a classifier is created by the GA, its parameters are initialized to: $exp = 1$, $num = 1$, $cs = (cs_{p_1} + cs_{p_2})/2$ (where p_1 and p_2 denote each of the parents), $acc = 1$ and $F = 1$.

3.2 Why Should We Not Share Fitness?

UCS showed to perform competitively with other machine learning techniques in both artificial and real-world domains [1], even without fitness sharing. However, there are some analyses in the literature that demonstrate the advantages of sharing fitness in learning classifier systems [3] and, in general, in genetic algorithms [9].

Thus, we introduce a new fitness computation scheme that shares fitness, similarly to XCS, with the aim of comparing its advantages and disadvantages with a non sharing scheme. In the remainder of the paper, UCS without sharing is referred as UCSns, and UCS with sharing as UCSs.

Parameters update with fitness sharing works as follows. Experience, correct set size and accuracy are computed as in UCSns. However, fitness is shared among all classifiers in [M]. Firstly, a new accuracy k is calculated, which discriminates between accurate and inaccurate classifiers. For classifiers belonging to ![C], $k_{cl \in ![C]} = 0$. For classifiers belonging to [C], k is computed as follows:

$$k_{cl \in [C]} = \begin{cases} 1 & if \quad acc > acc_0 \\ \\ \alpha(acc/acc_0)^{\nu} & otherwise \end{cases}$$

Then, a relative accuracy k' is calculated:

$$k'_{cl} = \frac{k_{cl} \cdot num_{cl}}{\sum_{cl_i \in [M]} k_{cl_i} \cdot num_{cl_i}} \tag{5}$$

And fitness is updated from k':

$$F = F + \beta \cdot (k' - F) \tag{6}$$

Let's note that, under this scheme, the computed fitness corresponds to the macroclassifier's fitness, as numerosities are involved in the formulas. Whenever the microclassifier's fitness is needed (e.g., in the deletion algorithm), we use the relation of formula 3.

4 XCS and UCS in Binary-Input Problems

4.1 Methodology

The aim of this section is to analyze different aspects of UCS and XCS learning behavior. We base our analysis on five artificial problems that gather some complexity factors said to affect the performance of LCSs [12,1]: a) a binary-class problem; b) a multiclass problem; c) an imbalanced binary-class problem; d) an imbalanced multiclass problem; and e) a noisy problem. For a detailed description of each problem the reader is referred to appendix A. Each problem was run with UCSns, UCSs and XCS.

If not stated differently, we used the following *standard configuration* for XCS: $P_{\#} = 0.6$, $\beta = 0.2$, $\alpha = 0.1$, $\nu = 5$, $\theta_{GA} = 25$, *selection* = tournament, $\chi = 0.8$, $\mu = 0.04$, $\theta_{del} = 20$, $\delta = 0.1$, GA_{sub} = true, $[A]_{sub}$ = false, $\theta_{sub} = 20$. When specify was enabled, $N_s = 0.5$ and $P_s = 0.5$. Parameters for UCS had the same values as in XCS, with acc_0=0.999 and ν=10.

The maximum population size of XCS and UCS was configured depending on the size of the optimal population that the systems were expected to evolve. XCS evolves a complete action map [O] [12], which consist of both highly rewarded rules[1] and poorly rewarded rules[2] with low error. On the other hand, UCS evolves the best action map [B] [1], which includes only highly rewarded rules. As proposed in [1], we set population sizes to $N = 25 \cdot |[O]|$ in XCS, and to $N = 25 \cdot |[B]|$ in UCS For each problem, we sought to find the parameters' configuration that permitted to obtain the best results. Thus, any changes on the parameter settings will be properly stated.

Different metrics were used to evaluate performance. We discarded accuracy as the measure of performance, since it does not provide enough evidence of effective genetic search, as pointed out in [13]. Instead of accuracy, the proportion

[1] Rules with high prediction and low error.
[2] Rules with low prediction and low error.

of the optimal action map achieved %[O] [12] was proposed as being a better indicator of the progress of the genetic search [13]. However, UCS and XCS evolve different optimal populations: XCS creates the complete action map [O], whereas UCS represents a best action map [B]. To allow a fair comparison, we only consider the proportion of best action map %[B] achieved by each system. That is, we only count the percentage of consistently correct rules. Besides, the proportion of correct classifications of the minority class (TP rate) is also used in imbalanced binary-class problems.

4.2 Binary-Class Problem: Parity

We tested the LCSs on the parity problem. The parity problem is a binary-class problem, whose class is 1 if the number of ones in the input is odd and 0 otherwise. The problem does not allow any generalization in the condition rules. Please, see appendix A.1 for the details. We ran the parity problem with condition lengths from 3 to 9 and the standard configuration. To stress specificity, we enabled the specify operator in XCS and UCS.

Figure 1 depicts the proportion of the best action map %[B] achieved by UCSns, UCSs and XCS. Curves are averages of ten runs. The plots show that UCSs converges earlier than UCSns, which shows up the advantages of fitness sharing. This is specially observed for larger input lengths, such as for 8 and 9 inputs. UCSs also improves XCS, and in turn XCS slightly improves UCSns. Next, we discuss these results in more detail.

