Accuracy-Based Learning Classifier Systems: Models, Analysis and Applications to Classification Tasks

Ester Bernadó-Mansilla esterb@salleurl.edu
Enginyeria i Arquitectura La Salle, Ramon Llull University, Passeig Bonanova, 8. 08022 Barcelona, Spain
Now at: Computing Sciences, Bell Laboratories, Lucent Technologies, 600-700 Mountain Avenue, Murray Hill, NJ 07974-0636, USA

Josep M. Garrell-Guiu josepmg@salleurl.edu
Enginyeria i Arquitectura La Salle, Ramon Llull University, Passeig Bonanova, 8. 08022 Barcelona, Spain

Abstract

Recently, Learning Classifier Systems (LCS) and particularly XCS have arisen as promising methods for classification tasks and data mining. This paper investigates two models of accuracy-based learning classifier systems on different types of classification problems. Departing from XCS, we analyze the evolution of a complete action map as a knowledge representation. We propose an alternative, UCS, which evolves a best action map more efficiently. We also investigate how the fitness pressure guides the search towards accurate classifiers. While XCS bases fitness on a reinforcement learning scheme, UCS defines fitness from a supervised learning scheme. We find significant differences in how the fitness pressure leads towards accuracy, and suggest the use of a supervised approach specially for multi-class problems and problems with unbalanced classes. We also investigate the complexity factors which arise in each type of accuracy-based LCS. We provide a model on the learning complexity of LCS which is based on the representative examples given to the system. The results and observations are also extended to a set of real world classification problems, where accuracy-based LCS are shown to perform competitively with respect to other learning algorithms. The work presents an extended analysis of accuracy-based LCS, gives insight into the understanding of the LCS dynamics, and suggests open issues for further improvement of LCS on classification tasks.

Keywords

Learning classifier systems, accuracy-based fitness, knowledge representation, learning complexity, generalization, data mining.

1 Introduction

Since the introduction of Learning Classifier Systems (LCS) (Holland, 1975) there have been a number of investigations on their architecture and performance. Recently, XCS (Wilson, 1995; Wilson, 1998) has become one of the main representatives of LCS. It is the first learning classifier system where accurate and maximal generalizations are reported. Its success is due to two main changes to the LCS architecture: a fitness based on accuracy and a niche Genetic Algorithm (GA). XCS's ability to evolve accurate generalizations together with the proposal of new representations that are well suited to real world problems (Wilson, 1999; Saxon and Barry, 2000) have raised interest in applications of LCS to increasingly difficult single-step problems like data mining (Wilson, 2001; Bernadó-Mansilla et al., 2002). Generalization is a key point in classification tasks, pattern recognition and data mining in general. In this type of problem, the learner induces a hypothesis from a set of training data samples. This process can also be described as the generalization of the training examples. The extracted model can be applied to the prediction of new instances not previously seen by the system. Generalization also implies a knowledge representation that is more compact than the original training set.

The generalization ability of traditional LCS has not really been demonstrated as of yet, except for some limited examples where a bias towards general rules has been introduced (Wilson, 1987; Holmes, 1997; Bernadó-Mansilla and Garrell-Guiu, 2001). The lack of accurate generalizations in traditional LCS has been mainly attributed to the fact that fitness is based on *strength* (Wilson, 1995), which is a prediction of the reward received from the environment. In addition, traditional LCS only evolve a map of the best rewarded rules (called a *best action map*). On the contrary, XCS evolves a *complete action map*, which contains all the accurate rules, either high-rewarded or low-rewarded, as a consequence of basing fitness on accuracy rather than on prediction. A best action map is a partial map, which is at most n times smaller than its corresponding complete action map, where n is the number of available actions (or classes).

With the application of LCS to real world problems, a set of complexity factors may arise. These include large search spaces, high dimensional feature spaces, high number of classes, presence of noise, etc. When both the search space and the number of classes increase, it is possible to consider the evolution of a best action map instead of a complete action map. There has been a debate over the advantages and disadvantages of best action maps and complete action maps (Wilson, 1995; Kovacs, 2000; Kovacs, 2001), but experimental and theoretical studies are limited. Best action maps offer a more compact knowledge representation and could be more efficient in large search spaces. Nevertheless, they have been associated with strength-based classifier systems and thus, with poor performance.

This paper presents an alternative to XCS that, being based on accuracy, evolves a best action map. The system, called UCS (Bernadó-Mansilla, 2002), is close to XCS in many respects but changes the way in which accuracy –and thus fitness– is computed. The similarities between XCS and UCS allow us to isolate the fitness computation from other components of the system. This leads to the investigation of several key aspects of LCS in the framework of classification problems, such as the evolution of best action maps versus complete action maps. The experiments reported in this paper demonstrate that generalization occurs effectively in best action maps. Moreover, best action maps evolve more compact knowledge representations and can converge earlier in large search spaces. In LCS, the GA search towards accurate rules is guided by fitness. Thus, computing fitness in different ways may have different consequences on the GA search. The paper investigates how the fitness pressure operates in each case, and suggests what type of accuracy computation may be best suited depending on the characteristics of the problem. We also investigate what complexity factors arise in each classifier system. The results reveal a strong correlation between the learning complexity and the frequency with which representative examples are provided to the system. Based on these results, we propose a model on the learning complexity of LCS which enhances the studies reported in the literature (Kovacs and Kerber, 2001). The results are further extended to a varied set of real world classification problems, where we compare the LCS performance with other non-evolutionary classifier schemes. The different conclusions obtained from this series of studies lead us to a better understanding of the LCS dynamics and the role of some components of LCS.

The rest of this paper is structured as follows. Section 2 reviews the different architectures of LCS and motivates the design of UCS. Next, we briefly describe the XCS classifier system and the principal changes we have made to obtain UCS. Section 4 analyzes the performance of both systems and discusses their differences through the type of covering map evolved by each system. Section 5 revises the results, giving insight into the fitness pressure that operates in each system. Section 6 investigates the learning complexity of LCS. Next, the accuracy-based LCS are applied to a large set of real world classification problems. Finally, we summarize the paper, give our main conclusions and suggest directions for future work.

2 Models of Learning Classifier Systems

2.1 Strength-Based and Accuracy-Based Learning Classifier Systems

LCS learn a set of rules by means of the interaction with the environment, which is usually performed through a *reinforcement learning* scheme (Sutton and Barto, 1998). Whenever an input is presented to the system, the rules compete with each other to be able to perform their actions and get a reward from the environment. This reward is then used by the system to update the quality of the rules. The discovery component, usually a Genetic Algorithm (GA), is triggered eventually to improve the ruleset. The GA bases its search on the fitness of each rule, which is computed from the quality of the rules.

LCS can be classified in two main categories, depending on how fitness is defined: (1) *strength-based* classifier systems and (2) *accuracy-based* classifier systems. In strength-based classifier systems (also called traditional LCS), fitness is based on strength, which is an estimate of the reward that the rule will receive from the environment[1]. Therefore, the GA searches for the best rewarded rules. In accuracy-based LCS, fitness is based on the accuracy of the prediction, rather than on the prediction itself. This means that the GA searches for rules that are accurate in their prediction, independently from their prediction value, which can be high or low. As a consequence of these two different views of fitness, the knowledge representation that is evolved in each case is different. Strength-based classifier systems evolve a *best action map*, i.e., a map that contains only the high-rewarded rules. On the contrary, accuracy-based classifier systems evolve a *complete action map*, which consists of all accurate rules belonging to the different payoff levels defined by the environment.

The analysis of strength-based and accuracy-based classifier systems reveals that the number of payoff levels defined by the environment is a key point (Kovacs, 2000; Kovacs, 2001). Multiple payoff levels often appear in multi-step environments (as suggested in (Kovacs, 2002), all non-trivial multi-step environments have multiple payoff levels). In multi-step environments, the system has to learn a sequence of actions and the reward usually comes at the end of the sequence. This reward is passed back to the sequence of activated rules multiplied by a discount factor, defining thus different payoff levels which must be covered in order to have a complete description of the problem. Strength-based LCS tend to perform poorly in the presence of multiple payoff levels. One of the identified problems is the *greedy classifier allocation* (Cliff and

[1]Some strength-based systems use *sharing* (Horn et al., 1994): the reward is divided among the active classifiers. In this case, the individual strength does not predict the reward received by environment. Instead, the total amount of the shared strength in a niche predicts reward (see (Wilson, 1995) for a discussion).

Ross, 1995), which assigns higher reproductive opportunities to classifiers with higher reward (recall that fitness is based on strength), probably causing gaps in the covering map (specially in those states with the lowest rewards). The interaction of this effect with overgeneral classifiers, termed as the problem of *strong overgenerals* (Kovacs, 2001), may cause that overgeneral classifiers are reproduced more often than other reliable classifiers covering low-rewarded states. As a consequence, overgeneral classifiers may displace other reliable classifiers, and the performance of the system can be dramatically worsened. Accuracy-based classifier systems avoid these effects by basing fitness on the accuracy of the reward prediction.

In single-step problems like classification, only two payoff levels are usually defined by the reward function. A maximum reward R is returned when the system predicts the correct class, and a minimum reward (usually zero) otherwise. Such a reward function is called *unbiased*. In this case, the rules that always receive the highest reward are termed *consistently correct rules*, because they predict the correct class in all the states that they match. On the other hand, rules with zero reward correspond to *consistently incorrect rules*, i.e., rules that predict the incorrect class in all the states that they match. A best action map contains only the consistently correct rules, whereas a complete action map contains both the consistently correct and consistently incorrect rules. In classification problems, only the consistently correct rules are necessary to describe the problem. Therefore, the advantages of evolving a complete action map are still unclear. The next section discusses these issues and motivates the design of a new classifier system to clarify them.

2.2 Complete Action Maps and Best Action Maps

Since our interest is to study LCS on classification problems, we restrict the discussion of complete action maps and best action maps to the framework of single-step problems with unbiased reward functions. Some of the main questions that arise in this framework are: why should the system evolve a complete action map if some of its rules (the consistently incorrect rules) are not used for classification? Shouldn't the evolution of a best action map be more efficient? To answer these questions, we review different key aspects where a complete map and a best action map may differ.

Exploration. The evolution of a complete action map requires greater exploration of the search space than a best action map. A complete action map explores all the consistently correct and the consistently incorrect rules, while a best action map only searches for the former ones. This problem can be augmented if the search space is large, e.g. with high dimensionality or with high number of possible classes. As Wilson (1995) mentions, in these cases large regions of the complete action map are unremunerative (from the point of view of a classification problem). However, he mentions that strength-based classifier systems are not a feasible alternative because they are unable to generalize and they converge towards suboptimal solutions. Kovacs (2000) argues that maintaining the consistently incorrect rules by means of a complete action map may help the exploration process. If the system codifies and remembers the incorrect rules it can avoid exploring them repetitively. Keeping track of incorrect rules may also improve the exploit performance since these rules tell the system what not to do.

Population size. The size of a complete action map is greater than a best action map. In a categorization problem with n classes, a complete action map can be as much as n times larger than the best action map. Usually the population size required by the learning classifier system is proportional to the size of the covering map. Then, evolving a complete action map requires higher population sizes and consequently, more computational resources than a best action map (Kovacs, 2000).

Complexity. The size of the optimal population has been determined as a factor of complexity for LCS (Kovacs and Kerber, 2001). The larger the optimal population, the greater the number of cycles required for learning. Therefore, the evolution of complete action maps may last longer than best action maps.

Generalization. The evolution of best action maps might be erroneously associated with poor generalization, due to the fact that some strength-based systems evolving best action maps are unable to generalize. But the lack of generalization should be attributed to the learner rather than to the knowledge representation. In fact, LCS may be able to generalize only over the states of maximum payoff, while evolving a best action map. An example of generalization in a strength-based system can be found in (Kovacs, 2002).

