Toward a Theory of Generalization and Learning in XCS

Martin V. Butz, Tim Kovacs, Pier Luca Lanzi, and Stewart W. Wilson

***Abstract*—In this paper, we take initial steps toward a theory of generalization and learning in the learning classifier system XCS. We start from Wilson's generalization hypothesis, which states that XCS has an intrinsic tendency to evolve accurate, maximally general classifiers. We analyze the different evolutionary pressures in XCS and derive a simple equation that supports the hypothesis theoretically. The equation is tested with a number of experiments that confirm the model of generalization pressure that we provide. Then, we focus on the conditions, termed "challenges," that must be satisfied for the existence of effective fitness or accuracy pressure in XCS. We derive two equations that suggest how to set the population size and the covering probability so as to ensure the development of fitness pressure. We argue that when the challenges are met, XCS is able to evolve problem solutions reliably. When the challenges are not met, a problem may provide intrinsic fitness guidance or the reward may be biased in such a way that the problem will still be solved. The equations and the influence of intrinsic fitness guidance and biased reward are tested on large Boolean multiplexer problems. The paper is a contribution to understanding how XCS functions and lays the foundation for research on XCSs learning complexity.**

***Index Terms*—Evolutionary computation, evolutionary learning, generalization, learning classifier systems (LCSs), XCS.**

I. Introduction

XCS [43] represents a major development in learning classifier systems research. Since its inception, XCS has provided repeatable results that are generally better than those produced by the majority of models developed since Holland's ground-breaking work [19]. XCS overcomes most of the shortcomings of previous models and has proved effective in many domains [34]. In data analysis applications, XCS can perform better than some well-known traditional machine learning techniques [3], [12], [32], [47]. In addition, XCS provides within a single paradigm representations of target concepts that in machine learning usually belong to different approaches, e.g., attribute value and interval representations [3], [12], [47], like those provided by C4.5 [39], CN2 [10], RISE [13], as well as simple relations among attributes [32] similar to those provided by FOIL [38]. Barry's Java implementation of XCS, JXCS [1], was used during the COIL2000 competition for the analysis of insurance data [16]. The data analysis system built by coupling JXCS with a powerful commercial tool for data preprocessing (namely Model-1) scored second in the competition. NuTech technologies developed a commercial version of XCS for data analysis applications NuTech Solutions Inc. [36] that is based on Wilson's XCSI [46]. In multistep problems, such as grid environments, XCS has shown interesting performance. In simple environments [43], [44], XCS achieved optimal performance and provided a minimal representation of the optimal solutions evolved. In more complex environments [2], [31], XCS showed certain weaknesses that could be solved by adopting specific techniques [29]. XCS has also been applied with success to simple robotic problems involving the real and the simulated Khepera robot [30], [45].

XCS differs from traditional models in several respects. First, XCS has a simplified structure since it does not have an internal message list. In addition, XCS uses a modification of Q-learning [41] instead of the "bucket brigade" [18]. Most important, in XCS, classifier fitness is based on the accuracy of the classifier's payoff prediction instead of the prediction itself as in Holland's framework [18].

The principles underlying evolution in XCS were first outlined in Wilson's [43] *generalization hypothesis*, which suggested that classifiers in XCS would evolve to be as general as possible without losing accuracy. Reference [21] extended Wilson's explanation to an *optimality hypothesis*, supported experimentally, in which he argued that XCS will develop minimal representations of optimal solutions. Over time, researchers have studied various aspects of XCSs learning capabilities, such as: the relation between XCS performance and problem complexity [26]–[28], [44], the performance of accuracy based fitness with respect to that of strength-based fitness [24], and the development of Markov models for XCS's genetic algorithm (GA) [4], [5]. Though these results provide insight, there are still basic aspects of XCS that are not clear. In essence, Wilson's hypothesis has not been investigated theoretically, so that precisely how XCS evolves accurate, maximally general classifiers is not well understood.

In this paper, we investigate Wilson's hypothesis and take the first steps toward a fundamental theory of XCS. The analysis focuses on XCS as a pure classifier. Features of multistep environments such as the propagation of reward are not addressed here. However, since our analysis concerns very general issues of evolution in XCS, our results should be readily applicable to multistep problems.

Manuscript received June 6, 2002; revised June 9, 2003. This work was supported in part by the National Science Foundationa under Grant DMI-9908252 and in part by the German Research Foundation (DFG) under Grant DFG HO1301/4–3. The work of M. Butz was supported in part by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under Grant F49620-00-0163.

M. V. Butz is with the Department of Cognitive Psychology, University of Würzburg, Würzburg 97070, Germany (e-mail: butz@psychologie.uni-wuerzburg.de).

T. Kovacs is with the Department of Computer Science, The University of Bristol, Bristol BS8 1UB, U.K. (e-mail: kovacs@cs.bris.ac.uk).

P. L. Lanzi is with the Dipartimento di Elettronica e Informazione, Politecnico di Milano, Milan 20133, Italy (e-mail: pierluca.lanzi@polimi.it).

S. W. Wilson is with Prediction Dynamics, Concord, MA 01742 USA, and also with the Department of General Engineering, University of Illinois at Urbana–Champaign, IL 61761 USA (e-mail: wilson@prediction-dynamics.com).

Digital Object Identifier 10.1109/TEVC.2003.818194

The analysis is organized in two parts. In the first part, we analyze the different evolutionary pressures that are present in XCS: the *set* pressure, due to the application of the GA in environmental niches with deletion acting on the whole population; the *mutation* pressure; the *deletion* pressure; and the *subsumption* pressure. For the set pressure, we develop an equation which demonstrates that the interaction between the niched GA and panmictic deletion results in an intrinsic tendency toward general classifiers. The equation, which supports Wilson's hypothesis from a theoretical standpoint, is confirmed through a number of experiments. In the second part, we focus on the tendency toward accurate classifiers that should result from the use of accuracy-based fitness. In particular, we investigate the general conditions that favor the existence of fitness (i.e., accuracy) pressure in XCS. We develop two equations that identify two boundaries—or, figuratively, *challenges*—that can decrease or even eliminate the fitness pressure. We call them the *covering challenge* and the *schema challenge*. The equations are highly useful for effective parameter selection. We suggest that, once the two challenges are met XCS is able to evolve a complete, accurate, and maximally general problem representation reliably. When the two challenges are not met, the problem may still contain intrinsic fitness guidance or biased reward that will permit evolution of a solution. With a set of experiments using the Boolean multiplexer, we show the boundaries induced by the two challenges as they show up in practice. Other results illustrate the effect of intrinsic fitness guidance and biased reward functions.

A. Paper Contributions

This paper provides three major contributions. First, through the analysis of the evolutionary pressures in XCS, the paper provides theoretical support for Wilson's hypothesis by showing that the interaction between niched evolution and panmictic deletion results in a generalization pressure. Second, through the definition of the covering challenge and of the schema challenge, the paper provides guidelines to set two main XCS parameters (the population size N, and the don't-care probability $P_\#$). Finally, our analysis forms the conceptual and mathematical basis for further ongoing investigations [7].

B. Paper Organization

This paper is organized as follows. In Section II, we provide a short overview of XCS, with all details that are important for the remainder of the paper. In Section III, we investigate the evolutionary pressures in XCS separately, and then discuss their interaction. The analysis is confirmed experimentally in Section IV. In Section V, we investigate how the pressure induced by accuracy-based fitness provides guidance toward accurate classifiers, emphasizing the two challenges, and possible solutions. This analysis is confirmed experimentally in Section VI. Finally, in Section VII, we draw conclusions.

All experiments in this paper were carried out with Butz's [6] implementation of XCS and were repeated with Lanzi's `xcslib` [33]. The configuration files required to reproduce the results are available on request, as are the result files for the statistical analysis.

II. XCS Classifier System

In this section, we give a short description of XCS. For a complete description, we refer the interested reader to the original papers by Wilson [43], [44], and to the recent algorithmic description by Butz and Wilson [8].

XCS acts as a reinforcement learning agent [40]: it receives inputs describing the current state of the environment $s(t) \in \mathcal{S}$, it reacts with actions (or classifications) $a(t) \in \mathcal{A}$, and eventually receives reward $R \in \mathbb{R}$ as an indication of the value of its actions. The goal of XCS is to maximize the amount of reward gathered in the long run. XCS achieves this by learning an *action-value* function [40], which maps *state-action* pairs into a real number, called *payoff*, which is analogous to the Q value of Q-learning [41].

While XCS works for either single-step or multistep problems, for present purposes we will focus on single-step problems in which the task is to maximize the *immediate reward* received from the environment as a direct consequence of an action (and not the cumulative, discounted reward in the long run). Moreover, we restrict inputs from the environment to binary strings. Accordingly, the input space is denoted by $\mathcal{S} \subseteq \{0,1\}^L$ where L is the fixed length of the input string.

Classifiers in XCS consist of a condition, an action, and three main parameters: The condition $C \in \{0,1,\#\}^L$ specifies which input states $s \in \mathcal{S}$ the classifier is capable of *matching* ($\#$ is a "don't-care" symbol). The action a specifies the action for which the payoff is predicted. The prediction p estimates the payoff that the system expects if the classifier matches and its advocated action is executed. The prediction error ε estimates the error in the payoff prediction p. The fitness F estimates the accuracy of the payoff prediction p.

XCS interacts with the environment as follows. When the system receives an input from the environment it forms a *match set* [M] of classifiers whose conditions are satisfied by the current input. If the match set [M] contains less than θ_{mna} classifiers with different actions, *covering* classifiers are created with a condition that matches the current input and a random action is selected from among those not in [M]. Specifically, each attribute in the condition of a covering classifier is set to $\#$ with a probability $P_\#$ and to the corresponding input symbol, otherwise. For each action a in [M], XCS computes the *system prediction* $P(a)$, which is an estimate of the payoff that the system expects when action a is performed. It is computed by the fitness-weighted average of all matching classifiers that specify action a. Following the notation in [9]

$$P(a) = \frac{\sum_{cl.a=a \wedge cl \in [M]} cl.p \times cl.F}{\sum_{cl.a=a \wedge cl \in [M]} cl.F} \tag{1}$$

where $cl.a$ is the action of classifier cl; $cl.p$ is the prediction of classifier cl; and $cl.F$ is the fitness of classifier cl. The different values of $P(a)$ form the *prediction array*. XCS selects an action with respect to the values in the prediction array. The classifiers in [M] that advocate the selected action are put in the *action set* [A]. Next, the selected action is sent to the environment and a reward R is returned to the system.

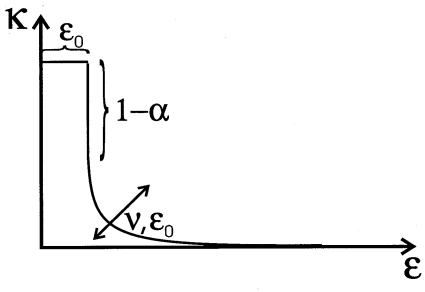

Fig. 1. The classifier accuracy κ as a function of the classifier prediction error ε. If the prediction error ε is below the threshold ε_0 the classifier is assumed to be accurate, i.e., it has an accuracy of one, otherwise, the classifier accuracy κ drops off quickly, based on the values of α and ν.

A. Reinforcement Component

During each cycle, XCS uses the reward R to update the parameters of the classifiers in $[\mathrm{A}]$. Note that *only* the classifiers in $[\mathrm{A}]$ are updated. Initially, the classifier prediction p is updated as follows:

$$p \leftarrow p + \beta(R - p) \tag{2}$$

where β $(0 < \beta \leq 1)$ denotes the *learning rate*; R the reward received from the environment. Next, the prediction error ε is updated

$$\varepsilon \leftarrow \varepsilon + \beta(|R - p| - \varepsilon). \tag{3}$$

The update of the classifier fitness F is slightly more complex. First, the classifier *accuracy* κ and the classifier *relative accuracy* κ' are computed as

$$\kappa = \begin{cases} 1, & \text{if } \varepsilon < \varepsilon_0 \\ \alpha\left(\frac{\varepsilon}{\varepsilon_0}\right)^{-\nu}, & \text{otherwise} \end{cases} \tag{4}$$

$$\kappa' = \frac{\kappa}{\sum\limits_{x \in [A]} \kappa_x}. \tag{5}$$

The parameter ε_0 $(\varepsilon_0 > 0)$ controls the tolerance for prediction error ε; the parameter α $(0 < \alpha < 1)$ and the parameter ν $(\nu > 0)$ are constants controlling the rate of decline in accuracy κ when ε_0 is exceeded. The classifier accuracy κ is calculated from the prediction error ε as follows [see (4)]: if the prediction error ε is below the threshold ε_0 the classifier is said to be *accurate* (it has an $\mathrm{accuracy} = 1$); otherwise, the accuracy κ drops off quickly, dependent on the values of α and ν. The accuracy values κ in the action set $[\mathrm{A}]$ are then converted to relative accuracies κ' [see (5)]. Finally, classifier fitness F is updated toward the classifier's current relative accuracy as follows:

$$F \leftarrow F + \beta(\kappa' - F). \tag{6}$$

The idea behind the accuracy calculation is visualized in Fig. 1. ε_0 is a threshold measuring the extent to which errors are accepted, α causes a strong distinction between accurate and not quite accurate classifiers, and the steepness of the succeeding slope is influenced by ν, as well as ε_0. To summarize, in XCS the classifier fitness is an estimate of the classifier's accuracy relative to other classifiers in $[\mathrm{A}]$ and behaves inversely to the reward prediction error; errors below the ε_0 threshold are regarded as having equal accuracy.

B. Discovery Component

In XCS, a GA [17] is applied to the classifiers in the current action set $[\mathrm{A}]$ if the average time since the last GA application to the classifiers in $[\mathrm{A}]$ exceeds a threshold θ_{ga}. For this purpose, each classifier keeps an additional parameter ts recording the last time that the classifier was in an action set to which the GA was applied. The GA selects two parental classifiers with probability proportional to their fitness. Two offspring are generated by reproducing, crossing, and mutating the parents. The offspring are inserted into the population. As happens in all the other models of classifier systems, parents stay in the population competing with their offspring. So far, with XCS, two types of mutation have been used, namely, *free mutation* [43] and *niche mutation* [8]. In free mutation, an attribute of the classifier condition is mutated to the other two possibilities with equal probability. In niche mutation, a classifier condition is mutated so that it still matches the current input, i.e., a don't-care symbol is mutated to the corresponding input value, while 0 or 1 is mutated to don't-care. Niche mutation generally results in a faster convergence time, whereas free mutation causes broader exploratory behavior, faster knowledge transfer and, thus, higher robustness.