Table 1. Accuracy and fitness of UCSns's classifiers along the generality-specifity dimension, depicted for the parity problem with $\ell = 4$

	Condition	Class	Accuracy	Fitness
1	####	0	0.5	0.00097
2	0###	0	0.5	0.00097
3	00##	0	0.5	0.00097
4	000#	0	0.5	0.00097
5	0000	0	1	1

a) Why is the Parity Problem Difficult for Both Systems?
The parity problem appears to be a hard problem for both learners. Its main difficulty can be attributed to the lack of *fitness guidance* toward optimal classifiers. As some generalization is introduced by the covering operator, XCS and UCS have to drive the population from generality to specificity. However, the fitness pressure does not correctly lead to specificity. That is, specifying one bit of an overgeneral classifier does not increase its accuracy unless all bits are specified. Table 1 shows the evolution that would suffer the most overgeneral classifier to become an optimal classifier in UCSns in the parity problem with ℓ=4. At each step, one of the *don't care* bits is specified. Note that accuracy, and so, fitness, remain constant during all the process until the optimal classifier is achieved. However, we would expect that the specification of one bit should result in a fitter classifier, since it

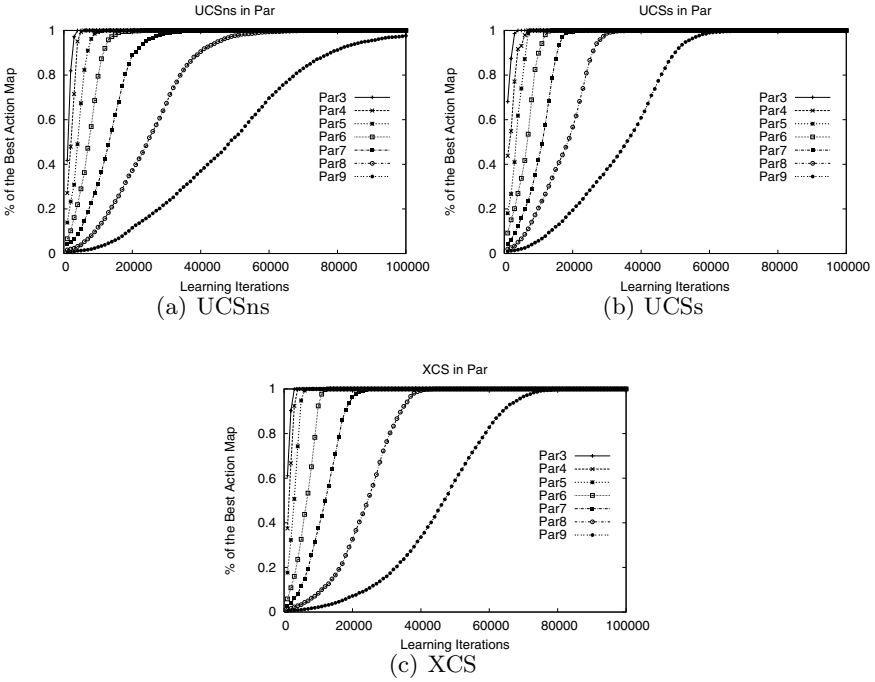


Fig. 1. Proportion of the best action map achieved by UCSns (a), UCSs (b) and XCS (c) in the parity problem with condition lengths from ℓ=3 to ℓ=9

approaches the optimum classifier (in which all the bits are specified). Therefore, the fitness is not guiding toward optimal optimal solutions, and so, an optimal classifier can only be obtained randomly. This example is also generalizable to XCS, for which fitness guidance is carefully analyzed in [6].

b) Exploring Only High-Rewarded Niches

Under a pure explore regime, XCS uniformly explores niches containing high-rewarded rules and niches with consistently incorrect rules (provided that examples come uniformly). Actually, the system maintains a complete action map. On the other hand, UCS explores only the high rewarded niches, i.e., the best action map.

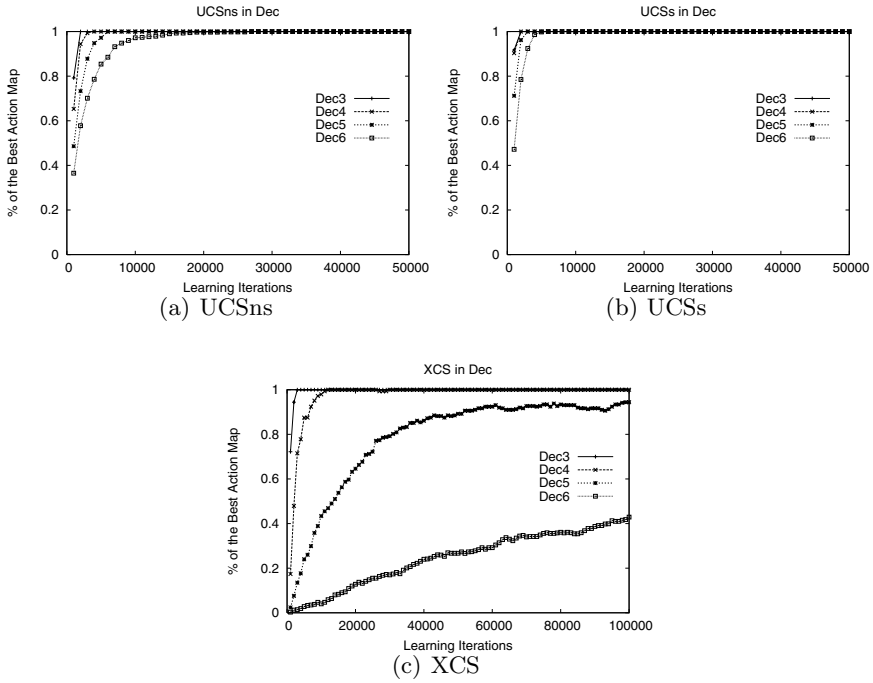
Thus, UCS is expected to evolve the best action map in half the time XCS does. However, results show that both systems need a similar number of learning iterations to converge. Our hypothesis is that the exploration of consistently incorrect rules may help XCS to discover consistently correct rules. For example, if XCS evolves a consistently incorrect classifier such as `0001:0`, XCS may discover the consistently correct classifier `0001:1` by mutation of the class bit. Thus, XCS can discover parts of [B] while exploring low-rewarded niches.

c) The Advantages of Sharing Fitness

The results show that the convergence curves in UCSs and XCS are steeper than in UCSns. After some initial iterations, the systems start to evolve some optimal classifiers. In the case of UCSs and XCS, once the first optimal classifiers are discovered, the convergence curves raise quickly. This effect is due to fitness sharing. Under fitness sharing, the discovery of a new optimal classifier makes the fitness of overgeneral classifiers that participate in the same action/correct set decrease quickly. This produces a) a higher pressure toward overgeneral deletion and b) a higher selective pressure toward specific classifiers in the GA. Thus, the GA is likely to produce new specific, and so, optimal classifiers. Without fitness sharing, overgeneral classifiers maintain the same fitness along the whole learning process. This effect is specially strong in imbalanced data sets, where overgeneral classifiers have higher accuracy.

