An Analysis of Generalization in XCS with Symbolic Conditions

Pier Luca Lanzi

Abstract— **We analyze generalization in the eXtended Classifier System (XCS) with symbolic conditions, based on genetic programming, briefly XCSGP. We start from the results presented in the literature, which showed that XCSGP could not reach optimality in Boolean problems when classifier conditions involved logical disjunctions. We apply a new implementation of XCSGP to the learning of Boolean functions and show that our version can actually reach optimality even when disjunctions are allowed in classifier conditions. We analyze the evolved generalizations and explain why logical disjunctions can make the learning more difficult in XCS models and why our version performs better than the earlier one. Then, we show that in problems that allow many generalizations, so that *or* clauses are less "convenient", XCSGP tends to develop solutions that do not exploit logical disjunctions as much as one might expect. However, when the problems allow few generalizations, so that *or* clauses become an interesting way to introduce simple generalizations, XCSGP exploit them so as to evolve more compact solutions.**

I. Introduction

Learning systems heavily rely on their generalization capabilities. A system that generalizes properly can (i) represent in a compact form what it has learned and can (ii) successfully apply what it has learned to previously unseen situations. In learning classifier systems [8] generalization is achieved through the evolution of *general* condition-action-prediction rules, called *classifiers*, which represent the solution to the target task. Generalization in learning classifier system depends on the language used to specify classifier conditions and, according to the most recent extensions, also on the function used to compute classifier prediction [21], and on the approach used to compute classifier actions [22]. Classifier conditions are usually represented by fixed-length strings on the ternary alphabet $\{0,1,\#\}$ which test a condition over a set of binary sensory inputs. However, along the years, several alternative representations [20], [12], [17] have been introduced which extended the original framework in many ways [9]. Among the several representations introduced, symbolic conditions, based on Genetic Programming [16], are probably the most general, the most versative, but unfortunately, also the most computational and memory demanding.

XCS with symbolic conditions, based on genetic-programming, XCSGP, was first introduced in [16]. The initial results showed that XCSGP performed optimally in some multistep problems and quite well, although not always optimally, in problems involving Boolean functions. The performance of XCSGP was analyzed and it was suggested that the use of *or* clauses was the cause of the unoptimal performance. In particular in [16], it was argued that *or* clauses can hide parts of classifier conditions and thus they may eventually favor the reproduction of untested (and thus risky) subexpressions. To support his argument it was shown that XCSGP could perfom optimally when classifier conditions did not include logical disjunctions. Later, XCSGP was also applied to simple classification problems showing that the system was competitive with other well-know classification methods [13].

None of the previous works on symbolic conditions [16], [13], not even in the most recent ones [17], presented an analysis of the evolved solutions. Accordingly, it is still not clear whether disjunctive clauses might be harmful and what types of solutions XCS with symbolic conditions evolves. In this paper, we follow up on our first work [16] and we analyze generalization in XCS with symbolic conditions. Our interest in two fold. First, we want to study whether logical disjunctions can be harmful to the evolution of optimal solutions, and why they might be harmful. Second, we want to analyze to what extent they are actually exploited by XCSGP for generalization since *or* clauses are per se a major distinguishing features between the concepts that XCS and XCSGP can evolve.

At first, we briefly describe the eXtended Classifier System XCS [18] (Section II) and we overview the modifications that the introduction of symbolic conditions requires (Section III). Then, we introduce the experimental design (Section IV) and illustrate the test problems considered in this paper (Section V). In Section VI, we study generalization in XCSGP. We show that in a simple Boolean problem our version of XCSGP can reach optimal performance even when logical disjuctions are employed, however, we also XCSGP does not fully exploit the potential that *or* clauses might provide. We note that, although XCS with symbolic conditions tends to evolve ovelapping classifiers, the introduction of logical disjunctions increases the amount of overlapped regions. As a consequence, the classifiers which share the same niche tend to have a very low fitness, so that it becomes more difficult for XCS to identify accurate classifiers. The introduction of potential fitness [2] as opposed to the original exponential fitness [18] used in the first implementation [16] increases the abolity of XCS in distinguishing among small fitness values, so that the performance of XCSGP can reach optimality. However, although logical disjunctions provide better generalization, they allow the building of complex concepts that are more difficult to maintain and to evolve. As a consequence, the evolved solutions exploit *or* clauses less than one might initially expect. Nevertheless, when the problem allows few generalizations, so that *or* clauses represent almost the only source of generalization, XCSGP can exploit them and evolve solutions that are slightly more

Pier Luca Lanzi is with the Dipartimento di Elettronica e Informazione, Politecnico di Milano, (email: pierluca.lanzi@polimi.it).

1-4244-1340-0/07$25.00 ©2007 IEEE

compact than those evolved by XCS.

II. The XCS Classifier System

XCS is a reinforcement learning algorithm that works on a rule based representation [14]. It maintains a population of rules (the classifiers) which represents the solution to a reinforcement learning problem. Classifiers consist of a condition, an action, and four main parameters [18], [3]: (i) the prediction p, which estimates the relative payoff that the system expects when the classifier is used; (ii) the prediction error ε, which estimates the error of the prediction p; (iii) the fitness F, which estimates the accuracy of the payoff prediction given by p; and (iv) the numerosity *num*, which indicates how many copies of classifiers with the same condition and the same action are present in the population.

At time t, XCS builds a *match set* [M] containing the classifiers in the population [P] whose condition matches the current sensory input s_t; if [M] contains less than θ_{nma} actions, *covering* takes place and creates a new classifier that matches s_t and has a random action. For each possible action a in [M], XCS computes the *system prediction* $P(s_t, a)$ which estimates the payoff that the XCS expects if action a is performed in s_t. The system prediction $P(s_t, a)$ is computed as the fitness weighted average of the predictions of classifiers in [M] which advocate action a [18]. Then XCS selects an action to perform; the classifiers in [M] which advocate the selected action form the current *action set* [A]. The selected action a_t is performed, and a scalar reward r_{t+1} is returned to XCS together with a new input s_{t+1}. When the reward r_{t+1} is received, the estimated payoff $P(t)$ is computed as follows:

$$P(t) = r_{t+1} + \gamma \max_{a \in [M]} P(s_{t+1}, a) \quad (1)$$

Next, the parameters of the classifiers in [A] are updated in the following order [3]: prediction, prediction error, and finally fitness. Prediction p is updated with learning rate β ($0 \leq \beta \leq 1$):

$$p_k \leftarrow p_k + \beta(P(t) - p_k) \quad (2)$$

Then, the prediction error ϵ and classifier fitness are updated as usual [18], [3]. On regular basis (dependent on parameter θ_{ga}), the genetic algorithm is applied to classifiers in [A]. It selects two classifiers, copies them, and with probability χ performs crossover on the copies; then, with probability μ it mutates each allele. The resulting offspring classifiers are inserted into the population and two classifiers are deleted to keep the population size constant.