C. Classifier Deletion

If the number of classifiers in the population $[\mathrm{P}]$ exceeds the threshold N, excess classifiers are deleted to keep the population size constant. The deletion process is applied to the classifiers in the whole population $[\mathrm{P}]$. It selects classifiers with probability proportional to an estimate of the size of the action sets that the classifiers occur in. This estimate is stored in the classifier *action set size* parameter as. If the classifier is sufficiently experienced and its fitness F is significantly lower than the average fitness of classifiers in $[\mathrm{P}]$, its deletion probability is further increased.

D. Macroclassifiers

In XCS, a *macroclassifier* technique is used to speed processing and provide a more perspicuous view of population contents. Macroclassifiers represent a set of classifiers with the same condition and the same action by means of a new parameter called *numerosity*. Whenever a new classifier is generated by the GA (or covering), $[\mathrm{P}]$ is scanned to see if there already exists a classifier with the same condition and action. If so, the *numerosity* parameter of the existing classifier is incremented by one, and the new classifier is discarded. If not, the new classifier is inserted into $[\mathrm{P}]$. The resulting population consists entirely of structurally unique classifiers, each with numerosity ≥ 1. If a classifier is chosen for deletion, its numerosity is decremented by 1, unless the result would be 0, in which case the classifier is removed from $[\mathrm{P}]$. All operations in a population of macroclassifiers are carried out as though the population consisted of conventional classifiers; that is, the numerosity is taken into account. In a macroclassifier population, the sum of numerosities equals N, the traditional population size. $[\mathrm{P}]$'s actual size in macroclassifiers, M is of interest as a measure of the population's space complexity.

E. Subsumption Deletion

Wilson [44] introduced two *subsumption deletion* procedures to broaden the generalization capability of XCS. The first procedure, *GA subsumption*, checks offspring classifiers to see whether their conditions are logically subsumed by the condition of an accurate and sufficiently experienced parent. If an offspring is "GA subsumed," it is not inserted in the population but the parent's numerosity is increased.

The second procedure, *action set subsumption*, searches in the current action set for the most general classifier that is both *accurate* and *sufficiently experienced*. Then, all the classifiers in the action set are tested against the general one to see whether it subsumes them and the subsumed classifiers are eliminated from the population.

III. Evolutionary Pressures in XCS

We begin our analysis by studying the different evolutionary pressures in XCS and their interaction. To accomplish this, we first analyze five different pressures in XCS separately: 1) set pressure; 2) mutation pressure; 3) deletion pressure; 4) subsumption pressure; and 5) fitness pressure. As will be seen over this and the next three sections, the interaction of the pressures leads toward a population consisting of accurate, maximally general classifiers.

A. Set Pressure

The basic idea behind the set pressure is that XCS reproduces classifiers in action sets [A], whereas it deletes classifiers from the whole population [P]. The *set pressure* is a *combination* of the selection pressure produced by the GA applied in [A] and the pressure produced by deletion applied in [P]. The set pressure was first identified by Wilson [43] in his *generalization hypothesis* and was later extended and experimentally investigated by Kovacs [22] who proposed the *optimality hypothesis*. Briefly, the reasoning behind the generalization hypothesis is that general classifiers appear more often in action sets [A] and, therefore, they are more often reproduced by the GA. This causes an intrinsic tendency toward generality that, combined with deletion from [P], we call the set pressure. Kovacs' *optimality hypothesis* [22] suggests that because of the set pressure, XCS can develop a *complete*, *accurate*, and *maximally compact* solution (i.e., an "optimal" solution) for a given problem.

To formalize the set pressure, we calculate the expected specificity $s([A])$ of the classifiers in an action set [A] with respect to the current expected specificity of the classifiers in the population [P], denoted by $s([P])$. Expected specificity measures the average proportion of non don't-care symbols in the conditions of classifiers of a particular classifier set.

At the beginning of an experiment, the specificity of the initial random population, $s([P])$ corresponds to the *don't-care probability* $P_\#$, i.e., $s([P]) = 1 - P_\#$. To calculate the specificity in the action set $s([A])$ from the specificity in the population, $s([P])$, we assume that the specificity in the population is binomially distributed, as is the case in a randomly generated population with don't-care probability $P_\#$. With this assumption, we can determine the probability that a randomly chosen classifier $cl \in [P]$ has specificity k/L as follows:

$$P\left(s(cl) = \frac{k}{L}\right) = \binom{L}{k} s([P])^k (1 - s([P]))^{L-k} \tag{7}$$

where cl is a classifier, L is the length of classifier conditions, and k is the number of *specified bits* in the condition, i.e., number of bits different from a don't-care symbol; and $s(cl)$ denotes the specificity of classifier cl. The equation essentially estimates the proportion of different specificities assuming an infinite population size.

The probability that a classifier cl matches a certain input S depends on the classifier specificity. To match, a classifier cl with specificity k/L must match all the k specific bits. This event has probability 0.5^k since each specific attribute matches with probability 0.5. Therefore, the proportion of classifiers in [P] with a specificity k/L that match in a specific situation is

$$\begin{aligned} &P\left(cl \text{ matches} \wedge s(cl) = \frac{k}{L}\right) \\ &= P\left(s(cl) = \frac{k}{L}\right) P\left(cl \text{ matches} | s(cl) = \frac{k}{L}\right) \\ &= P\left(s(cl) = \frac{k}{L}\right) 0.5^k \\ &= \binom{L}{k} \left(\frac{s([P])}{2}\right)^k (1 - s([P]))^{L-k}. \end{aligned} \tag{8}$$

To derive a specificity $s([M])$ of a match set [M], it is first necessary to specify the proportion of classifiers in [M] with specificity k/L given the population specificity $s([P])$. This proportion, $P(s([M]) = k/L | s([P]))$, can now be derived by

$$\begin{aligned} &P\left(s([M]) = \frac{k}{L} | s([P])\right) \\ &= \frac{P\left(cl \text{ matches} \wedge s(cl) = \frac{k}{L}\right)}{\sum_{i=0}^{L} P\left(cl \text{ matches} \wedge s(cl) = \frac{i}{L}\right)} \\ &= \frac{\binom{L}{k} \left(\frac{s([P])}{2}\right)^k (1 - s([P]))^{L-k}}{\sum_{i=0}^{L} \binom{L}{i} \left(\frac{s([P])}{2}\right)^i (1 - s([P]))^{L-i}} \\ &= \frac{\binom{L}{k} \left(\frac{s([P])}{2}\right)^k (1 - s([P]))^{L-k}}{\left(1 - \frac{s([P])}{2}\right)^L} \\ &= \binom{L}{k} \left(\frac{s([P])}{2 - s([P])}\right)^k \left(1 - \frac{s([P])}{2 - s([P])}\right)^{L-k}. \end{aligned} \tag{9}$$

To compute $s([M])$, we multiply actual specificity values k/L by the proportions $P(s([M]) = k/L | s([P]))$ and sum up the values to derive the resulting specificity of [M]. Since the action

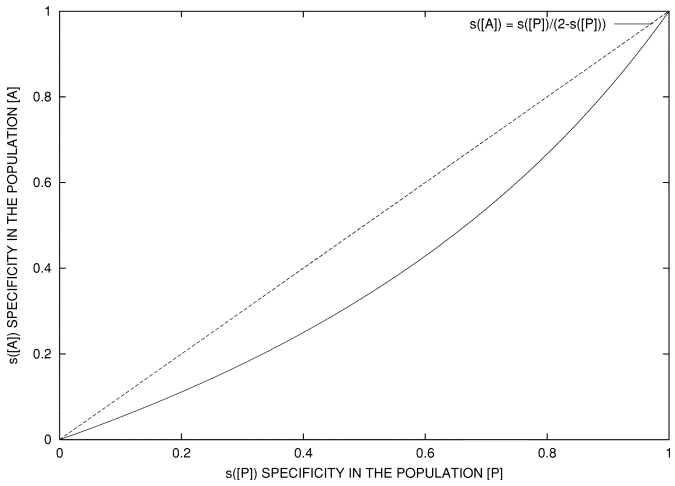

Fig. 2. Specificity of the action set [A] as a function of the specificity of the population [P]. Note that except at the end points $s([A])$ is always smaller than $s([P])$.

set [A] has *on average* the same specificity as the match set [M] ($s([A]) \approx s([M])$), $s([A])$ can be derived as follows:

$$\begin{aligned} s([A]) \approx & s([M]) \\ = & \sum_{k=0}^{L} \frac{k}{L} P(s([M]) = \frac{k}{L} | s([P])) \\ = & \sum_{k=1}^{L} \frac{k}{L} \binom{L}{k} \left(\frac{s([P])}{2-s([P])}\right)^k \left(1-\frac{s([P])}{2-s([P])}\right)^{L-k} \\ = & \sum_{k=1}^{L} \binom{L-1}{k-1} \left(\frac{s([P])}{2-s([P])}\right)^k \left(1-\frac{s([P])}{2-s([P])}\right)^{L-k} \\ = & \frac{s([P])}{2-s([P])} \\ & \cdot \sum_{j=0}^{L-1} \binom{L-1}{j} \left(\frac{s([P])}{2-s([P])}\right)^j \left(1-\frac{s([P])}{2-s([P])}\right)^{L-1-j} \\ = & \frac{s([P])}{2-s([P])}. \end{aligned} \tag{10}$$

The equation can be used to determine the mean specificity $s([A])$ in an action set [A] assuming a binomially distributed specificity with mean $s([P])$ in the population. This assumption is always valid in the beginning of an experiment when the population is initialized with respect to $P_{\#}$. In this case, the average specificity will be $1-P_{\#}$. Fig. 2 depicts (10). Note that (except at the end points) the specificity of [A] is always smaller than the specificity of [P]. Thus, since selection takes place in the action sets but deletion occurs in the population as a whole, there should be a tendency for the generality of the population to increase—in line with Wilson's generalization hypothesis. In the absence of fitness pressure, the equation provides an estimate of the difference in specificity of selected and deleted classifiers as long as an approximately binomial distribution is present. Equation (10) is enhanced in Section III-F and experimentally validated in Section IV.

B. Mutation Pressure

Although usually only a low mutation probability is applied, mutation still influences specificity. In the absence of other forces, mutation causes a population to tend toward specific proportions of zeros, ones, and don't-cares. Specifically, free mutation (Section II) pushes toward a distribution of *1:2* general:specific, while niche mutation pushes toward a distribution of *1:1* general:specific. The average change in specificity between the parental classifier $s(cl(t))$ and the mutated offspring classifier $s(cl(t+1))$ for the niche mutation case can be written as

$$\begin{aligned} \Delta_{mn} = & s(cl(t+1)) - s(cl(t)) \\ = & s(cl(t))(1-\mu) + (1-s(cl(t)))\mu - s(cl(t)) \\ = & \mu(1-2s(cl(t))). \end{aligned} \tag{11}$$

For free mutation, it is

$$\begin{aligned} \Delta_{mf} = & s(cl(t+1)) - s(cl(t)) \\ = & s(cl(t))\left(1-\frac{\mu}{2}\right) + (1-s(cl(t)))\mu - s(cl(t)) \\ = & 0.5\mu(2-3s(cl(t))). \end{aligned} \tag{12}$$

Thus, by itself, mutation pushes the population toward a specificity of 0.5 when niche mutation is applied and 0.66 with free mutation. The intensity of the pressure depends on the mutation type, on the frequency of the GA application (influenced by the parameter θ_{ga}), and on the mutation probability μ.

C. Deletion Pressure

The probability of a classifier being deleted depends on its action set size estimate *as* and (depending on classifier experience) its fitness F (Section II). Due to the resulting bias toward deleting classifiers that occupy larger action sets, deletion pushes the population toward an equal distribution of classifiers in each environmental niche. Therefore selection, from [P], of classifiers for deletion is essentially random, and there is no particular deletion pressure for or against general classifiers. In the absence of other biases, the specificity of deleted classifiers will be, *on average*, equal to the average specificity in the population $s([P])$.

D. Subsumption Pressure

The subsumption deletion mechanism adds another evolutionary pressure to XCS. The mechanism applies only to classifiers that are accurate and sufficiently experienced. Once accurate classifiers are found, subsumption deletion pushes toward maximal *syntactic* generality in contrast to the set pressure which only pushes toward generality if $\mathcal{S}$ contains inputs which permit more-general classifiers to be more active. In particular, *GA subsumption deletion* prevents the insertion into [P] of offspring whose parents are accurate, experienced, and formally more general than the offspring. *Action set subsumption* is stronger than GA subsumption since it allows an accurate more general classifier in an action set to eliminate all classifiers in the set that are more specific.

To summarize, subsumption pressure is an additional pressure toward accurate, maximally general classifiers (i.e., classifiers that are as general as possible while still being accurate)

from the over-specific side. It applies only once accurate classifiers are found. Thus, subsumption pressure is helpful mainly later in the learning process once accurate classifiers are found. It usually results in a strong decrease of population size.

E. Fitness Pressure

Until now, we have not considered the effect of fitness (accuracy) pressure which, as noted previously, can influence several other pressures. Fitness pressure is highly dependent on the particular problem being studied and is therefore difficult to formalize. In general, fitness results in a pressure which pushes [P] from over-general classifiers toward accurate classifiers as is further analyzed in Section V. Broadly speaking, fitness pressure toward accuracy counters the set pressure toward generality resulting in a population consisting primarily of accurate, maximally general classifiers.

F. Interaction of Pressures

We will now begin to combine the evolutionary pressures discussed so far and analyze their interaction. Initially, we consider the interaction of set pressure, mutation pressure, and deletion pressure which yields an important relationship we call the *specificity equation*. Next, we consider the effect of subsumption pressure. Finally, we provide a visualization of the interaction of all the pressures. The analyses are experimentally tested in Section IV.

Specificity Equation: Set pressure, mutation pressure, and deletion pressure all influence the average specificity in the population. Due to the problem dependence of fitness pressure, we cannot formulate that pressure and consequently will assume a similar fitness of all classifiers in our analysis. As shown later in Section IV, this assumption *holds* when all classifiers are accurate and nearly holds when all are similarly inaccurate.

Despite the fitness equality assumption, deletion is also dependent on the action set size estimate as of a classifier. However, in accordance with Kovacs's insight on the relatively small influence of this dependence [23], we assume a random deletion from the population in our formulation. Thus, as stated above, a deletion results on average in the deletion of a classifier with a specificity equal to the specificity of the population $s([P])$. The generation of an offspring, on the other hand, results in the insertion of a classifier with an average specificity of $s([A]) + \Delta_{mx}$ ($x \in f, n$) dependent on the type of mutation used. Putting the observations together, we can now calculate the average specificity of the resulting population after one time step

$$s([P(t+1)]) = s([P(t)]) + f_{\text{ga}} \times \frac{2(s([A]) + \Delta_{mx} - s([P(t)]))}{N}. \tag{13}$$

The parameter f_{ga} denotes the frequency of a GA application per time step. The formula adds to the current specificity in the population $s([P(t)])$ the expected change in specificity calculated as the difference between the specificity of the two reproduced and mutated classifiers, i.e., $s([A]) + \Delta_m$ and $s([P(t)])$. Note that although the frequency f_{ga} is written as a constant in the equation, f_{ga} actually depends on $s([P(t)])$, as well as the specificity distribution in the population. Thus, in general f_{ga} cannot be written as a constant. However, by setting θ_{ga} to 1, it is possible to force f_{ga} to be 1 since the average time since the last application of the GA in an action set (not generated by covering) will always be at least 1.

