Analysis and Improvement of Fitness Exploitation in XCS: Bounding Models, Tournament Selection, and Bilateral Accuracy

Martin V. Butz, David E. Goldberg, and Kurian Tharakunnel
Illinois Genetic Algorithms Laboratory (IlliGAL)
University of Illinois at Urbana-Champaign
Urbana, IL, 61801
{butz,deg,kurian}@illigal.ge.uiuc.edu

Abstract

The evolutionary learning mechanism in XCS strongly depends on its accuracy-based fitness approach. The approach is meant to result in an evolutionary drive from classifiers of low accuracy to those of high accuracy. Since, given inaccuracy, lower specificity often corresponds to lower accuracy, fitness pressure most often also results in a pressure towards higher specificity. Moreover, fitness pressure should cause the evolutionary process to be innovative in that it combines low-order building blocks of lower accurate classifiers, to higher-order building blocks with higher accuracy. This paper investigates how, when, and where accuracy-based fitness results in successful rule evolution in XCS. Along the way, a weakness in the current proportionate selection method in XCS is identified. Several problem bounds are derived that need to be obeyed to enable proper evolutionary pressure. Moreover, a fitness dilemma is identified that causes accuracy-based fitness to be misleading. Improvements are introduced to XCS to make fitness pressure more robust and overcome the fitness dilemma. Specifically, (1) tournament selection results in a much better fitness-bias exploitation, and (2) bilateral accuracy prevents the fitness dilemma. While the improvements stand for themselves, we believe they also contribute to the ultimate goal of an evolutionary learning system that is able to solve decomposable machine-learning problems quickly, accurately, and reliably. The paper also contributes to the further understanding of XCS in general and the fitness approach in XCS in particular.

Keywords

Learning Classifier Systems, XCS, evolutionary pressures, bounding models, tournament selection, bilateral accuracy.

1 Introduction

The accuracy-based learning classifier system XCS (Wilson, 1995) has become known as the most reliable Michigan-style learning classifier system (LCS) (Holland, 1976; Holland & Reitman, 1978) for solving typical data mining and machine learning problems. It has been shown that XCS is able to learn accurate representations of oblique data and real-value data input (Wilson, 2000; Wilson, 2001). Other publications have shown

favorable comparisons of XCS with other machine learning algorithms such as C4.5 (Bernadó, Llorà, & Garrell, 2002; Dixon, Corne, & Oates, 2002). Despite these promising results, a detailed understanding of why XCS works so well and whether improvement is possible remains elusive.

In response to this need for more mechanistic understanding of XCS workings, a body of facetwise theory is being developed to (1) analyze extant LCS performance and (2) design better XCS mechanisms. The purpose of this paper is to build on this growing base of facetwise theory and better understand important aspects of the fitness-based learning process in XCS.

We have developed a general theory of when and how XCS can exploit its accuracy-based fitness measure for evolving accurate classifiers. We show that if the fitness pressure applies in the right direction, mutation can lead to a proper evolution of accurate classifiers. However, due to disruptive mutation, recombination is essential for solving some problems. This finding also shows the utility of crossover in LCS. Ultimately, as in genetic algorithms (GAs), crossover should become the major discovery operator propagating and recombining low-order building blocks to evolve accurate classifiers (Holland, 1975; Goldberg, 1989). This should lead to the evolution of high-order building blocks by a learning process comparable to a basic innovative process (Goldberg, 2002).

The paper is structured as follows. First, we provide a brief introduction to XCS. Next, we give an overview of previously identified evolutionary pressures in XCS. The remainder of the paper is dedicated to the analysis and improvement of the fitness pressure. Specifically, we show that fitness pressure can be exploited much more efficiently by using *tournament selection* with a tournament size which is relative to the current action set size. Next, we analyze when fitness pressure applies, identifying a *reproductive opportunity bound (ROP-bound)* and a *representative bound*. After the experimental verification of the bounds, we show how existing fitness pressure is exploited by XCS, evolving more specialized and more accurate classifiers. Finally, we identify problem characteristics in which fitness guidance can be misleading causing an *accuracy dilemma*. We introduce XCS with *bilateral accuracy* as an approach that can alleviate the accuracy dilemma. Summary, conclusions, and future work conclude the paper.

2 XCS in a Nutshell

The accuracy-based learning classifier system XCS was introduced by Wilson (1995). Immediately, XCS formed something of a cornerstone in LCS research. All previous learning classifier systems suffered from *strong over-generals* (Kovacs, 2001) in one way or another. The only way to overcome the problem of strong over-generals is to alter the fitness approach. Bull and Hurst (2002) showed that fitness-sharing techniques in combination with a large learning rate can overcome the problem in strength-based LCSs. However, a high learning rate destroys the fundamental concept of temporal difference learning techniques that require small learning rates to accurately approximate the average value encountered.

XCS goes one step further and derives fitness from the accuracy of a classifier's reward prediction error instead of from the reward prediction itself. Consequently, XCS is intended to not only evolve an optimal and compact classification policy (or behavioral policy) but rather a complete, accurate, and maximally general *payoff map* of an environment (out of which an optimal policy can be derived). Thus, it does not only evolve rules (the classifiers) that represent best classifications, but it evolves rules for all available classifications predicting accurate payoff for each.

Previous analyses of XCS form the basis of this work. Kovacs (1999) introduced an improvement of the deletion mechanism. Lanzi and Colombetti (1999) showed that stochastic properties can be detected in an environment and enhanced XCS in this respect to handle some noisy environments. Next, Kovacs (Kovacs, 2000; Kovacs, 2001) presented his theory of strong over-generals that revealed the basic idea behind the accuracy-based fitness approach in XCS. The main point is that strength-based learning classifier systems can suffer from strong over-generals—classifiers that encounter high reward on average but mislead the decision processes in XCS in the low reward cases. With this insight in mind, Butz and Pelikan (2001) revealed five major evolutionary pressures in XCS: (1) set pressure, (2) mutation pressure, (3) deletion pressure, (4) subsumption pressure, and (5) fitness pressure and they analyzed the interaction of set pressure and mutation pressure. Butz, Kovacs, Lanzi, and Wilson (2001) showed when fitness pressure might not apply and suggested approaches to work around the shortcomings. Finally, Butz, Kovacs, Lanzi, and Wilson (2003) wrapped these ideas together in a fundamental theory of generalization and learning in XCS solving the set pressure in XCS analytically.

To keep our analyses herein as simple as possible, we introduce XCS as a pure classifier that receives independent problem instances of string length l and classifies them. We want to mention, though, that XCS was also successfully applied to several multi-step problems in which reinforcement propagation is necessary for learning an optimal behavior and the successive states (or problem instances) depend on the history of previous states and actions (Lanzi, 1999; Lanzi, 2000). While this study is restricted to classification problems, the same arguments made in this paper hold in multi-step problems and thus all conclusions drawn in this work should be transferable to multi-step and adaptive behavior applications of XCS.

This section provides a general introduction to XCS. The sections thereafter build on this knowledge to develop our theory of an accuracy-based fitness approach. Readers familiar with XCS might want to skip this section and only use it for reference purposes. For a more complete introduction to XCS the interested reader is referred to the original XCS paper (Wilson, 1995) and a more recent algorithmic description (Butz & Wilson, 2001).

2.1 Classification Problems

We define a classification problem as a problem that consists of problem instances $s \in \mathcal{S}$ that need to be classified by XCS with one of the possible classifications $a \in \mathcal{A}$. The problem then provides scalar payoff $R \in \Re$ with respect to the made classification. The goal for XCS is to choose the classification that results in the highest payoff. To do that, XCS is designed to learn a complete mapping from any possible $s \times a$ combination to an accurate payoff value. Since we restrict ourselves in this paper to Boolean input, $\mathcal{S} \subseteq \{0,1\}^l$ where l denotes the fixed string length of a problem instance. The action/classification space may be defined by $\mathcal{A} = \{a_1, ..., a_n\}$.

2.2 Knowledge Representation

XCS evolves a population $[P]$ of rules or *classifiers*. Each classifier in XCS consists of five main components. The condition $C \in \{0,1,\#\}^l$ specifies the subspace of the problem instances in which the classifier is applicable (i.e. it *matches*). The "don't care" symbol $\#$ matches in all input cases. The action part $A \in \mathcal{A}$ specifies the advocated action. The payoff prediction p approaches the average payoff encountered after executing action A in situations in which condition C matches. The prediction error ε estimates the

mean absolute deviation, or error, of the payoff prediction p. The fitness F reflects the average relative accuracy of the classifier with respect to other overlapping classifiers. Moreover, each classifier keeps an experience counter exp which counts the number of fitness updates of the classifiers, a time stamp ts which records the last GA invocation in a set the classifier was part of, an action set size estimate as which estimates the average action set size the classifier is part of, and a numerosity num which counts the number of actual micro-classifiers this *macroclassifier* represents.

2.3 Parameter Updates

With each problem instance, XCS updates its knowledge base. Given a current input s, XCS forms a *match set* $[M]$ consisting of all classifier in $[P]$ whose conditions match s. If an action is not represented in $[M]$, a covering classifier is created that matches s and specifies the unrepresented action. The probability of a don't care symbol in each attribute of the covering classifier is determined by the don't care parameter $P_{\#}$. For all classifications a, XCS forms the fitness-weighted average of all reward prediction estimates of the classifiers in $[M]$ that advocate a, i.e. the so-called *payoff prediction* $P(a)$. With respect to the payoff prediction, XCS can select the appropriate classification. After the classification is selected and sent to the problem, payoff R is provided according to which XCS updates all classifiers in the current *action set* $[A]$, which is the subset of classifiers in $[M]$ that advocate the chosen classification a. After the update and a possible GA invocation, the next iteration begins.
Classifier parameters in $[A]$ are updated as follows:

$$p \leftarrow p + \beta(R - p) \tag{1}$$

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

where β ($0 < \beta \leq 1$) denotes the *learning rate*. The fitness value of each classifier in $[A]$ is updated according to the current relative accuracy κ' of the classifier:

$$\kappa = \begin{cases} 1 & \text{if } \varepsilon < \varepsilon_0 \\ \alpha(\varepsilon_0/\varepsilon)^{\nu} & \text{otherwise} \end{cases} \tag{3}$$

$$\kappa' = \frac{\kappa \cdot num}{\sum_{cl \in [A]} cl.\kappa \cdot cl.num} \tag{4}$$

$$F \leftarrow F + \beta(\kappa' - F) \tag{5}$$

Parameters of a classifier are referred to using the dot notation ($cl.\kappa$ refers to the accuracy κ of classifier cl). Parameter ε_0 ($\varepsilon_0 > 0$) controls the tolerance for prediction error ε[1]; parameters α ($0 < \alpha < 1$) and ν ($\nu > 0$) are constants controlling the rate of decline in accuracy κ when ε_0 is exceeded. Accuracies κ in $[A]$ are then converted to set-relative accuracies κ'. Finally, classifier fitness F is updated towards the classifier's current set-relative accuracy κ'. All parameters except for fitness are updated using the *moyenne adaptive modifiée* technique (Venturini, 1994) according to which parameters are set directly to the average of the so far encountered cases as long as the average update causes a stronger change than the update due to the learning rate β would (i.e. $exp < 1/\beta$). Finally, each time the parameters of a classifier are updated, the experience counter exp of the classifier is increased by one.

[1]Note that parameter ε_0 depends on the maximal reward possible in the environment. In the remainder of the paper, provided ε_0 values refer to the fraction of maximal reward possible in the specific problem

2.4 Discovery Component

Genetic discovery is invoked in the current action set $[A]$ if the average time since the last GA application (recorded in the time stamp ts) upon the classifiers in $[A]$ exceeds a threshold θ_{ga}. 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. Parents stay in the population competing with their offspring. Free mutation is applied in which each attribute of the offspring condition is mutated to the other two possibilities with equal probability. Uniform crossover is applied as the crossover operator. Parameters of the offspring are inherited from the parents, except for the experience counter exp which is set to one, the numerosity num which is set to one, and the fitness F which is multiplied by 0.1 being rather pessimistic about the quality of the offspring.

In the insertion process, *subsumption deletion* (Wilson, 1998) may be applied to stress generalization. *GA subsumption* checks offspring classifiers to see whether their conditions are logically subsumed by the condition of another *accurate* and *sufficiently experienced* classifier in $[A]$. If an offspring is subsumed, it is not inserted in the population but the subsumer's numerosity is increased. *Action set subsumption* searches in the current action set for the most general classifier that is both *accurate* and *sufficiently experienced*. All the classifiers in the action set are tested against the general one to see whether it subsumes them (increasing again the numerosity of the subsumer and deleting the subsumed classifiers). Due to the possible disruptive effects of over-general classifiers that become temporarily accurate, we did not apply action set subsumption in our experiments.

The population of classifiers $[P]$ is of fixed size N in XCS. Excess classifiers are deleted from $[P]$ with probability proportional to an estimate of the size of the action sets that the classifiers occur in (stored in parameter as). If the classifier is sufficiently experienced ($exp > \theta_{del}$) and its fitness F is significantly lower than the average fitness of classifiers in [P], its deletion probability is further increased.

3 Evolutionary Pressures

With the evolutionary processes in mind, we can now give an overview of all identified evolutionary pressures in XCS. Each pressure is briefly introduced and analyzed. This section provides a short introduction to previous insights. The relevant pressures for this paper are analyzed in detail in the successive sections.