III. Adding Symbolic Conditions to XCS

XCS with symbolic conditions (XCSGP) was first introduced in [16], see also [13]. The version considered in this paper is a recent reimplementation of the original system based on the Open Beagle GP framework [5], [6]. The extension of XCS with symbolic, GP-like, conditions is straightforward and it basically requires the modification of four XCS components: the representation of classifier conditions, the matching, the covering, and the genetic operators.

Representation. The classifiers in XCSGP have the same structure as the classifiers in XCS. The only difference is in the representation of the conditions that in XCSGP is based on Genetic Programming [11]. Conditions in XCSGP are trees that represent programs obtained by composing functions, taken from the set of the available functions F, and terminals, taken from the set of the available terminals T. In this paper, the terminals represent the problem variables while the available functions are limited to the basic Boolean operators, i.e., *and*, *or*, and *not*.

Matching. This operation is straightforward. Classifier conditions are evaluated on the current input using the typical GP evaluation procedure implemented in Open Beagle [6].

Covering. The covering operator is applied when no classifier in the population matches the current input. It creates a new classifier with a condition that matches the current input and a randomly generated action. The covering operator works as follows. First, a fixed number of simple expressions, which match the current input, is generated. Then, these expressions are joined through the function *or*. The former step is application dependent. In the case of Boolean functions, each expression is a *minterm* which matches a random number of the current sensory inputs, determined by the same don't care probability $P_{\#}$ used in XCS [18]. The latter step introduces an additional source of generalization in the initial population. As in [16], covering generates conditions with *three* conjunctive expressions.

Genetic Operators are implemented using the standard GP operators available in Open Beagle [5], [6]. In the experiments presented here, recombination is implemented using the constrainted crossover, mutation is implemented using the constrained operators [5], [6]. The parameters of the offspring classifiers are updated as in XCS.

Implementation. The version of XCSGP used in this paper is a reimplementation of the original code [16], [13] based on the more recent XCS Library (`xcslib`) [15] and the Open Beagle framework [5], [6].

IV. Design of Experiments

In the experiments presented in this paper, we have followed the standard settings used in the literature [18]. Each experiment consists of a number of problems that the system must solve. Each problem is either a *learning* problem or a *test* problem. During *learning* problems, the system selects actions randomly from those represented in the match set. During *test* problems, the system always selects the action with prediction. The genetic algorithm is enabled only during *learning* problems, and it is turned off during *test* problems. The covering operator is always enabled, but operates only if needed. Learning problems and test problems alternate. The performance is computed as the percentage of correct answers during the last 100 test problems. All the reported statistics are averages over 20 experiments.

V. Test Problems

To study generalization in XCSGP we focused on Boolean functions. These are an important testbed to study learning classifier systems which are also well-suited to analyze generalization in the symbolic domains. For any Boolean function is relatively easy to compute its minimal, i.e., optimal, representation in *disjunctive normal form* as sum of prime implicants [7]. In addition, many tools are available which allow the simplification of complex Boolean expressions into its correspondent disjunctive normal form [7]. Accordingly, it is relatively easy to compare the solutions evolved by XCSGP with their corresponding minimal representations.

We considered on two types of problems, the Boolean multiplexer and the equality function. The former are well-know test functions which allow many generalization and for which XCS can easily evolve maximally general, maximally accurate, solutions. The latter are Boolean functions which allow very few generalizations, so that the typical XCS can only learn an extensional solution.

Boolean Multiplexer. These are defined over binary strings of length n where $n = k + 2^k$; the first k bits, $x_0, \ldots, x_{k-1}$, represent an address which indexes the remaining 2^k bits, $y_0, \ldots, y_{2^k-1}$; the function returns the value of the indexed bit. For instance, in the 6-multiplexer function, mp_6, we have that $mp_6(100010) = 1$ while $mp_6(000111) = 0$. The Boolean multiplexer involving 6, 11, and 20 inputs represented as sums of *prime implicants* are showed in Figure 1; the product corresponds to the logical *and*, the sum to the logical *or*, and the overline to logical *not*.

Equality. The function $eq_{n,k}$ is defined over n variables, $\{x_0, \ldots, x_{n-1}\}$ and returns one $x_0 + \ldots x_{n-1}$ is equal to k; zero otherwise. The minimal representation of $eq_{5,3}$ as sum of prime implicants is reported in Figure 2. Boolean multiplexer allow many generalizations in that their prime implicants (Figure 1) are defined over a small subset of the input variables. In contrast, $eq_{n,k}$ does not allow any generalization, in fact, its prime implicants coincide with the function minterms, i.e., no semplification of the minterms is possible. XCS cannot generalize much in $eq_{n,k}$, e.g., to represent the positive concept all the inputs must be taken into account to compute the function output; accordingly don't care can be rarely included into the classifier conditions.

VI. Generalization in XCSGP

We now analyze generalization in XCS with symbolic conditions. We start from the 6-multiplexer and show that, although XCSGP can potentially evolve conditions involving logical disjunctions, the evolved solutions contain fewer disjunction as the number of available generalizations increases. We argue that this has two main causes, the lower fitness that largely overlapping classifiers have and the intrinsic difficulty in maintaining complex concepts when simple concepts provide comparable generalization. We extend the results to bigger multiplexer and show that the findings we report for the 6-multiplexer still apply. Finally, we show

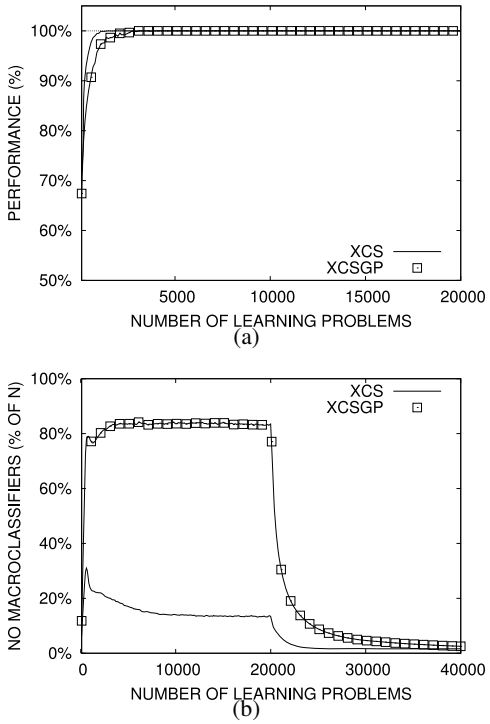

Fig. 3. XCSGP applied to the 6-multiplexer: (a) performance; (b) macroclassifiers in the population.

that, although XCSGP does not exploit logical disjunctions as much as we might expect, yet, in problems allowing a little generalization, i.e., when *or* clauses represent a major source of generalization, XCSGP successfully exploits them providing solutions that are slightly more compact than those evolved by XCS.