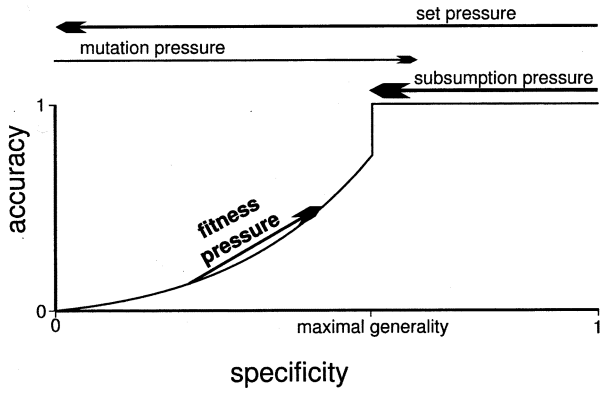

Fig. 3. Interaction of different pressures in XCS: 1) the fitness pressure pushes [P] toward accuracy where the slope and the initial gap are controlled by parameters ν, ε_0, and α as shown in Fig. 1; 2) the set pressure pushes [P] toward classifiers that are general with respect to the input set $\mathcal{S}$; 3) the optional subsumption pressure pushes the population toward classifiers that are *syntactically* general; and 4) the mutation pressure pushes to a fixed proportion of symbols in classifier conditions. Overall, these pressures lead the population toward a population of accurate maximally general classifiers.

Subsumption: XCS's tendency toward accurate, maximally general classifiers is not dependent on the use of the subsumption deletion operations which, as noted earlier, are optional. For this reason, and because we have not developed an analysis of subsumption pressure, it is left out of the specificity equation and is included only qualitatively in Fig. 3. Subsumption is not used in the experiments of Section IV.

Interaction of all Pressures: The interaction of all the pressures is visualized in Fig. 3. In particular, the fitness pressure pushes [P] toward more accurate classifiers; the set pressure pushes [P] toward more general classifiers; the subsumption pressure pushes the population toward classifiers that are accurate and syntactically maximally general; the mutation pressure pushes toward a fixed proportion of symbols in classifier conditions. Deletion pressure is implicitly included in the notion of set pressure. More detailed effects of deletion are not depicted. Overall, these pressures lead the population toward a population of accurate maximally general classifiers. While set pressure and mutation pressure (free mutation is represented) are independent of classifier accuracy, subsumption pressure and of course fitness pressure are influenced by accuracy.

IV. Validation of the Specificity Equation

We now present a set of experiments to validate the influences of the evolutionary pressures identified in Section III. In particular, we validate the specificity equation, formulated in (13), which summarizes the effect of the three main evolutionary pressures in XCS: set pressure, mutation pressure, and deletion pressure.

We apply XCS to Boolean strings of length $L = 20$ with different settings. The following figures show runs with mutation rates varying from 0.02 to 0.20. In each plot, solid lines denote the result from (13); while dotted lines represent the result of actual XCS runs. Curves are averages over 50 runs. If

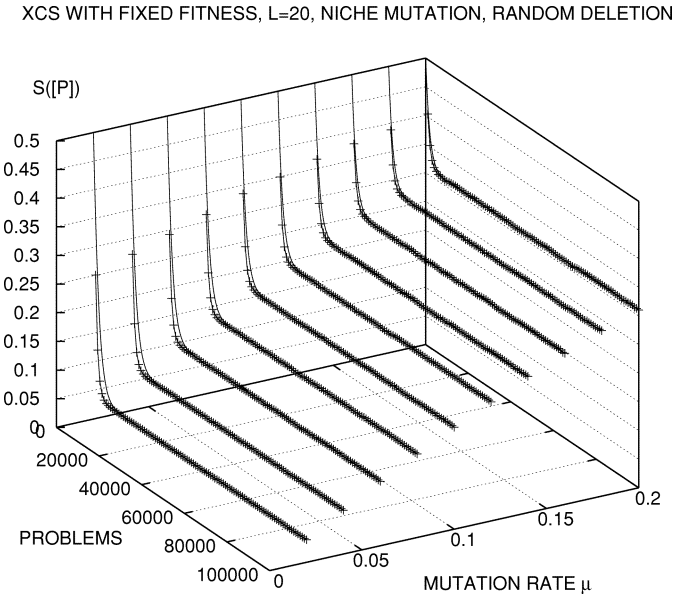

Fig. 4. Specificity of [P] when the classifier fitness is fixed, deletion is random, and niche mutation is used. Solid lines represent the specificity as predicted by (13). Dotted lines represent the actual specificity in [P]. Note that without the fitness influence, the actual specificity in XCS behaves nearly exactly as predicted by the model given in (13).

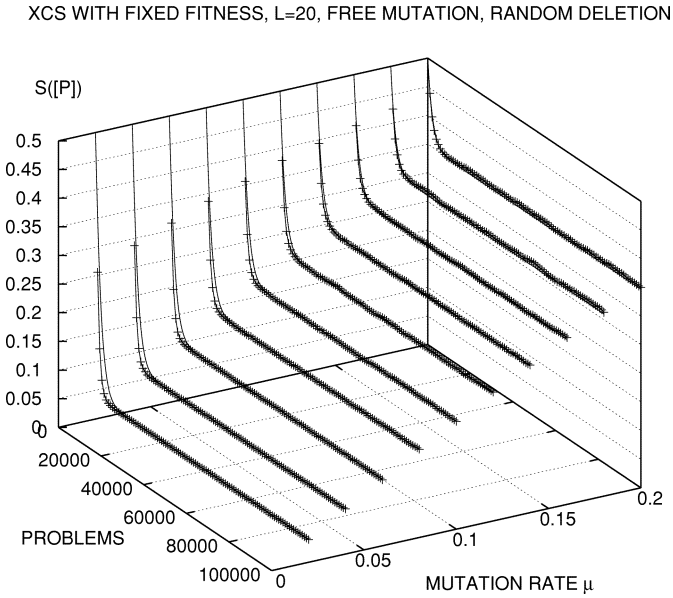

Fig. 5. Specificity of [P] when the classifier fitness is fixed, deletion is random, and free mutation is used instead of niche mutation. Solid lines represent the specificity as predicted by (13). Dotted lines represent the actual specificity in [P]. As predicted, the pressure toward specificity caused by free mutation is higher than the one caused by niche mutation.

not stated differently, the population is initially filled up with random classifiers with don't-care probability $P_{\#} = 0.5$. *Niche mutation* is applied. The other XCS parameters are set as follows: $R = 1000$; $N = 2000$; $\beta = 0.2$; $\theta_{mna} = 2$; $\alpha = 0.1$; $\varepsilon_0 = 10$; $\nu = 5$; $\theta_{\rm ga} = 1$; $\chi = 0.8$, $\theta_{del} = 20$; $\delta = 0.1$; and $\theta_{\rm sub} = \infty$. Note that the discount factor γ is irrelevant here since this are classification or single-step problems. Since this section is concerned with the set pressure, subsumption is turned off to prevent the additional generalization effect due to the subsumption pressure.

A. Fixed Fitness

We begin the validation of (13) examining runs where there is neither fitness pressure nor deletion pressure. Fitness pressure as well as deletion pressure are eliminated by deleting classifiers randomly. With these settings, we now investigate the influence of: 1) free mutation; 2) niche mutation; 3) the GA threshold $\theta_{\rm ga}$; and 4) the initialization of the population.

Experiment 1. Niche Mutation: Fig. 4 depicts the specificity of the population $[{\rm P}]$ when the fitness is fixed, deletion is random, and niche mutation is used. As the plot in Fig. 4 shows, the runs match very closely to the model expressed in (13). The initial specificity of 0.5 drops off quickly in the beginning due to the strong set pressure. However, soon the effect of the mutation pressure becomes visible and the specificity in the population converges as predicted. Furthermore, we note that the higher the mutation rate μ, the stronger the influence of mutation, which is manifested in the higher convergence value in the curves with higher μ.

Experiment 2. Free Mutation: Fig. 5 depicts the specificity of the population $[{\rm P}]$ when free mutation is used. Although the mutation pressure becomes visible in the variation of μ, Fig. 5 further reveals the influence of mutation. As formulated in (12), the mutation pressure is slightly higher when applying free mutation. When directly comparing Figs. 4 and 5, we note that the higher the parameter μ, the higher the influence of mutation pressure and, thus, the higher the differences in the two mutation types.

Experiment 3. The $\theta_{\rm ga}$ Threshold: As noted in Section III-F, the GA frequency $f_{\rm ga}$ in (13) should not be written as a constant value, since it actually depends on the specificity $s([P(t)])$ of the population. However, the GA frequency $f_{\rm ga}$ equals 1 when the GA threshold $\theta_{\rm ga}$ is set to 1. When setting $\theta_{\rm ga}$ to a higher value (100), Fig. 6 reveals the lower GA frequency effect. Once the specificity in the population has dropped, the action set sizes increase since more classifiers match a specific state. Consequently, more classifiers take part in a GA application, more time stamps ts are updated, the average time since the last GA application in the population and in the action sets decreases, and finally, the GA frequency decreases. The decrease is observable in the slower specificity decrease. However, as predicted by (13), despite its dependence on the actual specificity, $f_{\rm ga}$ does not influence the convergence value.

Experiment 4. The Initialization of $[{\rm P}]$: Until now, we initialized the population with random classifiers. This hypothesis assures a perfect binomial specificity distribution in the beginning of the run. However, the hypothesis of an initial random population appears not to be strictly necessary. Fig. 7 reports runs in which this hypothesis is relaxed. The population is initially empty and first classifiers are generated by covering. The only noticeable effect in Fig. 7, with respect to Fig. 4, is that in the very beginning of a run the specificity drops off slightly faster than in the case of an initial random population (see Fig. 4). This is easily explained: since the population does not contain 2000 classifiers initially, the specificity pressure is stronger, as also observable in (13) when N is initially smaller than 2000.

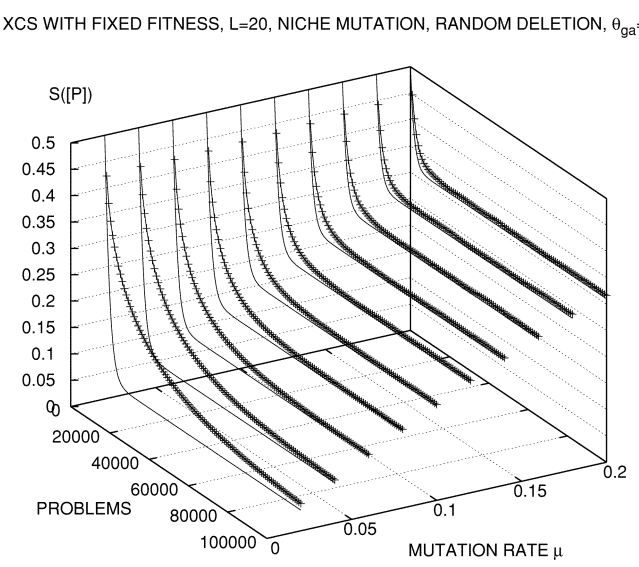

Fig. 6. Specificity of $[\mathrm{P}]$ when the classifier fitness is fixed, deletion is random, niche mutation is used, and the threshold $\theta_{\mathrm{ga}} = 100$. Solid lines represent the specificity as predicted by (13). Dotted lines represent the actual specificity in $[\mathrm{P}]$. Note that when using a threshold θ_{ga} of 100 instead of 1, the GA frequency and, consequently, the specificity pressure decreases once the specificity drops.

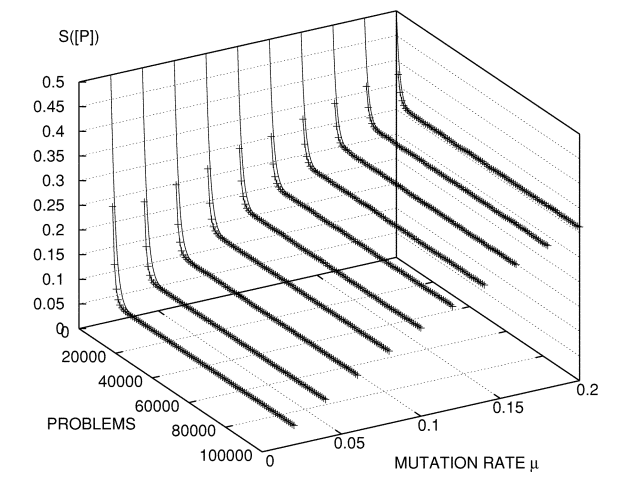

Fig. 7. Specificity of $[\mathrm{P}]$ when the classifier fitness is fixed, deletion is random, niche mutation is used, and population is initially empty. Solid lines represent the specificity as predicted by (13). Dotted lines represent the actual specificity in $[\mathrm{P}]$. With an initially empty population, the specificity drops slightly faster in the beginning.

B. Constant Function

While in the previous section fitness has been intentionally omitted, in this section and in the next one we study the actual influence of fitness on $s([P])$. In particular, in this section, we apply XCS to a constant Boolean function which always returns a reward of 1000. With these settings, all classifiers turn out to be accurate since their prediction error is always zero. Note, however, that a zero prediction error does not necessarily mean constant fitness values. In fact, since fitness is determined as the classifier's *relative accuracy*, fitness can still influence evolutionary pressure. Fig. 8 reports runs in which random deletion is used (i.e., deletion is proportional to numerosity). It also shows

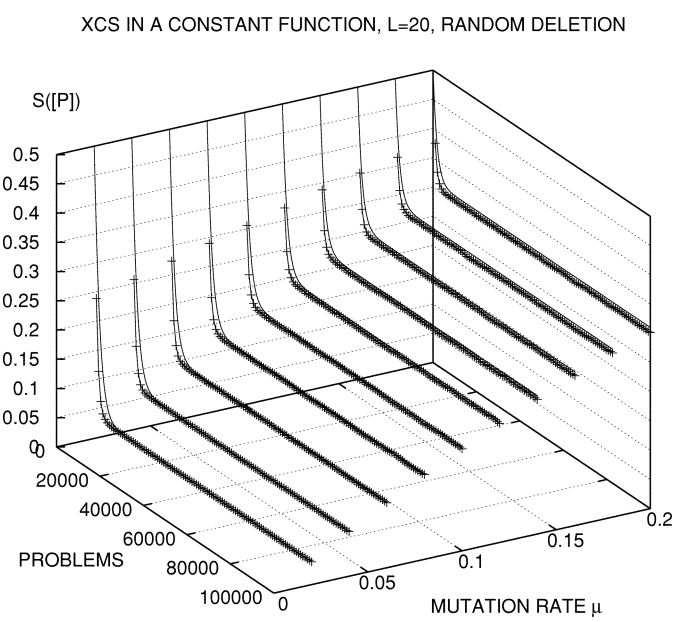

Fig. 8. Specificity of $[\mathrm{P}]$ when XCS is applied to a constant function. Deletion is random, niche mutation is used. Solid lines represent the specificity as predicted by (13). Dotted lines represent the actual specificity in $[\mathrm{P}]$. Note that when applied to a constant function with random deletion, the changing specificity still matches the proposed theory.