Changing environment. Hartley (1999) demonstrates the advantages of evolving complete action maps when the environment changes abruptly. He compares an accuracy-based classifier system (XCS) with a strength-based classifier system (NEWBOOLE). He shows that the complete action map evolved by XCS allows the system to recover faster from an abrupt change of the environment than the best action map evolved by NEWBOOLE. Nevertheless, his experiment is limited to a binary classification problem, where the change does not affect the rules but only their prediction values. That is why the evolution of a complete map allows a faster recovery: XCS has to relearn only the quality of the rules, while NEWBOOLE has to rediscover the rules. When the changes of the environment alter the underlying rules, the evolution of complete action maps may have no benefits over best action maps, as noted in (Bull et al., 2002).

Although some of these aspects have been discussed elsewhere (Wilson, 1995; Kovacs, 2000; Kovacs and Kerber, 2001) there is not any comparative study supporting these hypotheses. Moreover, the comparison between an accuracy-based classifier system and a strength-based classifier system might be limited by the lack of generalization of the latter. This paper introduces UCS, an accuracy-based classifier system close to XCS that evolves a best action map. The system keeps the same generalization mechanisms as XCS, so it is expected to not suffer from lack of generalization. By the comparison of XCS with UCS, we provide insight into the aspects mentioned previously as well as other related aspects. We restrict our analysis to stationary classification problems, leaving the study on non-stationary environments for future work.

3 Description of the Systems

3.1 XCS

In this section, we give a brief description of XCS, restricted to single-step problems, which is the scope of this paper. For more details, the reader is referred to (Wilson, 1995; Wilson, 1998; Butz and Wilson, 2001).

Representation

XCS evolves a population [P] of classifiers, where each classifier has a rule and a set of associated parameters estimating the quality of the rule. Each rule consists of a condition and an action (or class[2]): $condition \rightarrow action$. For binary inputs, the condition

[2]We will use the terms *action* and *class* indistinguishably.

is usually represented in the ternary alphabet: $\{0, 1, \#\}^l$, where l is the length of the input string. The symbol #, called *don't care*, allows us to express generalizations in the rule's condition. The action is codified as an integer.

Each classifier has three main parameters: a) the payoff prediction p, an estimate of the payoff that the classifier will receive if its action is selected, b) the prediction error ϵ, which estimates the error between the classifier's prediction and the received payoff and c) the fitness F, computed as an inverse function of the prediction error.

Performance Component

At each time step, an input example x is presented and a match set [M] is built, consisting of those classifiers whose conditions are satisfied by the example. If the match set is empty, or the number of covered actions is less than the parameter θ_{mna}, covering occurs. The covering operator creates a new classifier with a condition matching the current example and an action that is chosen randomly from those not represented in [M]. Once the match set is obtained, a payoff prediction $P(a)$ is computed for each action a available in [M]. It is computed as a fitness weighted average of the predictions of those classifiers advocating a. The winning action can be selected from a variety of regimes, ranging from the *pure-explore mode* (random action) to the *pure-exploit mode* (the action with the highest prediction). Under classification problems, pure-explore is used during training, while pure-exploit is used when the system is predicting new unseen examples. The chosen action specifies the action set [A], formed by all the classifiers in [M] advocating this action.

Update

Once the action is selected, the environment returns a reward r, which is used to adjust the parameters of the classifiers in [A]. First, the prediction p is adjusted: $p \leftarrow p + \beta(r - p)$, where β $(0 \leq \beta \leq 1)$ is the learning rate. Next, the prediction error ϵ is updated: $\epsilon \leftarrow \epsilon + \beta(|r - p| - \epsilon)$. Then, the classifier's accuracy is computed as an inverse function of the classifier's error: $k = \alpha(\epsilon/\epsilon_0)^{-\nu}$ for $\epsilon \geq \epsilon_0$, otherwise 1. The parameter ϵ_0 $(\epsilon_0 > 0)$ determines the threshold error under which a classifier is considered to be accurate. The parameters α $(0 < \alpha < 1)$ and ν $(\nu > 0)$ control the degree of decline in accuracy if the classifier is inaccurate (Butz et al., 2001). Then, the relative accuracy k' is computed by dividing the accuracy k by the total amount of accuracies in the action set. Finally, the fitness F is updated according to the classifier's relative accuracy: $F \leftarrow F + \beta(k' - F)$. Note that basing fitness on the relative accuracies provides sharing (Horn et al., 1994) among the classifiers belonging to the same action set. Each classifier also maintains an experience parameter exp, which is increased by one each time the classifier's parameters are updated.

Discovery Component

The GA in XCS is applied to the action sets, rather than panmictically (i.e., over all the population). First, it selects two parents from the actual [A] with probability proportional to fitness. Then, the parents are crossed and mutated with probabilities χ and μ respectively.

The resulting offspring are introduced into the population. First, each offspring is checked for subsumption (Wilson, 1998) with its parents. If either of the parents is sufficiently experienced, accurate and more general than the offspring, then the offspring is not introduced and its parent's numerosity is increased. This process is called *GA subsumption*. If the offspring classifier can not be subsumed, it is inserted in the population, deleting another classifier if the population is full (Kovacs, 1999). The deletion probability of a classifier is proportional to the average of the size of the action sets in which it has participated (stored in the parameter as). Also, if the classifier is sufficiently experienced and its fitness is low, its deletion probability is higher.

Specify

The specify operator (Lanzi, 1997) was designed for environments which allow few generalizations, where XCS's generalization tendency can prevent XCS from learning correctly. The specify operator is applied to the current action set [A]. It triggers automatically when the action set is sufficiently experienced and its average error $\epsilon_{[A]}$ exceeds a certain threshold, defined as twice the average population error $\epsilon_{[P]}$. Then, it selects a rule from [A] with probability proportional to the error, and specifies some of its don't care bits according to the input example.

3.2 UCS

UCS[3] is specifically designed for supervised environments, so many of the design criteria are optimized for this framework. It keeps the principal features of XCS, a fitness based on accuracy and a niche GA, but changes the way in which accuracy is computed. This section describes UCS, emphasizing the main differences with XCS.

Representation

UCS evolves a population [P] of classifiers, where each classifier has a rule, represented as in XCS, and a set of parameters. Two main parameters describe the quality of the rule: a) *acc*, the accuracy of the rule, which is an estimate of the probability that the rule predicts the correct class, and b) the fitness F, which is based on the accuracy *acc*.

The rest of the classifier's parameters are inherited from XCS. These are: the numerosity (*num*), which is the number of copies of the rule, the niche set size (ns), which is based on the parameter action set size (as) from XCS, and the experience (exp).

Performance Component

In UCS, learning is performed using a supervised learning scheme, which means that the input example comes along with the associated class: $x : a$. This differs from XCS where, according to a reinforcement learning scheme, the input example x is presented, the system responds with an action and the environment returns a reward. In classification problems, this reward is maximum if the action proposed by the system is correct and zero otherwise. Observe that XCS is a reinforcement learner, while UCS is a supervised learner. In classification problems, UCS benefits from a supervised scheme as it is described in the following.

During learning, an input example x with the associated class a is presented. From x, a match set [M] is formed, consisting of those classifiers whose condition matches x. Those classifiers in [M] which predict the correct known class a form the *correct set* [C]. The rest of the classifiers in [M] belong to the *incorrect set* labeled [!C]. Covering is applied in UCS when the correct set [C] is empty. In this case, a classifier covering the current input is created, with the same class a provided with the input.

In exploit or test mode, an input x is presented and the system has to predict its associated classification. In UCS, once the match set [M] is formed, the best action is selected from the vote (weighted by fitness) of all classifiers in [M].

[3]UCS stands for sUpervised Classifier System.

Update

The classifier's parameters in UCS are updated in the following way. The classifier's accuracy is computed as the proportion of correct classifications with respect to the number of matches:

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

This value is updated each time a classifier belongs to a match set, and thus it is an average over all the examples that the classifier has matched. The fitness is computed as a function of accuracy:

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

where ν is a constant. If we compare this fitness computation with that of XCS, two key points can be highlighted. First, in UCS, the accuracy parameter *acc* directly estimates the accuracy rate of each classifier. Second, UCS does not perform any fitness sharing. Although this could be incorporated, we have preferred to start with a simpler approach in order to understand the system's behavior fully.

Since the classifier's parameters are updated in the match sets, the classifier's experience exp is increased by one every time that the classifier participates in a match set. The parameter ns is the average size of the correct sets where the classifier participates. It is updated whenever the classifier belongs to a correct set.

Discovery Component

The GA in UCS is also inherited from XCS, but it is applied to the correct set [C] only. It selects two classifiers from [C] with probability proportional to fitness and applies crossover and mutation. The resulting offspring are inserted in the population. Subsumption is also included in UCS in the same way as in XCS. The only difference is that a classifier is considered to be accurate when its accuracy *acc* is greater than a threshold $acc0$. The deletion algorithm operates similarly to XCS. The specify operator is incorporated in UCS as well.

Summary of UCS

UCS maintains the same structure as XCS, but uses a different accuracy computation. It inherits the generalization algorithms from XCS, which are mainly based on the fact that the GA is applied on niches rather than on the whole population. Niches in UCS are defined by the correct rule sets [C], and therefore it is expected that UCS will generalize over the search space of correct rule sets, leaving the incorrect rules out of exploration.

UCS also shares some features with other LCS. The way in which UCS divides the classifiers in [M] into the correct set [C] and the incorrect set [!C] resembles NEWBOOLE (Bonelli et al., 1990). The accuracy computation is equivalent to Frey&Slate's classifier system (Frey and Slate, 1991). Both systems were also designed for supervised problems.

4 Learning Performance and Evolution of Covering Maps in XCS and UCS

4.1 Methodology

Our aim is to study the performance of accuracy-based classifier systems when they are applied to difficult classification problems. We begin our study with three artificial problems: a) a binary class problem, b) a multiclass problem and c) a multiclass problem with different proportions of examples per class. Besides being representative of three types of real world classification problems, this test bed allows us to emphasize the differences between the two types of accuracy-based classifier systems. For the moment, we restrict our analysis to problems with binary attributes.

The choice of a good metric of performance is vital to the analysis of our results. In the literature, the percentage of accurate classifications has been widely used. Nevertheless, this metric is poor when we study problems with unequal distribution of examples per class, since the results seem better if the system covers mainly the majority class. Some authors (Kovacs and Kerber, 2001) have used the percentage of the achieved optimal population, or percentage of covering map, denoted by %[O]. This is more related to our objective, because we want to analyze how XCS and UCS evolve their respective covering maps. Nevertheless, since the optimal populations are different in XCS and UCS, we restrict our metric to the percentage of the best action map covered by each system, which we denote as %[B]. This metric is an indication of the number of "useful" rules that are evolved in each system.