A. Experiments with the 6-Multiplexer

In the first experiment, we applied XCSGP to the 6-multiplexer with 1000 classifiers ($N = 1000$) and the following parameters, β=0.2, θ_{GA}= 0, ϵ_0=10.0, χ=0.8, μ=0.04, and θ_{nma}=2; *condensation* is activated after 20000 learning problems and last for 20000 learning problems; all the other parameters as set as in [18]. Figure 3a compares the performance of XCS (solid line) with that of XCSGP (dotted line), computed as the percentage of examples that have correctly classified; curves are averages over 20 runs. XCSGP reaches the optimal performance by 2500 learning problems and remains stable even when condensation starts, after 20000 learning problems. The number of macroclassifiers in the population (Figure 3b) rapidly reaches the 80% due to the typical bloating phenomenon which affects variable-length representations. Then it quickly drops as condensation starts; after 20000 condensation problems the average population size is around 25 classifiers. Overall, in 20000 condensation problems, the average population size drops from 800 classifiers to an average of 25 classifiers. Table I reports

$$\text{6-multiplexer}(x_0, x_1, y_0, y_1, y_2, y_3) = \overline{x_0}\ \overline{x_1} y_0 + \overline{x_0}\ x_1 y_1 + x_0\ \overline{x_1} y_2 + x_0 x_1 y_3 \quad (3)$$

$$\text{11-multiplexer}(x_0, x_1, x_2, y_0, \ldots, y_7) = \overline{x_0}\ \overline{x_1}\ \overline{x_2}\ y_0 + \overline{x_0}\ \overline{x_1}\ x_2\ y_1 + \overline{x_0}\ x_1\ \overline{x_2}\ y_2 + \overline{x_0}\ x_1\ x_2\ y_3 + x_0\ \overline{x_1}\ \overline{x_2}\ y_4 + x_0\ \overline{x_1}\ x_2\ y_5 + x_0\ x_1\ \overline{x_2}\ y_6 + x_0\ x_1\ x_2\ y_7 \quad (4)$$

$$\text{20-multiplexer}(x_0, x_1, x_2, x_3, y_0 \ldots, y_{15}) = \overline{x_0}\ \overline{x_1}\ \overline{x_2}\ \overline{x_3}\ y_0 + \overline{x_0}\ \overline{x_1}\ \overline{x_2}\ x_3\ y_1 + \overline{x_0}\ \overline{x_1}\ x_2\ \overline{x_3}\ y_2 + \overline{x_0}\ \overline{x_1}\ x_2\ x_3\ y_3 + \overline{x_0}\ x_1\ \overline{x_2}\ \overline{x_3}\ y_4 + \overline{x_0}\ x_1\ \overline{x_2}\ x_3\ y_5 + \overline{x_0}\ x_1\ x_2\ \overline{x_3}\ y_6 + \overline{x_0}\ x_1\ x_2\ x_3\ y_7 + x_0\ \overline{x_1}\ \overline{x_2}\ \overline{x_3}\ y_8 + x_0\ \overline{x_1}\ \overline{x_2}\ x_3\ y_9 + x_0\ \overline{x_1}\ x_2\ \overline{x_3}\ y_{10} + x_0\ \overline{x_1}\ x_2\ x_3\ y_{11} + x_0\ x_1\ \overline{x_2}\ \overline{x_3}\ y_{12} + x_0\ x_1\ \overline{x_2}\ x_3\ y_{13} + x_0\ x_1\ x_2\ \overline{x_3}\ y_{14} + x_0\ x_1\ x_2\ x_3\ y_{15} \quad (5)$$

Fig. 1. The multiplexer functions represented as sum of *prime implicants*: the product corresponds to the logical *and*, the sum to the logical *or*, and the overline to logical *not*.

$$eq_{5,3}(x_0, x_1, x_2, x_3, x_4) = x_0 x_1 x_2 \overline{x_3}\ \overline{x_4} + x_0 x_1 \overline{x_2} x_3\ \overline{x_4} + x_0 x_1 \overline{x_2}\ \overline{x_3} x_4 + x_0 \overline{x_1} x_2 x_3\ \overline{x_4} + x_0 \overline{x_1} x_2 \overline{x_3} x_4 + x_0 \overline{x_1}\ \overline{x_2} x_3 x_4 + \overline{x_0} x_1 x_2 x_3 \overline{x_4} + \overline{x_0} x_1 x_2 \overline{x_3} x_4 + \overline{x_0} x_1 \overline{x_2} x_3 x_4 + \overline{x_0}\ \overline{x_1} x_2 x_3 x_4 \quad (6)$$

Fig. 2. The function $eq_{5,3}$ as sum of prime implicants.

id	condition	action	P	ϵ	F	N
51950	$\overline{x_0} y_0 y_1 y_2 + \overline{x_1} y_0 y_2 + \overline{x_0} x_1 y_1 + \overline{x_0}\ \overline{x_1} y_0$	1	1000	0.00	.35	52
55677	$\overline{x_0} x_1 y_1$	1	1000	0.00	.50	53
55268	$x_0 x_1 y_3 + x_0 \overline{x_1} y_2 + \overline{x_0}\ \overline{x_1} y_0$	1	1000	0.00	.38	66
54033	$x_0 x_1 y_3 + x_0 \overline{x_1} y_2 + \overline{x_0}\ \overline{x_1} y_0$	1	1000	0.00	.52	91
54217	$x_0 \overline{x_1}\ \overline{y_2}$	0	1000	0.00	.45	54
54818	$\overline{x_0}\ \overline{y_0}\ \overline{y_1}\ \overline{y_2} + \overline{x_0}\ \overline{x_1}\ \overline{y_0}\ \overline{y_1} + x_1 \overline{y_1}\ \overline{y_3} + x_0 \overline{x_1}\ \overline{y_2}$	0	1000	0.00	.35	66
55308	$x_0 x_1 \overline{y_3} + \overline{x_0} x_1 \overline{y_1} + \overline{x_0}\ \overline{x_1}\ \overline{y_0}$	0	1000	0.00	.79	159