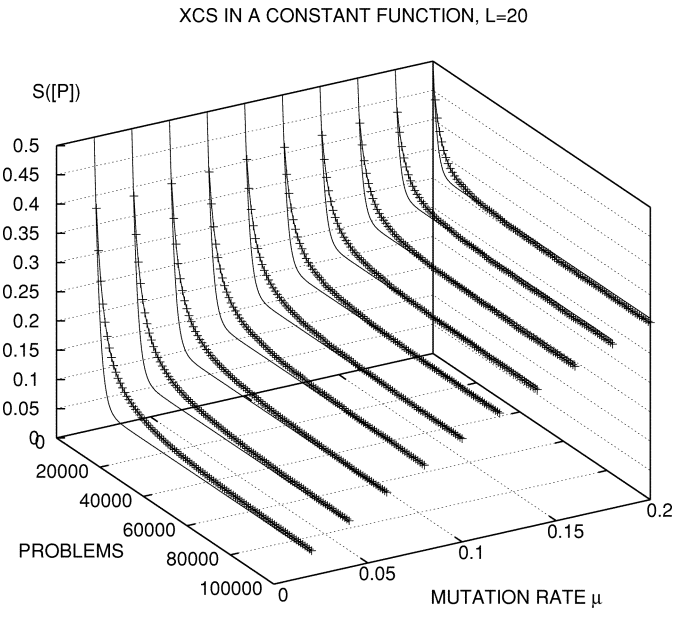

Fig. 9. Specificity of $[\mathrm{P}]$ when XCS is applied to a constant function; deletion is based on the action set size parameter as, niche mutation is used. Solid lines represent the specificity as predicted by (13). Dotted lines represent the actual specificity in $[\mathrm{P}]$. Due to the slower adaptation of the action set size estimate parameter, specificity convergence takes longer when the usual deletion is applied.

that the assumption of a binomial distribution indeed holds later in the run, or is at least not too harsh since the specificity behaves exactly as predicted.

But the behavior of $s([P])$ changes when we apply the deletion method used in XCS, based on the action set size parameter as. Fig. 9 reports runs in which the population is initially empty and the usual deletion is used. Note that in Fig. 9, the slope of the curves decreases. However, in the end the specificity of $[\mathrm{P}]$

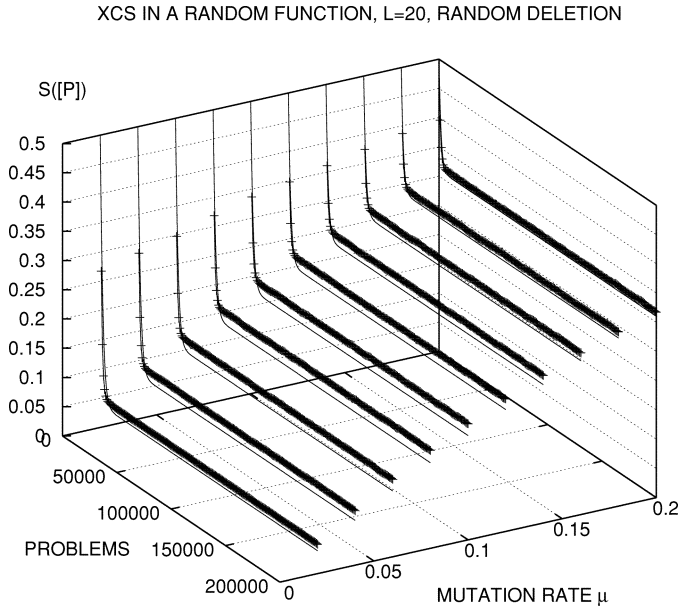

Fig. 10. Specificity of [P] when XCS is applied to a random Boolean function; deletion is random; niche mutation is used. Solid lines represent the specificity as predicted by (13). Dotted lines represent the actual specificity in [P]. Applied to a random function, the specificity stays on a higher level due to the continuous larger noise in more specific classifiers.

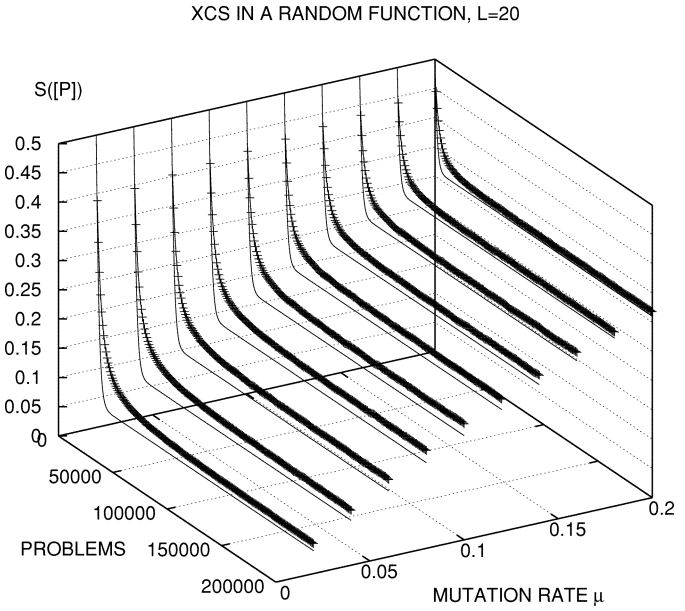

Fig. 11. Specificity of [P] when XCS is applied to a random Boolean function; niche mutation is used. Solid lines represent the specificity as predicted by (13). Dotted lines represent the actual specificity in [P]. The fitness biased deletion method further influences the specificity.

converges near the value predicted by the theory. The difference between the results in Fig. 9 and those in Fig. 8 is easily explained noting that the difference can only be caused by the bias of the deletion method toward classifiers in larger niches. As the specificity of [P] decreases, the action set size increases as noted before. Thus, since more general classifiers are more often present in action sets, their action set size estimate as is more sensitive to the change in the action set size and, consequently, it is larger in more general classifiers while specificity drops. Eventually, all as values will have adjusted to the change and the predicted convergence value is met. This explanation is further confirmed by the fact that the difference between the actual runs and the curves given by (13) become smaller and equal faster for higher mutation rates μ since the specificity slope is not as steep as in the curves with lower μ values.

C. Random Function

The results in the previous section show that the influence of the fitness in XCS with a constant function is rather small. Accordingly, we now apply XCS with the two different deletion strategies, to a much more challenging problem: a random Boolean function which randomly returns rewards of 1000 and 0. Fig. 10 reports the runs in which XCS with random deletion is applied to the random function. Fig. 10 shows that in the case of a random function the fitness influences the specificity slope as well as the convergence value. In fact, the convergence value is larger than that predicted by the model in (13). Two factors cause this effect: 1) the parameter initialization technique and 2) the high variance in low-experience classifiers. Since the possible rewards are 0 and 1000 and assuming accurate parameter estimates in a classifier, classifier predictions fluctuate around 500, and consequently also the prediction errors fluctuate around 500. As in the case of the more sensitive action set size estimates in Section IV-B, here, the sensitivity is manifested in the prediction error ε. More specific classifiers have a less sensitive ε and, consequently, a higher variance in the ε values. Since the accuracy calculation expressed in (4) scales the prediction error to the power ν—which is set to the usual value 5—the higher variance causes *on average* higher accuracy and, thus, higher fitness. Different parameter initialization techniques in combination with the moyenne adaptive modifiée technique enhance this influence. The more a classifier is inexperienced, the more the classifier parameters are dependent on the most recent cases. This, in combination with the scaled fitness approach, can make the effect even stronger. Since we set the experience $\exp$ of a new classifier to 1 (which differs from the algorithmic description in which $\exp$ is set to 0), XCS keeps the decreased parental parameter estimates so that fitness overestimation is prevented. In fact, experimental runs with $\exp = 0$ show that the specificity can increase to a level of even 0.2 independent of the mutation setting when $\exp = 0$ or the prediction error estimate ε is set to 0 initially.

When applying the usual deletion strategy, based on as and the fitness estimate f, deletion causes an increase in the specificity of [P] early on as shown in Fig. 11. This longer convergence time is attributable to the bias on as as already observed in Fig. 9. The additional fitness bias causes hardly any observable influence.

Overall, it can be seen that in a random function fitness causes a slight intrinsic pressure toward higher specificity. This pressure is due to the parameter initialization method and the higher variance in more specific classifiers. The on-average higher fitness in more specific classifiers causes fitness pressure and deletion pressure to favor those more-specific classifiers; thus, the resulting undirected slight pressure toward higher specificity.

Note that the specificity change and the convergence to a particular specificity level observed in Fig. 11 should essentially take place in all problems that are similar to a random function. As we will see, this is particularly the case if classifiers are over-general and the investigated problem provides no fitness guidance from the over-general side in the form of layered reward or biased generality.

V. Evolving Accurate Classifiers

In the previous sections, we analyzed the different evolutionary pressures in XCS. With (13), we provided a model to predict the change of specificity in the population. The experiments in Section IV confirmed the validity of the model.

Although the experiments revealed fitness influences, we did not consider the fitness pressure explicitly since, as noted in Section III, fitness pressure depends heavily on the problem definition. Therefore, it is impossible to formalize a problem-independent fitness pressure. Nevertheless, we can still investigate what are the *general conditions* in XCS that guarantee that fitness pressure *applies*. In this section, we derive conditions which must be satisfied to guarantee that there *is* fitness pressure. The derived conditions, termed *challenges*, are worst case bounds. Several common problem properties are outlined that ease the derived boundaries. Note that we do not provide a formal model of how fitness pressure *acts*; instead we analyze the conditions which favor the *existence* of fitness pressure in XCS.

There are two general conditions which must be satisfied to guarantee that fitness pressure exists. The first (rather obvious) condition is that classifier fitness must be *meaningful*, i.e., the XCS parameters must be set so as to guarantee that the GA acts and that classifiers stay in the population long enough to have their fitness adequately evaluated through on-line experience. The second (more problem-dependent) condition is that the application of the GA must result in an effective pressure toward high fitness classifiers, i.e., toward *accurate* classifiers. Each of these conditions can be viewed as a *challenge* which must be met to guarantee that fitness pressure exists. We call these challenges, respectively: the *covering* challenge and the *schema* challenge.

A. Covering Challenge

The covering challenge deals with the problem of setting XCS parameters so that the GA takes place on meaningful fitness values. In fact, classifier fitness is not just *computed* as in GAs, but must be evaluated through online experience. This requires that classifiers be applied enough times to allow an adequate evaluation of their fitness. If this is not the case, classifier fitness does not provide any information about the problem solution and therefore fitness pressure does not apply through the GA.

We remember from Section II that in XCS covering usually occurs at the beginning of a run. Then, when most of the possible input configurations are *covered* by some classifier, the GA starts acting in the action sets. However, under certain circumstances, covering might go on indefinitely because there continue to be some inputs that are not covered. When this happens, XCS enters a *cover-delete* cycle which prevents classifiers from remaining in [P] long enough to have their fitness evaluated. More precisely, at the beginning, [P] has an average specificity which depends on the specificity of initial classifiers, determined by the parameter $P_\#$. When XCS receives an input configuration, it builds the match set [M]. If the specificity of [P] is too high due to a small $P_\#$, [P] tends to contain overspecific classifiers and the complete input space might not be well covered. As a consequence, covering is likely to be applied when building [M] and one or more classifiers are inserted in the population. At the same time, excess classifiers must be deleted to keep the population size constant. Note that in this early stage classifiers have little experience and therefore both their fitness F and their action set size estimate as are basically meaningless. Thus, deletion will select classifiers essentially at random. Consequently, important classifiers that cover other input configurations will often be deleted. The result is a continual cover-delete cycle in which XCS tries to cover the new inputs with classifiers that are overspecific, while it randomly deletes other covering classifiers to make room for the new ones. Separately, the GA selects classifiers for reproduction randomly, since classifier fitness is not reliable due to limited experience. Overall these three effects cause a lack of significant fitness pressure among the classifiers in [P]. Clearly, the covering challenge can be met by setting $P_\#$ and N so that inputs have a reasonable probability of being matched so as to avoid the firing of the covering operator.

The situation can be formalized by determining the probability $P(\text{cover})$ that an input is covered by at least one classifier in a randomly generated population, as a function of the specificity in the population $s([P])$ (initially equal to $1 - P_\#$). First, we compute the probability $P(\text{match}))$ that a random classifier matches an input

$$\begin{aligned} P(\text{match}) &= \left(\frac{s([P])}{2} + (1 - s([P]))\right)^L \\ &= \left(\frac{2 - s([P])}{2}\right)^L \end{aligned} \tag{14}$$

where L is the length of the input string (see Section II). Then, we compute the probability $P(\text{no match in [P]})$ that no classifier in a randomly generated population matches the current input string

$$P(\text{no match in } [P]) = (1 - P(\text{match}))^N$$

where N denotes the size of the population. Finally, the probability $P(\text{cover})$ that at least one classifier matches the current input string is derived from the previous equations as

$$\begin{aligned} P(\text{cover}) &= 1 - P(\text{no match in } [P]) \\ &= 1 - (1 - P(\text{match}))^N \\ &= 1 - \left(1 - \left(\frac{2 - s([P])}{2}\right)^L\right)^N. \end{aligned} \tag{15}$$

Equation (15) is correct for the usual ternary coding when the population [P] is initially filled up with random classifiers. Moreover, the equation assumes a uniform and complete distribution of all possible 2^L binary problem instances. In real-world problems, this property can be often relaxed since a data set usually does neither contain all expressible problem instances nor

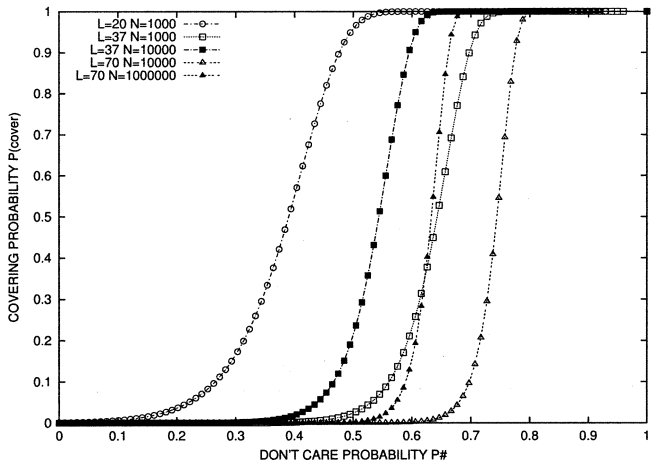

Fig. 12. $P(\text{cover})$ as a function of the don't-care probability $P_\#$ (initially $s([P]) = 1 - P_\#$) for different values of L and N.

a uniform distribution over the expressible instance space. Note that the property is also violated in multistep problems in which the behavioral strategy causes a further skew in the instance distribution. Equation (15) provides a rough approximation of the real situation. In fact, (15) does not take into account many factors which influence the covering challenge, such as the use of the MAM technique for speeding up the estimate of classifier parameters, the learning rate β, and the parameters used for the evaluation of classifier fitness (ε_0, ν, and α).