The five evolutionary pressures identified elsewhere (Butz & Pelikan, 2001) are as follows:

1. *Set pressure* represents the general tendency to evolve more general classifiers due to selection in the action set and deletion in the population.

2. *Mutation pressure* pushes towards an equal number of zeros, ones, and don't cares in the binary case.

3. *Deletion pressure* emphasizes the evolution of equally distributed classifier subsets and high-fitness classifiers additional to the set pressure influence.

4. *Subsumption pressure* contributes mainly to speed-up convergence in the population. It only applies if the environment is deterministic or only slightly stochastic.

5. *Fitness pressure* is the main concern of this paper. Fitness pressure is intended to be

the pressure from inaccuracy to accuracy which often coincided with from overgeneral to accurate, maximally general classifiers.

The five pressures are discussed in more detail in the following subsections.

3.1 Set Pressure

The set pressure was the main subject of analysis of Butz and Pelikan (2001). This pressure represents the previously hypothesized pressure towards generality stated in Wilson's *generalization hypothesis* (Wilson, 1995). It was further investigated and emphasized in Kovacs' *optimality hypothesis* (Kovacs, 1997).

Intuitively, the set pressure formalizes that the average specificity in an action set $[A]$ will be lower than the average specificity in the population $[P]$ since more general classifiers match with a higher probability. Butz, Kovacs, Lanzi, and Wilson (2003) show that the average specificity in an action set (or match set) given the average specificity in the current population can be approximated by

$$s([A]) = \frac{s([P])}{2 - s([P])} \tag{6}$$

denoting $s([X])$ the average specificity in set $[X]$. Thus, during one GA application, it can be expected that the average specificity in the population (disregarding all other pressures for now) changes as follows:

$$\Delta_{set} s([P]) = 2 \frac{\left(\frac{s([P])}{2 - s([P])} - s([P]) \right)}{N} \tag{7}$$

where N denotes the number of (micro-) classifier in the population. The fraction is multiplied by two since two offspring classifiers are generated in one reproductive event.

3.2 Mutation Pressure

Although usually only a low mutation probability is applied, mutation still influences specificity. Generally, a random mutation process causes a tendency towards an equal number of symbols in a population. Thus, applying random mutations the result will be a population with an approximately equal proportion of 0, 1, and $\#$ symbols in the condition parts of the classifiers in a binary LCS. Thus, mutation results in a pressure towards an equal number and an equal distribution of symbols in classifier conditions.

The average change in specificity between the parental classifier $s(cl(t))$ and the mutated offspring classifier $(s(cl(t+1))$ can be written as

$$\begin{gathered} \Delta_{mut} = s(cl(t+1)) - s(cl(t)) = \\ s(cl(t))(1 - \mu/2) + (1 - s(cl(t))\mu - s(cl(t)) = \\ 0.5\mu(2 - 3s(cl(t))) \end{gathered} \tag{8}$$

Thus, mutation alone pushes the population towards a specificity of $0.6\overline{6}$. Note that although this influence might be considered weak, it was shown that mutation can significantly alter population specificity (Butz & Pelikan, 2001).

Combining set and mutation pressure to a general *specificity equation* (Butz, Kovacs, Lanzi, & Wilson, 2003) a general estimate of the change in the population's specificity can be derived.

$$\Delta_{spe} s([P]) = f_{ga} \frac{2\left(s([A]) + \Delta_{mut} - s([P])\right)}{N} \tag{9}$$

Table 1: Converged Specificities in Theory and in Empirical Results

μ	0.02	0.04	0.06	0.08	0.10	0.12	0.14	0.16	0.18	0.20
$s([P])$, theory	0.040	0.078	0.116	0.153	0.188	0.223	0.256	0.288	0.318	0.347
$s([P])$, empirical	0.053	0.100	0.160	0.240	0.287	0.310	0.329	0.350	0.367	0.394

Table 2: Mutation settings for desired specificities

$s([P])$	0.1	0.2	0.3	0.4	0.5	0.6	0.7
μ	0.051	0.107	0.168	0.240	0.333	0.480	0.840

The parameter f_{ga} denotes the average frequency of GA application. This formula allows for an accurate prediction of specificity change and convergence over time given no fitness influence (Butz & Pelikan, 2001; Butz, Kovacs, Lanzi, & Wilson, 2003).

From Equation 9 it is now possible to derive the specificity the population will converge to (regardless of the initial specificity) by setting the difference $(s[A])+\Delta_{mut}-(s[P])$ to zero:

$$\begin{aligned} s([A]) + \Delta_{\mu_f} &= s([P]) \\ \frac{s([P])}{2 - s([P])} + \frac{\mu}{2}\left(2 - 3\frac{s([P])}{2 - s([P])}\right) &= s([P]) \end{aligned} \tag{10}$$

Solving for $s([P])$:

$$\begin{aligned} s([P])^2 - (2.5\mu + 1)s([P]) + 2\mu &= 0 \\ s([P]) &= \frac{1 + 2.5\mu - \sqrt{6.25\mu^2 - 3\mu + 1}}{2} \end{aligned} \tag{11}$$

Solving for μ:

$$\begin{aligned} \mu\left(2 - \frac{5}{2}s([P])\right) &= s([P]) - s([P])^2 \\ \mu &= \frac{s([P]) - s([P])^2}{2 - \frac{5}{2}s([P])} \end{aligned} \tag{12}$$

Applying the above two equations enables us now to determine the expected specificity in the population given a fixed mutation probability. On the other hand, given a desired specificity in the population, we can determine an appropriate mutation probability to achieve that specificity. Table 1 shows the resulting specificities in theory and empirically determined on a random Boolean function (a Boolean function that returns randomly either zero or one). Note that the empirical results actually reveal slightly higher values than determined. This effect was investigated in detail in Butz and Pelikan (2001) and is due to the higher noise in more specific classifiers in combination with the biased accuracy function as well as the fitness biased deletion method. Table 2 shows the necessary mutation rates for desired specificities. In practice, slightly lower mutation rates should actually lead to the desired specificities due to intrinsic fitness influences. Note that apart from its implicit specialization pressure (when currently more general than $s([P]) = 0.6\overline{6}$) mutation can have a negative effect when set too high possibly disrupting subsolutions.

3.3 Deletion Pressure

The main part of the deletion pressure is already included in the set pressure. Additional to deleting classifiers from the population in difference to the reproduced classifiers in the action set, the deletion is biased towards deleting classifiers that populate large niches and classifiers with a fitness value which is significantly smaller than the average value of the population. Hereby, the bias towards large niche deletion has a rather small effect as pointed out by Kovacs (1999) and further shown in Butz and Pelikan (2001).

3.4 Subsumption Pressure

Subsumption pressure results in a further pressure towards generality. However, since this pressure is limited to complete accuracy, it only applies in deterministic environments once XCS evolved accurate classifiers. Thus, the innovative part of an evolutionary process does not apply in this pressure. While the main goal of the subsumption pressure is to decrease the population size boiling it down to the essential classifiers, subsumption must be applied with care. Too low thresholds of θ_{sub} as well as too high values of ε_0 can actually cause subsumption to cause fundamental loss of information in the population (by subsuming accurate classifiers with a temporarily accurate, over-general classifier).

Action set subsumption can be very strong and cause a whole niche to collapse. If an over-general classifier becomes temporarily accurate (due to low error probability, by chance biased sampling, and high β values) it will absorb all more specific classifiers and essentially all (more specific) accurate, maximally general classifiers.

3.5 Fitness Pressure

Finally, we arrive at the fitness pressure—the main drive towards accuracy from the inaccurate (and thus mostly over-general) side. Fitness pressure is analyzed in the next sections in more detail. For this section, it suffices to understand that fitness pressure is meant to continuously push the evolution of more-accurate classifiers. Thus, fitness is the major pressure in XCS that guides the evolutionary process towards higher accuracy. We will thus often talk of *fitness* or *accuracy guidance*.

3.6 Pressure Interaction

The interaction of the pressures is sketched out in Figure 1. As mentioned above, we see that apart from the (rather weak) push of mutation pressure, only fitness pressure pushes towards accuracy from the over-general side. However, while mutation is a completely random pressure towards higher specificity (assuming a specificity of less than $0.6\overline{6}$), in many problems fitness pressure should be able to detect structure, or *building blocks*, before reaching accuracy. As we will see, fitness pressure is consequently mandatory if it is not possible to cover the whole specificity level necessary for a successful evolution. This will become clearer in the next sections.

We now first focus on how fitness pressure can be made as strong and as reliable as possible assuming that accuracy, and consequently fitness, continuously increases the closer a classifier is to maximal accuracy. Next, we also question this assumption and show that in some environments this might actually not be the case. Finally we redefine fitness to overcome this problem.

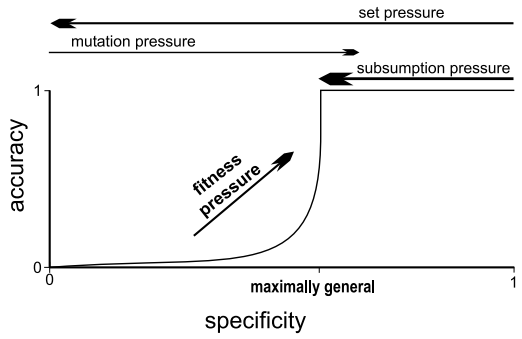

Figure 1: The sketched pressure interaction reveals how XCS is designed to evolve accurate, maximally general classifiers.

4 Strong and Stable Fitness Pressure: Tournament Selection

Selection in XCS and other LCSs has always been done by means of proportionate selection. Although it is known from GA literature that proportionate selection is subject to many pitfalls (Goldberg & Deb, 1991; Goldberg & Sastry, 2001), the LCS community has adhered to this selection method. This section shows that XCS can actually suffer from the pitfalls of proportionate selection as well. Moreover, we show that tournament selection solves the problem resulting in a strong and stable fitness pressure.

4.1 Proportionate vs. Tournament Selection

Proportionate selection was analyzed in Holland's original GA work (Holland, 1975). However, as known from GA literature (Baker, 1985; Goldberg & Deb, 1991), proportionate selection is strongly dependent on fitness scaling. Moreover, fitness pressure strongly depends on the current fitness distribution in the population effectively decreasing fitness pressure once fitness differences decrease. For example, Goldberg and Sastry (2001) note that proportionate selection schemes have a tendency to stall if used with an unscaled fitness approach.

Fitness of XCS classifiers is derived from the scaled, set-relative accuracy. While the scaling usually works well, what if all classifiers are currently similarly inaccurate with the better classifiers being only slightly more accurate? Similar accuracy of all classifiers in an action set should decrease or even annihilate fitness pressure since proportionate selection is used.

Tournament selection, on the other hand, does not care about the current relative fitness differences. What matters is fitness rank. Thus, tournament selection should not suffer from fitness scaling nor should it be impaired if the relative accuracy difference is very small.

4.2 Failure in the Multiplexer Problem

To reveal the problem of proportionate selection, we apply XCS to the multiplexer problem with altered parameter settings or with additional noise in the payoff function of the problem. We show that learning in XCS with proportionate selection is disrupted if the learning parameter β is set too low or if noise in the payoff function is too high. The multiplexer problem is introduced in the appendix.

Unless otherwise stated, all results herein are averaged over 50 experimental runs. Performance is assessed by test trials in which no learning takes place and the better

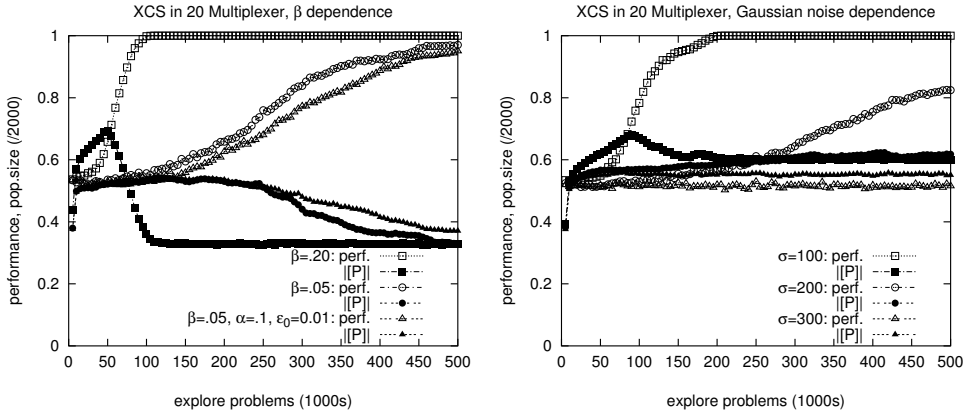

Figure 2: Decreasing learning rate β impairs XCS's learning performance (left-hand side). The setting of the accuracy function parameters has no immediate influence. Adding Gaussian noise to the payoff function of the multiplexer problem quickly deteriorates performance of XCS (right-hand side).

prediction array value is chosen as the classification. During normal learning, classifications are chosen at random. If not stated differently, parameters were set as follows: $N = 2000$, $\beta = 0.2$, $\alpha = 1$, $\varepsilon_0 = .001$, $\nu = 5$, $\theta_{GA} = 25$, $\chi = 0.8$, $\mu = 0.04$, $\theta_{del} = 20$, $\delta = 0.1$, $\theta_{sub} = 20$, and $P_{\#} = 1.0$. Note that we intentionally chose $P_{\#}$ very high to always start with an over-general population. Section 5.1 mathematically shows that a start with a fairly general population is definitely necessary in problems with large input strings since otherwise the high specificity may prevent useful reproductive events.