TABLE I
EXAMPLE OF FINAL POPULATION EVOLVED BY XCSGP FOR THE 6-MULTIPLEXER.

one of the final populations evolved by XCSGP over the 20 runs; for the sake of clarity, *only* the classifiers with *non zero* prediction are reported. Each row corresponds to a classifier: `id` is a unique identifier; `condition` is the classifier condition after it has been simplified with SIS [4]; `action` is the action; `P` is the prediction; ϵ is the prediction error; `F` is the fitness; and finally, `N` is the numerosity. The solution in Table I is quite compact in that it gives solution to the 6-multiplexer in 7 classifiers. It also heavily exploits *or* clauses (symbolized by a "+"). Four classifiers (55677, 55268, 54217, 55308) represent 6-multiplexer completely (Figure 1): classifier 55677 covers the first prime implicant of the minimal DNF representation, while classifier 55268 covers the three remaining prime implicants; similarly, classifiers 54217 and 55308 cover respectively one and three of the four conditions that represent the zeros of 6-multiplexer. The classifiers in Table I show also some redundancies. For instance, two classifiers (55268 and 54033) have equivalent conditions since their simplified forms are identical. Furthermore, the third and fourth term in the condition of classifier 51950 cover two prime implicants already covered by classifiers 55268 and 54033 while the first and the second terms of the same classifier are specific and thus *subsumed* by other prime implicants which appear in the population.

As was to be expected, although the classifiers in Table I are accurate (in fact, their prediction error is zero), they have a very low fitness. Symbolic conditions tend to produce overlapping conditions, so that classifiers in the population share their fitness with other classifiers. More important, the population in Table I does not contain any of the two *maximally* general classifiers which represent the most compact solution for 6-multiplexer, i.e.: the classifier with condition "$\overline{x_0}\ \overline{x_1} y_0 + \overline{x_0}\ x_1 y_1 + x_0\ \overline{x_1} y_2 + x_0 x_1 y_3$", action "1", and prediction 1000; and the classifier with condition "$\overline{x_0}\ \overline{x_1}\ \overline{y_0} + \overline{x_0}\ x_1 \overline{y_1} + x_0\ \overline{x_1 y_2} + x_0 x_1 \overline{y_3}$", action "0", and prediction 1000. Interestingly, none of the

twenty populations evolved contained one of the above maximally general classifiers.

B. Classifier Fitness and or Clauses

Classifier fitness in XCS estimates the classifiers *relative* accuracy. A classifier has a high fitness is: (i) it is accurate, i.e., its *prediction error* ϵ is smaller than the threshold ϵ_0; and (ii) it is "relevant" within the environmental niches it matches, i.e., its numerosity N is large with respect to that of other classifiers matching the same niches. Given a certain learning problem and the ternary representation, there are usually few *maximally general* classifiers which accurately represent the optimal solution for each niche. And because such classifers are the most general they usually do not overlap so that for each niche there are few relevant classifiers with high fitness values. On the other hand, when we introduce *or* clauses the number of classifiers that are accurate and maximally general with respect to a certain niche grows exponentially. Given, n accurate maximally general conditions $C_1, \ldots, C_n$ that can be built using only conjuctions and negations, we can construct $2^n - n - 1$ accurate conditions that are more general than each C_j using logical disjunctions; one of these is the maximally condition "C_1 *or* $\ldots$ *or* C_n". Each one of these new conditions has a lower fitness with respect to the original C_j, but it has a more complex structure which can be more difficult to evolve. Accordingly, when *or* clauses are introduced, *maximally general* classifiers (which give the most compact representation of the solution) are *less likely* to be evolved since (i) they have a smaller fitness than that obtained with a non overlapping solution; furthermore, (ii) they have a more complex structure which may more difficult to evolve and maintain. The overall results is that XCSGP tends to evolve populations of *general* classifiers while rarely evolves *maximally general* classifiers which exploit *or* clauses. This suggests an explanation about why, in all the ten experiments with the 6-multiplexer, as well as in the next experiments with more complex problems, XCSGP never evolved one of the two maximally general classifiers presented in the previous section. It also explains why in [16], XCSGP could not reach optimality with *or* clauses. The original XCS [18], and also the first XCSGP [16], used an exponential fitness calculation and a deletion based on the classifier average action set size. The subsequent introduction of the potential fitness [2] and accuracy based deletion [10] dramatically improved the ability of XCS in distinguishing between classifiers with similar fitness, a feature which has dramatic effects when the system is dealing with populations of classifiers with very small fitness values.

C. Experiments with Larger Multiplexer

We extended the previous results and applied XCSGP to the 11-multiplexer. The parameters were set as in the previous experiments except for the population size which was set to 1000, 2000, and 4000 classifiers; condensation was activated after 50000 learning problems. Figure 4a compares the performance of XCSGP with different population sizes (dotted lines) against the performance of XCS

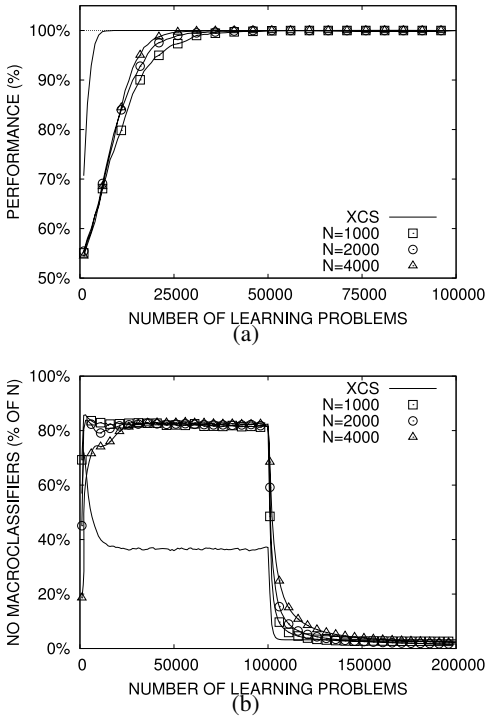

Fig. 4. XCSGP applied to the 11-multiplexer: (a) performance; (b) number of macroclassifiers in the population.