The covering challenge is overcome if $P(\text{cover})$ is large enough to avoid the cover-delete cycle. This essentially should already be the case for rather small values of $P(\text{cover})$ since once the GA kicks in, more general classifiers are generated and covering does not take place anymore. Fig. 12 depicts (15) for different values of the problem size L and of the population size N. When the problem is simple, e.g., $L = 20$, even for small values of $P_\#$ we have high values of $P(\text{cover})$ suggesting that the *covering* challenge is relatively easy to meet. As the problem complexity increases, we need a larger $P_\#$ to avoid the cover-delete cycle. For instance, when $L = 70$ and $N = 10\,000$, covering classifiers must contain at least 75% of don't-cares to have a reasonably high $P(\text{cover})$; with more classifiers, e.g., $N = 1\,000\,000$, smaller values of $P_\#$ can be used.

Note that the covering challenge can be easily met by setting $P_\#$ very high. However, as $P_\#$ increases, the population tends to be filled up with overgeneral classifiers which—because they contain few *specified* bits—can accumulate little information about the optimal solution. When this happens, XCS may find it difficult to develop an effective pressure toward accurate classifiers. Providing enough specificity in the population is the subject of the schema challenge, considered next.

B. Schema Challenge

Satisfaction of the covering challenge guarantees that the GA acts on reliable information, since classifiers stay in the population long enough to be evaluated adequately. Once the covering challenge is met, we can analyze how the proposed evolutionary pressures (see Section III) result in the evolution of accurate, maximally general classifiers in XCS. As shown in Fig. 3 above, fitness pressure is the only pressure that pushes toward specificity in a directed (accuracy dependent) way. Accuracy-based fitness favors reproduction of classifiers which are more accurate. However, the strength of the fitness pressure is highly problem dependent so that it is impossible to derive a fitness pressure measure in general.

Assuming the worst-case scenario in which there is no *fitness guidance*, we can derive a worst-case bound on the conditions that assure proper convergence. These conditions define what we call the *schema* challenge.

General Case: The essence of the schema challenge is that classifiers must contain enough specified bits in their conditions to give the GA sufficiently accurate classifiers to work with. In general, we cannot make any assumption about the problem we are tackling and, therefore, it is basically impossible to determine what number of specified bits is *enough* to allow the GA to work properly. One way to guarantee that XCS can reach an accurate solution from the overgeneral side consists of setting $P_\#$ small enough so that accurate classifiers for most environmental niches are present due to covering. Let us calculate the probability that an environmental niche is represented by at least one classifier in [P]. For this purpose, we think of an environmental niche as a schema (see, e.g., [15] and [17]) of order o combined with an action. A classifier represents an environmental niche if: 1) it has the same action and 2) all the o positions specified in the schema are also specified in the classifier condition. The probability $P(\text{representative})$ that a schema of order o is represented in [P] is computed as follows. First, we compute the probability $P(\text{one representative})$ that *one* classifier correctly represents an environmental niche as

$$P(\text{one representative}) = \frac{1}{n}\left(\frac{s([P])}{2}\right)^o$$

where n denotes the number of possible actions; o is the schema order of the environmental niche. It is now possible to determine the probability $P(\text{representative})$ that [P] contains at least one representative of the niche as:

$$\begin{aligned} P(\text{representative}) &= 1 - (1 - P(\text{one representative}))^N \\ &= 1 - \left(1 - \frac{1}{n}\left(\frac{s([P])}{2}\right)^o\right)^N. \end{aligned} \tag{16}$$

Fig. 13 depicts (16) for different values of o and N. As can be seen, generally speaking the schema challenge can be met by setting $P_\#$ small enough. In fact, small values of $P_\#$, in Fig. 13, correspond to high values of $P(\text{representative})$ suggesting that each niche will be represented in [P]. Accordingly, for small $P_\#$, XCS should be able to develop an accurate maximally general solution since accurate classifiers are already present in [P]. Note also that the specificity of classifiers converges to a certain specificity dependent on mutation type, mutation strength μ, population size N, and GA frequency f_{ga} as formulated in (13) and validated in Section IV. Thus, it needs to be assured that specificity is not lost initially when fitness values may not provide any meaningful values. This can for example be prevented by setting θ_{ga} large enough. Note also that while the *schema* challenge can be solved by setting $P_\#$ small enough, the *covering* challenge requires that $P_\#$ is set large enough to guarantee

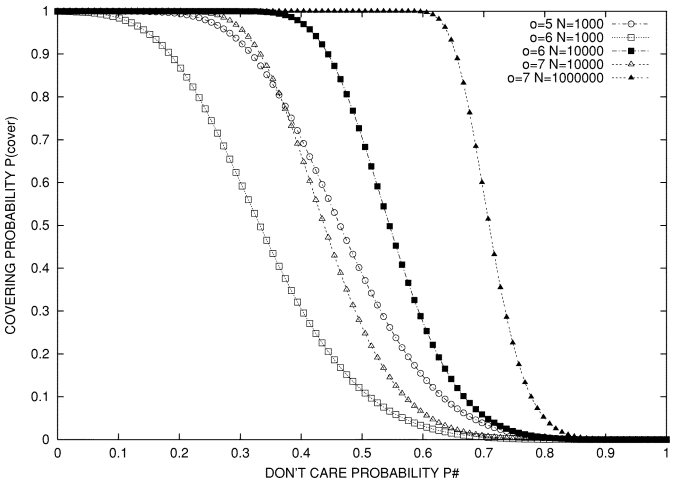

Fig. 13. $P(\text{representative})$ as a function of the don't-care probability $P_\#$ (initially, $P_\# = 1 - s([P])$) for different values of o and N.

that the input space is covered appropriately by $[\mathrm{P}]$. Thus, the interaction of two challenges induces boundaries which must be considered when applying XCS. These boundaries are further discussed in Section VI, where (15) and (16) are validated through a set of experiments.

As mentioned earlier, the *schema* challenge represents a worst-case bound which assumes that there exists no fitness guidance in a problem. However, in most problems this may not be the case. The following paragraphs point out some typical properties that ensure fitness guidance.

Layered Payoff: Strong fitness pressure that guides overgeneral classifiers toward accurate generalizations is often present in problems with *layered payoff* in which more than two reward values are provided. Classifier specialization toward accuracy may then decrease the possible payoff levels and, thus, increase accuracy.

Layered payoff could essentially be provided in classification problems in which the degree of class affiliation is known rather than only the class itself. For example, in medical data problems the degree of a sickness may be included in the payoff function. Interestingly, in multistep problems the payoff is inherently layered due to the discounted reward propagation.

Layered payoff was first used by Wilson [43] to simulate in a single-step problem the multiple payoff levels of typical multistep reinforcement learning problems. Kovacs [25] used layered payoff to develop a theory of strong overgenerals, and defined a function that creates such landscapes as a biased reward function. The easing of the schema challenge induced by layered payoff is further discussed and experimentally validated in Section VI-C.

Biased Generality: While layered payoff introduces an explicit bias in the reward function, *biased generality* is often intrinsically present, even when the reward function is not biased. The idea behind biased generality is that an overgeneral classifier will often be correct more often than wrong (or *vice-versa*). Any change in the classifier's condition due to the GA will be reflected in a change in its statistical correctness—and thus its error ε—and this will tend to guide the system toward increasingly accurate classifiers. Biased generality can be approached mathematically. Let us assume a two-level $R/0$ payoff landscape, where R is provided if the prediction is correct, 0, otherwise. Let $P_c(cl)$ denote the probability that classifier cl predicts the correct outcome. Due to the assumed payoff landscape and the assumption of a uniformly random encountering of both cases, the reward prediction p_{cl} of classifier cl eventually oscillates around $P_c(cl)\cdot R$,where the amount of oscillation can be influenced by β. Neglecting the oscillation and consequently setting p_{cl} equal to $P_c(cl)\times R$ the following derivation is obtained:

$$\begin{aligned}\varepsilon_{cl} =&(R - p_{cl}) \cdot P_c(cl) + p_{cl} \cdot (1 - P_c(cl)) \\ =&2R \cdot (P_c(cl) - P_c(cl)^2).\end{aligned} \tag{17}$$

Equation (17) sums the two cases of executing a correct or wrong action with the respective probabilities. It expresses the prediction error ε of classifier cl as a parabolic function of the probability $P_c(cl)$. The curve reaches its maximum of $0.5 \cdot R$ for the prediction error ε of classifier cl in the case of $P_c(cl) = 0.5$; it is 0 for $P_c(cl) = 0$ and $P_c(cl) = 1$. It is worth noting that (17) shares some similarities with the *entropy* of $P_c(cl)$, which is maximal for 0.5 and minimal for 0 and 1. In this respect, we might read the above equation as an estimate of the amount of information that $P_c(cl)$ conveys about the problem solution. If cl is completely incorrect or completely correct, $P_c(cl)$ conveys the largest amount of available information; in fact the "entropy" is minimal. If cl is 0.5, $P_c(cl)$ conveys almost no information about the problem solution; in fact the "entropy" which is a measure of confusion is maximal. Equation (17) shows that if the consistency of a correct/wrong prediction increases, the accuracy and, consequently, the fitness of a classifier increases. The easing of the schema challenge due to biased generality is further discussed and experimentally validated in Section VI-D.

VI. Validation of Fitness Guidance

The covering and schema challenges, as well as the fitness benefit due to layered payoff and biased generality will now be validated in several Boolean multiplexer problems. Boolean multiplexer problems have often been used to evaluate learning classifier systems (LCSs) performance [42], [43]. Jong and Spears [20] showed that they typically outperform other machine learning algorithms (such as C4.5) in these problems.

First, we study the boundaries induced by the covering challenge and by the schema challenge on the 20-multiplexer problem. We then move to the 37- and 70-multiplexers, which have more severe boundaries. Using a two-level R/0 payoff scheme, XCS is able to solve both problems, while layered payoff is shown to permit faster solutions. Finally, we show the effect of biased generality by applying XCS to a modified multiplexer function.

A. Boolean Multiplexer Problems

Boolean multiplexers are defined for strings of l bits, where $l = k + 2^k$ 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 six-multiplexer function mp_6, we have that $\mathrm{mp}_6(100\,010) = 1$,

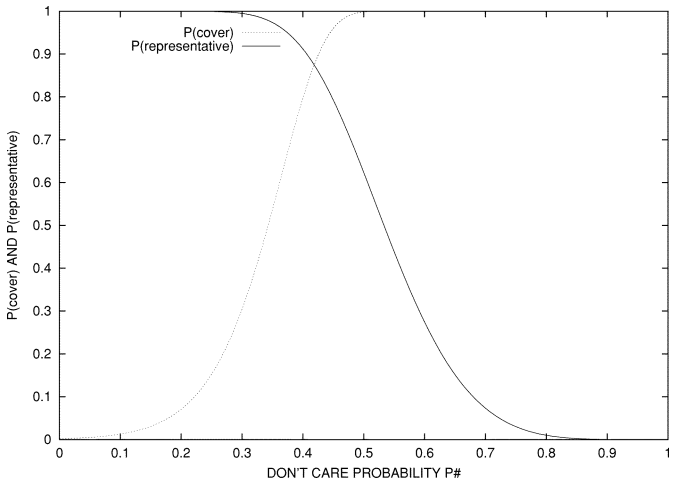

Fig. 14. The boundaries induced by the *covering challenge* and by the *schema challenge* in the 20-multiplexer when the population size is 2000. $P(\text{cover})$, dashed line, represents the covering challenge as stated in (15); $P(\text{representative})$, solid line, represents the schema challenge as stated in (16). L is 20; o is 5.

while $\text{mp}_6(000\,111) = 0$. More formally, the six-multiplexer can be represented by the following disjunctive normal form:

$$\text{mp}_6(x_0, x_1, y_0, \ldots, y_3) = \overline{x_0}\,\overline{x_1} y_0 + \overline{x_0}\, x_1 y_1 + x_0\,\overline{x_1} y_2 + x_0 x_1 y_3.$$

Because of their symmetry, Boolean multiplexers are biased neither toward over general nor toward overspecific classifiers, i.e., the probability that an over-general classifier is correct is close to 50%. Applying a 1000/0 reward regime, the payoff is not layered so that there is no biased reward benefit.

B. XCS in the 20-Multiplexer: The Two Challenges

To show the boundaries introduced by the covering challenge and by the schema challenge, we apply XCS to the 20-multiplexer for different values of $P_\#$ with a population of 2000 classifiers.

First, we consider the boundaries determined by $P(\text{cover})$ (15), and by $P(\text{representative})$ (16), respectively, for the covering and for the schema challenge. Fig. 14 reports the plots of $P(\text{cover})$, dashed line, and $P(\text{representative})$, solid line, when N is 2000, L is 20, o is 5.[1]

When applying XCS we are interested in those values of $P_\#$ which produce both a sufficiently high value of $P(\text{cover})$ and a high value of $P(\text{representative})$. A high value of $P(\text{cover})$ means that classifiers in [P] cover most of the input configurations to guarantee that the covering challenge is met. A high value of $P(\text{representative})$ means that [P] is more likely to guarantee that the schema challenge is met. Thus, with respect to Fig. 14, we are interested in the area below the intersection of the two plots where both $P(\text{cover})$ and $P(\text{representative})$ have non zero values. In particular, the values of $P_\#$ between 0.3 and 0.6 seem to guarantee reasonably high values of $P(\text{cover})$ and $P(\text{representative})$. Thus, we should expect that XCS will perform better for such values.

[1]The schema order o is 5 since in the 20-multiplexer accurate and maximally general classifiers have five specific bits, i.e., they belong to schemata of order 5.