Figure 2 reveals the strong dependence on parameter β. Decreasing the learning rate hinders XCS from evolving an accurate payoff map of the problem. The problem is that over-general classifiers occupy a big part of the population initially. A better offspring often loses against the over-general parents since the fitness of the offspring only increases slowly (due to the low β value) and small differences in the fitness F only have small effects when using proportionate selection. Altering the slope of the accuracy curve by changing parameters α and ε_0 does not have any positive learning effect either. Although one could say that β should simply not be set that low, Section 4.4.1 shows that a low β rate is necessary to be able to handle noisy problems. Another possibility for preventing the problem would be to have a separate learning rate β_F for the fitness estimate which could be set higher than the learning rate for the other estimates. Since tournament selection makes XCS more independent from parameter β in general we did not pursue the β_F idea any further in this paper.

In addition to the dependence on β we can show that XCS is often not able to solve noisy problems. We added Gaussian noise with mean zero and standard deviation σ to the payoff provided by the environment. Each time payoff is provided, instead of providing the actual payoff to XCS we provide the payoff plus a noise value generated from the Gaussian distribution. Figure 2 shows that when adding only a small amount of noise, XCS fails to evolve a complete and accurate classification model. One problem is that in the noisy case the prediction error ε of maximally accurate classifiers no longer reaches the threshold ε_0 so that the strong distinction between accurate and inaccurate classifiers does not apply (note also that subsumption does not apply so that the pop-

ulation size stays larger even if the problem is learned successfully). Additionally, the larger the noise the less the difference between accurate and inaccurate classifiers. Due to this decreasing difference in accuracy and thus in the fitness values, proportionate selection soon does not provide enough selection pressure anymore when Gaussian noise with a higher standard deviation is added and the population starts to drift at random. Lanzi and Colombetti (1999) proposed an extension to XCS that detects noise in environments and adjusts the error estimates accordingly. This approach, however, only works around the basic problem of proportionate selection.

Herein, we show that XCS with tournament selection is suitable for both cases. In fact, tournament selection enables XCS to solve a much broader class of problems in general.

4.3 Implementing Tournament Selection

As the name implies, tournament selection refers to a GA selection process in which tournaments are held among a subset of individuals of a population. Individuals that take part in the tournament are usually selected at random from the population. Usually, a fixed tournament size is used. The individual in the tournament with the highest fitness wins and is reproduced.

Three differences have to be considered between a common GA and the evolutionary approach in XCS. (1) The GA in XCS is a steady-state GA since only two classifiers are replaced in each GA application. (2) Selection is applied to subsets of the population (i.e. the action sets) which have varying sizes. (3) Initially, action sets are often overpopulated by highly over-general classifiers so that only a small fraction of an action set might have a significantly higher fitness than the average. This has several consequences for the appropriate selection method. (a) Some classifiers might not be considered at all for reproduction before being deleted since they are simply never part of an action set. (b) A high number of over-general classifiers can shadow the few more-accurate classifiers. Thus, a relatively strong selection pressure appears to be necessary which adapts to the current action set size.

Consequently, our tournament selection process holds tournaments of sizes dependent on the current action set size. The size of each tournament has the size of the fraction $\tau \in (0, 1]$ of the current action set size. Instead of selecting the two parental classifiers in a GA invocation by proportionate selection, two independent tournaments are now held in which the classifier with the highest fitness wins the tournament. We also experimented with fixed tournament sizes that were recently also used in Kovacs (2002). Several experiments showed, however, that performance is not reliable and thus inferior to set-relative tournament selection (Butz, Sastry, & Goldberg, 2003). Parameter τ is set to 0.4 in the subsequent experimental runs. The experimental study in Butz, Sastry, and Goldberg (2003) also shows that XCS performance is very robust with respect to τ resulting in similar performance for values ranging from 0.2 up to 0.8.

4.4 Empirical Results with Tournament Selection

This section compares XCS performance without and with tournament selection in the multiplexer problem and in the Count Ones and Layered Count Ones problem (please see the appendix for exact problem definitions). It is shown that XCS with tournament selection, referred to as *XCSTS* in the remainder of this work, outperforms XCS without tournament selection in all settings.

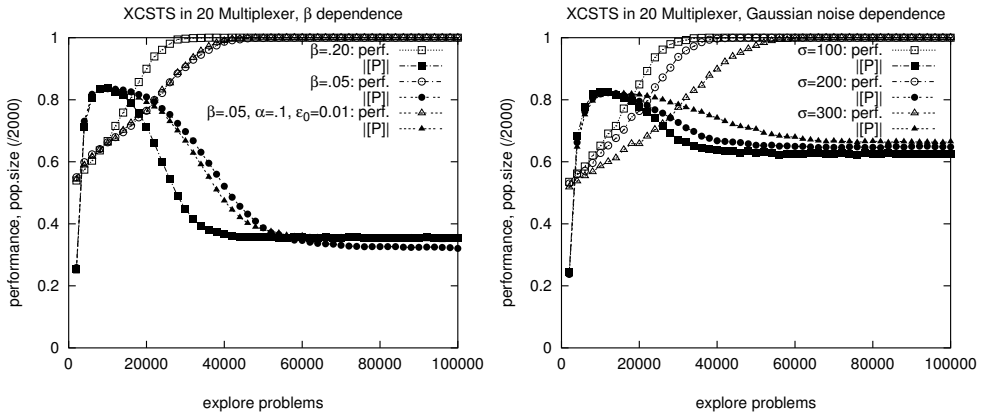

Figure 3: XCSTS is barely influenced by a decrease of learning rate β (left-hand side). Also the setting of the accuracy function parameters has no significant influence. Adding Gaussian noise influences performance of XCSTS to a much lesser degree than XCS with proportionate selection.

4.4.1 Parameter β and Noise Dependence in the Multiplexer Problem

Figure 3 shows that XCSTS can solve the 20-multiplexer problem even with a lower parameter value β. Although learning speed is still influenced, the curve shows that XCSTS is much more independent of parameter β. Due to the independence on fitness scaling, more accurate offspring are detected and propagated faster.

Figure 3 (right-hand side) also shows that XCSTS is much more robust in noisy problems. Especially in the Gaussian noise case, XCSTS is barely influenced by noise with a standard deviation of 250 whereas XCS was not able to solve a standard deviation of 100. As expected, the population sizes do not converge to the sizes achieved without noise since subsumption does not apply. Nonetheless, in both cases the population sizes decrease indicating the detection of accurate classifiers.

Figure 4 (left-hand side) shows to what extent XCSTS depends on parameter β. Adding Gaussian noise of $\sigma = 250$ does not alter performance when switching between $\beta = 0.2$ and $\beta = 0.05$. When decreasing β even further, though, XCSTS also takes longer to evolve a proper model or fails completely (when setting $\beta = 0.0005$, a rather extreme value). XCSTS is even able to handle Gaussian noise with standard deviation $\sigma = 500$. A β value of 0.2 is too high in the case of such high noise. Since temporal difference updates are applied to the learning parameters, the variance of payoff prediction p and prediction error estimate ε remains so high so that the fitness estimate remains meaningless. Consequently, fitness pressure is often misleading and the evolutionary process takes much longer. The smaller the parameter β, the more accurate the values will be on the long run (given a stationary environment). Thus, in very noisy environments parameter β needs to be decreased to generate accurate payoff estimates. However, setting β too low will hinder the offspring from becoming more accurate, as we observed when β was set to 0.0005. With the moderate setting of $\beta = 0.05$, the problem is solvable for XCSTS and 100% performance is reached.

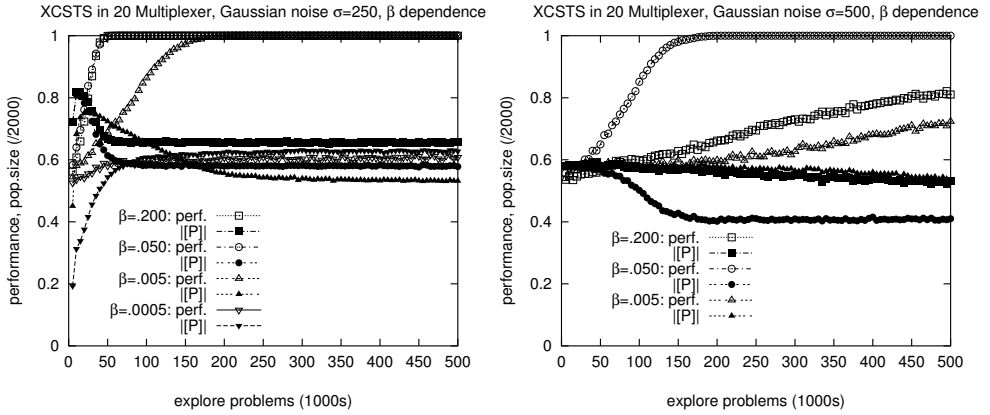

Figure 4: Performance of XCSTS in the Boolean multiplexer problem with additional Gaussian noise of $\sigma = 250$ (left-hand side) and $\sigma = 500$ (right-hand side) shows that a high parameter value β can decrease performance due to strong fluctuations in parameter values while a too small parameter value β can decrease performance since it takes too long to reach an accurate value.

4.4.2 Fitness Exploitation in the Count Ones Problems

GA research has actively studied crossover operators, linkage learning, and the probabilistic model building GAs (PMBGAs) (Pelikan, Goldberg, & Lobo, 2002), LCS research has somewhat neglected the investigation of the crossover operator. This section shows that XCSTS is able to exploit beneficial recombinatory events more efficiently.

The count ones problems (defined in the appendix) are designed to show the usefulness of recombination in XCS. Essentially, in the count ones problems the accuracy of a classifier gradually increases as the more relevant attributes are specified uniformly to one or to zero. Thus, recombination of classifiers can effectively result in the generation of higher-accurate offspring.

Figure 5 shows the performance of XCS and XCSTS in the count ones problem with a string length $l = 20$ and $k = 7$ relevant bits. It can be seen that XCSTS learns the problem faster since it is able to exploit the fitness guidance in the problem more effectively. Moreover, it is much less dependent on the chosen mutation rate. With a rate of $\mu = 0.04$ the supply of more specialized classifiers is increased so that XCS does not rely on proper fitness guidance as much. However, with the lower rate of $\mu = 0.01$ XCS's performance decreases much more than the performance of XCSTS which shows that XCSTS is able to exploit fitness guidance beginning from higher generality. Without the crossover operation both versions suffer which points out that XCS as well as XCSTS successfully recombine classifiers to generate better offspring.

In the *layered count ones problem* any specialization in the relevant bits results in fewer possible reward outcomes and essentially a lower mean absolute deviation. Thus, the more a classifier is able to specialize the relevant attributes, the higher its accuracy and consequently its relative fitness. Thus, this problem allows an even more effective fitness exploitation. Figure 6 shows runs in the layered count ones problem with string length $l = 20$ and $k = 7$ relevant bits. It can be seen that despite the high learning rate β performance of XCS with proportionate selection suffers from a low mutation rate. This indicates that XCS is hardly able to exploit the fitness guidance in

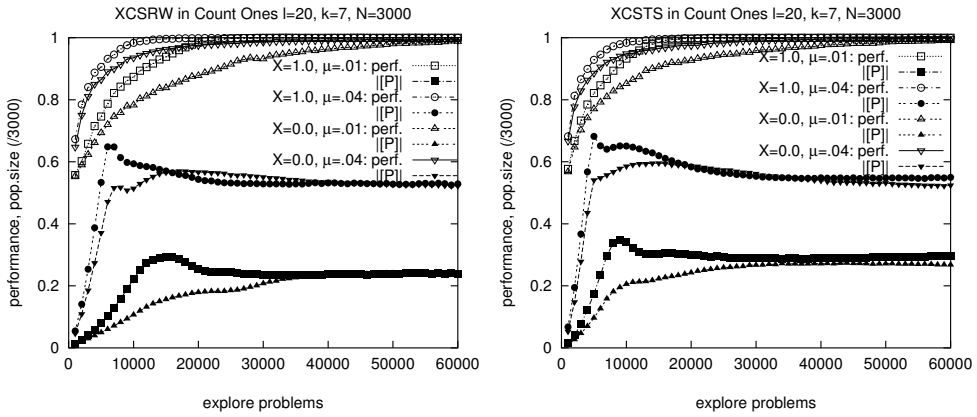

Figure 5: In the count ones problem, XCS with proportionate selection (left-hand side) performs nearly identical to XCSTS when mutation is set to $\mu = 0.04$. With a lower mutation rate, the supply of more accurate classifiers decreases. XCSTS is still able to exploit fitness guidance and consequently suffers less from a lower mutation rate. Regardless of the selection method used, though, performance decreases when crossover is not applied indicating the successful recombination of lower order building blocks.

more general classifiers. Additionally, XCS without crossover hardly differs from XCS with crossover application indicating only few successful recombinatory events. XCSTS, on the other hand, shows a proper fitness exploitation (figure 6, right-hand side). Setting the crossover probability χ to zero has a strong effect on performance suggesting successful recombinatory events when crossover is applied. Thus, XCSTS detects and exploits accuracy guidance much more reliably and enables proper recombination of sub-optimal building blocks.