(solid line). XCSGP is slower than XCS, as expected, but it always reaches optimal performance around 40000 learning steps. The population size improves the learning speed only slightly. As it happened in the experiments with the 6-multiplexer, the number of macroclassifiers rapidly grows and stabilizes around the 80% of N. When condensation is activated, the population rapidly shrinks and at the end the population size is around 37 classifiers. Table II reports one of the populations evolved in one of the experiments reported in Figure 4 when 2000 classifiers are involved; for the sake of clarity, only classifiers with non-zero predictions are reported. The first eleven classifiers represent the 11-multiplexer function completely. As with 6-multiplexer the solution does not exploit many *or* clauses; the solution is also redundant: classifiers 276511 and 276822 represent exactly the same condition, as well as classifiers 272699 and 273897. Interestingly, we have one classifier, 265471, which has many *or*; note however that most of the terms involved are specific and only the last two terms belong to the minimal representation (Figure 1).

In the last experiments with Boolean multiplexer, we applied XCSGP to the 20-multiplexer with the same parameters employed in the previous experiments and a population size of 6000 classifiers; condensation starts after 400000 learning problems. Figure 5 compares (a) the performance of XCSGP with that of XCS and (b) the number of macroclassifiers in the population. The plots confirm the previous results:

id	condition	action	P	ϵ	F	N
275596	$\overline{x_0}\ \overline{x_2}y_0y_2 + \overline{x_0}x_1\overline{x_2}y_2$	1	1000	0.00	0.76	109
276511	$x_0x_1y_6y_7 + x_0x_1\overline{x_2}y_6 + x_0\overline{x_1}\ \overline{x_2}y_4$	1	1000	0.00	0.47	92
276822	$x_0x_1y_6y_7 + x_0x_1\overline{x_2}y_6 + x_0\overline{x_1}\ \overline{x_2}y_4$	1	1000	0.00	0.32	63
277942	$x_0x_1x_2\overline{y_0}\ \overline{y_4}\ \overline{y_5}y_6y_7 + x_0x_1x_2\overline{y_0}\ \overline{y_1}\ \overline{y_4}y_6y_7 + x_0\overline{x_1}\ \overline{x_2}y_4 + \overline{x_0}\ \overline{x_1}\ \overline{x_2}y_0$	1	1000	0.00	0.32	63
272699	$\overline{x_0}\ \overline{x_1}x_2y_1$	1	1000	0.00	0.49	56
273897	$\overline{x_0}\ \overline{x_1}x_2y_1$	1	1000	0.00	0.51	56
250068	$x_0x_1y_2y_6y_7 + \overline{x_0}\ \overline{x_1}\ \overline{x_2}y_0y_4 + x_0x_1x_2y_7$	1	1000	0.00	0.29	54
274411	$\overline{x_0}x_1x_2y_3$	1	1000	0.00	0.52	54
271790	$x_0\overline{x_1}x_2y_5$	1	1000	0.00	0.51	52
252597	$x_0x_1x_2y_0y_7 + x_0\overline{x_1}x_2y_5$	1	1000	0.00	0.43	50
269886	$\overline{x_0}x_1x_2y_3$	1	1000	0.00	0.48	50
269844	$x_1x_2\overline{y_3}\ \overline{y_7} + x_0x_1x_2\overline{y_7} + \overline{x_0}x_1x_2\overline{y_3}$	0	1000	0.00	0.41	61
272955	$x_0\overline{x_1}x_2\overline{y_5} + \overline{x_0}\ \overline{x_1}x_2\overline{y_1}$	0	1000	0.00	0.36	57
278127	$\overline{x_0}x_1\overline{x_2}\ \overline{y_2} + \overline{x_0}\ \overline{x_1}\ \overline{x_2}\ \overline{y_0}$	0	1000	0.00	1.00	149
265471	$x_0x_1x_2\overline{y_2}y_3y_4\overline{y_7} + x_1x_2\overline{y_3}\ \overline{y_4}\ \overline{y_6}\ \overline{y_7} + x_0x_1x_2\overline{y_4}\ \overline{y_6}\ \overline{y_7} + x_0x_1x_2\overline{y_4}y_6\overline{y_7} +$ $+x_0x_1x_2\overline{y_3}y_6\overline{y_7} + x_0x_1x_2y_3y_6\overline{y_7} + x_1x_2\overline{y_2}\ \overline{y_3}y_4\overline{y_7} + \overline{x_0}x_1x_2\overline{y_3} + \overline{x_0}\ \overline{x_1}x_2\overline{y_1}$	0	1000	0.00	0.64	107
276464	$x_0\overline{x_1}x_2\overline{y_5}y_6 + x_0\overline{x_1}\ \overline{y_4}\ \overline{y_5} + x_0\overline{x_1}\ \overline{x_2}\ \overline{y_4}$	0	1000	0.00	0.86	105
276735	$x_0x_1\overline{x_2}\ \overline{y_6}$	0	1000	0.00	1.00	106

TABLE II
EXAMPLE OF FINAL POPULATION EVOLVED BY XCSGP FOR THE 11-MULTIPLEXER.

XCSGP is slower than XCS but it always reaches optimal performance; in XCSGP the number of macroclassifiers rapidly reaches the 80% of N and when condensation is turned oon, XCSGP stays optimal while the population becomes more and more compact. Table III reports part of one of the populations evolved in the experiments of Figure 5; for the sake of clarity, only classifiers with action one and non-zero predictions are reported. Note that *or* clauses are present only in three classifiers, and the solution is basically the same that would be evolved with the standard ternary representation [19]. Accordingly, many classifiers have an high fitness value, near to one, since they do not share their niche with other classifiers.

D. Experiments with Boolean Equality

Boolean multiplexer allow many generalizations and their optimal solutions are generally very compact if compared to the number of variables involved. For instance, the solution of the 20-multiplexer, which is defined over 20 variables, is represented by just 16 prime implicants (Figure 1).

In the last experiment, we applied XCSGP to the Boolean equality function $eq_{5,3}$ which do not allow many generalizations (Figure 2). In fact, the prime implicants of $eq_{5,3}$ contains all the five input variables, i.e., the prime implicants are equivalent to the function minterms. Accordingly, in $eq_{5,3}$ the logical disjunctions, the *or* clauses, represent a major source of generalization.