4.3 Multiclass Problem: Decoder

In this section we aim at analyzing the behavior of XCS and UCS on multiclass problems. For this purpose, we use the decoder problem, which has $\ell + 1$ classes, being ℓ the string length. The best action map contains 2^ℓ specific classifiers, while the complete action map has $(\ell+1)\cdot 2^\ell$ classifiers. See appendix A.2 for the details.



(a) UCSns (b) UCSs

(c) XCS

Fig. 2. Proportion of the best action map achieved by UCSns (a), UCSs (b) and XCS (c) in the decoder problem with condition lengths from ℓ=3 to ℓ=6. Note that UCS is shown for 50,000 explore trials, while XCS is shown for 100,000 trials.

Figure 2 depicts the proportion of the best action maps achieved by UCSns, UCSs and XCS. The results evidence a much better performance of UCS, specially for higher condition lengths. With a condition length ℓ=6, UCSs needs up to 5,000 learning iterations to discover the complete best action map, whereas XCS discovers only 40% of it in 100,000 iterations. This huge difference between XCS and UCS could mainly be explained by a) the explore regime and b) the accuracy guidance toward the optimal population.

a) Explore Regime

The first aspect that hinders the performance of XCS is the explore regime used. As XCS explores uniformly each class, only 1 of each $\ell + 1$ explores will be made on the class of the input instance. The other ℓ explores will be focused on classifiers that predict wrong classes. This issue, which appeared to have low effect in the parity problem, now takes importance since the number of classes increases. Moreover, the hamming distance between consistently incorrect classifiers and correct classifiers is bigger than in the parity problem. So, in the decoder problem, it is more difficult to create a high-rewarded classifier while exploring a low-rewarded niche.

b) Accuracy Guidance Toward Optimal Population

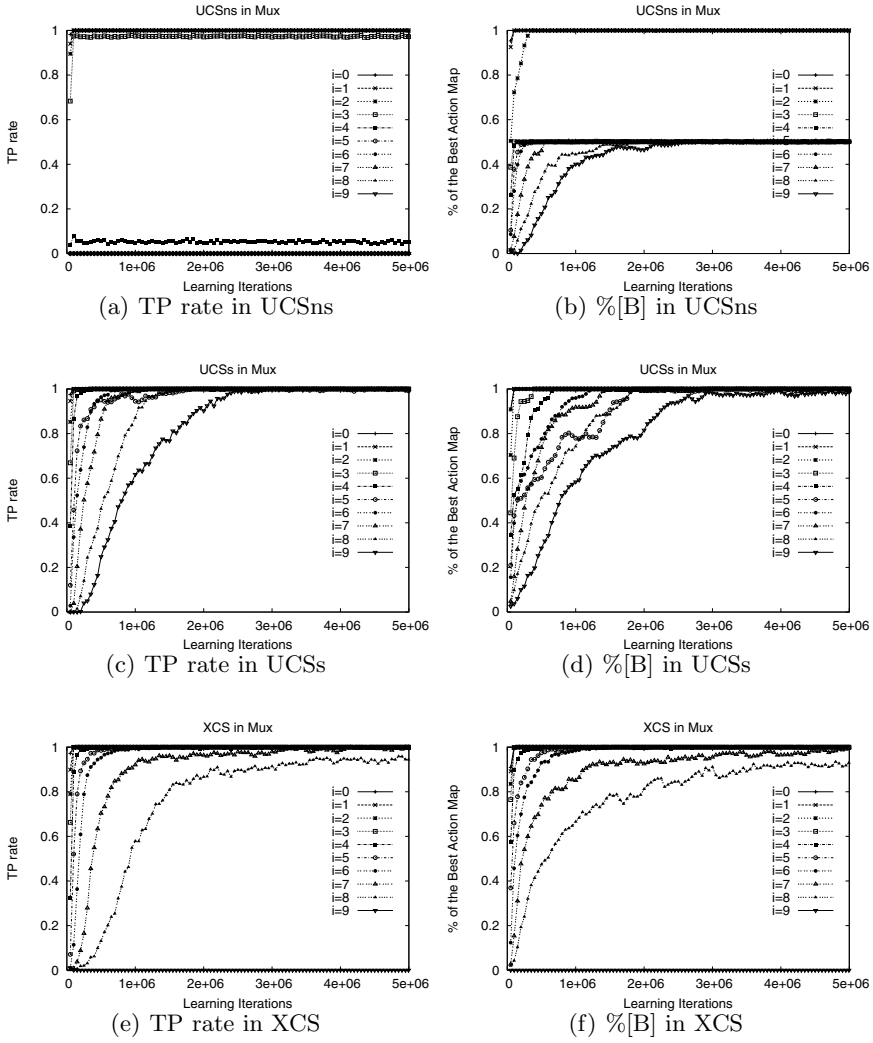
The second point that influences the convergence of XCS is the way in which fitness guides to the solution. This issue was initially termed by Butz et al. as the *fitness dilemma* [6]. Let's suppose we have the overgeneral classifier cl_1 `1###:8`, whose estimated parameters are P=125 and $\epsilon = 218.75$ [6]. XCS is expected to drive cl_1 to the classifier `1000:8`. Imagine now that, either mutation or specify generates the classifier cl_2 `10##:8`, whose parameter estimates are P=250 and $\epsilon = 375$. That is, as the classifier approaches the optimal one, the error increases. As long as the classifier moves towards the right direction, it gets smaller fitness, which consequently means fewer genetic event opportunities and higher deletion probabilities. Thus, there is no effective fitness pressure towards optimal classifiers.