4.5 The Consequences of Tournament Selection for XCS

Our study of XCS has shown that proportionate selection results in several drawbacks that can be easily remedied. First, a strong parameter dependence especially on the learning parameter β was observed which is due to the slower fitness adaptation of offspring. Second, noise degraded performance of XCS since proportionate selection is dependent on fitness scaling which is altered when noise is added. Third, fitness guidance in the count ones problems was not exploited well.

Simple tournament selection with tournament sizes proportionate to the actual action set size remedied all those drawbacks. The system became much more independent from parameter settings, noise was manageable to a much higher degree, and fitness guidance was exploited more efficiently. In fact, no problem was found in which tournament selection with a tournament size $\tau = 0.4$ in proportion to the current action set size had a worse performance than proportionate selection. We also showed that crossover is highly beneficial in the count ones problems. Since these problems are designed to exhibit successful recombinatory events and XCSTS benefited from crossover, we confirmed that tournament selection provides a good basis for the successful recombination of low-order building blocks.

We applied XCSTS in several other Boolean function problems and showed throughout that XCSTS outperforms XCS in all settings. Some of these runs can be

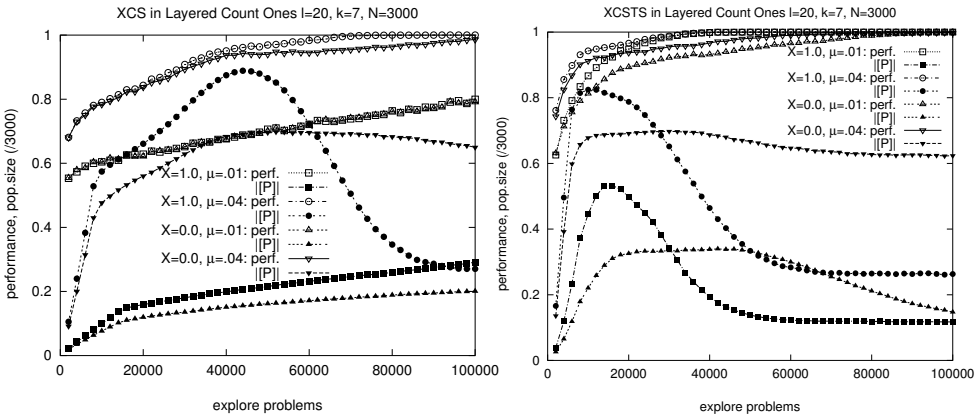

Figure 6: In the layered count ones problem, XCS without tournament selection is not able to exploit the fitness guidance efficiently when the average specificity in the population is too low ($\mu = 0.01$). XCSTS always detects available fitness guidance. When crossover is not applied, performance strongly decreases which indicates that XCSTS successfully recombines classifiers.

found in (Butz, Sastry, & Goldberg, 2003). Wilson (personal communication) successfully implemented tournament selection according to (Butz, Sastry, & Goldberg, 2003) and observed the same results. In smaller problems (the 11-multiplexer), however, he observed that the benefit is not observable anymore. This seems to be due to the fact that the 11-multiplexer is rather easy to solve. Tournament selection is a step towards solving larger problems and providing a basis for successful recombination in LCSs in general. In the remainder of this paper we will use XCSTS only.

5 When Fitness Pressure Applies

The previous section showed that fitness pressure can be made much more reliable by applying tournament selection. The evolving fitness values were exploited well, guiding the evolutionary process towards the desired accurate classifiers. This worked fine in the problems shown. This section investigates when fitness pressure applies in general with respect to the type of problem and chosen parameter settings.

Three preconditions need to be satisfied in order to ensure a proper fitness pressure: (1) higher accurate classifiers need to be part of a GA application, (2) higher accurate classifiers must exist or must be generated by mutation, and (3) parameter estimates need to be sufficiently accurate. These three aspects are addressed in the following.

5.1 Reproductive Opportunity

The first precondition assures that higher-accurate classifiers have a reproductive opportunity before they are deleted. To ensure this, the expected time until a reproductive opportunity should be larger than the expected time until deletion. This constraint effectively results in a population size bound since only a larger population size can increase the time until deletion in XCS.

To determine this bound, let's first determine the expected number of steps until deletion. Assuming neither any fitness estimate influence nor any action set size esti-

mate influence, the probability of deletion is random. It can be approximated by the probability of choosing the higher-accurate classifier for deletion among all possible N classifiers.

$$P(\text{deletion}) = \frac{1}{N} \tag{13}$$

To make the reproductive opportunity of a classifier with a certain specificity s_{cl} probable, the probability of reproduction should be higher than the one for deletion. A reproductive opportunity takes place if the classifier is part of an action set. The probability of being part of an action set can be determined by

$$P(\text{in}[A]) = \frac{1}{n} 0.5^{l \cdot s_{cl}} \tag{14}$$

Note that this probability assumes binary input strings and further that all 2^l possible binary input strings occur with equal probability. Combining Equation 13 with Equation 14, we can consequently determine a necessary condition for a classifier having at least one reproductive opportunity before deletion. Since two classifiers are deleted randomly from the population, and since classifiers are reproduced in the current action set and not in two different actions sets, the probability of a reproductive opportunity needs to be larger than approximately double the deletion probability:

$$\begin{aligned} P(\text{in}[A]) &> 2P(\text{deletion}) \\ \frac{1}{n} 0.5^{l \cdot s_{cl}} &> \frac{2}{N} \\ N &> n 2^{l \cdot s_{cl}+1} \end{aligned} \tag{15}$$

This bounds the population size by $O(n2^{ls})$. We will see, that in many problems the specificity s can be set to $s = 1/l$ so that the bound actually diminishes. However, in problems in which no fitness guidance can be expected from the over-general side, we have to derive s differently. To characterize problems without fitness guidance, we make use of the schema notation (Holland, 1975) in genetic algorithms.

A schema of a string of length l is usually defined as a string that specifies some of the positions and ignores others. The number of specified positions is termed the *order* of the classifier k. This actually corresponds to the notation of the condition parts in LCSs where the number of specialized attributes (i.e. the non-don't care symbols) corresponds to the order. As outlined in (Butz, Kovacs, Lanzi, & Wilson, 2001), a schema in learning classifier systems is represented by a classifier if the classifier correctly specifies at least all positions that are specified in the schema. Thus, a representative of a schema has a specificity of at least k/l. For example, a classifier with condition $C =$ `##10#0` is a representative of schema `**10*0`, but also of schemata `**10**`, `**1**0`, `***0*0`, `**1***`, `***0**`, `*****0`, and `******`.

Problems can now be said to be of order of difficulty k_d if the minimal-order schema that provides fitness guidance is of order k_d. A schema provides fitness guidance, if the probability of consistently classifying correct/incorrect is larger than it is for random classification. Section 6 is making this more concrete. For now, it is sufficient to assume that for a problem of order of difficulty k_d XCS needs to ensure reproductive opportunities to classifiers that represent proper order k_d schemata.

The expected specificity of such a representative of the order k schema can be estimated given a current specificity in the population of $s([P])$. Given that the classifier

specifies all k positions its expected average specificity will be approximately:

$$E(s(\text{representative of schema of order } k)) = \frac{k + (l-k)s([P])}{l} \tag{16}$$

Substituting s_{cl} in Equation 15 with the expected specificity of a representative of a schema of order k, the population size N can be bounded by

$$\begin{aligned} N &> n2^{l \cdot \frac{k+(l-k)s([P])}{l}+1} \\ N &> n2^{k+(l-k)s([P])+1} \end{aligned} \tag{17}$$

This bound ensures that the classifiers necessary in a problem of order of difficulty k_d get reproductive opportunities. Once the bound is satisfied, existing representatives of an order k_d schema are ensured to reproduce and XCS is enabled to evolve a more accurate population.

Note that this population size bound is actually exponential in schema order k_d and in string length times specificity $ls([P])$. This would mean, that XCS would scale exponentially in the problem length which is certainly highly undesirable. However, we can show that the necessary specificity in $[P]$ decreases with larger population sizes. Considering this, we show in the following that out of the specificity constraint a general *reproductive opportunity bound (ROP-bound)* can be derived that shows that population size grows in $O(l^{k_d})$.

5.2 Presence and Generation of Representative

While the above bound ensures the reproduction of *existing* classifiers that represent a particular schema, it does not assure the actual presence or generation of such a classifier. This relates to the previously identified *schema challenge* (Butz, Kovacs, Lanzi, & Wilson, 2001). Given a current specificity in the population $s([P])$ and assuming a uniform distribution of specified positions in the condition parts of the classifiers in $[P]$, the following derivation for the necessary specificity is possible ensuring the existence of at least one representative of a particular schema of order k in the population.

$$\begin{aligned} P(representative) &= \frac{1}{n}(s([P]))^k \\ E(representative) &= N\frac{1}{n}(s([P]))^k \\ E(representative) &> 1 \\ s([P]) &> (\frac{n}{N})^{1/k} \end{aligned} \tag{18}$$

Thus, the necessary specificity given a population size of N and the necessary representation of an unknown schema of order k is bounded by

$$s = O((\frac{n}{N})^{1/k}) \tag{19}$$

Substituting this bound in the O-notated dependence of the representative bound on string length l ($N = O(2^{ls([P])})$) and ignoring the additional constants, we can derive

the following enhanced population size bound.

$$\begin{aligned} N &> 2^{l(\frac{n}{N})^{1/k}} \\ \log_2 N &> l(\frac{n}{N})^{1/k} \\ N^{1/k}\log_2 N &> ln^{1/k} \\ N(\log_2 N)^k &> nl^k \end{aligned} \tag{20}$$

This *reproductive opportunity bound* (ROP-bound) essentially shows that population size N needs to grow approximately exponentially in the order of the minimal building block k_d (i.e., problem difficulty) and polynomial in the string length.

$$N \approx O(l^{k_d}) \tag{21}$$

Note that in the usual case, k_d is rather small and can often be set to one. Essentially, when k is greater than one, other classification systems in machine learning have also shown a similar scale up behavior. For example, the inductive generation of a decision tree in $C4.5$ (Quinlan, 1993) would not be able to decide on which attribute to expand first (since any expansion leads to the same probability distribution) and consequently would generate an inappropriately large tree.

The *representative bound* actually also extends in time given we start with a completely general population (i.e. $P_\# = 1.0$) since the *set pressure* needs to push the population towards the intended specificity before a representative can evolve.

Given a mutation probability μ, the probability can be approximated that a classifier will be generated that has all k relevant positions specified given a current specificity $s([P])$.

$$P(\text{generation of representative}) = (1-\mu)^{s([P])k} \cdot \mu^{(1-s([P]))k} \tag{22}$$

Out of the probability, we can determine the expected number of steps until at least one classifier may have the desired attributes specified. Since this is a geometric distribution,

$$\begin{aligned} &E(t(\text{generation of representative})) = \\ &1/P(\text{generation of representative}) = \\ &\left(\frac{\mu}{1-\mu}\right)^{s([P])k} \mu^{-k} \end{aligned} \tag{23}$$

Given a current specificity of zero, the expected number of steps until the generation of a representative consequently equals to μ^{-k}. Thus, given we start with a completely general population, the expected time until the generation of a first representative is less than μ^{-k} (since $s([P])$ increases over time). Requiring that the expected time until a classifier is generated is smaller than some threshold Θ, we can generate a lower bound on the mutation μ:

$$\begin{aligned} \mu^{-k} &< \Theta \\ \mu &> \Theta^{\frac{1}{-k}} \end{aligned} \tag{24}$$

This representative bound actually relates to the existence of a representative bound determined above in Equation 18. Setting Θ equal to N/n we get the same bound (since s can be approximated by $a \cdot \mu$ where $a \approx 2$ is a constant, see Equation 11 for a more exact derivation of s).

5.3 Sufficiently Accurate Values

Although we assume in the main parts of this paper that the estimation values of XCS are sufficiently accurate, we need to note that this assumption does not hold true necessarily. All classifier parameters are only an approximation of the average value. The higher parameter β is set, the higher the variance of the parameter estimates will be. Moreover, while the classifiers are younger than $1/\beta$, the estimation values are approximated by the average, so far encountered values. Thus, if a classifier is younger than $1/\beta$, its parameter variances will be even higher than the one for experienced classifiers.

This leads us to a stronger bound on the population size. Requiring that each offspring has the chance to be evaluated at least $1/\beta$ times to get as close to the real value as possible, the reproductive opportunity bound needs to be increased by the number of evaluations we require.

From Equation 13 and Equation 14, we can derive the expected number of steps until deletion and similarly, we can derive the expected number of evaluations during a time period t.

$$E(\text{\# steps until deletion}) = \frac{1}{P(\text{deletion})}/2 = \frac{N}{2} \tag{25}$$

$$E(\text{\# of evaluations in } t \text{ steps}) = P(\text{in } [A]) \cdot t = \frac{1}{n} 0.5^{ls_{cl}} t \tag{26}$$

The requirement for success can now be determined by requiring that the number of evaluations before deletion must be larger than Ψ where Ψ could for example be set equal to $1/\beta$.

$$\begin{aligned} &E(\text{\# of evaluations in (\# steps until deletion)}) > \Psi \\ &\frac{1}{n} 0.5^{ls_{cl}} \frac{N}{2} > \Psi \\ &N > \Psi n 2^{ls_{cl}+1} \end{aligned} \tag{27}$$

Setting Ψ to one, Equation 27 is equal to Equation 17 since one evaluation is equal to at least one reproductive opportunity. In general, the sufficient evaluation bound only increases the reproductive opportunity bound by a constant so that the overall bound stays the same.