We applied XCSGP to $eq_{5,3}$ with a population of 4000 classifiers; the remaining parameters are set as in the previous experiments; condensation was started after 120000 learning problems. Figure 6a compares the performance of XCSGP with that of XCS: as usual, XCS is faster than XCSGP, and they both reach optimal performance. In this case however, the difference in speed is more visible. This function allows a little generalization, since symbolic conditions tend to generalize more than typical ternary conditions, the learning is more difficult for XCSGP. On the other hand, generalization-wise, symbolic conditions can potentially provide better generalization since they can exploit the logical disjunctions. Figure 6b compares the number of classifiers in the population of XCSGP and XCS: as usual, the number of classifiers in the population of XCSGP rapidly reaches the 80% of N and it remains at the same level until condensation starts.

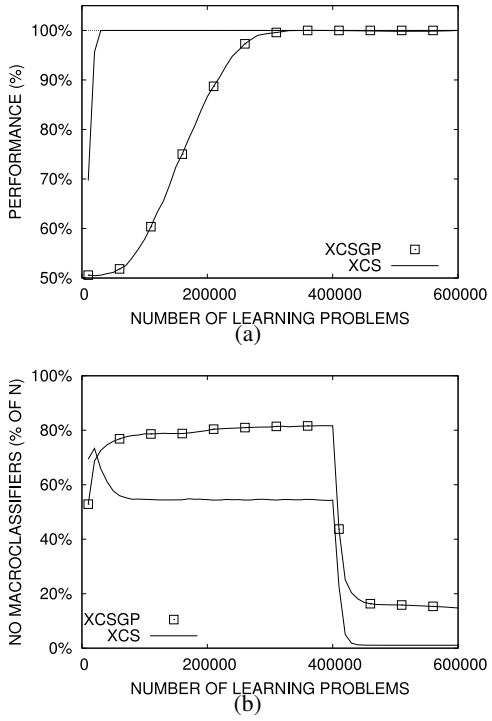

Fig. 5. XCSGP applied to the 20-multiplexer: (a) performance; (b) number of macroclassifiers in the population.

id	condition	action	P	ϵ	F	N
698500	$\overline{x_0}x_1x_2\overline{x_3}y_6$	1	1000	0.00	1.00	79
737937	$x_0x_1\overline{x_2}y_{12}y_{13} + x_0x_1\overline{x_2}x_3y_{13}$	1	1000	0.00	.87	78
736940	$x_0\overline{x_1}x_2x_3y_{11} + \overline{x_0}\ \overline{x_1}x_2\overline{x_3}y_2$	1	1000	0.00	.91	76
725614	$\overline{x_0}\ \overline{x_1}\ \overline{x_2}\ \overline{x_3}y_0$	1	1000	0.00	1.00	70
700885	$x_0x_1x_2x_3y_{15}$	1	1000	0.00	1.00	78
735086	$x_0\overline{x_1}\ \overline{x_2}\ \overline{x_3}y_8$	1	1000	0.00	1.00	67
738212	$\overline{x_0}x_1\overline{x_2}y_4y_5 + \overline{x_0}x_1\overline{x_2}x_3y_5$	1	1000	0.00	.84	66
695019	$\overline{x_0}x_1x_2x_3y_7\overline{y_{12}} + \overline{x_0}x_1x_2x_3y_7y_{12}$	1	1000	0.00	1.00	65
742769	$x_0\overline{x_1}\ \overline{x_2}x_3y_9$	1	1000	0.00	1.00	60
733211	$\overline{x_0}\ \overline{x_1}\ \overline{x_2}x_3y_1$	1	1000	0.00	1.00	59
690683	$x_0x_1x_2\overline{x_3}y_{14}$	1	1000	0.00	.76	52
737287	$x_0\overline{x_1}x_2\overline{x_3}y_{10}$	1	1000	0.00	.76	52
681771	$\overline{x_0}\ \overline{x_1}x_2y_2y_3 + \overline{x_0}\ \overline{x_1}x_2x_3y_3$	1	1000	0.00	.47	38
743063	$x_0x_1\overline{x_2}\ \overline{x_3}y_{12}$	1	1000	0.00	.59	34
734148	$\overline{x_0}\ \overline{x_1}x_2x_3y_3$	1	1000	0.00	.45	29
730923	$\overline{x_0}x_1\overline{x_2}\ \overline{x_3}y_4$	1	1000	0.00	.42	21
728723	$\overline{x_0}x_1\overline{x_2}\ \overline{x_3}y_4\overline{y_9} + \overline{x_0}x_1\overline{x_2}\ \overline{x_3}y_4\overline{y_5}$	1	1000	0.00	.38	19
739904	$x_0x_1x_2\overline{x_3}y_{14}$	1	1000	0.00	.24	17
733441	$x_0\overline{x_1}x_2\overline{x_3}y_{10}$	1	1000	0.00	.24	16

TABLE III
EXAMPLE OF POPULATION EVOLVED BY XCSGP FOR THE 20-MULTIPLEXER.

With a five bits problems, XCS needs just few classifiers to cover the whole search space and in fact the population always stays around the 5% of N, i.e., 150 classifiers. However, when condensation ends, the population evolved by XCSGP is more compact 1.1% of N instead of the 1.5% of N obtained by XCS.

Table IV reports one of the populations evolved in the experiments of Figure 6; as previously done, only classifiers with non-zero predictions are reported. In contrast with the solutions evolved for the Boolean multiplexer, the solutions for $eq_{5,3}$ highly exploit *or* clauses, both for action `1` and action `0`. Accordingly, classifiers have a low fitness since, as we discussed in Section VI-A, classifiers with *or* tend to share their niches with other classifiers. All the experiments produced solutions similar to that in Table IV. Thus, we argue that if the problem allows many generalizations, so that *or* clauses are less "convenient", XCSGP tends to develop solutions that exploit the available generalizations; if the problem allows few generalizations, so that *or* clauses become an interesting way to introduce simple generalizations, XCSGP tends to exploit them more.