UCS overcomes the *fitness dilemma*, because the accuracy is calculated as the percentage of correct classifications. Therefore, classifier cl_2 would have higher accuracy than cl_1. In that way, UCS's accuracy guidance does not mislead the search.

Finally, it can be observed that UCSs converges slightly quicker than UCSns. As in the parity problem, we ascribe this behavior to the fact that fitness sharing makes a stronger pressure toward the optimal classifiers as long as they are discovered.

4.4 Imbalanced Binary-Class Problem: Imbalanced Multiplexer

Real-world problems often have smaller proportion of examples of one class than the others. It has been widely accepted that this issue, addressed as the class imbalance problem [10], could hinder the performance of some well-known learners. Recent studies [16,18] have shown that UCSns suffers from class imbalances,



(a) TP rate in UCSns (b) %[B] in UCSns

(c) TP rate in UCSs (d) %[B] in UCSs

(e) TP rate in XCS (f) %[B] in XCS

Fig. 3. TP rate and % of [B] achieved by UCSns, UCSs and XCS in the 11-bit multiplexer problem with imbalance levels from i=0 to i=9

while different mechanisms acting at the classifier level or at the sampling level can alleviate the effect. On the other hand, XCS has shown to be highly robust to class imbalances if its parameters are set appropriately (see [17]). In this section we compare the behavior of XCS and UCS in a controlled imbalanced problem, and besides, analyze the influence of fitness sharing in such conditions.

We used the imbalanced 11-bit multiplexer (see appendix A.4), where minority class instances were sampled with a proportion P_{accept}. In the remainder of the paper we use i to refer to the imbalance level, that is, $P_{accept} = \frac{1}{2^i}$.

We ran XCS and UCS with the 11-bit multiplexer and imbalance levels from i=0 to i=9. Parameters were set as suggested in [17]: for high imbalance levels, β should be increased to allow more reliable parameter estimates, while θ_{GA} should be decreased to allow more uniform genetic opportunities among the different niches. Moreover, we only apply subsumption when the involved classifiers have received, at least, 2^i parameter updates. Thus, for XCS we used the standard configuration but setting $\beta = \{0.04, 0.02, 0.01, 0.005\}$, $\theta_{GA} = \{200, 400, 800, 1600\}$, and $\theta_{sub} = \{64, 128, 256, 512\}$ for i=$\{6, 7, 8, 9\}$ respectively. As UCS appeared to be less sensitive to parameters' settings in previous experiments (not reported here), we maintained the standard configuration. Only for i$\geq$6, $\theta_{GA} = 50$ and $\beta = 0.02$.

Figure 3 shows the proportion of correct classifications of the minority class (TP rate) and the proportion of the best action map achieved by each system. We do not show the proportion of correct classifications of the majority class since it raises to 100% in few iterations. The results show that UCSns fails at low imbalance levels (it only converges for i$\leq$3), whereas XCS solves the problem up to i=8, and UCSs up to i=9.

Let's analyze the behavior of UCSns. Figure 3(b) shows that, for i$\geq$4, UCSns evolves only half of the best action map. Looking at the populations evolved (see table 2 for $i = 6$) we observed that UCSns discovered all optimal classifiers predicting the minority class (class 1). However, optimal classifiers predicting the majority class were replaced by the most overgeneral classifier cl_{ovg}: `###########:0`. This behavior is related to the accuracy computation and the lack of fitness sharing in UCSns. At any imbalance level, the accuracy expected for cl_{ovg} is $acc_{cl_{ovg}} = 1 - P_{accept}$, and the fitness is computed as a power of the accuracy. For i=6, $acc_{cl_{ovg}} = 0.9843$, which gives a high fitness value. The classifier's accuracy is still lower than acc_0 (recall that acc_0 is set to 0.999); thus, the classifier would be considered as inaccurate for subsumption purposes. However, since cl_{ovg} participates in $1 - P_{accept}$ of all correct sets, the classifier receives many genetic opportunities, and finally overtakes the population. Increasing ν

Table 2. Most numerous rules evolved in a single run of UCSns with the 11-bit imbalanced multiplexer for i=6. Cond. is the classifier's condition, C. the class it predicts, and Acc., F., and Num. are the accuracy, fitness and numerosity of the classifier.

Cond.	C.	Acc.	F.	Num.
`###########`	0	0.98	0.86	142
`0001#######`	1	1.00	1.00	55
`001#1######`	1	1.00	1.00	74
`010##1#####`	1	1.00	1.00	60
`011###1####`	1	1.00	1.00	63
`100####1###`	1	1.00	1.00	57
`101#####1##`	1	1.00	1.00	69
`110######1#`	1	1.00	1.00	57
`111#######1`	1	1.00	1.00	68
...				

to make a stronger pressure toward accuracy does not significantly improve the results for high imbalance levels.

This effect does not appear in XCS and UCSs with appropriate parameter settings. As both systems share fitness, the fitness of overgeneral classifiers would considerably diminish when an optimal classifier is discovered. Figures 3(c)-3(f) show the improvement obtained with UCSs and XCS. UCSs shows to perform slightly better at high imbalance levels, being able to solve the 11-bit multiplexer even for i=9, in which XCS fails. At these regimes of such low supply of minority class examples, exploring only the correct class is crucial to maximally benefit from exploration. In fact, as XCS explores half of the correct actions for a given number of iterations (with respect to UCS), one could expect that UCSs solves the multiplexer up to one imbalance level greater than XCS does.

4.5 Imbalanced Multiclass Problem: Position

After analyzing the effects of class imbalance on an artificially imbalanced problem, we introduce the position problem, which intrinsically has unequal distribution of examples per class, and a number of classes that increases linearly with the condition length (see appendix A.3 for a description of the problem).