As a last point in this section, we want to point out that fitness is actually not computed by averaging, but the Widrow-Hoff delta rule is used from the beginning. Moreover, fitness is always set to 10% of the parental fitness value (to prevent disruption). Thus, fitness is derived from two approximations and it starts off with a disadvantage so that the early evolution of fitness strongly depends on fitness scaling and an accurate approximation of the prediction and prediction error estimates. To ensure a fast detection and reproduction of superior classifiers it is consequently necessary to choose initial classifier values as accurate as possible.

5.4 Bound Verification

We show that the derived bounds hold using a Boolean function problem of order of difficulty k_d where k_d is larger than one. The hidden parity function, originally investigated in Kovacs and Kerber (2001), is very suitable to manipulate k. The basic problem is represented by a Boolean function in which k relevant bits determine the outcome. If the k bits have an even number of ones, the outcome will be one and zero otherwise.

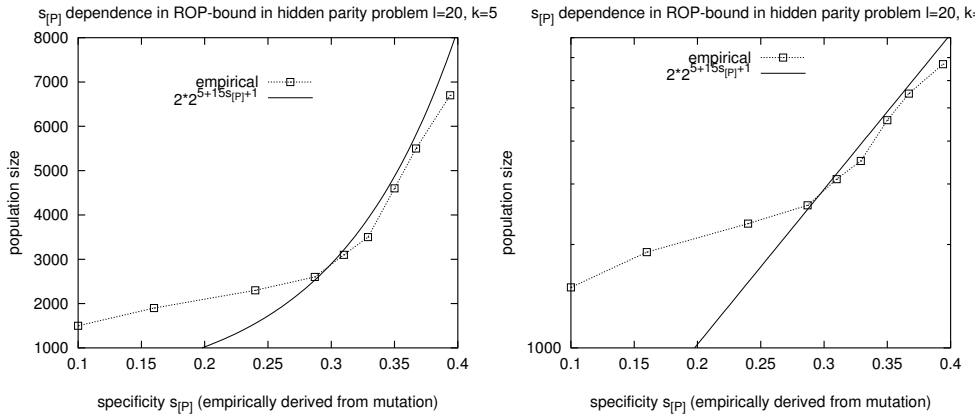

Figure 7: Minimal population size depends on applied specificity in the population (manipulated by varying mutation rates). When mutation and thus specificity is sufficiently low, the bound becomes obsolete.

The order of difficulty k_d is equivalent to the number of relevant attributes k. The major problem in the hidden parity is that there is no fitness-guidance whatsoever before the k relevant hidden parity bits are specified. That is, if a classifier does not specify the values of all o relevant positions, it has a $50/50$ chance of getting the output correct. Thus, the accuracy for all classifiers that do not have all k bits specified is approximately equal (more information on the hidden parity problem can be found in the appendix).

Figure 7 shows that the reproductive opportunity bound is nicely approximated by Equation 17. The experimental values are derived by determining the population size needed to reliably reach 100% performance. The average specificity in the population is derived from the applied mutation using Table 1. XCSTS is assumed to reach 100% performance reliably when all 50 runs reach 100% performance after $200,000$ steps. Although the bound corresponds to the empirical points when specificity is high, in the case of lower specificity the actual population size needed departs from the approximation. The reason for this is the *representative generation bound* evaluated above.

For $k = 5$ and a mutation rate of 0.08 the representative generation bound determines a population size of more than 2000 so that the reproductive opportunity bound becomes obsolete. For even lower specificities, the generation of representative bound becomes a major factor as well so that the scale up curve starts to expand in time as well. This can be seen in Figure 8 in which performance becomes rather independent of the chosen population size when mutation, and thus specificity, is sufficiently low.

To validate the $O(l^k)$ bound of Equation 21, we ran further experiments in the hidden parity problem with $k = 4$, varying l and using an optimal mutation rate μ (and thus specificity). Results are shown in Figure 9. The bound was derived from experimental results averaged over 50 runs determining the population size for which 98% performance is reached after 300000 steps. With appropriate constants added, the experimentally derived bound is well-approximated by the theory. Showing the same bound on a $\log$-scale confirms that the population size needs to grow polynomial in the string length l.

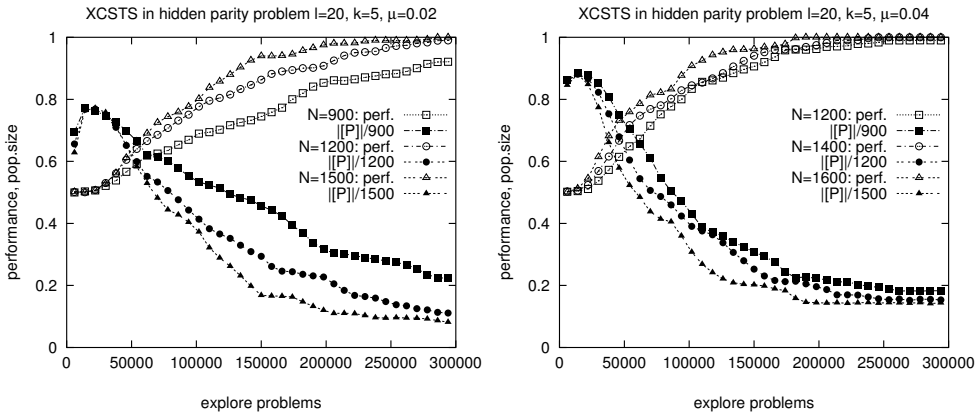

Figure 8: In the case of small mutation rates, different population sizes hardly affect performance and the *representative bound* takes over.

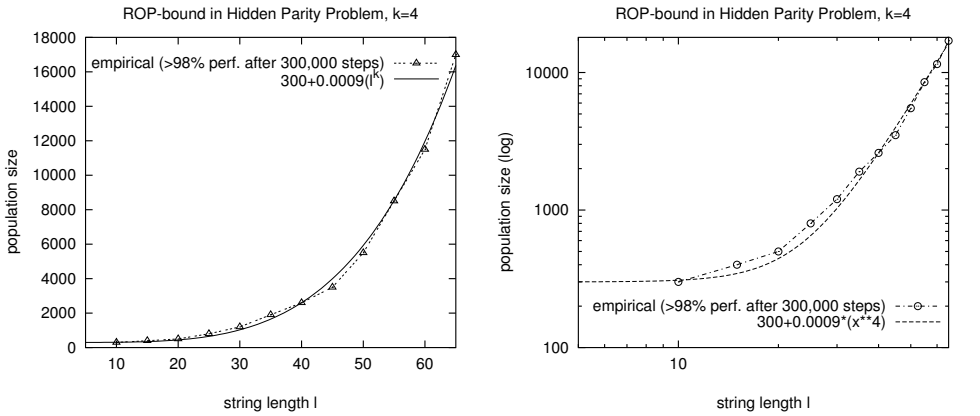

Figure 9: The *reproductive opportunity bound* can be observed altering problem length l and using optimal specificity s, given a problem of order of difficulty k_d. The experimental runs in the hidden parity problem of order $k = 4$ ($k_d = k$) confirm that the population size necessary in XCS to ensure a proper evolutionary process indeed scales up in $O(l^k)$.

6 Accuracy Guidance

While the last section determined the scale-up behavior requiring the supply of classifiers that represent order-k schemata, this section investigates in further detail the fitness guidance when order-k schemata are supplied. We will see that in many experiments $k = 1$ and fitness leads towards higher accuracy from the beginning.

6.1 Reward and Reward Error Estimation

The introduction to the parameter update mechanisms in Section 2.3 gives hints on what XCS parameters actually approximate. This section evaluates on this and shows when this approximation might thwart the evolutionary fitness pressure.

Using the Widrow-Hoff delta rule, the reward prediction of a classifier approximates the mean reward it encounters in a problem. Thus, the reward prediction can be approximated by the following estimate.

$$cl.p \approx \sum_{\{S|cl.C \text{ matches } S\}} P(S)P(cl.A|S)R(S, cl.A) \tag{28}$$

S denotes hereby a state or sensory input to XCS and $P(S)$ denotes the probability that state S is presented in the problem and $P(A|S)$ specifies the conditional probability that action (or classification) A is chosen given state S.

Given that reward prediction is well-estimated, we can derive the prediction error estimate in a similar fashion.

$$cl.\varepsilon \approx \sum_{\{S|cl.C \text{ matches } S\}} P(S)P(cl.A|S)(|cl.p - R(S, cl.A)|) \tag{29}$$

Thus, the reward prediction error estimate essentially approximates the mean reward encountered in a problem and the reward prediction error estimates the mean absolute deviation (MAD) from the reward prediction.

Given that there are only two reward levels possible (usually 0 and 1000) and denoting the probability that a classifier classifies matching input correctly by $P_c(cl)$ we can derive the estimated reward prediction simply by

$$cl.p = 1000P_c(cl) + 0(1 - P_c(cl)) = 1000P_c(cl) \tag{30}$$

Similarly, we can derive the prediction error by

$$\begin{aligned} cl.\varepsilon = P_c(cl)(1000 - 1000P_c(cl)) + (1 - P_c(cl))(1000P_c(cl)) = \\ 2000(P_c(cl) - P_c^2(cl)) \end{aligned} \tag{31}$$

This prediction to prediction error relation is depicted in Figure 10. The main idea behind this function is that given a $50/50$ probability of classifying correctly, the mean absolute error will be on its highest value. The more consistently a classifier classifies problem instances correctly/incorrectly the lower the reward prediction error. The next section shows how fitness strongly depends on this reward prediction error estimate.

6.2 Fitness Estimation

As specified in Section 2, the fitness estimate is derived from the action-set-relative accuracy of a classifier where the accuracy is derived from the scaled reward prediction error estimate. At a first glimpse, it may seem somewhat counterintuitive to use the

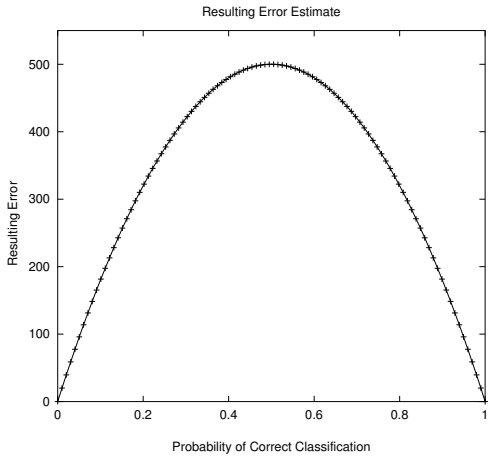

Figure 10: Assuming a $1000/0$ reward scheme, it can be seen that the prediction error estimate peaks at a probability of a correct classification of 0.5. Given that this is the probability of a completely general classifier, the more probable a classification is correct or wrong, the lower the error will be.

additional fitness measure. Using tournament selection, we could simply select the classifier with the lowest error estimate $cl.\varepsilon$ in a tournament. However, important differences between the fitness estimate and the reward prediction error estimate can be derived.

Similarly to reward prediction and reward prediction error, fitness estimates an average value of all matching cases of the classifier. In contrast to the other estimates, though, the classifier estimate is dependent on the other classifier's accuracies. Thus, the fitness measure is not an absolute value of the classifier but a measure which depends on the current state (partially indirectly) of the whole population:

$$cl.F = \sum_{\{S|cl.C \text{ matches } S\}} P(S)P(A|S)cl.\kappa'(S)$$
$$\text{where } \kappa'(S) = \frac{\kappa \cdot num}{\sum_{cl_i \text{ matches } S} cl_i.\kappa \cdot cl_i.num} \tag{32}$$

Parameter κ is determined according to Equation 3 defined in the XCS overview in Section 2.

The equation shows that fitness essentially approximates the *mean relative accuracy* of a classifier. Thus, each classifier gets a fitness value which depends not only on its own accuracy but also on the number of other classifiers in the sets the classifier is a part of and on their accuracy. If the set is overpopulated, the classifier will get a lower fitness value. How much lower it will be depends on the accuracy of the other classifiers in the set as well as on how the accuracy is scaled.

Let's assume a set with one accurate classifier cl and other inaccurate classifiers. In this case, the impact of the other classifiers will depend on how accuracy is scaled. In the scaling with the power function in XCS, the impact of inaccurate classifiers is rather low. However, when using other selection schemes like tournament selection where the scaling is not needed, the impact might be (and experimentally showed to be) stronger. Variances in the size of the sets could even push the average fitness (i.e. its average

relative accuracy with all sets it matches in) of an accurate, maximally general classifier below the fitness of some (slightly) inaccurate classifier that matches in a subset of this classifier's matching states and in other under-populated sets in which it receives a high fitness share.

On the other hand, if the accurate classifier cl shares the niche with other accurate classifiers, the competition changes in a different way. Three different types of a complete, accurate, and maximally general problem representation can be distinguished. (1) There are only non-overlapping solutions possible (this is the case in the hidden parity problem, the layered multiplexer problem, or the layered count ones problem). (2) A maximally general non-overlapping solution is possible, but others interfere (this is the case in the multiplexer problem). (3) Only a maximally general overlapping solution is possible (this is the case in the count ones problem).

In the case of a non-overlapping maximally general solution, over-specialized but accurate classifiers will get less reproductive opportunities and eventually die out (caused by the set pressure). It is still possible that, caused by noise, over-specialized classifiers might sometimes have a higher fitness (due to inaccurate classifiers that eat up fitness shares in the sub-niche of which the over-specialized classifier is not a part) but the implicit generalization pressure manifested in the set pressure takes care of that.