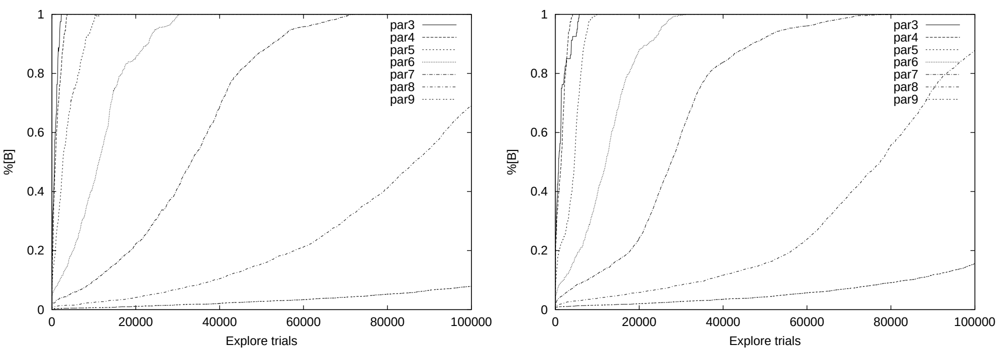
The parameter settings in XCS are fixed in all the problems, except for the population size which is set to $N = 25 \cdot |[O]|$. The parameters are set as follows (see (Butz and Wilson, 2001) for the notation): $\beta = 0.2$, $\theta_{mna} =$ number of actions, $P_{\#} = 0.33$, $\epsilon_0 = 0.001$, $\alpha = 0.1$, $\nu = 5$, $\theta_{GA} = 25$, $\chi = 0.8$, $\mu = 0.04$, $\theta_{del} = 20$, $\delta = 0.1$, $doGASubsumption = yes$, $doActionSetSubsumption = no$, $\theta_{sub} = 20$, $doSpecify = yes$, $N_{sp} = 20$, $P_{sp} = 0.5$, $reward = 1000/0$. The UCS parameters that are shared with XCS have the same values as indicated. Note that the population size is also set to $N = 25 \cdot |[O]|$, which results in a smaller population, since $|[O]|$ is smaller in UCS than in XCS. GA subsumption and the specify operator are used in XCS as well. The parameters of UCS which differ from those of XCS are: $\nu = 10$, $acc_0 = 0.99$. We used our own implementation of XCS and UCS, both codified in C++.

4.2 Analysis on a Binary Class Problem

As a binary class problem we have chosen the parity problem, which has been widely used as a benchmark for learning classifier systems (Kovacs, 1997; Kovacs and Kerber, 2001). The parity problem is defined as follows: given a binary string of fixed length, the output is one if the number of ones in the string is odd, otherwise the output is zero. The problem is denoted by *parl*, where l is the number of inputs. The problem does not allow any generalization, unless some irrelevant bits[4] are added in the input examples. To keep the study simpler, we have not added any irrelevant bits. Therefore, the optimal ruleset only consists of specific rules, of type 0010:1. The optimal population size in UCS is $|[O]|_{UCS} = 2^l$. In XCS, $|[O]|_{XCS} = 2^{l+1}$, which corresponds to twice the size of UCS's map due to the consistently incorrect rules. In order to study the scalability of XCS and UCS with respect to the problem complexity, we use a test set of parity problems ranging from three to nine input bits.

Figure 1 shows the results obtained by XCS and UCS in the parity problem. The curves show the percentage of the best action map %[B] achieved by each system along the explore trials. Curves are averaged over ten runs. Note that UCS can effectively learn the optimal set of rules corresponding to the parity problem. The convergence speed is almost the same as XCS for almost all parity problems. It seems though that for the largest problems (par8, par9) UCS's learning slope is steeper, but this might not be significant since there is high variance in these problems. From the performance

[4]An irrelevant bit has no influence on the output class. Therefore, in the ternary alphabet, the rules can generalize it with the don't care symbol '#'.



(a) XCS (b) UCS

Figure 1: Proportion of the best action map covered by XCS (left) and UCS (right) in the parity problem, ranging from three to nine input bits. Curves are averages over ten runs. The x axis depicts the number of explore trials.

Table 1: Population sizes (number of macroclassifiers) in XCS and UCS in the parity problem at iteration 100,000. The numbers are averages of ten runs.

Problem	XCS	UCS
par3	20.76	12.40
par4	49.92	33.68
par5	116.16	76.32
par6	271.04	175.84
par7	593.92	396.32
par8	2100.48	839.04
par9	2800.80	1633.28

point of view, both systems are equivalent, since they learn the best action map (the useful rules) with a similar speed. In this problem, the advantages of evolving only a best action map are not clearly seen. In fact, since XCS is learning a complete action map, which is twice the size of the best action map, XCS is internally faster, but from the user's point of view both systems are equally fast.

Table 1 shows the size of the final populations for XCS and UCS at iteration 100,000. These sizes represent the number of macroclassifiers (i.e., the number of different rules in the population), averaged over ten runs. Note that UCS always obtains less rules. This is reasonable since the optimal population in UCS only consists of the correct rules while XCS codifies both the correct and the incorrect rules. The size of the optimal population in UCS is half the size of the optimal population in XCS. That is why UCS has approximately half of the population required by XCS, as shown in Table 1.

4.3 Analysis on a Multiclass Problem

Real world classification problems are often characterized by a high number of classes (e.g., a letter recognition task). For this reason, we want to study the behavior of the accuracy-based LCS in multiclass problems: not only their performance, but also their scalability with respect to the number of classes. As the number of classes increases, the ratio between the size of the complete action map and the best action map also

Table 2: Best action map (first column) and complete action map (all columns) of dec4. The notation of each rule is: *condition : action* $\rightarrow$ *payoff*.

0000:0 $\rightarrow$ 1000	1###:0 $\rightarrow$ 0	#1##:0 $\rightarrow$ 0	##1#:0 $\rightarrow$ 0	###1:0 $\rightarrow$ 0
0001:1 $\rightarrow$ 1000	1###:1 $\rightarrow$ 0	#1##:1 $\rightarrow$ 0	##1#:1 $\rightarrow$ 0	###0:1 $\rightarrow$ 0
0010:2 $\rightarrow$ 1000	1###:2 $\rightarrow$ 0	#1##:2 $\rightarrow$ 0	##0#:2 $\rightarrow$ 0	###1:2 $\rightarrow$ 0
0011:3 $\rightarrow$ 1000	1###:3 $\rightarrow$ 0	#1##:3 $\rightarrow$ 0	##0#:3 $\rightarrow$ 0	###0:3 $\rightarrow$ 0
0100:4 $\rightarrow$ 1000	1###:4 $\rightarrow$ 0	#0##:4 $\rightarrow$ 0	##1#:4 $\rightarrow$ 0	###1:4 $\rightarrow$ 0
0101:5 $\rightarrow$ 1000	1###:5 $\rightarrow$ 0	#0##:5 $\rightarrow$ 0	##1#:5 $\rightarrow$ 0	###0:5 $\rightarrow$ 0
0110:6 $\rightarrow$ 1000	1###:6 $\rightarrow$ 0	#0##:6 $\rightarrow$ 0	##0#:6 $\rightarrow$ 0	###1:6 $\rightarrow$ 0
0111:7 $\rightarrow$ 1000	1###:7 $\rightarrow$ 0	#0##:7 $\rightarrow$ 0	##0#:7 $\rightarrow$ 0	###0:7 $\rightarrow$ 0
1000:8 $\rightarrow$ 1000	0###:8 $\rightarrow$ 0	#1##:8 $\rightarrow$ 0	##1#:8 $\rightarrow$ 0	###1:8 $\rightarrow$ 0
...	...	...	...	...
1111:15 $\rightarrow$ 1000	0###:15 $\rightarrow$ 0	#0##:15 $\rightarrow$ 0	##0#:15 $\rightarrow$ 0	###0:15 $\rightarrow$ 0

increases. If this ratio is very high, the evolution of a best action map may have some advantages with respect to a complete action map.

We use a multiclass problem which allows us to increase the number of classes gradually. The problem, denoted as *decl*, consists of the n-ary decoding of a binary string of length l:

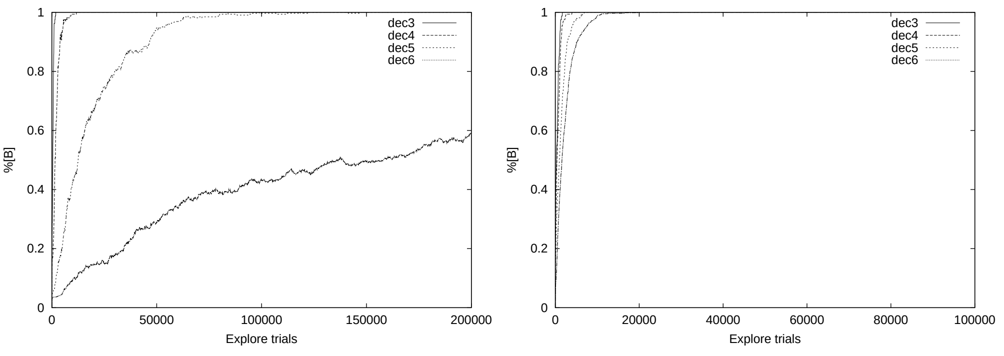
$$f_{dec}(x) = \sum_{i=1}^{l} x_i \cdot 2^{i-1} \tag{3}$$

where x_i is the ith input bit (starting from the rightmost bit towards the leftmost bit) and $f_{dec}(x)$ is the output class. The number of classes n is 2^l. Similarly to the parity problem, the decoder function does not allow any generalization from the input examples (to be exact, in the consistently correct rules). Table 2 shows the best action map (only the first column) and the complete action map (all columns) corresponding to the dec4 problem. Observe that the best action map consists of one rule per class. Therefore $\|[O]\|_{UCS} = n = 2^l$. The complete action map adds l consistently incorrect rules for each class. Thus, $\|[O]\|_{XCS} = n \cdot (l+1) = 2^l \cdot (l+1)$.

The experiments were performed on a series of decoder problems ranging from three to six input bits. Figure 2 shows the results obtained by XCS and UCS. Note that UCS achieves a faster convergence than XCS in the decoder problem, which is especially evident in the largest problems (i.e., dec5, dec6). This behavior may be associated with the increasing ratio between the size of the complete action map and the best action map. Note that this ratio is: $r_{decl} = l + 1$, which, in this series of problems, corresponds to: $r_{dec3-dec6} = \{4, 5, 6, 7\}$. If we compare the convergence time in XCS with respect to UCS, we observe that the convergence ratio increases much faster than the map size ratio. This might indicate the presence of other complexity metrics operating in this problem besides the map size ratio.

Let's analyze this observation by a closer examination of the type of complete map corresponding to the decoder problem. Looking again at Table 2, we observe the following particularity: the incorrect rules have greater generalization than the correct rules. The system can generalize over the states with zero payoff, but not over the states with maximum payoff. The presence of so many general rules in the population may lead to an excessive pressure towards generalization and consequently, the correct rules –which must be specific– may be difficult to discover.

In order to study this hypothesis, we examine the population that XCS has evolved



(a) XCS (b) UCS

Figure 2: Proportion of the best action map covered by XCS (left) and UCS (right) in the decoder problem. Curves are averages over ten runs. The x axis depicts the number of explore trials (note that the two plots are scaled differently in order to facilitate the analysis).

in a certain stage of learning. Table 3 plots the population predicting class one, evolved by XCS in the dec6 problem at iteration 100,000. Rules are sorted by their prediction values. Observe that the consistently incorrect rules are all evolved (rules 1 to 6, with prediction zero). Nevertheless, the consistently correct rule corresponding to class one 000001:1 is not evolved yet. Instead of the correct rule there are some overgeneral rules trying to predict accurately class number one. These rules range from the most overgeneral rule ######:1 (rule 7) to near-optimal rules like 00000#:1 (rule 13). Notice that the most overgeneral rule has a high numerosity. This type of rule can be produced by the influence of the other incorrect rules in the population. The Hamming distance between either of the six incorrect rules and the overgeneral rule ######:1 is only one. Thus, the probability that a recombination or mutation of the incorrect rules produces this overgeneral rule is very high. This indicates that the incorrect rules with high generalization tend to produce an additional pressure towards generalization, resulting in the creation of overgeneral rules.