VII. SUMMARY

In this paper, we analyzed generalization in XCSGP, the version of XCS with symbolic conditions based on genetic programming. We started from the earlier results reported in [16] where it was shown that XCSGP could not reach optimality when logical disjunctions were involved. We analyzed generalization in XCSGP and suggested why logical disjunctions make learning more difficult. Symbolic conditions tend to be highly overlapping especially when disjunctions are involved so that classifiers in XCSGP tend to have low fitness values. Accordingly, disjunctions make the learning more difficult since they tend to lower the fitness of classifiers in the population. For the same reason, in problems

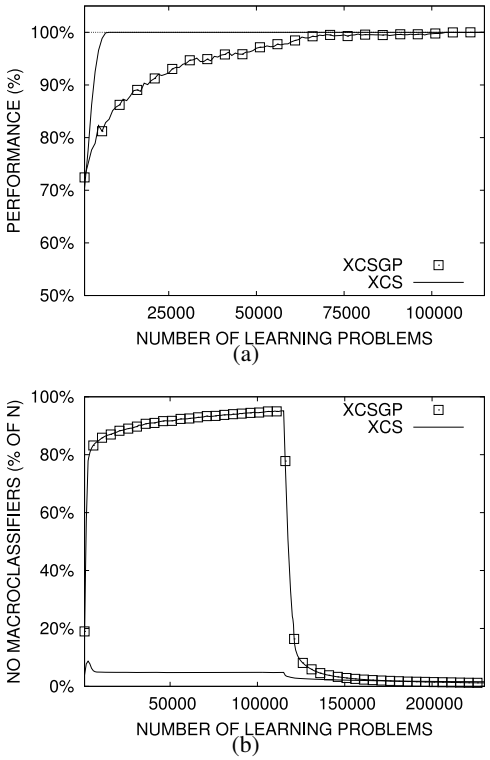

Fig. 6. XCSGP applied to the $eq_{5,3}$: (a) performance; (b) number of macroclassifiers in the population.

id	condition	action	P	ϵ	F	N
627089	$x_0x_1\overline{x_2}x_3\overline{x_4} + x_0\overline{x_1}x_2x_3\overline{x_4} + \overline{x_0}x_1x_2\overline{x_3}x_4$	1	1000	0.00	.49	165
634337	$x_0x_1x_2\overline{x_3}\ \overline{x_4} + \overline{x_0}x_1x_2x_3\overline{x_4} + \overline{x_0}x_1\overline{x_2}x_3x_4 + \overline{x_0}\ \overline{x_1}x_2x_3x_4$	1	1000	0.00	.24	98
621067	$x_0x_1\overline{x_2}\ \overline{x_3}x_4 + x_0\overline{x_1}\ \overline{x_2}x_3x_4$	1	1000	0.00	.24	66
629956	$x_0x_1\overline{x_2}\ \overline{x_3}x_4 + x_0\overline{x_1}x_2\overline{x_3}x_4 + \overline{x_0}x_1x_2\overline{x_3}x_4$	1	1000	0.00	.29	94
595256	$x_0\overline{x_1}\ \overline{x_2}x_3x_4 + \overline{x_0}\ \overline{x_1}x_2x_3x_4$	1	1000	0.00	.63	21
596192	$x_0x_1\overline{x_2}x_3\overline{x_4} + x_0\overline{x_1}x_2x_3\overline{x_4} + \overline{x_0}x_1x_2\overline{x_3}x_4$	1	1000	0.00	.15	56
604917	$x_0x_1x_2\overline{x_3}\ \overline{x_4} + \overline{x_0}x_1x_2x_3\overline{x_4} + \overline{x_0}x_1\overline{x_2}x_3x_4 + \overline{x_0}\ \overline{x_1}x_2x_3x_4$	1	1000	0.00	.06	23
605162	$x_0x_1\overline{x_2}\ \overline{x_3}x_4 + x_0\overline{x_1}\ \overline{x_2}x_3x_4$	1	1000	0.00	.03	9
643758	$\overline{x_1}\ \overline{x_3}\ \overline{x_4} + \overline{x_1}\ \overline{x_2}\ \overline{x_4} + \overline{x_1}\ \overline{x_2}\ \overline{x_3}$	0	1000	0.00	.39	154
643925	$\overline{x_0}\ \overline{x_1}\ \overline{x_3}x_4 + x_1x_2x_3x_4 + x_0x_1x_3x_4 + \overline{x_2}\ \overline{x_3}\ \overline{x_4} + \overline{x_1}\ \overline{x_2}\ \overline{x_4} + \overline{x_0}\ \overline{x_2}\ \overline{x_4}$	0	1000	0.00	.19	140
595543	$x_0x_1x_3x_4 + x_0x_1x_2x_3 + x_1x_2x_4 + \overline{x_0}\ \overline{x_3}$	0	1000	0.00	.28	205
635743	$x_0\overline{x_1}x_2x_3x_4 + \overline{x_0}\ \overline{x_1}x_2\overline{x_4} + \overline{x_1}\ \overline{x_2}\ \overline{x_4} + \overline{x_1}\ \overline{x_2}\ \overline{x_3}$	0	1000	0.00	.52	188
643831	$x_0x_1x_3x_4 + x_0x_1x_2x_4 + \overline{x_1}\ \overline{x_2}\ \overline{x_3} + \overline{x_0}\ \overline{x_2}\ \overline{x_3} + \overline{x_0}\ \overline{x_1}\ \overline{x_2}$	0	1000	0.00	.26	170
632262	$\overline{x_0}\ \overline{x_1}\ \overline{x_3}x_4 + x_1x_2x_3x_4 + x_0x_1x_3x_4 + \overline{x_2}\ \overline{x_3}\ \overline{x_4} + \overline{x_1}\ \overline{x_2}\ \overline{x_4} + \overline{x_0}\ \overline{x_2}\ \overline{x_4}$	0	1000	0.00	.16	121
631549	$\overline{x_0}\ \overline{x_1}\ \overline{x_3}x_4 + x_1x_2x_3x_4 + x_0x_1x_3x_4 + \overline{x_2}\ \overline{x_3}\ \overline{x_4} + \overline{x_1}\ \overline{x_2}\ \overline{x_4} + \overline{x_0}\ \overline{x_2}\ \overline{x_4}$	0	1000	0.00	.81	60
643190	$\overline{x_0}x_1\overline{x_2}\ \overline{x_3} + \overline{x_0}\ \overline{x_3}\ \overline{x_4} + \overline{x_0}\ \overline{x_2}\ \overline{x_4} + \overline{x_0}\ \overline{x_1}\ \overline{x_4} + \overline{x_0}\ \overline{x_1}\ \overline{x_2}$	0	1000	0.00	.16	92
643758	$\overline{x_1}\ \overline{x_3}\ \overline{x_4} + \overline{x_1}\ \overline{x_2}\ \overline{x_4} + \overline{x_1}\ \overline{x_2}\ \overline{x_3}$	0	1000	0.00	.39	154
638655	$\overline{x_0}\ \overline{x_1}\ \overline{x_3}x_4 + x_1x_2x_3x_4 + x_0x_1x_3x_4 + \overline{x_2}\ \overline{x_3}\ \overline{x_4} + \overline{x_1}\ \overline{x_2}\ \overline{x_4} + \overline{x_0}\ \overline{x_2}\ \overline{x_4}$	0	1000	0.00	.68	51
630566	$\overline{x_0}x_1\overline{x_2}\ \overline{x_3} + \overline{x_0}\ \overline{x_3}\ \overline{x_4} + \overline{x_0}\ \overline{x_2}\ \overline{x_4} + \overline{x_0}\ \overline{x_1}\ \overline{x_4} + \overline{x_0}\ \overline{x_1}\ \overline{x_2}$	0	1000	0.00	.08	45
608634	$\overline{x_0}x_1\overline{x_2}\ \overline{x_3} + \overline{x_0}\ \overline{x_3}\ \overline{x_4} + \overline{x_0}\ \overline{x_2}\ \overline{x_4} + \overline{x_0}\ \overline{x_1}\ \overline{x_4} + \overline{x_0}\ \overline{x_1}\ \overline{x_2}$	0	1000	0.00	.13	73