In the case of interference the problem becomes more delicate. Consider the 3-multiplexer problem. Here the two optimal solutions for classification one are cl_1=`1#1-1` and cl_2=`01#-1`. However, classifier cl_3=`#11-1` is as accurate and as often applicable as the other two. The problem is that the whole space for classification one can only be covered with cl_1 and cl_2 or with cl_3 and classifiers `010-1` and `101-1` insisting on a non-overlapping population. The average fitness effect takes care of this problem since cl_3 will be only present in sets where either cl_1 or cl_2 will also be present. Thus, this set is (on average) overpopulated. However, the other two niches (`010` and `101`) are less crowded so that cl_1 and cl_2 will get a higher fitness share in those niches and thus cl_1 and cl_2 will approximate a higher fitness value than cl_3. Thus, cl_3 will get less opportunities for reproduction and eventually dies out.

In the case of overlapping maximally general solutions the problem becomes most severe. Essentially, more specific classifiers that only match in a less crowded subset of a classification niche might get an on average higher fitness. This is essentially the problem in the count ones problem where $\lfloor l/2 \rfloor + 1$ ones or zeros need to be specified. The more ones in the actual relevant positions, the more a niche will be overpopulated with classifiers that specify $\lfloor l/2 \rfloor + 1$ bits in different positions. A classifier that specifies $\lfloor l/2 \rfloor + 1$ `1`s and an additional `0` assures that its sets will be less crowded and thus it will get a higher fitness on average than the maximally general classifiers. Thus, in this case the evolving population is actually not necessarily maximally general.

6.3 Fitness Guidance Exhibited

In many problems, fitness guidance can be observed by displaying the evolving specificity of classifiers. This section analyses several Boolean function problems and shows that XCSTS indeed exploits the inherent fitness guidance in those problems. Furthermore, we show that our derived knowledge allows us to solve much larger problems by choosing population size N as well as mutation μ well.

6.3.1 Count Ones Problems

In the count ones problem fitness guidance is rather straight-forward. Initially, the probability of classifying input correctly equals 0.5 given no attribute of the k relevant

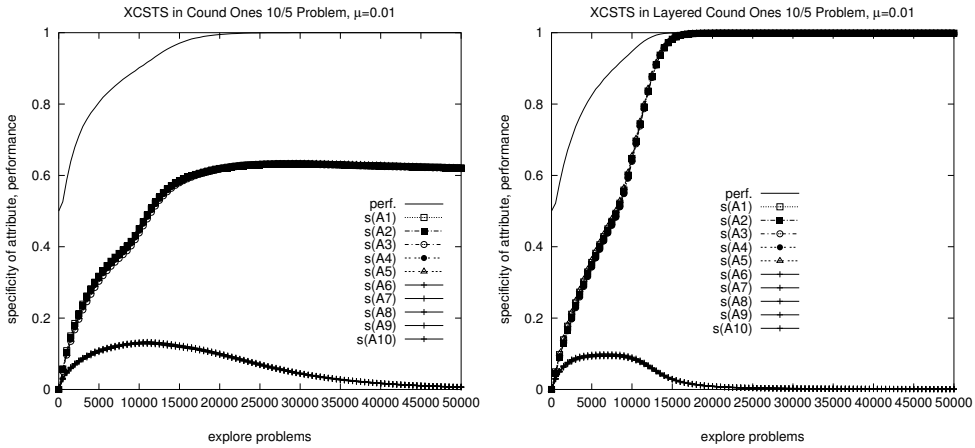

Figure 11: In both the count ones problem (left-hand side) and the layered count ones problem (right-hand side), the k relevant attributes are independent of each other so that the evolving specificity is indistinguishable in all k attributes as well as in all $l-k$ irrelevant attributes. Accuracy pressure pushes the specificity in the right direction and 100% performance is reached fast.

bits is specified. In the count ones problem with $k=5$, for example, the specification of one attribute results in a probability of $11/16$ of being correct or incorrect dependent on whether or not the classification (i.e. the action) of the classifier corresponds with the specified value, respectively. Thus, using Equation 31 the average error estimate will approximate 429.6875 and thus any classifier that has one correct position specified will have a higher accuracy on average and thus fitness than a classifier that has no position specified. Thus, specificity will increase and eventually the evolutionary process results in an accurate and maximally general representation of the whole problem. Figure 11 shows the progressive change in specificity in the count ones problem with $l=10$ and $k=5$ relevant bits setting $\mu=0.01$ to keep the influence on specificity due to mutation small. Results are averaged over 1000 runs. It can be seen that the specificity of all k relevant bits uniformly increases while the specificity of the other $l-k$ bits uniformly decreases. Since it is sufficient to specify $\lceil k/2 \rceil$ attributes of the k relevant attributes in the count ones problem, the specificity in the relevant attributes does not increase to 100%.

In the layered count ones problem all k positions need to be specified to be able to predict accurately. Thus, the average specificity in the k relevant positions reaches 100% (Figure 11, right-hand side). Again we set $\mu=0.01$ and average over 1000 independent runs. Note that again all k relevant positions are independent of each other so that there is no difference in the evolving specificities of the k positions in the layered count ones problem nor in the $l-k$ irrelevant attributes. Interestingly, the specificity of the irrelevant attributes decreases much faster than in the (not layered) count ones problem. This appears to be an additional effect of the overlapping niches in the count ones problem. XCSTS takes longer to reach a stable state since the population can switch between different complete and accurate but overlapping representations of the problem and as a result, the specificity of the $l-k$ irrelevant attributes does not decrease as fast.

The curves above show that in both count ones problems, the order of the problem

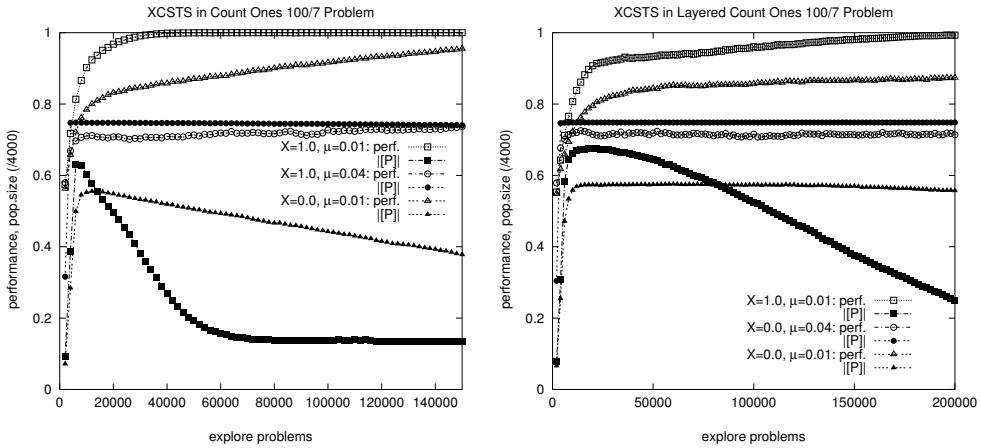

Figure 12: In both the count ones problem (left-hand side) and the layered count ones problem (right-hand side), a problem of size $l = 100$ is solvable with a maximal population size $N = 3000$. Mutation needs to be set low enough to obey the reproductive opportunity bound. Without recombination, learning is seriously impaired in both problems showing the benefit and strong importance of recombination in these problems.

equals to one. Thus, mutation can be set very low to make sure that the reproductive opportunity bound is obeyed. Figure 12 shows runs in the count ones problem (left-hand side) and the layered count ones problem for $l = 100$ and $k = 7$. Although the order of the problem is one, high mutation rates still result in the violation of the reproductive opportunity bound and learning is nearly impossible. The curves also show the importance of recombination. Since mutation needs to be small, crossover becomes much more important to ensure a proper recombination of lower order building blocks.

6.3.2 Multiplexer Problems

In the multiplexer problem without layered reward, fitness guidance is rather small, and actually first pushes in a rather unexpected direction. In fact, it is not the $k - 1$ relevant address bits that cause fitness guidance at first, but the 2^{k-1} addressed bits. That is, while in the specification of an address bit the probability of classifying correctly remains unchanged given all other bits are don't care symbols, specifying one of the 2^{k-1} remaining bits changes the probability. For example, in the 6-multiplexer a classifier with condition `1#####` is correct with a probability of 0.5 while a classifier with a condition `##1###` is correct/incorrect with a probability of 0.625 dependent on whether its action is one or zero, respectively. The specificity change can be observed in Figure 13 for the 11-multiplexer (average of 1000 experiments, $\mu = 0.01$). As the theory predicts, initially, the specificity of the eight addressed bits increases stronger than the specificity of the three address bits. Once the specificity is sufficiently high in those positions (and consequently most classifiers specify only zeros and don't care symbols or only ones and don't care symbols in those positions), the specialization of the position bits can increase the accuracy of the classifier further. In the relatively easy 11-multiplexer problem, this happens very fast. Soon, the specificity of the position bits is much higher than the specificity in the remaining bits and the problem is solved. Note that the specificity in the remaining bits actually stays larger than in the count ones

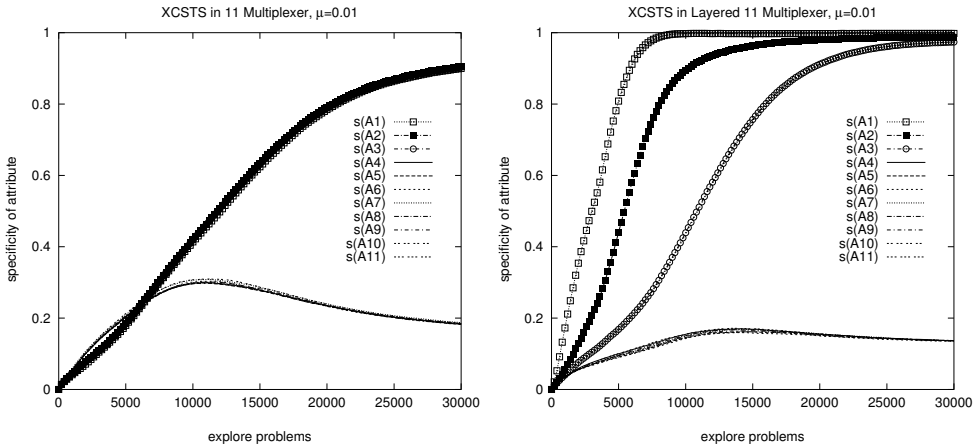

Figure 13: In the 11-multiplexer problem fitness guidance is rather small. Initially, fitness actually pushes classifiers that consistently specify ones/zeros in the eight position bits. Soon however, the specificity in the three address bits takes over. In the layered 11-multiplexer, on the other hand, much stronger fitness guidance is available due to the different payoff levels. The pressure results in a progressive increase in specificity starting with the highest position bits.

problems. This is the case since always one of the remaining bits (i.e. the addressed bit) needs to be specified as well, so that the average specificity in the remaining bits will stay above $1/8$. The analysis of the specificity evolution actually shows that XCSTS is able to overcome local optima (e.g. the specialization of all addressed bits in the multiplexer problem) and reach the global optimum confirming its robustness and dynamic capabilities.

In the layered multiplexer problem, fitness guidance is much stronger. In difference to the count ones problems where each of the k relevant bits is independent of each other, in the layered multiplexer problem the first bit cuts the range of possible outcome values in half, the next bit cuts the first half in half and so forth. Thus, fitness guidance for a specific bit is only available if all previous bits have been specified. This is observable in Figure 13 (right-hand side) where the specificity increases successively in each of the position bits.

Similar to the large count ones problems, we are now able to solve larger layered multiplexer problems with reasonable small population sizes. Figure 14 shows experiments in the 70 and 135 multiplexer problem. Again, the results are averaged over 50 experiments. Mutation is set to the low value of 0.05 to obey the reproductive opportunity bound. The layered 70-multiplexer was previously solved with XCS (Butz, Kovacs, Lanzi, & Wilson, 2001) but with a much larger population size. The layered 135-multiplexer was not solved with XCS before.

7 Fitness Dilemma and Bilateral Accuracy

The last section showed how XCSTS properly exploits accuracy guidance of a problem. We also saw that supply as well as reproductive events need to be ensured by obeying the reproductive opportunity bound as well as the existence of representative bound. This section is concerned with problems in which accuracy guidance actually leads

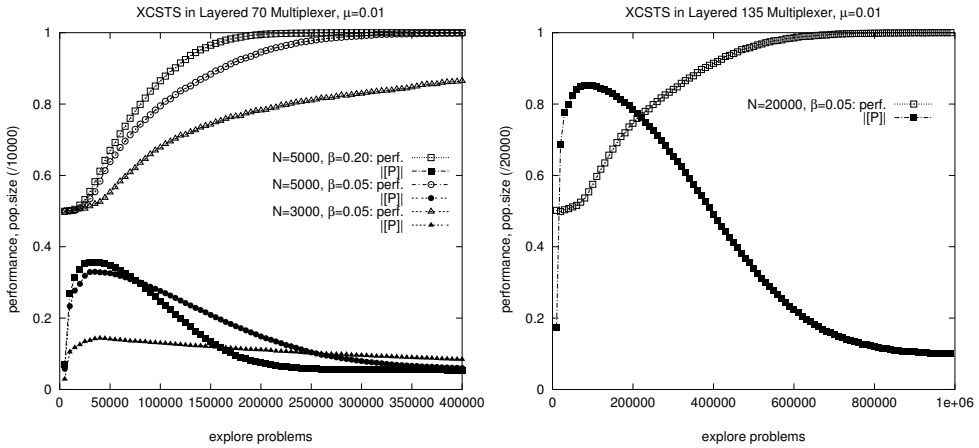

Figure 14: With our knowledge of the reproductive opportunity bound and the additional knowledge that the minimal schema that provides accuracy guidance in the layered multiplexer problem is of order one, we are able to solve large layered multiplexer problems using a low-enough mutation rate μ and a high-enough population size N.

in the wrong direction. That is, instead of pushing towards higher accuracy in the relevant positions, we show that problems exist in which accuracy guidance actually pushes towards higher generality. This *fitness dilemma* can prevent the evolution of an accurate representation of a problem. *Bilateral accuracy* is introduced as a first approach to alleviate the problem.