Once some overgeneral rules are discovered, how can the system evolve the correct specific rules? Let's analyze how XCS can discover the rule 000001:1 given the population depicted in Table 3. Assume that the input 000001 is presented to the system and an action set $[A]_1$ is formed by all the classifiers that advocate class one. There are three main ways of creating a new rule, the covering operator, the genetic algorithm and the specify operator:

- The covering operator will not occur in this case, since the input example is covered by several rules, i.e., the action set $[A]_1$ is not empty.
- The genetic algorithm is unlikely to produce such a rule. The selection mechanism is based on fitness, and fitness is an inverse function of the prediction error. Observe that as a rule becomes more specific (see rules 7 to 13) its prediction increases, but also its error. Therefore, the most general rules have greater probabilities of being selected, leading to offspring which are likely to be general too. The way in which fitness is distributed over these rules does not favor that the GA and thus

Table 3: Population predicting class number one in dec6, evolved by XCS after 100,000 explore trials. Columns show respectively the rule's number, the rule (condition:class) and its parameters: prediction, error, fitness, action set size estimate, experience, numerosity and generality, measured as the percentage of don't cares in the rule's condition.

id	rule	pred	error	F	as	exp	num	gen
1	1#####:1	0.000	0.000	0.218	97.558	644	16	0.833
2	#1####:1	0.000	0.000	0.366	97.132	568	26	0.833
3	##1###:1	0.000	0.000	0.517	91.695	754	32	0.833
4	###1##:1	0.000	0.000	0.254	100.922	418	20	0.833
5	####1#:1	0.000	0.000	0.369	101.108	736	25	0.833
6	#####0:1	0.000	0.000	0.368	103.121	737	32	0.833
7	######:1	208.796	0.235	0.181	67.909	43	13	1.000
8	####0#:1	226.844	0.281	0.019	64.313	14	3	0.833
9	###0##:1	233.554	0.284	0.009	59.430	18	1	0.833
10	#####1:1	241.943	0.294	0.009	55.050	14	1	0.833
11	#00###:1	286.999	0.346	0.004	52.978	30	1	0.667
12	000###:1	351.674	0.412	0.004	38.929	15	2	0.500
13	00000#:1	716.815	0.485	0.006	27.934	22	2	0.167

the search are guided towards accuracy. This problem has been addressed elsewhere (Butz et al., 2001) and has been termed the *schema challenge*. Section 5 gives more insight into this problem and analyzes how UCS overcomes it.

- The most probable way of obtaining the desired specific rule is by means of the specify operator. The specify operator (Lanzi, 1997) selects a rule with probability proportional to the prediction error and specifies some of its don't care bits, according to the input example. In this case, rules closer to accuracy (more specific) are more likely to be selected, and thus the consistently correct rule is easier to obtain. Nevertheless, the discovery of the consistently correct rule depends on the presence of rules closer to it. But the deletion mechanism might interfere with such a pressure, because it deletes rules with probability inversely proportional to fitness (Kovacs, 1999). So, rules closer to accuracy are more likely to be deleted. Therefore, the presence of these quasi-specific rules is spurious in the population, and it can be difficult for XCS to evolve the consistently correct rules.

In summary, the slower convergence of XCS with respect to UCS in the decoder problem may be attributed to a combination of several factors: a) the size of the complete action map with respect to the best action map, b) the presence of consistently incorrect rules with high generality that tend to produce generalization pressure and c) the absence of effective fitness pressure which does not guide the search towards accurate rules. This generalization pressure is counterbalanced by the specify operator. Note that the parity problem is much easier for XCS to learn. In fact, in the parity problem, both the correct and the incorrect rules are specific and the fitness pressure is not biased towards generalization in the same way as in the decoder. Section 5 expands on these observations by analyzing how the fitness pressure works in XCS and UCS.

4.4 Analysis on an Unbalanced Multiclass Problem

Most of the studies of LCS in single-step problems have been conducted on problems with the same proportion of examples per class (e.g., the multiplexer problem, the par-

Table 4: Best action map (first column) and complete action map (all columns) of pos6. The column labeled → *1000* plots the rules with payoff 1000, and columns labeled → *0* plot the rules with payoff zero.

→ 1000	→ 0	→ 0	→ 0	→ 0	→ 0	→ 0
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	####0#: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					

ity problem, etc.). But this feature is not necessarily present in most real world problems: sometimes, the occurrence of examples in a certain class is much higher than in another class. In this section, we study the behavior of LCS when the problem has unequal distribution of examples per class. This type of problem is expected to produce a series of rules with different levels of generality, which allows us to study the generalization mechanisms more deeply.

We have designed a problem with several classes and different distribution of examples per class, which we call *position* and denote as *posl*. Given a binary input string $(x_l, x_{l-1}, ...x_1)$ of fixed length l, the output class corresponds to the position of the leftmost bit with value one. If there is not any one in the input string, then the output class is zero. Table 4, column one, describes the optimal correct ruleset for the pos6 problem. The rest of the columns belong to the incorrect rules. Notice that for l input bits, there are $l + 1$ different classes and the optimal correct ruleset consists of just $l + 1$ rules (one rule per class). Regarding the optimal correct ruleset, it can be emphasized that the most general rule covers half of the examples, while the most specific rule covers only one example. The experiments performed on the position problem range from three input bits to nine input bits.

Figure 3 shows the results of XCS and UCS in the position problem. Observe that again UCS scales better than XCS. In this problem, the ratio between the size of the complete action map and the best action map increases gradually, as follows: $r_{pos3-pos9} = \{3.5, 4, 4.5, 5, 5.5, 6, 6.5\}$. However, the convergence ratio between XCS and UCS increases in a much higher proportion, as observed in the decoder problem.

The curves of XCS in the largest problems (pos6-pos9) show that XCS is learning fast during the first iterations (i.e., first 50,000 explore trials), but after that the learning curve improves very slightly. This suggests that some rules are difficult to obtain and these are probably the most specific rules. Specific rules activate less often and thus they have less opportunities to reproduce. In a problem with rules belonging to different levels of generality, the system has more opportunities to learn the general rules than the specific ones. This is even more problematic if we increase the number of input bits. As the input bits increase, the specific rules activate even less often while the number of bits to discover in the rule's condition is higher. This behavior observed in XCS can also explain the learning curves obtained in UCS. Notice that the curve corresponding to the pos9 problem in UCS has a similar shape. The curve is steep during the first quarter of the iterations but after that it improves slowly, probably because there are still some specific rules to find.

The fact that specific rules activate less frequently influences the learning conver-

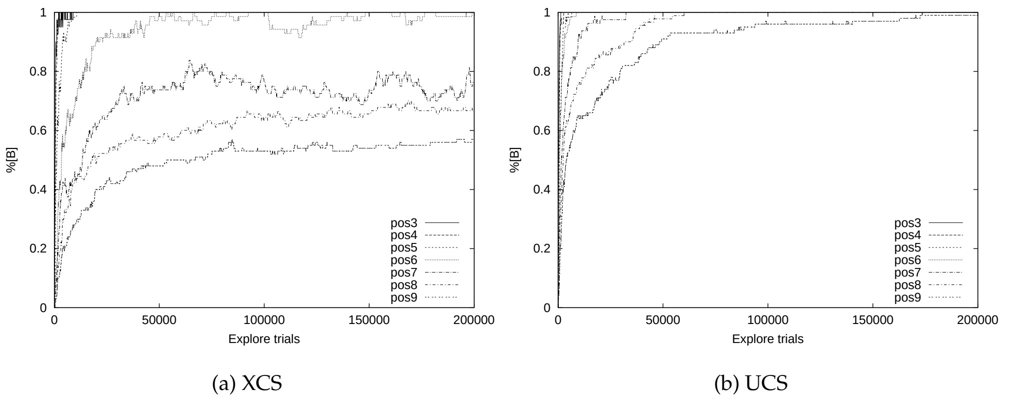


Figure 3: Proportion of the best action map covered by XCS (left) and UCS (right) in the position problem, along the explore trials. Curves are averages over ten runs.

gence, but since UCS is able to evolve such rules much faster than XCS (in a relation not explainable by the ratio between the map sizes), there must be other complexity factors for XCS. In order to study them we analyze the properties of the complete action map. Table 4 shows the complete action map belonging to the pos6 problem. The correct rules range from totally specific rules (000000:0 and 000001:1) to general rules covering half of the examples (1#####:6). For each correct rule advocating a certain class, there are a number of incorrect rules advocating erroneously the same class. These incorrect rules have a high generalization (5 bits out of 6). Note that the contents of the map corresponding to class 0 and class 1 are the same as the decoder's map. This leads us to the same conclusions as those obtained in the decoder problem: there is a generalization pressure contrary to the discovery of the most specific and accurate rules which makes learning in XCS difficult.

The decoder problem is also relevant for studying the generalization ability of UCS, which has not been shown so far. Since UCS achieves the optimal population in all cases, it can be argued that its generalization mechanism (which is inherited from XCS) also works well, even though it is exploring only a best action map. Table 5 shows the population evolved by UCS at iteration 200,000 in a run of the pos9 problem. From this table several observations can be drawn:

- The population has evolved towards the best action map, which is clearly marked by the rules with maximum fitness and high numerosity (rules 1 to 10).
- The rules are sorted in decreasing order of numerosity. Note that there is a high correlation between the numerosity of correct rules, their generality level and their experience: the more general the rule, the more experience it has and the more numerous it is. This proves that since general rules tend to participate in more correct sets, they are reproduced more often and thus have greater numerosities.
- The column labeled *tb* shows the iteration in which each rule was created. Regarding this column, the consistently correct rules, which were stable in earlier iterations, are clearly separated from the rules that the system is exploring currently. This means that the system evolves a very compact ruleset (the consistently

Table 5: Population evolved by UCS in pos9, at iteration 200,000. The classifiers are sorted by descending numerosity. Columns show the rule's number, the rule (condition:class), and the parameters: accuracy, fitness, correct set size estimate, experience, the iteration where the rule is discovered (tb), numerosity and generality.

id	rule	acc	F	ns	exp	tb	num	gen
1	1########:9	1.000	1.000	40.000	99668	942	40	0.889
2	001######:7	1.000	1.000	37.200	24724	2088	37	0.667
3	01#######:8	1.000	1.000	36.000	49473	2206	36	0.778
4	0001#####:6	1.000	1.000	32.161	12358	2524	32	0.556
5	00001####:5	1.000	1.000	28.994	6206	3107	28	0.444
6	000001###:4	1.000	1.000	17.321	2834	8741	17	0.333
7	0000001##:3	1.000	1.000	15.131	1555	3465	14	0.222
8	00000001#:2	1.000	1.000	12.112	692	19203	12	0.111
9	000000000:0	1.000	1.000	11.251	214	93912	11	0.000
10	000000001:1	1.000	1.000	9.461	297	38296	9	0.000
11	0#00001##:3	0.750	0.000	29.869	1	199991	1	0.333
12	0#0001##1:4	0.000	0.000	29.508	2	199818	1	0.333
13	0#1######:7	1.000	1.000	33.823	2	199997	1	0.778
14	000######:7	0.000	0.000	29.633	5	199969	1	0.667
15	000000###:3	0.375	0.000	19.238	9	199484	1	0.333
16	00001####:1	0.000	0.000	29.905	2	199962	1	0.444
17	00001001#:2	0.000	0.000	30.374	6	199569	1	0.111
18	0001####1:5	0.000	0.000	30.308	13	199665	1	0.444
19	0010001##:8	0.000	0.000	30.907	3	199524	1	0.222
20	0011##0##:6	0.000	0.000	29.259	5	199861	1	0.444
21	01#######:0	0.000	0.000	30.436	18	199907	1	0.778
22	01#####1#:5	0.000	0.000	30.239	8	199950	1	0.667
23	010001###:4	0.000	0.000	29.508	4	199818	1	0.333
24	1####0###:6	0.000	0.000	30.370	16	199927	1	0.778

correct rules). The rest of the rules result from the continuous search process that is performed during learning.