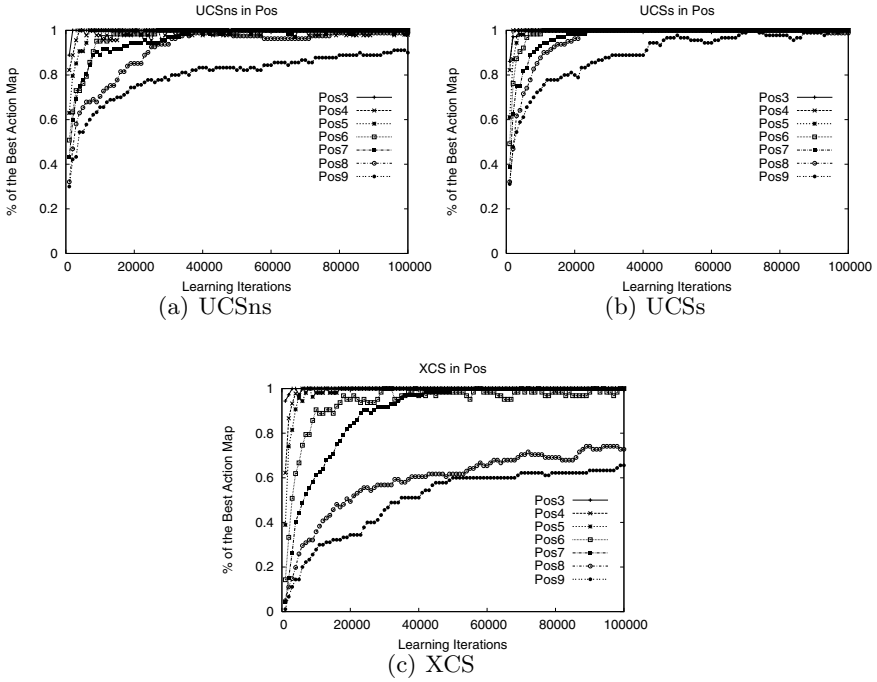


Fig. 4. Proportion of the best action map achieved by UCSns (a), UCSs (b) and XCS (c) in the position problem with condition lengths from l=3 to l=9

We ran UCSns, UCSs and XCS with the position problem and the standard configuration. Specify was disabled to introduce more generalization pressure. Figure 4 shows the proportion of the best action map achieved.

The position problem combines the effects of class imbalances and fitness dilemma in a single problem. The best action map consists of classifiers with different levels of generality. Thus, the most specific rules are expected to suffer from fewer genetic opportunities, since they are activated less often [1]. The results obtained support this hypothesis. In all cases, the shape of the curves have a steep increase at the beginning —where the most general rules are discovered—, and afterward, the curves improve slowly. Examining the populations (not shown for brevity), we confirmed that the most specific rules were difficult to discover.

In addition, the results obtained with XCS are poorer than with UCS. This can be again ascribed to the fitness dilemma, as in the decoder problem. That is, there is a misleading pressure toward optimal rules. Figure 5 shows an example of how the prediction error increases when driving the overgeneral rule $\#\#\ldots\# : 0$ to the optimal classifier $00\ldots : 0$ for different input lengths ℓ. This, coupled together with the few genetic opportunities that the most specific classifiers receive, makes the discovery of these classifiers very hard.

Finally, UCSs slightly outperforms UCSns. For position with $\ell = 9$, after 100,000 iterations, UCSs is able to discover all the best map while UCSns discovers 90% of the best action map. Again, fitness sharing helps to discover the most specific classifiers.

4.6 Noisy Problem: Multiplexer with Alternating Noise

To conclude the study, we analyze the effects of training XCS and UCS on noisy environments. To do that, we used the 20-bit multiplexer introducing alternating noise (denoted as mux_{an}). That is, the class of a input instance was flipped with probability P_x, as proposed in [5].

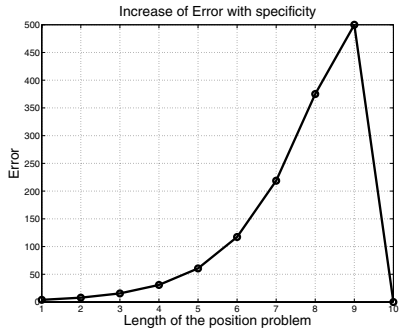
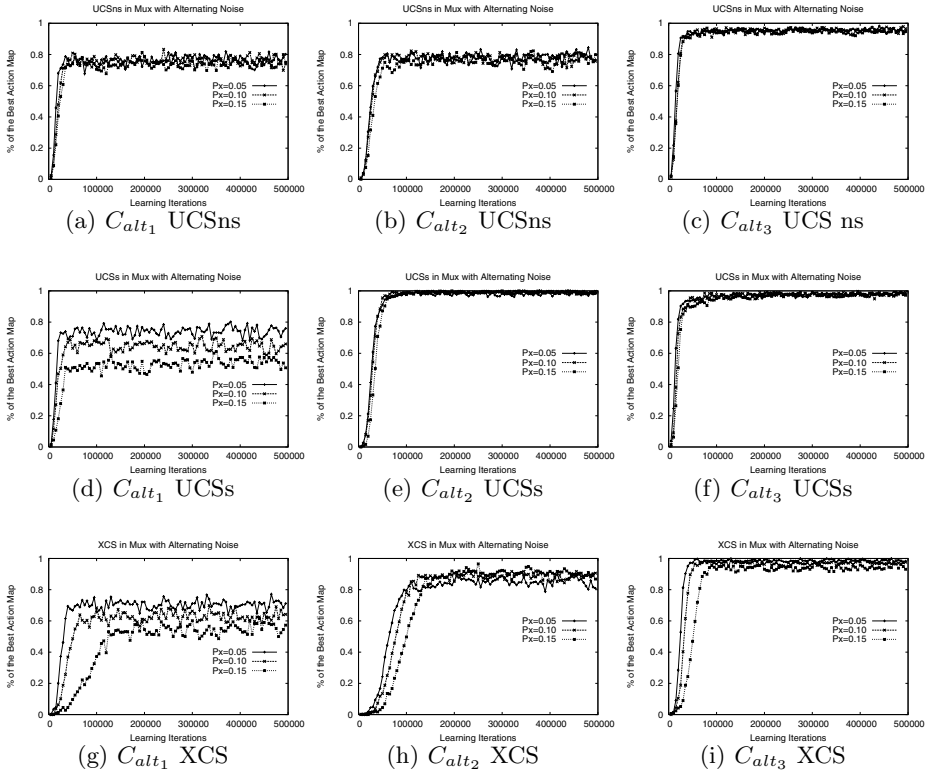