7.1 Misleading Accuracy: The Fitness Dilemma

In Section 6.1 we saw how the reward error estimate approximates the mean average deviation of the reward prediction. Figure 10 in Section 6.1 shows the exact relation. The subsequent section then showed how accuracy and thus fitness directly depends on the reward error estimation. The essential point is that the more consistently a classifier predicts the correct or incorrect outcome, the more accurate the classifier will be.

The question we ask now is, what if the initial probability of being correct is above/below 0.5? In this case the estimate actually gets into a dilemma. For example, given that the maximally general classifier has a probability of 0.2 of classifying input correctly, specifying parts of the relevant bits correctly might, for example, result in a probability of 0.5 of classifying input correctly. This, however has the consequence that the error is actually higher and, consequently, accuracy and fitness will be lower than the completely general classifier. Thus, instead of having a fitness pressure towards higher accuracy, the pressure goes in the opposite direction towards lower accurate classifiers. We illustrate this problem in the concatenated multiplexer problem.

A concatenated multiplexer problem is a problem in which several multiplexer problems are concatenated. The class of an input string is determined by the individual multiplexer functions. For example, Table 3 shows a sample of inputs and outputs of a concatenated multiplexer function made of three 3-multiplexer functions. We call this problem a 3×3 concatenated multiplexer problem. In general, a $x \times y$ concatenated multiplexer problem is a combination of x multiplexer problems of type y.

Table 3: A sample input/output mapping of the 3×3 concatenated multiplexer problem

Input	Output
000 000 000	000
010 100 111	101
110 101 001	010
111 110 001	100
111 111 111	111

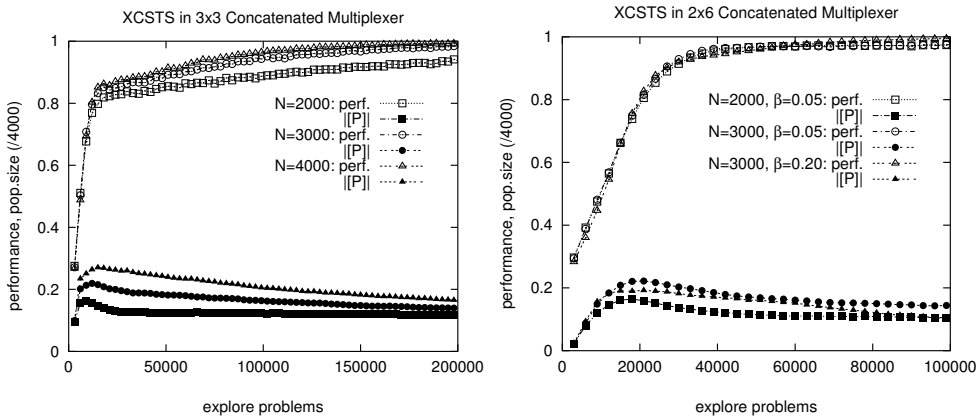

Figure 15: XCSTS performance in the 3×3 and the 2×6 concatenated multiplexer problem shows that XCSTS suffers from the identified fitness dilemma. Due to the easy identification of incorrect cases, it is still able to reach a considerable high performance.

We use the normal $1000/0$ reward scheme for the problem which means that a reward of 1000 is provided if the classification corresponds to the m multiplexer outputs. If one or more multiplexers are classified incorrectly, the output is 0.

XCSTS performance in the 3×3 as well as in the 2×6 multiplexer problem is shown in Figure 15. It can be seen that XCSTS hardly reaches 100% performance even after $200,000$ steps in the 3×3 case (XCS with proportionate selection shows similar behavior). In the 2×6 case, it is necessary to set learning rate β to 0.2 in order to reach 100% performance.

Part of the optimal classifier population for the 3×3 problem is shown in Table 4. Note that for the wrong classifiers the wrong output of one component is sufficient for the entire problem to have a wrong output. Thus, classifiers that specify a correct classification accurately, need to have a higher specificity than classifiers that accurately specify an incorrect classification.

To see why XCS is not able to perform well in this problem, let us look at the accurate and maximally general classifiers in the 3×3 case. For example, cl_1=`01# 01# 01#`→`111` is accurate and completely general (the spaces between condition part of component multiplexers are for clarity). One possible way for this classifier to evolve in the population is by the recombination of the partially accurate classifiers: cl_2=`01# 01# ###`→`111` and cl_3=`### ### 01#`→`111`. We can show that such an event, however, is extremely unlikely since the probability of being correct of classi-

Table 4: In the 3×3 multiplexer problem the necessary classifiers for an accurate and correct classification need to be much more specific than those for an accurate but incorrect classification (classifiers that always suggest a wrong classification)

Correct Classifiers		Wrong Classifiers	
Condition	Action	Condition	Action
00# 00# 00#	000	### ### 01#	000
00# 00# 1#0	000	### ### 1#1	000
00# 1#0 00#	000	### 01# ###	000
00# 1#0 1#0	000	### 1#1 ###	000
1#0 00# 00#	000	01# ### ###	000
1#0 00# 1#0	000	1#1 ### ###	000
1#0 1#0 00#	000	### ### 00#	001
1#0 1#0 1#0	000	...	...
00# 00# 01#	001	...	...
...	...	...	...

fiers cl_2 and cl_3 is actually closer to 0.5 than the one for the completely general classifier. In fact the probability for the completely general classifier of being correct equals 0.125 so that its reward prediction error $\varepsilon = 0.21875$ (see Equation 31). The probability of being correct for cl_1 equals 0.5 and for cl_2 equals 0.25. Thus, the reward prediction errors are expected to approximate $cl_1.\varepsilon = 0.5$ and $cl_2.\varepsilon = 0.375$. Both classifiers have consequently a lower accuracy than the completely general classifier so that selection and recombination is very improbable. Thus, besides the inherent generalization pressure in XCS, in this problem also fitness pushes towards generality instead of towards specificity so that it is highly unlikely that XCS will ever evolve an accurate representation of the problem.

It seems somewhat surprising that XCSTS still reaches close to 100% performance as shown in Figure 15. A more detailed analysis revealed that XCSTS learns the (easy) classifiers that specify incorrect classifications (shown on the right column of Table 4). Thus, given a current match set, XCSTS eventually 'knows' that seven of the eight possible classifications will be incorrect and thus chooses the correct classification most of the time. Due to noise and the continuous supply of parts of correct classifications from the incorrect classifiers, eventually the correct classifiers also evolve.

7.2 Bilateral Accuracy

In order to solve the fitness dilemma identified in the concatenated multiplexer problem it is necessary to reshape the error estimate shown in Figure 10 so that the most-general classifier ends up with the highest error estimate. Doing this promises gradual improvement of the accuracy and thus the necessary fitness guidance to break the otherwise exponential scale-up problem due to the reproductive opportunity bound (see Section 5.1). Moreover, we give way to successful recombination and thus enable XCSTS to solve decomposable problems.

As a first approach to solving the fitness-dilemma in XCS, we divide the error estimation into two parts and derive two classifier fitness values from the bilateral error estimates. The following section introduces the changes to XCS. Next, we provide several experimental investigations that confirm the better performance with this further

enhancement.

7.2.1 Bilateral Estimates

As the name implies, we divide the reward prediction error ε in XCS in two error estimates ε_u and ε_d that measure the positive error and the negative error respectively. That is, if the currently encountered reward is higher than the predicted reward $cl.p$ of a classifier cl, then $cl.\varepsilon_u$ is updated, otherwise ε_d is updated. Thus, while reward prediction still approximates the average payoff, the error now estimates up and down error. Due to this change, the slope for each error curve is ascertained to decrease towards higher accuracy. By estimating the resulting $cl.\varepsilon_u$ and $cl.\varepsilon_d$ in our previous scenario in which a classifier predicts the resulting reward correctly with a probability $P_c(cl)$, we show that the slope is now always decreasing for each error measure.

$$cl.\varepsilon_u = (1000 - 1000P_c(cl)) = 1000(1 - P_c(cl)) \tag{33}$$

$$cl.\varepsilon_d = 1000P_c(cl) \tag{34}$$

Note that the error measures are linearly dependent on the current probability of correctness. While this might seem like a simple relationship it must be noted that the error estimation, of course, only holds in a deterministic environment with a $1000/0$ payoff scheme. We will show, however, that the bilateral error estimate can also improve performance in more layered payoff schemes.

Preliminary experiments that only split the error and used the currently updated error measure to update the fitness have already shown promising behavior. More detailed experiments, however, show that it is more robust to also divide the fitness estimate in an up and down fitness estimate. Thus, dependent on if up or down error was updated, up fitness F_u or down fitness F_d is updated. Determination of accuracy and relative accuracy works as in the original XCS only that now either the up or down error is used. Note that if layered payoff is encountered, it is actually possible that for some classifiers the up error is used and for other classifiers the down error is used during the same fitness update iteration.

As with prediction error and fitness, the experience measure is also divided in two, accounting for the number of respective updates for the down and the up case. Thus, the moyenne adaptive modifiée technique can be used for the respective cases independently.

Finally, XCS uses the fitness estimate in GA selection that was updated in this step. The intuition behind this is that in a particular classification case, classifiers that predict the reward in this niche best should get the reproductive opportunities. This is measured by the corresponding F_u or F_d fitness measure. To keep the spirit of deletion, which is to ensure an equal distribution of classifiers over problem niches, the respective fitness and experience being considered are the ones that matches in the current input case.

7.2.2 Experimental Evaluation

Applying bilateral accuracy shows that fitness guidance can be exploited more efficiently in the concatenated multiplexer problem. Figure 16 shows that XCSTS with additional bilateral accuracy outperforms XCSTS with the normal fitness approach in the 3×3 as well as in the 2×6 multiplexer problem. It can be observed how the regained fitness pressure due to the bilateral accuracy approach increases the population size to a higher value in the beginning so that correct and accurate classifiers are evolved much faster. XCSTS with bilateral accuracy is also more independent of parameter β.

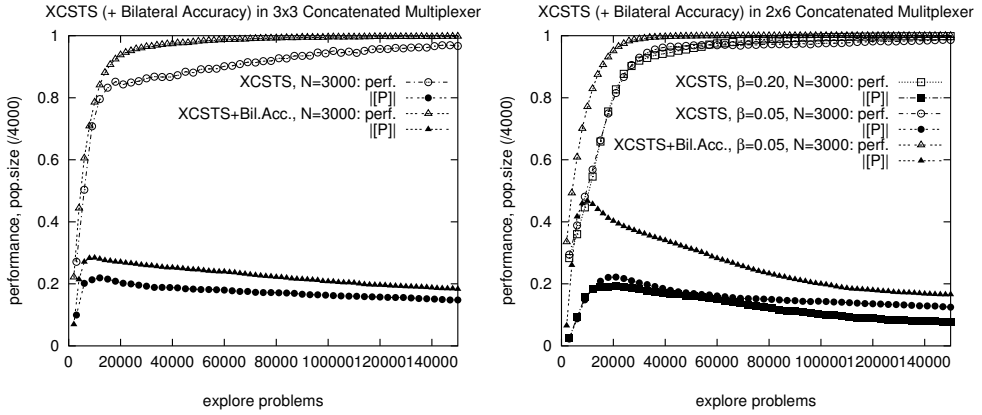

Figure 16: XCSTS with bilateral accuracy outperforms XCSTS in the 3×3 and the 2×6 concatenated multiplexer problem. Moreover, XCSTS with bilateral accuracy stays more parameter independent and solves the 2×6 multiplexer despite the small β value.

We sought additional confirmation for our approach by comparing the bilateral accuracy approach with the normal XCSTS in the count ones problems with biased input. That is, instead of providing problem instances uniformly randomly, we bias the input towards one class. In Figure 17 we show runs in the count ones problem in which class one occurs with a probability of 0.8. Due to the bias, the fitness dilemma occurs and XCSTS takes much longer to evolve an accurate problem representation. Bilateral accuracy allows XCSTS to exploit the accuracy guidance properly and consequently evolves a complete knowledge much faster.

8 Summary, Conclusions, and Future Work

This paper has extended XCS research in several ways: (1) tournament selection was introduced, (2) population size bounds were derived to enable fitness pressure, (3) the influence of fitness pressure on the evolutionary process was analyzed, (4) a fitness dilemma was identified and, (5) bilateral accuracy was introduced to solve the fitness dilemma. This section summarizes these points and provides possibilities for future work for each of them. The paper concludes with a few remarks on the general impact of this study and the applied facetwise analysis in this study.