- Another important observation can be extracted from the *tb* values of the correct rules. Observe that the most general rules are created earlier, while the most specific ones are created later. There is almost a direct relation between the generality and the discovery iteration of the rule. This relationship is studied in detail in Section 6 and leads us to propose a new complexity metric for LCS.

5 Fitness Pressure in XCS and UCS

The evolution of rules in XCS and UCS is produced by the interaction of several pressures which tend to guide the population towards accurate, maximally general classifiers. Some recent studies (Butz et al., 2001; Butz and Pelikan, 2001; Kovacs and Kerber, 2001) have noted that classifiers can be evolved from two sides: the overspecific side and the overgeneral side. Going from an overspecific classifier towards its maximally general point is explained by Wilson's generalization hypothesis (Wilson, 1995). This pressure towards generalization has been effective in our results with XCS as well as with UCS. The evolution from the overgeneral side towards accurate classifiers is mainly attributed to the *fitness pressure*: since fitness is based on accuracy, it is expected that the search will be guided towards accurate classifiers. In our results with the decoder and position problems, XCS has shown some difficulties in evolving accurate classifiers from the overgeneral side, whereas these difficulties do not seem to appear in UCS. This section analyzes how the fitness pressure operates in both systems and gives more insight into the explanation of our results.

XCS's fitness is based on accuracy, where accuracy is measured as an inverse function of the error between the classifier's prediction and the reward received by the environment. Given an unbiased R/0 reward function, the prediction error ϵ_{cl} of a classifier cl can be estimated according to (Butz et al., 2001):

$$\epsilon_{cl} = \frac{1}{R} \cdot (|R - p_{cl}| \cdot P_c(cl) + |0 - p_{cl}| \cdot (1 - P_c(cl))) \tag{4}$$

where p_{cl} is the prediction of the classifier and $P_c(cl)$ is the classifier's probability of being correct. The factor $\frac{1}{R}$ is introduced to normalize the error with respect to the maximum payoff R. Assuming a uniform distribution of correct and incorrect states, the prediction p_{cl} of a classifier can be approached as $R \cdot P_c(cl)$. Thus, the prediction error can be derived as:

$$\epsilon_{cl} = 2 \cdot (P_c(cl) - P_c(cl)^2) \tag{5}$$

This formula estimates the prediction error as a parabolic function with respect to $P_c(cl)$, with its maximum at $P_c(cl) = 0.5$, and two minima at $P_c(cl) = 0$ and $P_c(cl) = 1$.

In fact, we are interested in the variation of the error ϵ_{cl} as the classifier moves from the overgeneral side to the correct classifier. For example, let's analyze the error function in the dec6 problem, when we move from the overgeneral classifier ######:1 towards the accurate classifier 000001:1 (see Table 3). An overgeneral classifier cl with g don't care bits matches 2^g states and is correct in only one of them, resulting in $P_c(cl) = 1/2^g$. In each move from the overgeneral side to the accurate classifier, g is decreased by one and consequently $P_c(cl)$ is doubled. Thus, $P_c(cl)$ does not change linearly but exponentially. Plotting equation 5 with respect to $P_c(cl)$ in logarithmic scale gives us a closer explanation of the behavior of the error function in the decoder problem. Figure 4 plots the classifier's prediction error as a function of $P_c(cl)$ or, equivalently, as a function of the Hamming distance between the overgeneral classifier and the correct classifier. Moving along the x axis, from left to right, we are specifying bits in the direction of the correct classifier. Notice that each move from the overgeneral side towards the correct classifier produces an increase in prediction error, until the prediction error becomes 0.5. Then, the next move leads directly to the correct classifier. In terms of $P_c(cl)$, meanwhile $P_c(cl)$ is less than 0.5, a move in the accuracy direction always produces an increase in prediction error. In this case, the fitness pressure does not lead the search in the direction of accuracy.

The prediction error function in the position problem has the same behavior as in the decoder. But the prediction error function in the parity problem is different. The probability of any overgeneral classifier of being correct is 0.5. Each move from the overgeneral side to the correct classifier does not modify $P_c(cl)$ and thus the prediction error is constant ($\epsilon_{cl} = 0.5$). This means that there is not any pressure towards specific classifiers, but at least there is no implicit pressure towards generalization caused by the fitness landscape. This is one of the reasons why the parity problem is easier for XCS to learn than the decoder and the position problems.

Let's analyze how the fitness pressure operates in UCS. In UCS, fitness is also based on accuracy, however accuracy is calculated differently. Rather than considering the accuracy of the prediction, accuracy is directly measured as the percentage of correct

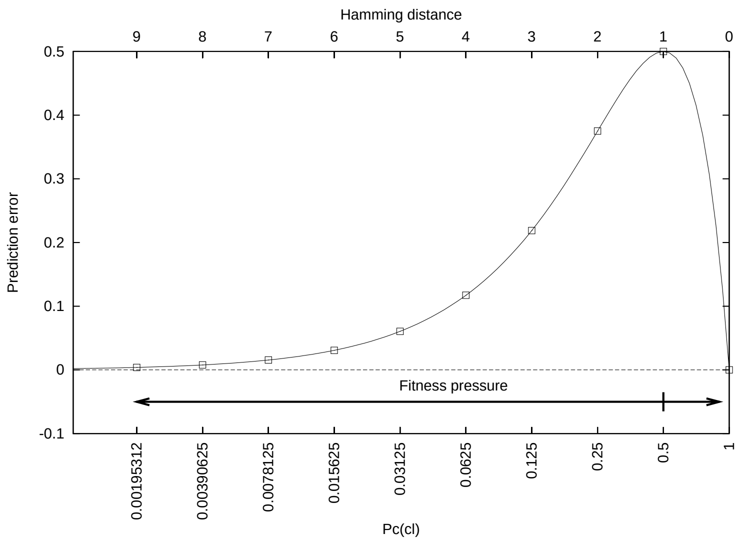


Figure 4: XCS's prediction error ϵ_{cl} in the decoder problem. The error is plotted as a function of $P_c(cl)$ –in log_2 scale– and as a function of the Hamming distance between an overgeneral classifier and an accurate classifier.

matches. This is possible because UCS uses a supervised learning approach. Thus, accuracy estimates $P_c(cl)$. In the decoder problem, it has been noted that $P_c(cl)$ increases exponentially while moving from the overgeneral side to the accuracy side. Since accuracy estimates $P_c(cl)$, and fitness is a function of accuracy, fitness also increases as we move from the overgeneral side towards accuracy. This means that UCS's fitness is producing an effective pressure towards accurate classifiers in the decoder and position problems, which explains why UCS has a faster convergence than XCS. In the parity problem, $P_c(cl)$ is constant for all the overgeneral classifiers, so in this case UCS and XCS have the same type of accuracy pressure.

The conclusions obtained in these artificial problems lead us to consider how the fitness pressure operates in real world classification problems. Given a certain problem, our aim is to predict whether there may exist a *contrary-to-accuracy* fitness landscape similar to the situation detected in the decoder and position problems. In these cases, UCS may operate as a better suited classifier system. So the crucial question is: what kind of problem characteristics lead to a contrary-to-accuracy fitness pressure? As it was observed previously, this effect arises in XCS when $P_c(cl)$ is less than 0.5, i.e., the proportion of correct states covered by the classifier is less than the proportion of incorrect states. If an overgeneral classifier has $P_c(cl) < 0.5$, it does not have any fitness guidance towards accuracy. The presence of such classifiers depends on the internal structure of the problem which is often difficult to analyze, but we hypothesize that problems with high number of classes or with unequal proportions of examples per class are the most probable to suffer from this effect. For example, assume a binary class problem, with class A covered by much fewer examples than class B (see Figure 5, where class A is represented with triangles and B with crosses). An overgeneral rule classifying class A will probably cover more examples of class B than class A. In this case, there won't be any pressure towards accuracy, until the rule covers at least the same percentage of A examples than B examples. Figure 5 represents this situation.

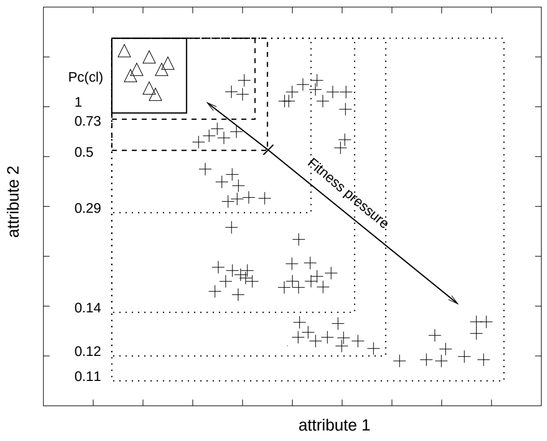


Figure 5: Distribution of examples in a binary class problem with two real attributes and different proportions of examples per class. The figure shows the coverage of some rules predicting class $\triangle$, along with their associated $P_c(cl)$ value. The overgeneral rules are drawn with dashed and dotted lines, while the maximally general rule is drawn with a solid line. XCS's fitness pressure only applies, in the direction of accuracy, to those rules with $P_c(cl) < 0.5$ (dashed lines).

Notice that the same effect may occur with multiple classes even if each class covers the same percentage of examples.

6 Complexity Metrics

One of the most interesting questions that arises in the study of any learning system is: what factors of complexity are present in a problem for the system of interest? Or equivalently, can we characterize the complexity of a problem for a certain learning system? Several studies have addressed this issue for XCS (Wilson, 1998; Kovacs and Kerber, 2001; Butz et al., 2001). Although they are limited to binary classification problems, the learning complexity has been attributed to several factors: the string length, the size of the optimal population $|[O]|$, the reward range, the mean Hamming distance among the rules in [O] and the schema challenge, which addresses the problem that fitness does not always lead towards accuracy.

This paper has also made some attempts in this direction. In fact, the learning dependence on $|[O]|$ noted in (Wilson, 1998; Kovacs and Kerber, 2001) has also been observed in our results, both in XCS and UCS: as we increase the size of the optimal population, learning becomes gradually more difficult. Besides this dimension of complexity, other complexity factors have arisen in our results with XCS, like the pressure towards generalization caused by some incorrect rules of the complete action map and the lack of effective fitness pressure, which we hypothesized to appear in problems with multiple classes or with highly unbalanced classes.

In this section, we seek for a better understanding of the learning dependency on $|[O]|$, which has been tailored as one of the main factors of complexity in LCS. We study how this dependence is reflected in the three types of problems that constitute our test bed and whether there exists any correlation of our results with respect to $|[O]|$. In order to isolate the dependence of learning on $|[O]|$ from other complexity metrics, we will focus our study on UCS since it does not suffer from the effect of other complexity factors as observed in XCS.

UCS has been shown to depend on the size of the optimal population. The parity problem, which ranges from three to nine input bits, has the following optimal population sizes: 8, 16, 32, 64, 128, 256 and 512. Similarly, in the decoder problem, ranging from three to six input bits, |[O]| corresponds to 8, 16, 32 and 64 respectively. In both cases, UCS's learning becomes slower as the problem increases. Moreover, note that for a certain |[O]|, UCS has a similar convergence in the parity and the decoder problem to some extent (see Figures 1 and 2). In fact, as noted in Section 5, the decoder problem is easier for UCS due to a more effective fitness pressure. Taking the position problem into account, the optimal populations are much smaller than the previous problems. From three to nine input bits, |[O]| corresponds to: 4, 5, 6, 7, 8, 9 and 10. This should indicate that the position problem is much easier than the parity and decoder problems. However, this was not observed in our results. For example, regarding Figures 3 and 1, the pos8 problem, which has $|[O]| = 9$, presents more learning difficulty than par6, with $|[O]| = 64$. So, relating the learning complexity to |[O]| leads to some inconsistencies in the results, which we seek to clarify in the following.