Fig. 5. Error of XCS's classifiers along the generality-specificity dimension. The curve depicts how the error of the most overgeneral classifier $\#\#\ldots\# : 0$ evolves until obtaining the maximally accurate rule $00\ldots : 0$.



(a) C_{alt_1} UCSns (b) C_{alt_2} UCSns (c) C_{alt_3} UCS ns

(d) C_{alt_1} UCSs (e) C_{alt_2} UCSs (f) C_{alt_3} UCSs

(g) C_{alt_1} XCS (h) C_{alt_2} XCS (i) C_{alt_3} XCS

Fig. 6. Proportion of the best action map achieved by UCSns, UCSs, and XCS in the 20-bit multiplexer with alternating noise and with configurations C_{alt_1}, C_{alt_2}, and C_{alt_3}

The effect of noise may be related to the effect of undersampled classes to some extent. The system receives wrongly labeled instances in a percentage P_x of the explore trials. The key difference here is that each instance is correctly labeled in a high proportion of samples (1 - P_x), and only in small proportion of cases (P_x), the instance comes with the wrong class. Thus, we aim at analyzing the ability of each system to obviate noisy instances in favor of non-noisy ones.

We ran UCSns, UCSs and XCS in the 20-bit multiplexer with alternating noises $P_x = \{0.05, 0.10, 0.15\}$ and three different parameter configurations C_{alt_1}, C_{alt_2} and C_{alt_3}. C_{alt_1} is the standard configuration with specify disabled. C_{alt_2} sets β=0.01 and $\theta_{GA} = 100$ to have more reliable parameters estimates before the GA triggers. Finally, C_{alt_3} is based on C_{alt_1}, setting $acc_0 = 1 - P_x$ and $\epsilon_0 = P_x * R_{max}$ to let the system admit the noise of the data. In all runs, the population size was set to 2,000 in both UCS and XCS[3]. Figure 6 shows the percentage of optimal population achieved for each setting.

[3] UCS population size is equal to XCS, because the effect of wrongly labeled instances makes UCS try to evolve consistently incorrect rules to fit to noise.

With C_{alt_1} setting, UCSns gets higher performance than UCSs and XCS, specially for the highest levels of noise (see figures 6(a), 6(d) and 6(g)). This behavior can be attributed to the way that classifier parameters are estimated. Both XCS and UCSs compute a windowed average of fitness by means of the learning parameter β. In noisy environments, the parameter averages oscillate and cannot stabilize properly. So, high levels of noise require low values of β. As UCSns computes fitness as a power of the accuracy, parameter values of experienced classifiers remain steady.

Figures 6(b), 6(e) and 6(h) show the proportion of the best action map achieved with C_{alt_2} setting, which is configured to lead to more stable parameters estimates. The results show a clear improvement of UCSs and XCS with respect to configuration C_{alt_1}. This proves our hypothesis that higher levels of noise require lower β values to allow for better parameter stabilization, coupled with higher θ_{GA} to let the genetic algorithm operate with better estimates. UCSs specially benefits from that, nearly reaching 100% of the best action map in few iterations. Results of UCSns are the same as those obtained with C_{alt_1} setting, as it does not use β in the fitness estimate.

Finally, figures 6(c), 6(f) and 6(i) show the experiments with C_{alt_3} setting, which fits acc_0 and ϵ_0 to consider noise as negligible. This setting allows optimal classifiers, which include a certain percentage of noise, to be considered as accurate. This lets optimal classifiers have higher accuracy estimates (in the case of UCSs and XCS). Besides, subsumption would be applied toward optimal classifiers increasing the pressure toward them. Results appear better to those obtained with C_{alt_1} in all LCSs. With respect to C_{alt_2}, we obtained better results than with C_{alt_3} in UCSns and XCS. Combining C_{alt_2} and C_{alt_3} benefits from the advantages of each approach (not shown for brevity).

5 Summing Up

In this section, we briefly sum up the differences observed when sharing is introduced in UCS, as well as the different dynamics between UCS and XCS.

Explore Regime. Exploring only the class of the input instance (as UCS does) appeared to be beneficial in problems with high number of classes, e.g., decoder and position. Moreover, it helped to solve the imbalanced multiplexer up to one imbalance level higher than XCS, in an extreme low supply of minority class instances. Finally, the effects were nearly imperceptible in the parity problem, since the exploration of the low-rewarded niches could lead to discover high-rewarded optimal classifiers.

We used a pure explore regime in XCS, where each available class is uniformly explored for each possible input. Since UCS proves that a more directed search toward the correct class generally speeds up convergence, we could also think of other explore regimes in XCS. Training in XCS could be based on an exploration regime that gradually changes from pure exploration toward increasing exploitation, similarly to schemes such as softmax [20]. This remains an open issue for further work.

Accuracy Guidance. The results of XCS in some domains showed the lack of fitness guidance toward accurate classifiers. This problem, already observed in previous studies [1,6], was termed the fitness dilemma in [6]. The problem does not exist in UCS since accuracy is computed directly as the percentage of correct classifications. We showed that XCS strongly suffers from the fitness dilemma in the decoder, and to a lower degree, in the position. In these cases, UCS clearly outperformed XCS. To alleviate this effect, bilateral accuracy was proposed for XCS [6]. As a future work, we aim to investigate how this approach compares with UCS.