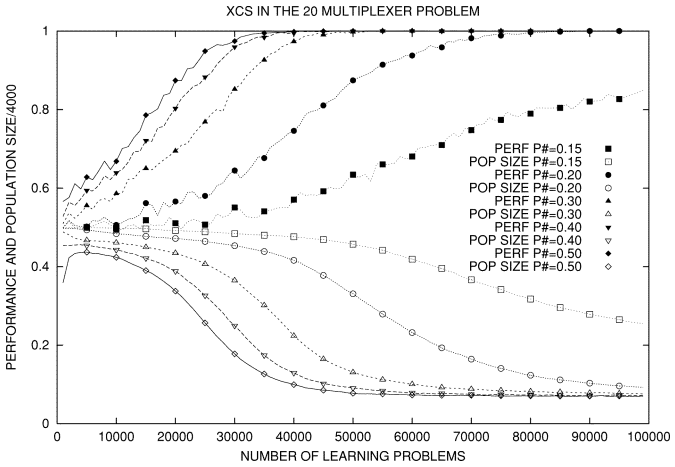

Fig. 15. XCS in the 20-multiplexer when $P_\#$ is 0.15, 0.20, 0.30, 0.40, and 0.50. Population size is 2000 classifiers. Curves are averages over 50 runs.

We apply XCS to the 20-multiplexer for different values of $P_\#$. XCS parameters are set as follows: $R = 1000$, $N = 2000$, $\beta = 0.2$, $\theta_{mna} = 2$, $\alpha = 0.1$, $\varepsilon_0 = 10$, $\nu = 5$, $\theta_{\text{ga}} = 25$, $\chi = 0.8$, $\mu = 0.04$, $\theta_{\text{del}} = 20$, $\delta = 0.1$, and $\theta_{\text{sub}} = 20$; both GA subsumption and action set subsumption are applied. Since we are now interested in how fast XCS evolves accurate classifiers, subsumption is applied to ensure fast convergence once accurate classifiers evolved. The performance measure of XCS is not influenced by subsumption.

The covering challenge is easily observable in Fig. 15. For values of $P_\#$ near to 0, XCS slowly converges to an optimal solution. In particular, when $P_\#$ is 0.1 (not reported here), XCS does not reach the optimum, but its performance remains at the random level 0.5 indicating that it is trapped in the proposed covering/deletion loop. As $P_\#$ increases, XCS performance improves. When $P_\#$ is 0.15, after 80 000 problems, XCS performance is near .85, i.e., XCS correctly classifies around 85% of the input configurations. When $P_\#$ reaches 0.3 the performance dramatically improves and as $P_\#$ further increases, XCS learns even faster. The relation between the proposed model in Fig. 14 and the experiments in Fig. 15 is quite evident. For a don't-care probability of 0.15, $P(\text{cover})$ is very close to 0 predicting that the covering challenge will show up, which actually happens in the experiments. With increasing values of $P_\#$, $P(\text{cover})$ increases predicting that the effect of covering challenge will diminish and eventually disappear, which is confirmed by the experiments.

Fig. 16 illustrates the schema challenge. When $P_\#$ is close to 1, the initial populations tend to be filled up with over-general classifiers; thus, it is more difficult for XCS to converge to the optimal solution made of accurate and maximally general classifiers. However, as $P_\#$ decreases, the performance of XCS improves and the system learns faster. It is interesting to note that XCSs best performance is reached when $P_\#$ approximates 0.6 even though the maximally general solution for the 20-multiplexer involves only classifiers with $5/20 = 0.75$ don't-care symbols. This validates the schema challenge and essentially the benefit of an appropriate supply of accurate classifiers since XCS performs better for values of $P_\#$ that are significantly smaller than the percentage of don't cares found in the optimal

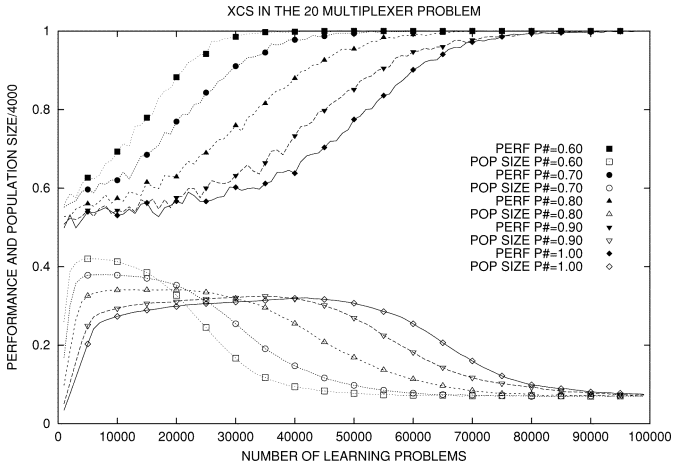

Fig. 16. XCS in the 20-multiplexer when $P_\#$ is 0.60, 0.70, 0.80, 0.90, and 1.00. Population size is 2000 classifiers. Curves are averages over 50 runs.

solution. The relation between the proposed model, Fig. 14, and the experimental results, Fig. 16, is evident.

Note, however, that even for values of $P_\#$ near to 1, which yields $P(\text{representative})$ close to 0, XCS is still able to evolve a complete and accurate model and, thus, reaches 100% performance. Two effects account for this observation.

1) The set pressure investigated in Section III causes the specificity to increase early on. In particular, mutation increases the specificity of completely general classifiers as expressed in (13). Moreover, the inaccuracy of the fitness estimate in the more specialized classifiers causes a further bias as shown in Fig. 11.
2) The multiplexer problem provides fitness guidance. Although it is a hard problem, a small (but sufficient) degree of fitness guidance is present. Essentially, any classifier that has some of the 2^k referenced bits specified will have a more probable outcome of either 1 or 0. Consequently, according to (17), such a classifier will approximate a lower prediction error ε, so that its accuracy and consequently its fitness will be higher. Thus, the multiplexer problem provides implicit fitness guidance albeit small and rather undirected which results in a small biased generality benefit.

Statistical Analysis: We performed an analysis of variance (ANOVA) to test whether the influence of $P_\#$ on XCS performance, as reported in Figs. 15 and 16, is statistically significant. Note that we do not test just the final performance, as usually done to test machine learning algorithms (e.g, [3], [12], [32], and [35] for learning classifier systems), but the differences among the entire learning curves. At the very first step, we apply a two-way ANOVA [14] to the learning curves obtained for the values of $P_\#$ between 0.15 and 1.00. For this purpose, we use the basic setting discussed in [11] and [37]. For every value of $P_\#$, we consider the 50 curves produced by each run; we sample the curves and consider only one point every 10 000 problems; overall we analyze 5000 curves, each one consisting of 100 points; as in [11] and [37], the first factor is the value of $P_\#$, the second factor is the number of problem considered (i.e., 10000, 20000, etc.); the confidence level is 10^{-4}. For the first factor, the two-way ANOVA returns $F = 6332.0$, $p = 0.000$ indicating that the effect of $P_\#$ values on the curves in Figs. 15 and 16 is statistically significant.

Since $P_\#$ has some significant effect on XCS performance, we applied appropriate *multiple comparison procedures* (also known as *post hoc tests* [14]) to find for which values of $P_\#$ XCS performance is statistically significant. We considered four *post hoc* tests: Tukey HSD, Scheffé, Bonferroni, and Student-Neumann-Keuls Glantz and Slinker [14]. The first three tests (Tukey HSD, Scheffé, and Bonferroni) individuate for which values of $P_\#$ XCS performs significantly different; in addition Tukey HSD and Scheffé tests also find groups of $P_\#$ values with similar performance; the Stewart–Neunmann–Keuls homogeneity test only finds groups of $P_\#$ values which result in similar XCS performance. When applied to the data from Figs. 15 and 16, *all* the four *post hoc* tests identified the same two groups of $P_\#$ values for which the difference in XCS performance *was not* statistically different; A group with $P_\# = 0.2$ and $P_\# = 0.9$ for which the first three tests returned $p = 1.000$, while the Stewart–Neunmann–Keuls homogeneity test returned $p = 0.561$; a group with $P_\# = 0.5$ and $P_\# = 0.6$ for which the first three tests returned $p = 1.000$, the Tukey HSD homogeneity test returned $p = 0.897$, the Sheffé homogeneity test returned $p = 0.988$, while the Stewart–Neunmann–Keuls homogeneity test returned $p = 0.136$. Note that the difference in XCS performance for *all* the remaining $P_\#$ values is statistically significant.

Discussion: In sum, we note that the boundaries induced by our theoretical model apply quite well as a lower and upper bound for $P_\#$ and, thus, for an initial specificity. As suggested by Fig. 14, XCS performed best for values of $P_\#$ between 0.3 and 0.6; in particular, the statistical analysis of the reported results showed no significant difference between the performance of XCS when $P_\#$ is 0.5 and 0.6. However, the theoretical boundaries are not as strong as suspected by the theory. For the covering challenge this is due to the fact that the covering challenge only applies early in the run. Once some more general classifiers are generated, the GA kicks in and the challenge does not apply anymore. In addition, the boundary provided by the schema challenge appears to be less severe than that provided by the covering challenge. In fact, XCS performs better for high values of $P_\#$, when the schema challenge should be more difficult, than for smaller values of $P_\#$, when the covering challenge should be more severe. The weak boundary of the schema challenge can be accounted for by the neglected set pressure and the slight implicit fitness guidance in the multiplexer problem which causes the biased generality benefit. Finally, we note that since the *covering challenge* depends only on general problem settings (i.e., the population size N and the input size L) it can be actually exploited to select adequate values of N and $P_\#$ that can favor evolution in XCS. In contrast, the *schema challenge* requires knowledge about the problem solution and, therefore, the boundary that the *schema challenge* provides is less useful in practice. On the other hand, from a theoretical standpoint, this boundary helps in understanding why, even in simple problems, high $P_\#$ values might be less effective.

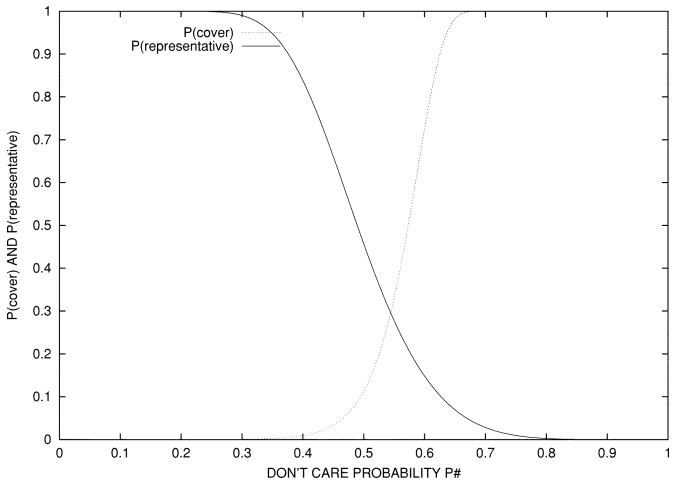

Fig. 17. The boundaries induced by the *covering challenge* and by the *schema challenge* in the 37-multiplexer when the population size N is 5000. $P(\text{cover})$, dashed line, identifies the covering challenge as stated in (13); $P(\text{representative})$, solid line, identifies the schema challenge as stated in (16); L is 37; o is 6.

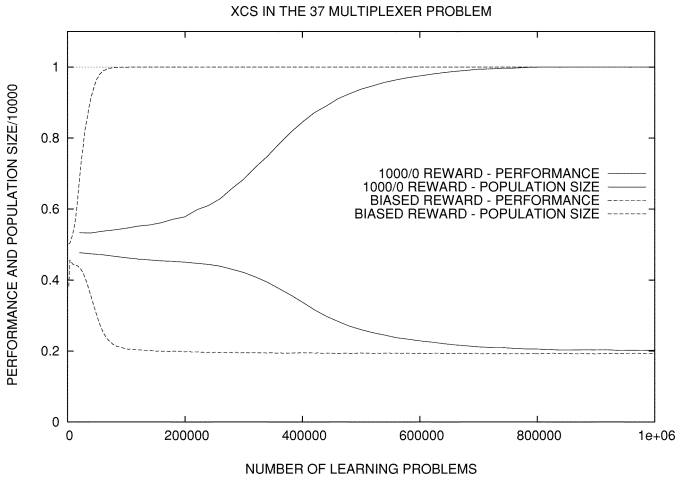

Fig. 18. XCS in the 37-multiplexer with 1000/0 reward, solid line; XCS with biased reward, dashed line. Population size is 5000 classifiers. For 1000/0 reward, $\alpha = 0.1$ and $\varepsilon = 10$; for biased reward, $\alpha = 0.1$ and $\varepsilon = 1$. Curves are averages over 50 runs. XCS with 1000/0 reward reaches the optimum after 700 000 learning problems, but XCS with biased reward learns much faster.

C. XCS in the 37-/70-Multiplexer: Layered Payoff Benefit

To study how XCS benefits from *layered payoff*, we compare XCS on the 37-multiplexer and on the 70-multiplexer with two different reward functions. The former is the usual 1000/0 reward function; the latter is a biased reward function that provides layered payoff.

The 37-Multiplexer: Fig. 17 reports the plots of $P(\text{cover})$, dashed line, and $P(\text{representative})$, solid line, when the population size N is 5000, $L = 37$, and $o = 6$.[2] As before, to solve the problem, we are interested in the values of $P_\#$ which correspond to high values of $P(\text{cover})$ and $P(\text{representative})$. Fig. 17 suggests that the 37-multiplexer is far more difficult than the 20-multiplexer. In fact, the interval of feasible $P_\#$ values is smaller and the feasible values of $P_\#$ correspond to smaller values of $P(\text{cover})$ and $P(\text{representative})$. Nonetheless, XCS is able to solve the problem with a population size of $N = 5000$ classifiers. Fig. 18 reports the performance of XCS with the usual 1000/0 reward function (solid upper curve) and the percentage of macroclassifiers in the population (solid lower curve); don't-care probability $P_\#$ is set to 0.65. As can be noted, XCS solves the 37-multiplexer problem with a close to 100% performance after 700 000 problem instances which is far below the 2^{37} actual possible instances.

We now illustrate the possible benefit due to layered payoff. For this purpose, we define the following biased reward function:

$$\text{reward} = (\text{value of the } k \text{ address bits} + \text{value of addressed bit}) * 100 + (\text{correctness}) * 300$$

where *correctness* is 1 if the action is correct, 0, otherwise. This biased reward function has the advantage that any specification of one of the k address bits results in a decrease in the number of possible rewards and therefore in a decrease in the prediction error. As a result, classifiers that are closer to accuracy are more likely to be selected. We apply XCS with this biased reward function to the 37-multiplexer with a population of 5000 classifiers. Fig. 18 reports the performance of XCS with biased reward (dashed upper curves) and the percentage of macroclassifiers in the population (dashed lower curves) when $\alpha = 0.1$ and $\varepsilon_0 = 1$; don't-care probability $P_\#$ is set to 0.65. The curves show a strong improvement in XCS performance when biased reward replaces the 1000/0 regime. While XCS with the 1000/0 reward reached 100% performance after about 700 000 problems, XCS with the biased reward function reaches 100% performance just after 100 000 problems (dashed line in Fig. 18). Note that with biased reward the difference between successive reward levels is smaller than with 1000/0 reward. Accordingly, XCS appears to have problems distinguishing between the successive levels unless $\varepsilon_0 = 1$; completely stable 100% performance was not reached when $\varepsilon_0 = 10$. To test whether the difference between the performance of XCS with 1000/0 reward and XCS with biased reward (as depicted in Fig. 18) is statistically significant, we applied the same procedure used for the 20-multiplexer. Note that we did not apply *multiple comparison procedures* (or *post hoc tests*[14]) since only two settings are compared. The ANOVA, performed according to the basic settings in [11] and [37] indicates that the difference is significant, as we would expect from the two plots.