Let's analyze the main complexity factor in the position problem. For this purpose, we view a learning classifier system as a system which is trying to evolve simultaneously a set of environmental niches. We define an environmental niche as a region of the search space covered by a set of examples sharing the same class, and which can be represented by an optimal (accurate and maximally general) rule. For example, the rule 0001##:3 represents the environmental niche covered by all the examples with class 3 in the pos6 problem. The goal of the system is to discover the optimal rule belonging to each niche, i.e., to find the set of optimal rules [O]. Depending on the characteristics of the problem, some niches may be more difficult to find than others. To achieve complete learning, the system must discover all the necessary niches and thus it must "wait" until the "slowest" niche is found. This can be formulated as follows:

$$t_{learning} = max_{o \in [O]}\, t_o \tag{6}$$

where $t_{learning}$ is the number of iterations needed to achieve learning, o is an optimal rule belonging to the optimal population [O], and t_o is the number of iterations needed to discover the rule o.

The number of iterations needed to discover an optimal rule for a certain niche depends on the type of rule and the problem itself. It is difficult to characterize, but it certainly depends on the number of examples provided to update the rules for that niche along the iterations. If a niche is updated often, it has more reproductive opportunities and therefore it has more chances to have its optimal rule discovered. A niche is updated when it is activated by its representative examples. Given a training set $\mathcal{T}$, the percentage of representative examples belonging to a certain environmental niche can be formulated as $|s_o|/|\mathcal{T}|$, where $|s_o|$ is the number of examples of the niche in $\mathcal{T}$, and $|\mathcal{T}|$ is the size of the training set. Assuming a uniform sampling of examples during learning, the niche examples will appear to the classifier system in a frequency $f_o \approx |s_o|/|\mathcal{T}|$. For example, in the pos6 problem the rule 0001##:3 has four representative examples: {000100, 000101, 000110, 000111}. The sample frequency of this niche corresponds to $4/2^6 = 1/16$ (assuming that all the examples are available in $\mathcal{T}$ and are provided uniformly). As the sample frequency of a niche is higher, fewer iterations are needed to learn the niche. Therefore, we hypothesize that the number of iterations needed to discover an optimal rule depends inversely on the sample frequency f_o multiplied by a complexity factor c_o:

$$t_o = \frac{c_o}{f_o} = c_o \cdot \frac{|\mathcal{T}|}{|s_o|} \tag{7}$$

This equation models the fact that specific rules are more difficult to find. As we mentioned previously, the chances LCS has of discovering an optimal rule depend on the opportunities the niche has been given to be updated, and thus to be reproduced by the GA (although covering and specify also play an important role, which also depends on the number of updates). Kovacs and Kerber (2001) also hypothesized that specific rules are harder to find for XCS, and provided the frequency of updates as one of the main reasons. The most general rules are represented by more frequent examples. On the other hand, the most specific rules have few representative examples which appear less often and thus, the total number of iterations to encounter such examples should increase. While some general rules have already been discovered, the system may still need more learning cycles to find good representatives for the most specific rules. Therefore, the learning complexity can be formulated as:

$$t_{learning} = max_{o \in [O]} \frac{c_o}{f_o} \tag{8}$$

where c_o and f_o are particular to each niche.

This hypothesis explains our results with the position problem. UCS's convergence depends mainly on the most specific rules, which have a sample frequency $f_o = 1/2^l$. The most general rule has $f_o = 1/2$ (one out of two examples represent this rule) and therefore it is much easier to evolve. Thus, the main factor of complexity in the position problem can be attributed to the small proportion of representative examples provided to the most specific rules.

How can these results be related to the $|[O]|$ metric observed in the parity and decoder problems? Note that these problems are characterized by rules with the same generality level. This means that there is the same proportion of representative examples per niche. Assuming a uniform distribution of examples, the value of f_o is the same for all niches and consequently it can be formulated as $f_o = 1/|[O]|$. This leads to:

$$t_{learning} = max_{o \in [O]} \frac{c_o}{f_o} = |[O]| max_{o \in [O]} \, c_o = |[O]| \cdot c \tag{9}$$

In this equation, we have assumed that c_o is the same for the optimal rules in [O], since all optimal rules in the parity and decoder problems share the same characteristics. This may not apply in all the problems, so the simplification performed in equation 9 may not be valid everywhere. Viewed in this way, learning depends on the size of the optimal population and a certain complexity factor which depends on the problem. This complexity factor c may model the mean Hamming distance, the fitness pressure, etc., and explains why two different problems with the same $|[O]|$ have different $t_{learning}$.

Regarding the example introduced earlier, where we compared pos8 with par6, the model of formula 8 can also explain why pos8 is more difficult to learn than par6. Note that in pos8 the lowest sample frequency corresponds to $f_o = 1/2^8 = 1/256$, while in par6 $f_o = 1/2^6 = 1/64$. Relating the learning complexity to f_o explains why pos8 needs more training cycles than par6, which could not be explained by the $|[O]|$ metric.

To conclude, the dependence of learning on $|[O]|$ can be devised as a particular case of the model where learning depends on the sample frequency of each environmental niche. This is consistent with the results and observations obtained throughout the paper. Although the study is based on UCS's results, this observation also applies to XCS, since the rule discovery mechanisms –which depend on the sample frequency– are basically the same.

7 Application to Real World Classification Problems

So far we have analyzed the performance of accuracy-based classifier systems on a set of artificial problems. This has allowed us to give insight into the behavior of the LCS related to the characteristics of the problem. In this section, our aim is to test the performance of the accuracy-based LCS on real world classification problems. The understanding of the LCS behavior on this type of problem is more complicated, since these problems may have intrinsic complexities, which are difficult to identify, limiting the maximum achievable accuracy rate. In an effort to determine the level of complexity of a given problem, we trained different types of learning systems. Besides having a reference value of the problem complexity, this will allow us to compare the LCS with respect to other well-known classification algorithms. The analysis was performed on a variety of problems, summarized in 30 different datasets. Some of the results presented here also appear, for a reduced set of problems, in (Bernadó-Mansilla et al., 2002). We have also extended the comparison between XCS and UCS to this framework.

7.1 Methodology

Thirty varied datasets were selected for the present study. They include different characteristics that may imply a factor of complexity for learning systems, such as high dimensionality, small number of available instances, missing attributes, real-valued attributes, high number of classes, etc. These features are summarized in Table 6. All the selected datasets belong to real world domains, except for *mux*, *led* and *tao* which were generated artificially. These represent three different case studies, where the comparison of the LCS performance with respect to other learning algorithms is interesting: a) the multiplexer problem (*mux*) with 11 attributes, where evolutionary learning systems have proved to be well suited, b) the *led* problem (Blake and Merz, 1998), with binary inputs and 10% added noise and c) the *tao* problem (Bernadó-Mansilla et al., 2002) with two real attributes and curved boundaries between classes. Most of the real world domains are obtained from the UCI repository (Blake and Merz, 1998), with an exception made for *bps* and *mmg* (see (Bernadó-Mansilla et al., 2002) for the details).

The accuracy rate of each learning system is used as the metric of performance. To have a good estimate of the accuracy rate, we use the following methodology. For large datasets (with a high number of instances) we use the holdout estimate. This applies to *krk*, *led*, *mus* and *sic* datasets, where 33.3% of the available instances are placed in the training set and 66.6% in the test set. The rest of the datasets are run on a stratified tenfold crossvalidation test. To test if the differences between the learning algorithms are statistically significant, we use a test for the difference of two proportions, if holdout is used, and a paired t-test from the ten-fold crossvalidation results (Dietterich, 1998). The Wilcoxon signed rank test (Conover, 1971) is used as well to test the overall statistical significance of the observed differences between two methods.

The performance of the LCS is compared to a variety of learning algorithms: ZeroR, IB1, IBk, Naive Bayes, C4.5, PART and SMO. Each will be discussed briefly as follows. ZeroR is a simple classifier system which predicts the majority class deduced from the training set. IB1 and IBk (Aha et al., 1991) are nearest neighbor algorithms: an example is classified by the majority class of the k nearest neighbor training instances

Table 6: Description of the datasets. The columns describe respectively: the name which we use to refer the dataset (Id), the extended name (DS), number of instances (Inst), percentage of missing attributes (missing), number of total attributes (#atr), number of binary (#bin), nominal (#nom) and real (#real) attributes and number of classes (#clas).

Id	DS	Inst	missing (%)	#atr	#bin	#nom	#real	#clas
aud	Audiology	226	2.0	69	61	8	-	24
aus	A.credit card	690	0.6	14	4	4	6	2
bal	Balance-scale	625	0.0	4	-	4	-	3
bpa	Bupa	345	0.0	6	-	-	6	2
bps	Biopsies	1027	0.0	24	-	-	24	2
bre	Breast-w	699	0.3	9	-	-	9	2
cmc	Cmc	1473	0.0	9	3	4	2	3
gls	Glass	214	0.0	9	-	-	9	6
h-c	Heart-c-14	303	0.2	13	3	4	6	5
h-h	Heart-h-14	294	20.4	13	3	4	6	5
hep	Hepatitis	155	5.6	19	13	-	6	2
irs	Iris	150	0.0	4	-	-	4	3
krk	kr-vs-kp	3196	0.0	36	34	2	-	2
lab	Labor	57	3.9	16	3	5	8	2
led	Led (10%n)	6000	0.0	7	7	-	-	10
mmg	Mammograms	216	0.0	21	-	-	21	2
mus	Mushrooms	8124	1.4	22	4	18	-	2
mux	11-Multiplexer	2048	0.0	11	11	-	-	2
pmi	Pima-indians	768	0.0	8	-	-	8	2
prt	Primary-tumor	339	3.9	17	14	3	-	21
seg	Segment	2310	0.0	19	-	-	19	7
sic	Sick	3772	5.5	29	21	1	7	2
soy	Soybean	684	2.9	35	16	19	-	19
tao	Tao	1888	0.0	2	-	-	2	2
thy	Thyroid	2237	5.2	29	21	1	7	18
veh	Vehicle	846	0.0	18	-	-	18	4
vow	Vowel	990	0.0	13	2	1	10	11
vte	Vote	435	5.6	16	16	-	-	2
wne	Wine	178	0.0	13	-	-	13	3
zoo	Zoo	101	0.0	17	15	1	1	7

(where k is 1 and 3 respectively). Naive Bayes (John and Langley, 1995) estimates the predicted class from the Bayes' rule of conditional probabilities. C4.5 (revision 8) (Quinlan, 1993) is an induction tree which derives from the ID3 algorithm. PART (Frank and Witten, 1998) is a learning scheme which extracts rules from the combination of decision trees and the separate-and-conquer rule learning strategy. SMO (Platt, 1998) is a support vector classifier system, implementing the sequential minimal optimization algorithm. It only performs binary classifications. All these algorithms are run using the Weka package (Witten and Frank, 2000), available at *http://www.cs.waikato.ac.nz/ml/weka*.