Fitness Sharing. Fitness sharing speeds up the convergence in all problems tested. Specially, it appears to be crucial in highly imbalanced data sets to deter overgeneral classifiers from overtaking the population. UCSns only gave better performance when parameter estimates were unstable in UCSs due to an inappropriate setting of the learning parameter β. Anyway, this effect cannot be attributed to the presence or absence of fitness sharing, but rather to the way in which fitness is estimated. Recall that UCSns computes fitness as a power of accuracy, while UCSs computes fitness as a weighted windowed average with learning parameter β.

Population Size. In the tested problems, UCS evolved best action maps with less learning iterations. Also smaller population sizes were used in UCS in all the tested problems, except for the noisy problem. The population evolved by XCS is generally larger, but comparable to that of UCS in terms of legibility. In fact, by removing low-rewarded classifiers from XCS's final population, we get a set of rules similar to that of UCS (not shown for brevity). Thus, the advantage of having smaller populations in UCS lies in the reduction of computational resources.

6 Conclusions

This paper provided insight into the UCS learning classifier system. We improved the original UCS system as introduced in [1] by including tournament selection and fitness sharing. Robustness of the modified UCS was proved across different artificial domains. Specially, fitness sharing was necessary in the imbalanced multiplexer problem. We suspect that this behavior can be also generalizable to other imbalanced problems, where overgeneral classifiers can easily become strong. Using sharing, we allow overgenerals until optimal classifiers start to evolve. When this happens, fitness of overgeneral classifiers decreases fast by the effect of sharing fitness with better competing solutions.

The comparison with XCS allowed for a better understanding of the differences between the two approaches of accuracy-based classifier systems. UCS has an architecture specifically designed for supervised learning problems. XCS is more general, and can be applied to multi-step problems, learning only from the feedback about action consequences. Thus, it is reasonable that XCS does not perform as well as UCS in supervised environments.

Two key differences provide UCS with better results in some classification domains: exploration focuses on best action maps and correct fitness pressures towards accuracy. Suggestions to improve XCS's convergence are made, such as using search regimes with more exploitation guidance. Some methods such as those based on ϵ-greedy action-selection or softmax action-selection [20] have already been tested on reinforcement learners. Their introduction to XCS could lead to performance similar to UCS. To avoid the effects of the fitness dilemma in XCS, the use of bilateral accuracy is proposed, as suggested in [6].

The experiments on a noisy problem showed high performance for the LCSs. Both UCS and XCS could achieve optimal populations with a level of noise up to 15%. Future work could enhance this study by analyzing LCSs' tolerance to increasing levels of noise. Our experiments performed in noisy and imbalanced problems showed that UCS was less sensitive to parameter settings. In XCS, two parameters became critical for optimal performance: the learning rate β and the GA triggering threshold θ_{GA}.

Although our results and conclusions are limited to artificial problems, our experimental testbed contained many complexity factors present in real-world problems: multiple classes, noisy instances and imbalanced classes, among others. In this sense, the paper provided some guidelines for further improving learning classifier systems in increasingly challenging problems.

Acknowledgements

The authors thank the support of *Enginyeria i Arquitectura La Salle*, Ramon Llull University, as well as the support of *Ministerio de Ciencia y Tecnología* under project TIN2005-08386-C05-04, and *Generalitat de Catalunya* under Grants 2005FI-00252 and 2005SGR-00302.

References

1. Bernadó-Mansilla, E. and Garrell, J.M. Accuracy-Based Learning Classifier Systems: Models, Analysis and Applications to Classification Tasks. *Evolutionary Computation*, 11(3):209–238, 2003.
2. Brown, G., Kovacs, T., and Marshall, J.A.R. UCSpv: principled voting in UCS rule populations. In *GECCO'07*, pages 1774–1781, New York, NY, USA, 2007. ACM Press.
3. Bull, L. and Hurst, J. ZCS Redux. *Evolutionary Computation*, 10(2):185–205, 2002.
4. M.V. Butz, K. Sastry, and D.E. Goldberg. Strong, stable, and reliable fitness pressure in XCS due to tournament selection. *Genetic Programming and Evolvable Machines*, 6(1):53–77, 2005.
5. Butz, M.V. *Rule-Based Evolutionary Online Learning Systems: A Principled Approach to LCS Analysis and Design*, volume 109 of *Studies in Fuzziness and Soft Computing*. Springer, 2006.
6. Butz, M.V., Goldberg, D., and Tharankunnel, K. Analysis and improvement of fitness exploration in XCS: Bounding models, tournament selection, and bilateral accuracy. *Evolutionary Computation*, 11(3):239–277, 2003.