The 70-Multiplexer: The utility of biased reward is even more evident when we compare the 1000/0 reward policy and the biased reward policy on a much more difficult problem, the 70-multiplexer, that involves 2^{70} input configurations. Fig. 19 reports the plots of $P(\text{cover})$, dashed line, and $P(\text{representative})$, solid line, when the population size N is 50 000, $L = 70$, and $o = 7$.[3] The boundaries are extremely severe: feasible values of $P_\#$ range between 0.6 and 0.8; in such an interval the maximum values of $P(\text{cover})$ and $P(\text{representative})$ are below 10%. Fig. 20 compares the performance of XCS with 1000/0 reward and XCS with biased

[2]The schema order o is 6 since in the 37-multiplexer, accurate and maximally general classifiers have six specific bits, i.e., they belong to schemata of order 6.

[3]The schema order o is 7 since in the 70-multiplexer, accurate and maximally general classifiers have seven specific bits, i.e., they belong to schema of order 7.

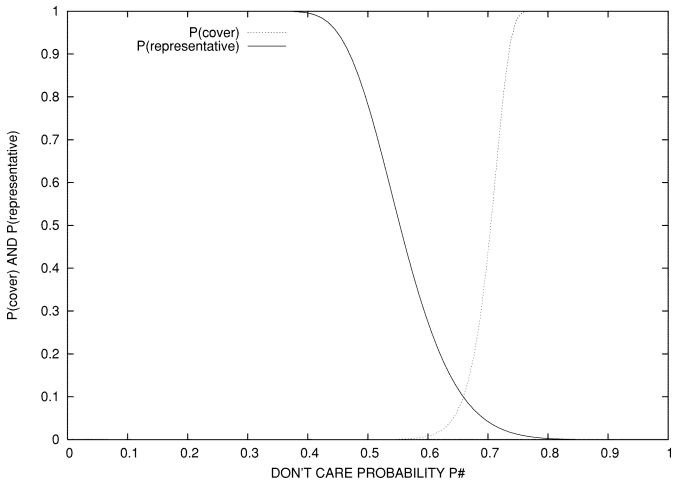

Fig. 19. The *covering challenge* and the *schema challenge* for the 70-multiplexer with a population size of $N = 50\,000$.

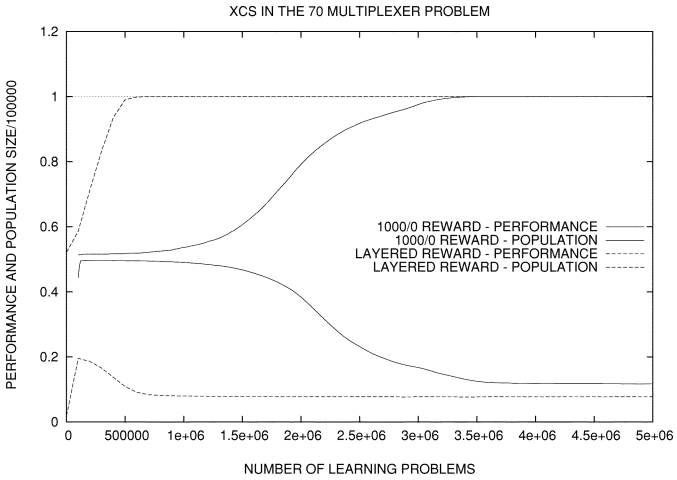

Fig. 20. XCS in the 70-multiplexer with 1000/0 reward (solid line) and biased (layered) reward (dotted line). Population size is 50 000 classifiers when 1000/0 is used; 20 000 classifiers when biased reward is used. In both cases, $P_{\#}$ is 0.75, $\alpha = 0.1$, and $\varepsilon = 1$.

reward in the 70-multiplexer. Population size N is 50 000 classifiers when 1000/0 reward is used but *just* 20 000 when biased reward is used; in both experiments $P_{\#}$ is .75, α is 0.1, and ε_0 is 1. Curves are averages over ten runs; both performance and population size are reported as the average over the last 100 000 test problems. This large moving window is needed to clearly highlight the initial learning of XCS; shorter moving windows would result in too much noise and would not allow a clear study of XCS behavior. Fig. 19 shows that XCS solves the 70-multiplexer with the 1000/0 reward regime, which so far has not been reported in the literature.[4] As in the case of the 37-multiplexer, with 1000/0 reward the convergence is much slower: it takes between 1 and 1.5 million problems before learning shows up and the performance begins to move away from 0.5 (i.e., 50%). However, XCS is learning during the first million problems. We can see this because inspection shows small groups of classifiers that solve part of the problem. However, since the problem is so large, the contribution of these small partial solutions is hardly visible from the performance plot. For instance, during the first million problems, XCS performance rises from .50 to around .53 over an average of ten runs. Once learning starts, the performance reaches the optimum after more or less 2.5 millions problems. That is quite fast if compared with the number of experiments needed at the beginning to rise from 0.5 performance. With the additional information provided through biased reward XCS learns much faster. After just one million problems, XCS with biased reward reaches optimal performance stably (Fig. 19, dotted line), just when XCS with 1000/0 reward is starting to learn (Fig. 19, solid line). Also, in this case, the ANOVA we performed according to the settings discussed in [11] and [37] indicates that again the difference of XCS performance in the case of 1000/0 reward and biased reward is statistically significant (as expected).

Finally, it is interesting to compare the learning curves obtained for the 37-multiplexer and the 70-multiplexer, in Fig. 18 and in Fig. 20, respectively. First, we note that the two curves of XCS performance for the 1000/0 reward policy are very similar in shape. This suggests that in the Boolean multiplexer, XCS has a similar learning behavior despite the size of the search space. Second, we note that although the input space of the 70-multiplexer is 2^{33} times bigger than the search space of the 37-multiplexer, to solve the 70-multiplexer XCS with 1000/0 reward needs only ten times more classifiers and five times more problems. These results are in part consistent with Wilson's claim Wilson [44] that in XCS the complexity of learning depends on the size of the solution as measured by the size of the Boolean formula, not on the size of the input space; see also Butz, Goldberg, and Tharakunnel [7].

D. xy-Biased Multiplexer: Biased Generality Benefit

Finally, we show the benefit of biased generality. For this purpose, we define an artificial problem, namely the xy-biased multiplexer, that is biased toward generality. First, we define a *biased multiplexer*, then we use a set of biased multiplexers to build an xy-biased multiplexer. Roughly, a biased multiplexer is a Boolean multiplexer whose output is biased toward a specific value: a *zero-biased multiplexer* is a modification of a Boolean multiplexer in which the zero output is more likely to be correct; a *one-biased multiplexer* is a modification of a Boolean multiplexer in which the one output is more likely to be correct. A biased multiplexer, is defined over $l = y + (2^y - 1)$ bits; as in the Boolean multiplexer the first y bits represent an address which indexes the remaining $2^y - 1$ bits. Note, however, that in a biased multiplexer, it is not possible to address one of the configurations: y address bits would address 2^y bits, but in this case only $2^y - 1$ bits are available. The missing configuration represents the bias. A *one-biased multiplexer* always returns one when the y address bits are all 1, i.e., the output one is correct for one configuration more than would be in the case in a Boolean multiplexer with y address bits. For instance, in a *one-biased multiplexer* with two address bits the condition 11### always corresponds to the output 1. A *zero-biased multiplexer* always returns to 0 when the y address bits are all 0s, i.e., the 0 output is more likely to be correct.

[4]Since the problem is huge, it is not practical to test optimal performance using a performance average. Instead, optimal performance was confirmed by examining the classifiers in the final populations.

A set of *biased multiplexers* can be used to build an xy-biased multiplexer as follows. An xy-biased multiplexer is a Boolean function which forms a hierarchy of depth two where: x refers to the number of bits used to address one of the 2^x biased multiplexers involved; y refers to the size of each one of the biased multiplexers involved, each one consisting of $y + 2^y - 1$ bits; half of the 2^x biased multiplexers are zero biased, half are one biased; overall, the conditions consist of $x + 2^x \times (y + 2^y - 1)$ bits. The bias comes in when an extreme position is referenced by the position bits. Dependent on whether a zero or a one bias is applied, the biased multiplexer, respectively, returns 1) always zero if the position bits are all zero or 2) always one if the position bits are all one. The bias depends on whether the value corresponding to the first x position bits is bigger than $(2^x - 1)/2$. In fact, the first half of the 2^x biased multiplexers in an xy-biased multiplexer will be *zero biased* while the second half will be *one biased*. Note that although the xy-biased multiplexer is an artificial problem, many classification problems have a similar bias, i.e., the specification of relevant bits of a problem usually results in a higher classification accuracy.

As an example, let us consider the *11-biased multiplexer* (*11bmp*). The first bit of the multiplexer, x_0, refers to one of the two biased multiplexers, which consist of 2 bits; the first biased multiplexer is *zero biased*, the second biased multiplexer is *one biased*. The *11-biased multiplexer* for input 00 111 outputs 0 because the first 0 refers to the (first) zero-biased multiplexer and the second 0 determines output 0; input 01 011 has output 0 as well; input 10 010 has output 1 since the one-biased multiplexer is now referenced; input 10 000 would be output 0. More formally, the *11-biased multiplexer* can be written in disjunctive normal form as

$$11\text{bmp}(x_0, x_1, x_2, x_3, x_4) = \overline{x_0} x_1 x_2 \vee x_0 x_3 \vee x_0 \overline{x_3} x_4.$$

To study how XCS can benefit from the biased generality, we apply XCS to the *xy-biased multiplexer* for the following values of x and y: $\langle 1,3\rangle$, $\langle 2,2\rangle$, and $\langle 3,1\rangle$. We set the don't-care probability $P_\#$ to 0.95 to guarantee that at the beginning [P] contains *only* overgeneral classifiers. To have an idea of how complex the three problems are, we use the function $||[O]||(.)$ introduced by Kovacs and Kerber [28]. This defines the complexity of a problem as the size of the minimal, accurate, nonoverlapping population that covers all environmental niches accurately. We note that $||[O]||(\langle 3,1\rangle) = 48$, $||[O]||(\langle 2,2\rangle) = 56$, and $||[O]||(\langle 1,3\rangle) = 60$, suggesting that $\langle 3,1\rangle$ is the simplest problem, while $\langle 1,3\rangle$ is the most difficult one. Results are depicted in Fig. 21. The three plots confirm the prediction of the function $||[O]||(.)$. In fact, Fig. 21 shows that in $\langle 3,1\rangle$ XCS learns faster, suggesting that this is the simplest problem; in $\langle 1,3\rangle$ XCS learns more slowly. Regarding the three plots it is worth noting that, apparently, the input length L does not influence the performance. In fact, although $\langle 1,3\rangle$ has the smallest input size, $L(\langle 1,3\rangle) = 21$, it still remains the most difficult to solve. Likewise, although $L(\langle 2,2\rangle) = 22$ and $L(\langle 3,1\rangle) = 19$ these problems are more easy to solve. However, why the performance in all three cases is much better than the performance in the 20 multiplexer, where $||[O]|| = 64$ and $L = 20$ is not explainable by either measure. Also, the fact that the $\langle 2,2\rangle$ problem appears to have difficulty similar to the $\langle 3,1\rangle$ problem is not explainable. Finally, the plateau in the $\langle 3,1\rangle$ curve is not explainable, either.

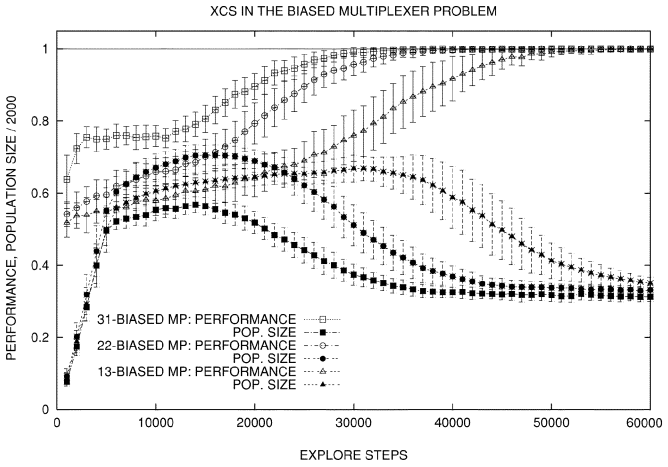

Fig. 21. The increasing consistency in classification when getting closer to being accurate helps XCS in evolving a complete and accurate representation—the biased generality benefit becomes obvious.

Overall, the curves in Fig. 21 suggest that XCS can exploit the implicit fitness guidance in the problem to solve the schema challenge which should be present when $P_\#$ is 0.95. In fact: 1) despite the low specificity in the beginning of all runs, XCS evolves the necessary specializations fast and 2) despite the higher $||[O]||$ measure in the $\langle 2,2\rangle$ run, XCS is able to exploit the problem bias in this setting more and, consequently, reaches a 100% knowledge as fast as in the 3,1 run. The plateau in the run for $\langle 3,1\rangle$ reveals that XCS first discovers the necessary specificity of the first bit, which results in a 0.75% correctness once specified in a classifier. However, due to the minor bias in the remaining three or four to be specified bits, it takes longer to proceed to a 100% knowledge. The $\langle 2,2\rangle$ case has a lower bias in the specification of the first bit, but a stronger bias in the remaining specifications. Thus, in this case, it takes longer to reliably evolve the first specializations but once the first positions are specified, the remaining positions are detected faster.

VII. Conclusion

This paper provides a foundation for the development of a theory of generalization and learning in XCS. In particular, it proposes a theoretical basis for the principal performance and generalization properties of the system, namely, the tendency to evolve classifiers that are as general as possible, while still predicting accurately. A pressure toward increasing generality was shown theoretically to result from the higher generality of classifiers in action sets versus the population as a whole. Selection from the action sets together with deletion from the whole population leads to the pressure. Mutation causes an additional, but small, pressure toward a fixed average specificity. The combined pressures were formulated in a *specificity equation* and demonstrated in several experiments. The simple equations we developed represent the first theoretical confirmation of Wilson's generalization hypothesis.