In order to apply XCS and UCS to the new datasets, a few changes are made with respect to the previous experiments. The rule codification is adapted to deal with real attributes and mixed types of attributes, as described in (Bernadó-Mansilla et al., 2002). The condition of the rule is codified as a sequence of tests, where each test fits to the type of attribute that it matches. If the attribute is binary, the test is codified in the ternary alphabet. If the attribute is real, the test is codified as an interval range. For simplicity, all real attributes are normalized to the range [0,1). Nominal attributes are translated into numeric values, and so considered as real attributes. The covering and GA operators are adapted accordingly. The parameter settings which are different from those previously indicated are: $N = 6400$, explore trials = 100000, $\theta_{GA} = 50$, $\theta_{del} = 50$, $\theta_{sub} = 50$, covering range in interval codification $r_0 = 0.6$ and mutation range in interval codification $m_0 = 0.1$ (see (Wilson, 2001) for the notation).

7.2 Results

Table 7 summarizes the accuracy rates of the different learning systems on all the datasets. Each number is either the mean of ten results, when crossvalidation was used, or the holdout estimate otherwise. Statistical differences are shown according to a two-tailed test at 95% confidence level, with the following notation. A significant improvement of XCS with respect to another method is denoted by • and a significant degradation by ◦. Similarly, a UCS improvement is denoted by ▲ and a degradation by △.

Several observations can be drawn from Table 7. Comparing UCS with XCS, we observe that there are no significant differences between them in 21 out of 30 datasets. UCS improves XCS significantly in six datasets: *aud*, *hep*, *seg*, *soy*, *thy* and *vow*. Observe that all these datasets (except for *hep*) are characterized by a high number of classes. There are 24 classes in *aud*, 7 classes in *seg*, 19 in *soy*, 18 in *thy* and 11 in *vow*. This confirms the observations made in the previous sections, where UCS was hypothesized to perform better than XCS in problems with a high number of classes, due to a more efficient accuracy pressure and a smaller covering map. XCS improves UCS in three datasets: *cmc*, *sic* and *tao*. These problems belong to binary and ternary classifications, and thus the advantages of UCS do not apply. Nevertheless, why XCS outperforms UCS is not explainable, but we hypothesize that this is a consequence of the fact that UCS does not have fitness sharing as XCS (see Section 3). The absence of fitness sharing may produce an excessive overtaking of some rules in the population, preventing other rules from being explored. Another problem associated with the lack of fitness sharing is that overlapping [O]s are not degraded, so this may favor the maintenance of redundant rules in the population. These effects have not arisen in our previous results, so we would benefit from further investigation along with the design of new test problems that emphasize this aspect.

Regarding the performance of the LCS related to the datasets, we note that the LCS have less accuracy rates than other methods in some datasets. Observe that some

Table 7: Accuracy rates of XCS and UCS along with different learning algorithms in a set of 30 datasets. Each number is the mean of the stratified ten-fold crossvalidation results or the holdout estimate. The statistical difference between XCS and the other methods is denoted by •, if XCS is better, and by ◦ if XCS is worse. Similarly, a significant improvement of UCS is denoted by ▲, and a degradation by △. The rows labeled *Xbw* and *Ubw* summarize respectively the number of datasets where XCS and UCS are better-worse than the method in the column. The last two rows show the confidence level of the Wilcoxon test for XCS and UCS with respect to the method in the column. See text for details.

	ZeroR	IB1	IBk	NB	C4.5	PART	SMO	XCS	UCS
`aud`	25.3 •▲	76.0 ◦△	68.4 ◦△	69.6 ◦△	79.0 ◦△	81.2 ◦△	-	41.6 ▲	57.8 ◦
`aus`	55.5 •▲	81.9	85.4	77.5 •▲	85.2	83.3	84.9	84.8	84.8
`bal`	45.0 •▲	76.2 •▲	87.2 ◦△	90.4 ◦△	78.5 •▲	81.9	-	81.6	82.9
`bpa`	58.0 •▲	63.5 ▲	60.6 ▲	54.3 •▲	65.8	65.8	58.0 •▲	65.4	68.7
`bps`	51.6 •▲	83.2	82.8	78.6 •▲	80.1 •▲	79.0 •▲	86.4 ◦△	83.2	84.2
`bre`	65.5 •▲	96.0	96.7	96.0	95.4	95.3	96.7	96.4	95.8
`cmc`	42.7 •▲	44.4 •▲	46.8 •▲	50.6 •	52.1 •	49.8 •	-	55.5 △	53.0 •
`gls`	34.6 •▲	66.3	66.4	47.6 •▲	65.8	69.0	-	70.8	70.8
`h-c`	54.5 •▲	77.4	83.2	83.6	73.6 •	77.9	-	80.3	79.3
`h-h`	63.9 •▲	78.3	82.4	83.7 △	80.3	79.6	-	79.9	79.3
`hep`	79.3	79.9	80.8	83.2 ◦	78.9	80.0	83.9 ◦	76.8 ▲	83.3 ◦
`irs`	33.3 •▲	95.3	95.3	94.7	95.3	95.3	-	94.7	94.7
`krk`	52.2 •▲	89.4 •▲	94.9 •▲	87.0 •▲	98.3	98.4	96.1 •▲	99.1	98.6
`lab`	65.4	81.1	92.1 ◦△	95.2 ◦△	73.3	73.9	93.2 ◦△	73.6	72.0
`led`	10.5 •▲	62.4 •▲	75.0	74.9	74.9	75.1	-	74.5	74.5
`mmg`	56.0 •	63.0	65.3	64.7	64.8	61.9	67.0	64.3	61.5
`mus`	51.8 •▲	100 ◦△	100 ◦△	96.4 •▲	100 ◦△	100 ◦△	100 ◦△	99.0	99.2
`mux`	49.9 •▲	78.6 •▲	99.8 •▲	61.9 •▲	99.9	100	61.6 •▲	100	100
`pmi`	65.1 •▲	70.3 •▲	73.9	75.4	73.1	72.6	76.7	75.4	75.6
`prt`	24.9 •▲	34.5 ▲	42.5	50.8 ◦△	41.6	39.8	-	39.9	40.1
`seg`	14.3 •▲	97.4 ◦△	96.1 ◦	80.1 •▲	97.2 ◦△	96.8 ◦	-	94.3 ▲	96.2 ◦
`sic`	93.8 •▲	96.1 •	96.3	93.3 •▲	98.4 △	97.0	93.8 •▲	97.6 △	96.1 •
`soy`	13.5 •▲	89.5 ◦△	90.3 ◦△	92.8 ◦△	91.4 ◦△	90.3 ◦△	-	24.8 ▲	51.2 ◦
`tao`	49.8 •▲	96.1 ◦△	96.0 ◦△	80.8 •▲	95.1 ◦△	93.6 ◦△	83.6 •▲	89.9 △	88.2 •
`thy`	19.5 •▲	68.1 ▲	65.1 ▲	80.6 ◦▲	92.1 ◦△	92.1 ◦△	-	66.8 ▲	86.9 ◦
`veh`	25.1 •▲	69.4 •	69.7	46.2 •▲	73.6	72.6	-	73.0	71.1
`vow`	9.1 •▲	99.1 ◦△	96.6 ◦△	65.3 •▲	80.7 ◦▲	78.3 ◦▲	-	69.2 ▲	86.1 ◦
`vte`	61.4 •▲	92.4 •▲	92.6 •▲	90.1 •▲	96.3	96.5	95.6	95.6	95.6
`wne`	39.8 •▲	95.6	96.8	97.8	94.6	92.9	-	95.1	97.2
`zoo`	41.7 •▲	94.6	92.5	95.4	91.6	92.5	-	94.6	93.6
Xbw	28-0	9-6	4-8	14-7	4-7	2-7	5-4	-	3-6
Ubw	27-0	9-6	6-7	14-6	3-7	2-5	5-3	6-3	-
XW	99.5	82.7	-87.0	74.9	-81.1	57.0	51.1	50.0	-81.7
UW	99.5	93.1	-73.6	87.9	-55.9	66.3	53.3	81.7	50.0

of these datasets coincide with the problems with the highest number of classes (*aud*, *seg*, *soy*, *thy* and *vow*). Although UCS improves XCS in these cases, LCS are still worse than other methods in this type of problems. The *tao* problem is also difficult for LCS. Since the boundaries between classes are curved in this problem, the approximation made by the LCS, which is based on hyperrectangles, is not as good as some methods like the nearest neighbor algorithms, which have the highest score. On the contrary, the *multiplexer* problem is better suited for LCS than the rest of the algorithms. The results in the *led* problem, which was generated with 10% noise, indicate that the presence of noise does not negatively affect LCS more than the other methods. Other differences between the LCS and the rest of the learning systems in some datasets are not easily explainable by the characteristics noted in Table 6, like the number or type of attributes, the number of instances, etc. This leads to future work whose goal would be to determine which measures characterize the complexity of real world problems. This characterization would allow us to investigate how learning performance relates to the complexity of the problem and would provide us with further explanations on the LCS behavior related to other methods.

The last rows in Table 7 summarize the overall performance of the LCS on all the datasets with respect to each of the learning methods. The row labeled *Xbw* counts the number of datasets where XCS is significantly better-worse than the method in the column. Similarly, the row labeled *Ubw* does the same computation for UCS. Note that XCS outperforms ZeroR, IB1 and Naive Bayes, is equivalent to SMO and worse than IBk, C4.5 and PART. UCS improves slightly these results, being superior to ZeroR, IB1 and Naive Bayes, equivalent to SMO and IBk and worse than C4.5 and PART. The two last rows show the confidence level of the Wilcoxon test for XCS and UCS respectively. A positive confidence value indicates that XCS/UCS are better than the method in the column, and a negative value indicates that XCS/UCS are worse. Regarding this test, few differences are found between XCS/UCS and the other methods. For example, UCS is observed to improve IB1 with a 93.1% confidence level, while all other differences have less confidence values (except for ZeroR). Regarding the training times, LCS usually tend to need higher training times than the other methods. One of the reasons can be attributed to the high number of rules that LCS maintain during training, especially in problems with real attributes. The hyperrectangle codification used in these cases tends to spread a high number of rules throughout the population. Although these rules are very similar, they are not similar enough to be condensed into a more compact ruleset. A method has been proposed recently to reduce the ruleset after training (Wilson, 2002) without affecting performance. The use of a similar method applied during learning might be considered in order to improve the LCS training time.

8 Conclusions

This paper investigates LCS on classification problems. Our study focused on two models of accuracy-based LCS: XCS and UCS. XCS bases fitness on accuracy computed from a reinforcement learning scheme, and in doing so it evolves a complete action map. We have designed a system close to XCS, where fitness is based on accuracy as well but calculated from a supervised learning perspective. The resulting system shares some properties with other LCS, such as Frey&Slate's and NEWBOOLE systems, but maintains the generalization mechanisms of XCS. This gives UCS the capability to evolve accurate generalizations through the states which are correct, that is, while evolving a best action map. The design of UCS is motivated by the need to understand and characterize the LCS dynamics on a set of difficult classification tasks. Its similarity with XCS allows us to isolate some key factors, such as knowledge representation and fitness pressure.

UCS has been shown to evolve accurate generalizations of best action maps efficiently, which proves that partial maps are an efficient alternative to complete maps. Nevertheless, our results demonstrate that this mostly occurs in large search spaces, when the ratio between the size of the complete action map and the best action map is significantly high. In this case, complete maps need more computational resources, such as larger populations and more learning cycles. In fact, the major factor in XCS's learning complexity is the absence of an effective fitness pressure towards accuracy, and this is worsened by the excessive generalization pressure induced by some incorrect rules of the complete map. This is hypothesized to occur in problems with a high number of classes or with highly unequal distribution of examples. These observations, made on a set of artificially designed problems, are confirmed by the results obtained on real world classification tasks.