TABLE IV

EXAMPLE OF FINAL POPULATION EVOLVED BY XCSGP FOR $eq_{5,3}$.

that allow many generalizations, when disjunctions are less important for generalization, XCSGP tends evolve solutions with few disjunctions. However, in problems that allow few generalizations, when disjunctions are more useful, XCSGP exploits logical disjunctions more.

REFERENCES

[1] Wolfgang Banzhaf, Jason Daida, Agoston E. Eiben, Max H. Garzon, Vasant Honavar, Mark Jakiela, and Robert E. Smith, editors. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99)*. Morgan Kaufmann, 1999.

[2] Martin V. Butz and Stewart W. Wilson. An Algorithmic Description of XCS. In Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson, editors, *Advances in Learning Classifier Systems*, volume 1996 of *LNAI*, pages 253–272. Springer-Verlag, Berlin, 2001.

[3] Martin V. Butz and Stewart W. Wilson. An algorithmic description of xcs. *Journal of Soft Computing*, 6(3–4):144–153, 2002.

[4] E. M. Sentovich and K. J. Singh and L. Lavagno and C. Moon and R. Murgai and A. Saldanha and H. Savoj and P. R. Stephan and R. K. Brayton and A. Sangiovanni-Vincentelli. SIS: A system for sequential circuit synthesis. Technical Report UCB/ERL M92/41, University of California – Berkeley, May 1992.

[5] Christian Gagne and Marc Parizeau. Open BEAGLE: A new versatile C++ framework for evolutionary computation. In *Proceeding of GECCO 2002, Late-Breaking Papers*, New York, NY, USA, 2002.

[6] Christian Gagne and Marc Parizeau. *Open BEAGLE manual*. Laboratoire de Vision et Systemes Numeriques Departement de Genie Electrique et de Genie Informatique Universite Laval, Quebec (QC), Canada, G1K 7P4 E-mail: cgagne,parizeau@gel.ulaval.ca, January 2003.

[7] Gary D. Hatchel and Fabio Somenzi. *Logic Synthesis and Verification Algorithms*. Kluwer Academic Press, 1996.

[8] John H. Holland. Escaping Brittleness: The possibilities of General-Purpose Learning Algorithms Applied to Parallel Rule-Based Systems. In Mitchell, Michalski, and Carbonell, editors, *Machine learning, an artificial intelligence approach. Volume II*, chapter 20, pages 593–623. Morgan Kaufmann, 1986.

[9] John H. Holland. *Hidden Order: How Adaptation Builds Complexity*. 1996.

[10] Tim Kovacs. Deletion schemes for classifier systems. In Banzhaf et al. [1], pages 329–336. Also technical report CSRP-99-08, School of Computer Science, University of Birmingham. http://www.cs.bham.ac.uk/˜tyk.

[11] John Koza. *Genetic Programming*. MIT Press, 1992.

[12] Pier Luca Lanzi. Extending the Representation of Classifier Conditions Part I: From Binary to Messy Coding. In Wolfgang Banzhaf, Jason Daida, Agoston E. Eiben, Max H. Garzon, Vasant Honavar, Mark Jakiela, and Robert E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 99)*, pages 337–344, Orlando (FL), July 1999. Morgan Kaufmann.

[13] Pier Luca Lanzi. Mining interesting knowledge from data with the xcs classifier system. In Lee Spector, Erik D. Goodman, Annie Wu, W.B. Langdon, Hans-Michael Voigt, Mitsuo Gen, Sandip Sen, Marco Dorigo, Shahram Pezeshk, Max H. Garzon, and Edmund Burke, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 958–965, San Francisco, CA 94104, USA, 7-11 July 2001. Morgan Kaufmann.

[14] Pier Luca Lanzi. Learning classifier systems from a reinforcement learning perspective. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, 6(3):162–170, 2002.

[15] Pier Luca Lanzi. The xcs library. 2002.

[16] Pier Luca Lanzi and Alessandro Perrucci. Extending the Representation of Classifier Conditions Part II: From Messy Coding to S-Expressions. In Wolfgang Banzhaf, Jason Daida, Agoston E. Eiben, Max H. Garzon, Vasant Honavar, Mark Jakiela, and Robert E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 99)*, pages 345–352, Orlando (FL), July 1999. Morgan Kaufmann.

[17] Drew Mellor. A first order logic classifier system. In Hans-Georg Beyer and Una-May O'Reilly, editors, *GECCO*, pages 1819–1826. ACM, 2005.

[18] Stewart W. Wilson. Classifier Fitness Based on Accuracy. *Evolutionary Computation*, 3(2):149–175, 1995. http://prediction-dynamics.com/.

[19] Stewart W. Wilson. Generalization in the XCS classifier system. In *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 665–674. Morgan Kaufmann, 1998.

[20] Stewart W. Wilson. Mining Oblique Data with XCS. volume 1996 of *Lecture notes in Computer Science*, pages 158–174. Springer-Verlag, April 2001.

[21] Stewart W. Wilson. Classifiers that approximate functions. *Journal of Natural Computating*, 1(2-3):211–234, 2002.

[22] Stewart W. Wilson. Three architectures for continuous action. Technical Report 2006019, Illinois Genetic Algorithms Laboratory – University of Illinois at Urbana-Champaign, 2006.