7. Butz, M.V., Goldberg, D.E., and Lanzi, P.L. Effect of pure error-based fitness in XCS. In *Learning Classifier Systems*, volume 4399/2007 of *LNCS*, pages 104–114. springer, 2007.
8. Butz, M.V. and Wilson, S.W. An algorithmic description of XCS. In P.L. Lanzi, W. Stolzmann, and S.W. Wilson, editors, *Advances in Learning Classifier Systems: Proceedings of the Third International Workshop*, volume 1996 of *Lecture Notes in Artificial Intelligence*, pages 253–272. Springer, 2001.
9. Harik, G. Finding Multiple Solutions in Problems of Bounded Difficulty. Technical report, IlliGAL Report No. 94002, Urbana-Champaign IL 61801, USA, May 1994.
10. Japkowicz, N. and Stephen, S. The Class Imbalance Problem: A Systematic Study. *Intelligent Data Analisis*, 6(5):429–450, November 2002.
11. Kovacs, T. Strength or Accuracy? Fitness Calculation for Classifier Systems. In *In Lanzi, Stolzmann, and Wilson Eds, Learning Classifier Systems. From Foundations to Applications. Lecture Notes in Artificial Intelligence 1813*, pages 143–160. Springer-Verlag, 2000.
12. Kovacs, T. and Kerber, M. What makes a problem hard for XCS. In *Lanzi, P. L., Stolzmann, W., & Wilson, S. W. (Eds.), Advances in Learning Classifier Systems: Third International Workshop, IWLCS*, pages 80–99. Springer-Verlag, 2000.
13. Kovacs, T. and Kerber, M. High Classification Accuracy does not Imply Effective Genetic Search. In *GECCO'04*, pages 785–796, Seattle, WA, USA. 26-30 June, 2004. Springer, LNCS 3103.
14. Lanzi, P.L. A Study of the Generalization Capabilities of XCS. In Thomas Bäck, editor, *Proc. of the Seventh Int. Conf. on Genetic Algorithms*, pages 418–425, San Francisco, CA, 1997. Morgan Kaufmann.
15. Lanzi, P.L. *Learning Classifier Systems: A Reinforcement Learning Perspective*, volume 183/2005 of *Studies in Fuzziness and Soft Computing*, pages 267–284. Springer, 2005.
16. Orriols-Puig, A. and Bernadó-Mansilla, E. The Class Imbalance Problem in UCS Classifier System: Fitness Adaptation. In *Congress on Evolutionary Computation*, volume 1, pages 604–611, Edinburgh, UK, 2-5 September 2005. IEEE.
17. Orriols-Puig, A. and Bernadó-Mansilla, E. Bounding XCS Parameters for Unbalanced Datasets. In *GECCO'06*, pages 1561–1568. ACM Press, 2006.
18. Orriols-Puig, A. and Bernadó-Mansilla, E. The Class Imbalance Problem in UCS Classifier System: A Preliminary Study. In *Learning Classifier Systems*, volume 4399/2007 of *LNCS*, pages 161–180. Springer, 2007.
19. Quinlan, J.R. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, San Mateo, California, 1995.
20. Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*. Cambridge, MA: MIT Press, 1998.
21. Vapnik, V. *The Nature of Statistical Learning Theory*. Springer Verlag, New York, 1995.
22. Wilson, S.W. Classifier Fitness Based on Accuracy. *Evolutionary Computation*, 3(2):149–175, 1995.
23. Wilson, S.W. Generalization in the XCS Classifier System. In *Genetic Programming: Proceedings of the Third Annual Conference*, pages 665–674. Morgan Kaufmann, 1998.
24. Wilson, S.W. Classifiers that approximate functions. *Journal of Natural Computing*, 1(2):211–234, 2002.

A Problem Definitions

In this section, we introduce the problems used in the paper. For each one, a short explanation of its characteristics and the sizes of the best action map $\%[B]$ and the complete action map $\%[O]$ are provided.

A.1 Parity

The parity is a problem that has widely been used as a benchmark in LCS since it was originally introduced in [12] to show the dependence of XCS's performance on the optimal population size. Given a binary string of length ℓ, the number of ones modulo two determines the output. Thus, the problem does not permit any generalization.

The best action map size consists of all the specific rules predicting the correct class, that is, $|[B]| = 2^\ell$. The complete action map doubles the best action map, as it also maintains specific rules predicting the wrong class. Then $|[O]| = 2^{\ell+1}$.

A.2 Decoder

The decoder problem is an artificial problem with binary inputs and multiple classes. Given an input of length ℓ, the output is determined by the decimal value of the binary input. The number of classes increases exponentially with the condition length — $num_{classes} = 2^\ell$. The best action map consists of all possible binary inputs $|[B]| = 2^\ell$ with their corresponding decimal value as output. The complete action map adds ℓ consistently incorrect rules per each consistently correct rule of the best action map. Thus, $|[O]| = 2^\ell \cdot (\ell + 1)$.

A.3 Position

Position is an imbalanced multiclass problem defined as follows. Given a binary-input instance of length ℓ, the output is the position of the left-most one-valued bit.

The best action map consists of $\ell + 1$ rules with different levels of generalization. The complete action map needs to maitain a set of wrong-labeled rules. The size of this set depends on the level of generalization of each class. Table 3 shows the best and the complete action map for position with $\ell = 6$.

A.4 Multiplexer

The multiplexer problem is one of the most used benchmarks in accuracy-based learning classifier systems [22]. The multiplexer is defined for binary strings of size ℓ, where $\ell = k + 2^k$. The first k bits of the conditions are the *position bits*. The output of the multiplexer is the value of the bit referred by the position bits.

Table 3. Best action map (first column) and complete action map (all columns) of position with ℓ=6

000000:0	1#####:0	#1####:0	##1###:0	###1##:0	####1#:0	#####1:0
000001:1	1#####:1	#1####:1	##1###:1	###1##:1	####1#:1	#####0:1
00001#:2	1#####:2	#1####:2	##1###:2	###1##:2	####1#:2	
0001##:3	1#####:3	#1####:3	##1###:3	###0##:3		
001###:4	1#####:4	#1####:4	##0###:4			
01####:5	1#####:5	#0####:5				
1#####:6	0#####:6					

Imbalanced Multiplexer. The imbalanced multiplexer was introduced in [17] to analyze the effects of undersampled classes in XCS. Departing from the original multiplexer problem, the imbalanced multiplexer undersamples one of the classes —labeled as the minority class— in the following way. When required, a new input example is selected randomly. If the example belongs to the class labeled as the majority class, it is given to the system. Otherwise, it is accepted with probability P_{accept}. If it is discarded, a new input example is chosen, which undergoes the same process. Regarding the notation used in the paper, $P_{accept} = \frac{1}{2^i}$.

Multiplexer with Alternating Noise. The multiplexer with alternating noise was firstly used in [5] to show that XCS with tournament selection is able to handle data sets with inconsistent data. The problem is as follows. When a new input instance corresponding to the multiplexer problem is sampled, its action is flipped with probability P_x.