Studying problems with unequal distribution of examples per class has also provided a better understanding of the learning complexity of LCS. In fact, accurate generalizations are reached at different levels, but the most specific rules are harder to achieve. We provide a model where the learning complexity depends on the percentage of representative examples for each optimal rule. The model includes a learning dependence on the size of the optimal population, which has already been observed in previous experiments reported in the literature.

Although some differences between XCS and UCS were highlighted throughout the paper, their actual performance in some real world classification problems is very close. This means that there are some complexity issues that affect both systems in a similar way. The characterization of real world problems by a set of complexity measures would allow us to model the LCS performance related to problem complexity, and would provide more insight into the kinds of problems where LCS are well suited and poorly suited. The overall performance of XCS and UCS is comparable to other well-known learning algorithms, like nearest neighbor algorithms, statistical methods, induction trees and support vector classifiers. Nevertheless, LCS still suffer from higher training times, so further investigation should be done to reduce their training time. Some issues in this direction may include the use of reduction techniques that minimize the number of rules during training, especially in problems with real attributes.

This paper proposes UCS as an accuracy-based architecture specifically designed for supervised learning problems. XCS is more general, and thus, it is reasonable that it does not perform as well as UCS in environments where supervision is provided. However, XCS can be applied to problems with layered reward and online environments, i.e., interacting with the environment and only getting feedback about action consequences. UCS only applies to single-step tasks, where supervision is available. Although its results are promising, its architecture still needs to be studied and improved. One further difference with XCS is the lack of fitness sharing, which might produce an excessive overtaking of some good rules in the population during training. In future work, we need to study how fitness sharing operates on LCS. In addition to this framework for further comprehension of the LCS components, UCS also provides an efficient alternative for difficult classification tasks.

Acknowledgments

The authors would like to acknowledge Tin K. Ho, Alfred V. Aho and three anonymous reviewers for their comments. We also thank Ian H. Witten and Eibe Frank for providing the Weka package and the donors of the datasets. This work was partially supported by *Enginyeria i Arquitectura La Salle*, *Ministerio de Ciencia y Tecnología* under grant TIC2002-04036-C05-03, and *Generalitat de Catalunya* under grant 2002SGR 00155.

References

Aha, D., Kibler, D., and Albert, M. (1991). Instance-based Learning Algorithms. *Machine Learning, Vol. 6*, pages 37–66.

Bernadó-Mansilla, E. (2002). *Contributions to Genetic Based Classifier Systems*. PhD thesis, Enginyeria i Arquitectura La Salle, Ramon Lull University, Barcelona, Spain.

Bernadó-Mansilla, E. and Garrell-Guiu, J. M. (2001). MOLeCS: Using Multiobjective Evolutionary Algorithms for Learning. In Zitzler, E., Deb, K., Thiele, L., Coello, C., and Corne, D., editors, *First International Conference on Evolutionary Multi-Criterion Optimization*, volume 1993 of *Lecture Notes in Computer Science*, pages 696–710. Springer Verlag.

Bernadó-Mansilla, E., Llorà-Fàbrega, X., and Garrell-Guiu, J. M. (2002). XCS and GALE: a Comparative Study of Two Learning Classifier Systems on Data Mining. In Lanzi, P., Stolzmann, W., and Wilson, S., editors, *Advances in Learning Classifier Systems, 4th International Workshop*, volume 2321 of *Lecture Notes in Artificial Intelligence*, pages 115–132. Springer.

Blake, C. and Merz, C. (1998). UCI Repository of machine learning databases, [http://www.ics.uci.edu/∼mlearn/MLRepository.html]. University of California, Irvine, Department of Information and Computer Sciences.

Bonelli, P., Parodi, A., Sen, S., and Wilson, S. W. (1990). NEWBOOLE: A fast GBML System. In Porter, B. and Mooney, R., editors, *Seventh International Conference on Machine Learning*, pages 153–159. Morgan Kaufmann.

Bull, L., Wyatt, D., and Parmee, I. (2002). Initial Modifications to XCS for Use in Interactive Evolutionary Design. In Merelo Guervós, J. J., Adamidis, P., Beyer, H.-G., Fernández-Villacañas, J. L., and Schwefer, H.-P., editors, *Parallel Problem Solving from Nature - PPSN VII*, pages 568–577. Springer.

Butz, M. and Wilson, S. (2001). An algorithmic description of XCS. In Lanzi, P., Stolzmann, W., and Wilson, S., editors, *Advances in Learning Classifier Systems: Proceedings of the Third International Workshop*, volume 1996 of *Lecture Notes in Artificial Intelligence*, pages 253–272. Springer-Verlag Berlin Heidelberg.

Butz, M. V., Kovacs, T., Lanzi, P. L., and Wilson, S. W. (2001). How XCS Evolves Accurate Classifiers. In Spector, L., Goodman, E., Wu, A., Langdon, W., Voigt, H., Gen, M., Sen, S., Dorigo, M., Pezeshk, S., Garzon, M., and Burke, E., editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'2001)*, pages 927–934. San Francisco, CA: Morgan Kaufmann.

Butz, M. V. and Pelikan, M. (2001). Analyzing the Evolutionary Pressures in XCS. In Spector, L., Goodman, E., Wu, A., Langdon, W., Voigt, H., Gen, M., Sen, S., Dorigo, M., Pezeshk, S., Garzon, M., and Burke, E., editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'2001)*, pages 935–942. San Francisco, CA: Morgan Kaufmann.

Cliff, D. and Ross, S. (1995). Adding Temporary Memory to ZCS. *Adaptive Behavior*, 3(2):101–150.

Conover, W. (1971). *Practical Nonparametric Statistics*. New York: John Wiley, pages 206-209, 383.

Dietterich, T. G. (1998). Approximate Statistical Tests for Comparing Supervised Classification Learning Algorithms. *Neural Computation*, 10(7):1895–1924.

Frank, E. and Witten, I. H. (1998). Generating Accurate Rule Sets Without Global Optimization. In Shavlik, J., editor, *Machine Learning: Proceedings of the Fifteenth International Conference*, pages 144–151. Morgan Kaufmann.

Frey, P. and Slate, D. (1991). Letter recognition using Holland-style adaptive classifiers. *Machine Learning*, 6:161–182.

Hartley, A. H. (1999). Accuracy-based fitness allows similar performance to humans in static and dynamic environments. In Banzhaf, W., Daida, J., Eiben, A., Garzon, M. H., Honavar, V., Jakiela, M., and Smith, R. E., editors, *Proceedings of the Genetic and Evolutionary Computation Conference, (GECCO-99)*, pages 266–273. Morgan Kaufmann.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor.

Holmes, J. H. (1997). Discovering Risk of Disease with a Learning Classifier System. In Bäck, T., editor, *Proceedings of the Seventh International Conference of Genetic Algorithms (ICGA97)*, pages 426–433. Morgan Kaufmann.

Horn, J., Goldberg, D. E., and Deb, K. (1994). Implicit Niching in a Learning Classifier System: Nature's Way. *Evolutionary Computation, 2(1)*, pages 37–66.

John, G. H. and Langley, P. (1995). Estimating Continuous Distributions in Bayesian Classifiers. In Besnard, P. and Hanks, S., editors, *11th Conference on Uncertainty in Artificial Intelligence* , pages 338–345. Morgan Kaufmann.

Kovacs, T. (1997). XCS Classifier System Reliably Evolves Accurate, Complete and Minimal Representations for Boolean Functions. In Roy, R., Chawdhry, P., and Pant, R., editors, *Soft Computing in Engineering Design and Manufacturing*, pages 59–68. Springer-Verlag.

Kovacs, T. (1999). Deletion Schemes for Classifier Systems. In Banzhaf, W., Daida, J., Eiben, A., Garzon, M. H., Honavar, V., Jakiela, M., and Smith, R. E., editors, *Proceedings of the Genetic and Evolutionary Computation Conference, (GECCO-99)*, pages 329–336. Morgan Kaufmann.

Kovacs, T. (2000). Strength or Accuracy? A comparison of two approaches to fitness calculation in learning classifier systems. In Stolzmann, W., Lanzi, P., and Wilson, S., editors, *Learning Classifier Systems: From Foundations to Applications*, volume 1813 of *Lecture Notes in Computer Science*, pages 143–160. Springer-Verlag Berlin Heidelberg.

Kovacs, T. (2001). Towards a Theory of Strong Overgeneral Classifiers. In *Foundations of Genetic Algorithms*, volume 6, pages 165–184. Morgan Kaufmann.

Kovacs, T. (2002). *A Comparison of Strength and Accuracy-based Fitness in Learning Classifier Systems*. PhD thesis, School of Computer Science. University of Birmingham. Birmingham, UK.

Kovacs, T. and Kerber, M. (2001). What Makes a Problem Hard for XCS? In Lanzi, P., Stolzmann, W., and Wilson, S., editors, *Advances in Learning Classifier Systems: Proceedings of the Third International Workshop*, volume 1996 of *Lecture Notes in Artificial Intelligence*, pages 80–99. Springer-Verlag Berlin Heidelberg.

Lanzi, P. L. (1997). A Study of the Generalization Capabilities of XCS. In Bäck, T., editor, *Proceedings of the Seventh International Conference of Genetic Algorithms (ICGA97)*, pages 418–425. Morgan Kaufmann.

Platt, J. (1998). Fast training of support vector machines using sequential minimal optimization. In Schlkopf, B., Burges, C., and Smola, A., editors, *Advances in Kernel Methods-Support Vector Learning*. Cambridge, MA:MIT Press.

Quinlan, R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers.

Saxon, S. and Barry, A. (2000). XCS and the Monk's Problem. In Lanzi, P., Stolzmann, W., and Wilson, S., editors, *Learning Classifier Systems: From Foundations to Applications*, volume 1813 of *Lecture Notes in Computer Science*, pages 223–242. Springer-Verlag Berlin Heidelberg.

Sutton, R. and Barto, A. (1998). *Reinforcement Learning: An Introduction*. Cambridge, MA: The MIT Press/Bradford Books.

Wilson, S. W. (1987). Classifier Systems and the Animat Problem. *Machine Learning Journal*, 2:199–228.

Wilson, S. W. (1995). Classifier Fitness Based on Accuracy. *Evolutionary Computation*, 3(2):149–175.

Wilson, S. W. (1998). Generalization in the XCS Classifier System. In Koza, J., Banzhaf, W., Chellapilla, K., Deb, K., Dorigo, M., Fogel, D., Garzon, M., Goldberg, D., Iba, H., and Riolo, R., editors, *Genetic Programming: Proceedings of the Third Annual Conference*, pages 665–674. San Francisco, CA: Morgan Kaufmann.

Wilson, S. W. (1999). Get Real! XCS with Continuous-Valued Inputs. In Booker, L., Forrest, S., Mitchell, M., and Riolo, R., editors, *Festschrift in Honor of John H. Holland*, pages 111–121. Center for the Study of Complex Systems, University of Michigan.

Wilson, S. W. (2001). Mining Oblique Data with XCS. In Lanzi, P., Stolzmann, W., and Wilson, S., editors, *Advances in Learning Classifier Systems: Proceedings of the Third International Workshop*, volume 1996 of *Lecture Notes in Artificial Intelligence*, pages 158–176. Springer-Verlag Berlin Heidelberg.

Wilson, S. W. (2002). Compact Rulesets from XCSI. In Lanzi, P., Stolzmann, W., and Wilson, S., editors, *Advances in Learning Classifier Systems, 4th International Workshop*, volume 2321 of *Lecture Notes in Artificial Intelligence*, pages 197–210. Springer.

Witten, I. H. and Frank, E. (2000). *Data Mining. Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann Publishers.