Next, the inherent pressure toward accurately predicting classifiers due to XCS's fitness function was examined. Two

qualifications, or *challenges*, came to light that might prevent this pressure. One, the *covering* challenge, was a requirement for sufficient generality (#'s) in the initial classifiers to permit covering of inputs to cease and allow created classifiers to have sufficient experience so that their fitnesses get adequately evaluated. The other, the *schema* challenge, was a requirement that—in contrast to generality—the initial classifiers contain sufficient specified bits to permit the GA to build classifiers specific enough to be accurate. Satisfaction of these conflicting challenges roughly defines an interval of initial don't-care probabilities ($P_{\#}$) and population sizes (N) for which XCS is likely to reach accurate, maximally general solutions.

The effect of the challenges was tested on several large Boolean multiplexer problems, where it was found that the region of solution was indeed given by $P_{\#}$ values where the two challenges were simultaneously met. The influence of mutation formulated in the specificity equation was located as well. Consistent data for N were found. The extent to which the reward landscape could provide fitness guidance for the GA was also examined. Experiments with a correspondingly layered payoff landscape and a biased multiplexer function confirmed that the guidance was present.

The results presented in this paper lead to a broader understanding of XCS and its learning and generalization power. Some of the findings form the basis for further analyses in [7]. Moreover, the formulas for the challenges offer useful rules of thumb for setting XCS parameters in actual problems. From a theoretical standpoint, this paper provides a foundation for future investigations of XCS's learning complexity.

ACKNOWLEDGMENT

M. V. Butz and P. L. Lanzi are more than grateful to D. E. Goldberg, X. Llora, M. Pelikan, K. Sastry for their help and the useful discussions.

P. Luca wishes to thank M. Colombetti and S. Ceri for invaluable support; F. Ferrandi and the Computer Architecture Laboratory for the CPU time provided to run massive experiments.

M. V. Butz, T. Kovacs, and P. Luca wish to thank S. W. Wilson for invaluable discussions and inspiration.

The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon.

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Office of Scientific Research, the National Science Foundation, or the U.S. Government.

REFERENCES

[1] A. Barry. (2003) Java Implementation of XCSC. [Online]. Available: http://www.cs.bath.ac.uk/ amb/LCSWEB/computer.htm

[2] A. M. Barry, "The stability of long action chains in XCS," *J. Soft Comput.*, vol. 6, no. 3-4, pp. 183–199, 2002.

[3] E. Bernadó, X. Llorà, and J. M. Garrell, "XCS and GALE: A Comparative study of two learning classifier systems with six other learning algorithms on classification tasks," in *Proc. 4th Int. Workshop Learning Classifier Systems (IWLCS-2001)*, Short version published in *Proc. Genetic and Evol. Comput. Conf. (GECCO2001)*, 2001, pp. 337–341.

[4] L. Bull, "Simple Markov models of the genetic algorithm in classifier systems: Accuracy-based fitness," in *Advances in Learning Classifier Systems: Proc. 3rd Int. Workshop*, P. L. Lanzi, W. Stolzmann, and S. W. Wilson, Eds. Berlin, Germany: Springer-Verlag, 2001a, LNAI 1996, pp. 21–28.

[5] ——, "Simple Markov models of the genetic algorithm in classifier systems: Multi-step tasks," in *Advances in Learning Classifier Systems: Proc. 3rd Int. Workshop*, P. L. Lanzi, W. Stolzmann, and S. W. Wilson, Eds. Berlin, Germany: Springer-Verlag, 2001b, LNAI 1996, pp. 29–36.

[6] M. V. Butz, "An implementation of the XCS classifier system in C," The Illinois Genetic Algorithms Laboratory, Univ. Illinois, Urbana-Champaign, IL, Tech. Rep. 99 021, 1999.

[7] M. V. Butz, D. E. Goldberg, and K. Tharakunnel, "Analysis and improvement of fitness exploitation in XCS: Bounding models, tournament selection, and bilateral accuracy," *Evol. Comput.*, vol. 11, no. 3, pp. 239–278, 2003.

[8] M. V. Butz and S. W. Wilson, "An algorithmic description of XCS," in *Advances in Learning Classifier Systems: Proc. 3rd Int. Workshop*, P. L. Lanzi, W. Stolzmann, and S. W. Wilson, Eds. Berlin, Germany: Springer-Verlag, 2001a, LNAI 1996, pp. 253–272.

[9] ——, "An algorithmic description of XCS," in *Advances in Learning Classifier Systems*, P. L. Lanzi, W. Stolzmann, and S. W. Wilson, Eds. Berlin, Germany: Springer-Verlag, 2001b, vol. 1996, LNAI, pp. 253–272.

[10] P. Clark and T. Niblett, "The CN2 induction algorithm," *Mach. Learn.*, vol. 3, no. 4, pp. 261–283, 1989.

[11] P. R. Cohen, *Empirical Methods for Artificial Intelligence*. Cambridge, MA: MIT Press, 1995.

[12] P. W. Dixon, D. W. Corne, and M. J. Oates, "A preliminary investigation of modified XCS as a generic data mining tool," in *Advances in Learning Classifier Systems*, P. L. Lanzi, W. Stolzmann, and S. W. Wilson, Eds. Berlin, Germany: Springer-Verlag, 2002, vol. 2321, LNAI, pp. 133–150.

[13] P. Domingos, "Rule induction and instance-based learning: A unified approach," in *Proc. 14th Int. Joint Conf. Artificial Intelligence (IJCAI 95)*, Montreal, QC, Canada, Aug. 1995, pp. 1226–1232.

[14] S. A. Glantz and B. K. Slinker, *Primer of Applied Regression & Analysis of Variance*, 2nd ed. New York: McGraw-Hill, 2001.

[15] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading, MA: Addison-Wesley, 1989.

[16] A. Greenyer, "CoIL Challenge 2000: The Insurance Company Case," Sentient Machine Research, Amsterdam and Leiden Inst. Advanced Comput. Sci., Leiden, The Netherlands, Tech. Rep. 2000–09, P. van der Putten and M. van Someren, Eds., 2000.

[17] J. H. Holland, *Adaptation in Natural and Artificial Systems*, 2nd ed. Ann Arbor, MI: Univ. Michigan Press, 1992.

[18] ——, "Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems," in *Machine Learning, an Artificial Intelligence Approach*, Mitchell, Michalski, and Carbonell, Eds. San Mateo, CA: Morgan Kaufmann, 1986, vol. II, ch. 20, pp. 593–623.

[19] J. H. Holland and J. S. Reitman, "Cognitive systems based on adaptive algorithms," in *Reprinted in: Evolutionary Computation. The Fossil Record*, D. B. Fogel, Ed. Piscataway, NJ: IEEE Press, 1978.

[20] K. A. D. Jong and W. M. Spears, "Learning concept classification rules using genetic algorithms," in *Proc. 12th Int. Conf. Artificial Intelligence IJCAI-91*, vol. 2, Sydney, Australia, 1991, pp. 651–656.

[21] T. Kovacs, "Evolving Optimal Populations with XCS Classifier Systems," M.S. thesis, School Comput. Sci., Univ. Birmingham, Birmingham, U.K., 1996.

[22] ——, "XCS classifier system reliably evolves accurate, complete, and minimal representations for boolean functions," in *Soft Computing in Engineering Design and Manufacturing*, Roy, Chawdhry, and Pant, Eds. London, U.K.: Springer-Verlag, 1997, pp. 59–68.

[23] ——, "Deletion schemes for classifier systems," in *Proc. Genetic and Evolutionary Computation Conf. (GECCO-99)*, W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, Eds., San Francisco, CA, 1999, pp. 329–336.
[24] ——, "Strength or accuracy? Fitness calculation in learning classifier systems," in *Learning Classifier Systems: From Foundations to Applications*, P. L. Lanzi, W. Stolzmann, and S. W. Wilson, Eds. Berlin, Germany: Springer-Verlag, 2000a, LNAI 1813, pp. 143–160.
[25] ——, "Toward a theory of strong overgeneral classifiers," in *Foundations of Genetic Algorithms (FOGA)*, W. Martin and W. M. Spears, Eds. San Mateo, CA: Morgan Kaufmann, 2000b, vol. 6, pp. 165–184.
[26] ——, "What should a classifier system learn?," in *Proc. 2001 Congress on Evolutionary Computation (CEC01)*, 2001, pp. 775–782.
[27] T. Kovacs and M. Kerber, "Some dimensions of problem complexity for XCS," in *Proc. 2000 Genetic and Evolutionary Computation Conf. Workshop Program*, A. S. Wu, Ed., 2000, pp. 289–292.
[28] ——, "What makes a problem hard for XCS?," in *Advances in Learning Classifier Systems: Proc. 3rd Int. Workshop*, P. L. Lanzi, W. Stolzmann, and S. W. Wilson, Eds. Berlin, Germany: Springer-Verlag, 2001, LNAI 1996, pp. 80–99.
[29] P. L. Lanzi, "A study of the generalization capabilities of XCS," in *Proc. 7th Int. Conf. Genetic Algorithms (ICGA97)*, T. Back, Ed., 1997, pp. 418–425.
[30] ——, "Reinforcement Learning by Learning Classifier Systems," Ph.D. dissertation, Politecnico di Milano, Milan, Italy, 1998.
[31] ——, "Extending the representation of classifier conditions Part I: From binary to messy coding," in *Proc. Genetic and Evolutionary Computation Conf. (GECCO-99)*, W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, Eds., 1999, pp. 337–344.
[32] ——, "Mining interesting knowledge from data with the XCS classifier system," in *Proc. Genetic and Evolutionary Computation Conf. (GECCO-2001)*, L. Spector, E. D. Goodman, A. Wu, W. Langdon, H. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. H. Garzon, and E. Burke, Eds., San Francisco, CA, July 7–11, 2001, pp. 958–965.
[33] ——, (2003) The XCS Library. [Online]. Available: http://xcslib.sourceforge.net
[34] P. L. Lanzi, W. Stolzmann, and S. W. Wilson, *Learning Classifier Systems. From Foundations to Applications*. Berlin, Germany: Springer-Verlag, 2000, vol. 1813, LNAI.
[35] T. M. Mitchell, *Machine Learning*. New York: McGraw-Hill, 1997.
[36] (2003). NuTech Solutions Inc. [Online]. Available: http://www.nutech-solutions.com/
[37] J. H. Piater, P. R. Cohen, X. Zhang, and M. Atighetchi, "A randomized ANOVA procedure for comparing performance curves," in *Proc. 15th Int. Conf. Machine Learning (ICML)*, Madison, WI, July 1998, pp. 430–438.
[38] R. Quinlan, "Learning first-order definitions of functions," *J. Artif. Intell. Res.*, vol. 5, pp. 139–161, Oct. 1996.
[39] R. J. Quinlan, *C4.5 Programs for Machine Learning*. San Mateo, CA: Morgan Kauffmann, 1993.
[40] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An introduction*. Cambridge, MA: MIT Press, 1998.
[41] C. J. C. H. Watkins, "Learning from Delayed Rewards," Ph.D. dissertation, King's College, Cambridge, U.K., 1989.
[42] S. W. Wilson, "Classifier systems and the animat problem," *Machine Learn.*, vol. 2, pp. 199–228, 1987.
[43] ——, "Classifier fitness based on accuracy," *Evol. Comput.*, vol. 3, no. 2, pp. 149–175, 1995.
[44] ——, "Generalization in the XCS classifier system," in *Proc. 3rd Annu. Conf. Genetic Programming 1998*, J. R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, M. H. Garzon, D. E. Goldberg, H. Iba, and R. Riolo, Eds., 1998, pp. 665–674.
[45] ——, "XCS tutorial," presented at the Int. Conf. Genetic and Evolutionary Computation 1999 (GECC099), Orlando, FL, 1999.
[46] ——, *State of XCS Classifier Syst. Res.*, pp. 63–82, 2000.
[47] ——, *Mining Oblique Data with XCS*, P. L. Lanzi, W. Stolzmann, and S. W. Wilson, Eds. New York: Springer-Verlag, 2001, vol. 1996, Lecture Notes in Computer Science, pp. 158–176.

Martin V. Butz received the Diploma degree in computer science from the University of Würzburg, Würzburg, Germany, in 2001 and is working toward the Ph.D. degree in computer science at the University of Illinois at Urbana-Champaign.

He is working at the Illinois Genetic Algorithms Laboratory (IlliGAL), University of Ilinois at Urbana-Champaign, and at the Department of Cognitive Psychology, University of Würzburg. His major research interest lies in the study of anticipatory learning and anticipatory behavior. The goal is to analyze such processes in a facet-wise manner increasing understanding and applicability. The processes are related to general learning theories in machine learning, as well as cognitive mechanisms. Applications span from data classification to the simulation of competent adaptive behavior and learning.

Tim Kovacs received the B.A. (honors) degree in psychology from Carleton University, Ottawa, ON, Canada, in 1995, and the M.Sc. and Ph.D. degrees in computer science from the University of Birmingham, Birmingham, U.K., in 1996 and 2002, respectively.

He joined the University of Bristol, Bristol, U.K., as a Lecturer in machine learning in 2001. His research interests are in machine learning, artificial intelligence, cognitive science, and more specifically, in evolutionary computation, reinforcement learning, and learning classifier systems. He has published on game playing, the complexity of learning, methodological issues in machine learning, and various aspects of learning classifier systems.

Dr. Kovacs was awarded a British Computer Society/Conference of Professors and Heads of Computing Distinguished Dissertation Award for his Ph.D. dissertation which is to be published by Springer-Verlag.

Pier Luca Lanzi was born in Turin, Italy, in 1967. He received the Laurea degree in computer science from the Università degli Studi di Udine, Udine, Italy, in 1994 and the Ph.D. degree in computer and automation engineering from the Politecnico di Milano, Milan, Italy, in 1999.

Since 2001, he has been an Assistant Professor with the Department of Electronics and Information, Politecnico di Milano. His research areas include evolutionary computation, reinforcement learning, and machine learning. He is interested in applications to data mining and autonomous agents. He is a Member of the Editorial Board of the *Evolutionary Computation Journal*.

Stewart W. Wilson was born in Rochester, NY, in 1937. He received the B.S. degree in physics and the Ph.D. degree in electrical engineering from the Massachusetts Institute of Technology, Cambridge, in 1960 and 1967, respectively.

From 1960 to 1983, he did research on educational technology and machine learning at Polaroid Corporation, Boston, MA. From 1983 to 1998, he was a Senior Scientist at the Rowland Institute for Science, Cambridge, MA. Since 1998, he has headed the research and consulting firm of Prediction Dynamics, Concord, MA. He is an Associate Editor of *Adaptive Behavior* and is a Member of the Editorial Boards of *Evolutionary Computation* and *Artificial Life*. He is an Adjunct Professor in the Department of General Engineering, University of Illinois at Urbana–Champaign and a Technical Consultant for NuTech Solutions, Inc., Boston, MA. He is interested in intelligence and learning in machines and organisms. As a vehicle for this, his research is focused primarily on learning classifier systems.