This paper showed when and how fitness pressure applies in the XCS classifier system. By applying *tournament selection* instead of proportionate selection, we showed that fitness pressure can be made much more stable, reliable, and parameter independent. Problems with much higher noise were solvable due to tournament selection. Moreover, it was shown that tournament selection provides a basis for the effective recombination of classifiers and thus the generation of higher-order building blocks out of lower-order building blocks. While this paper only showed successful recombinatory events in problems with building blocks of schema of order one using simple uniform crossover, one future research topic is the pursuit of higher order building blocks and more competent crossover operators in XCS. Tournament selection might also be applicable for deletion, thereby promising a more stable and reliable deletion pressure.

The paper also determined population and specificity bounds that ensure the supply of classifiers necessary for a proper fitness pressure. We showed that there is a

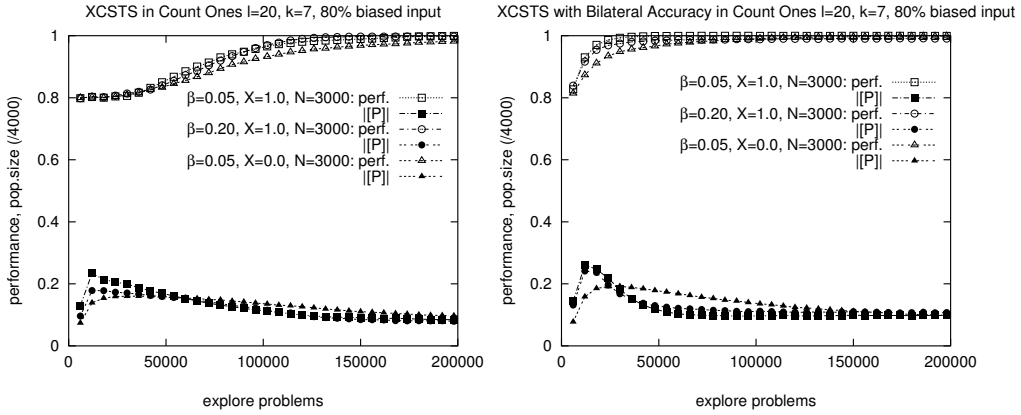

Figure 17: When biasing the problem instance distribution towards one class, XCSTS with bilateral accuracy suffers less from this bias. In the count ones problem in which 80% of the presented problem instances belong to class 1 XCSTS with bilateral accuracy evolves a complete solution faster and more reliably. Crossover improves performance in both cases indicating successful recombinatory events.

reproductive opportunity bound (ROP-bound) which needs to be obeyed in order to enable fitness pressure. Additionally, the supply of *representatives* needs to be ensured to initiate fitness pressure. The ROP-bound showed that the population size of XCS scales in slightly less than $O(l^{k_d})$ where l denotes the length of a problem instance and k_d the order of difficulty (i.e. the necessary order of the schema of minimal order that needs to be represented by a classifier). The bounds were confirmed in the hidden parity problem in which the order of the problem difficulty k_d is equal to the number of relevant bits. Several simplifying assumptions have been made to derive the ROP-bound and the specificity bounds. One of them is the assumption that all representable problem instances will actually be encountered. If the set of possible problem instances $\mathcal{S}$ is smaller than $\{0,1\}^l$ then the ROP-bound still holds as an upper bound but the problem becomes easier as $|\mathcal{S}|$ becomes smaller. Other means of stressing the reproduction of necessary schema representatives are imaginable such as biasing the selection mechanism more directly towards promising new classifiers or biasing the deletion method towards the deletion of more general classifiers when average accuracy is low. However, any such improvement can improve the ROP-bound maximally by a constant amount so that the basic population size requirement remains in $O(l^k)$.

Other bounds such as the quantification of the minimal classifier list size to ensure the reliable and stable representation of an optimal problem representation $|[O]|$ need to be investigated. Moreover, until now most derived bounds bound population size and not time until convergence. Although small population sizes were shown to delay performance instead of directly preventing learning completely (as could be expected from our derivations), future work needs to look at the time constraints in XCS in more detail as well. Also the relation to previously derived bounds such as the *schema challenge* bound (Butz, Kovacs, Lanzi, & Wilson, 2001) need to be established. In the experiments herein we set $P_\#$ to one and had mutation do all the initial specialization work. Setting $P_\#$ to the minimal value that still obeys the ROP-bound might speed up XCS's behavior significantly. However, for the purpose of this study, we chose not to experiment

with this.

We also showed how fitness pressure causes the progressive specialization of relevant attributes. The knowledge of the bounds and the dependence of fitness pressure on mutation probability (and thus specificity) and population size enabled us to solve large problem instances such as the layered 135-multiplexer and the unlayered and layered count ones problem with a problem length of $l = 100$ and seven relevant bits. A knowledge of specificity change in a problem might be exploitable for more informed or competent genetic operators in the future.

The more detailed analysis of the fitness pressure with respect to certain problem categories showed that in some problems the pressure can actually be misleading causing a *fitness dilemma*. The problem is that classifiers whose conditions are more different from desired accurate classifiers actually have a lower reward prediction error than classifiers that are more similar to accurate classifiers. Thus, fitness pressure is actually misleading and can seriously impair performance. We introduced *bilateral accuracy* as a first approach to alleviate this problem. The results indicated that bilateral accuracy conquers the fitness dilemma and ensures more reliable learning. Further analyses of the bilateral accuracy approach need to be pursued in future work. Nonetheless, the identification of the problem and the initial results confirmed that bilateral accuracy enables proper learning behavior in problems that cause the fitness dilemma.

Although this summary and future work suggestions show that there is still much more work to be done on the road to a proper understanding of XCS functioning, we have shown that XCS scales up in a manner that is machine learning competitive. By contrast to many other machine learning methods, XCS learns iteratively and thus can be applied online. That is, XCS can be used as a classifier and can be trained at the same time. Moreover, we showed that XCS is able to overcome initially misleading fitness values which makes it more dynamic and applicable to a wider range of problems. Future research needs to show to what extent these properties make XCS superior in comparison with other machine learning algorithms.

More generally we hope that these contributions to XCS theory and design lead towards XCS-based systems that can solve machine learning problems of bounded difficulty quickly, reliably, and accurately with only a modicum of human intervention. In the GA literature, GAs with this property are called *competent* (Goldberg, 2002), and we view this paper as a step towards competent genetics-based machine learning (GBML). Interestingly, facetwise design theory similar to that successfully applied in GAs over the last decade is now paying off in GBML. Thus, our parting hope is that other GBML researchers adopt this style of analysis and design to advance the state of GBML art.

Acknowledgment

We are grateful to Xavier Llora, Martin Pelikan, Kumara Sastry, Abhishek Sinha, and the whole IlliGAL lab for their help and the useful discussions.

The work was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant F49620-00-0163. Research funding for this work was also provided by a grant from the National Science Foundation under grant DMI-9908252. The US Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation thereon. Additional support from the Automated Learning Group (ALG) at the National Center for Supercomputing Applications (NCSA) is acknowledged. Further funding from the German research foundation (DFG) under grant DFG HO1301/4-3 is acknowledged.

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.

A Test Problem Definitions

A short definition of all used problems is provided with augmenting examples. We also provide a short explanation of the characteristics of each problem. All problems used are Boolean functions. Some of them are augmented with a special reward scheme.

A.1 Multiplexer

Multiplexer problems have been subject of study for a long time in learning classifier system research (De Jong & Spears, 1991) showing the superiority of those systems in comparison to other machine learning approaches such as C4.5. XCS was applied to multiplexer problems beginning with its first publication (Wilson, 1995).

The multiplexer problem is a Boolean function. The function is defined for binary strings of length $k + 2^k$. The output of the multiplexer function is determined by the bit located at the position referred to by the binary value of the k position bits. For example, in the six multiplexer $f(100010) = 1$, $f(000111) = 0$, or $f(110101) = 1$. Any multiplexer can also be written in disjunctive normal form. The disjunctive normal form of the 6-multiplexer is

$$6MP(x_1, x_2, x_3, x_4, x_5, x_6) = \neg x_1 \neg x_2 x_3 \vee \neg x_1 x_2 x_4 \vee x_1 \neg x_2 x_5 \vee x_1 x_2 x_6 \quad (35)$$

A correct classification results in a payoff of 1000 while an incorrect classification results in a payoff of 0.

In terms of fitness guidance, the multiplexer problem provides slight fitness guidance although not directly towards specifying the k position bits. Specifying any of the 2^k remaining bits gives the classifier a bias towards class zero or one. Thus, classifiers with more ones or zeros specified in the 2^k remaining bits have a lower error estimate on average. Specifying position bits (initially by chance) restricts the remaining relevant bits. These properties result in the fitness guidance in the multiplexer problem. The multiplexer problem needs $|[O]| = 2^{k+2}$ accurate, maximally general classifiers in order to represent the whole problem accurately.

A.2 Layered Multiplexer

A layered reward scheme for the multiplexer problem was introduced by Wilson (1995) intending to show that XCS is able to handle more than two reward levels (particularly relevant for multi-step problems in which reward is discounted and propagated). Later, it was shown that the layered reward scheme is actually easier for XCS since the used scheme provides stronger fitness guidance.

Reward of a specific instance-classification case is determined by

$$R(S, A) = (\text{value of } k \text{ position bits } + \text{ return value }) \cdot 100 + \text{ correctness } \cdot 300 \quad (36)$$

This function assures that the more position bits are specified, the less different the resulting reward values can be. Thus, XCS successively learns to have all position bits specified beginning with the left-most one. Similar to the 'normal' multiplexer problem above, the layered multiplexer needs $|[O]| = 2^{k+2}$ accurate, maximally general classifiers in order to represent the whole problem accurately.

A.3 Concatenated Multiplexer

In a concatenated multiplexer, x y-multiplexers are concatenated referring usually to a $x \times y$-multiplexer problem. There are 2^x classes in such a multiplexer. The resulting class is defined by the combination of the binary number that results when concatenating all x multiplexer problems together. For example, in a 2x3-multiplexer two three-multiplexers are concatenated. Thus, the string length equals six and there are four classes. An input string of `011000` belongs to class `10` as does the string `101001`. String `101010` would be class `11` and string `001110` would be class `00`.

The concatenated multiplexer problem was constructed to have different independent building blocks in a problem and work on recombining them effectively. Besides proper recombination, several problems arise such as the increased number of actions, the fitness dilemma when biasing fitness on the mean absolute deviation, and the difference in specificity between correct and wrong classifiers.

The $1000/0$ reward scheme results in a bias in the accuracy of the classifiers in that the probability of the most general classifier being correct equals to $1/2^x$. Efficient recombination in combination with the bilateral accuracy approach is very helpful in this problem since the recombination of subsolutions promises the evolution of better solutions. The number of accurate, maximally general classifiers in this problem is $|[O]| = 2^{x(k+1)} + k2^{x+k}$ (where k denotes the number of position bits in one y multiplexer) adding together the number of accurate correct classifiers and the number of accurate incorrect classifiers (those that accurately specify the wrong class).

A.4 Hidden Parity Function

Hidden parity functions were originally used by Kovacs and Kerber (2001) to show the dependence of the problem difficulty on the necessary number of accurate, maximally general classifiers $[O]$ in XCS. Hidden parity functions are the extreme case of a building block of size k. In the hidden parity function only k bits are relevant in a string of length l. The number of ones modulo two determines the class of the input string. For example, in a $k = 3$, $l = 6$ hidden parity problem (in which the first k bits are assigned to be the relevant ones) string `110000` as well as `011000` would be in class zero while string `010000` as well as string `111000` would be class one. The reward scheme is again `1000` for a correct classification and zero otherwise. Note that the hidden parity function is the extreme case for the reproductive opportunity bound introduced in Section 5.1 since any classifier that does not have all k relevant attributes specified will be over-general and consequently will not carry any useful fitness information. The number of accurate, maximally general classifiers necessary to represent the whole problem in this case is $|[O]| = 2^{k+1}$.

A.5 Count Ones

The count ones problem is similar to the hidden parity function in that only k positions in a string of length l are relevant. However, it is very much different in that the schema with minimal order that provides fitness guidance is of order one. Any specialization of only ones or only zeros in the k relevant attributes makes a classifier more accurate. The class in the count ones problem is defined by the number of ones in the k relevant bits. If this number is greater than half k, the class is one and otherwise the class is zero. We again use the $1000/0$ reward scheme in this problem. Due to the counting approach, even a specialization of only one attribute of the k relevant attributes biases the probability of correctness and consequently decreases the prediction error estimate. Thus, this problem provides a very strong fitness guidance and thus can be expected

to be rather independent of the string length. The number of accurate, maximally general classifiers $[O] = 4\binom{k}{\lceil \frac{k}{2} \rceil}$. Note that the optimal set of accurate, maximally general classifier is actually overlapping. This can cause additional difficulties.

A.6 Layered Count Ones

The layered count ones problem is identical to the count ones problem except for the reward scheme used. The received reward does now depend on the number of ones in the string. That is, a reward of $1000 \cdot \sum_{i=0}^{k} value[i]$ is provided if the classification was correct and 1000 minus that reward otherwise. This scheme causes any specialization in the k position bits to result in a lower deviation of reward outcomes and thus in a lower reward error estimate. Thus, as the count ones problem, the layered count ones problem provides a very strong fitness guidance and the evolutionary process can be expected to stay rather independent of the string length. However, note that there are more classes to distinguish than in the layered count ones problem. Since any change in one of the relevant bits changes the resulting reward, all k bits need to be specified in any classifier that specifies the input string accurately. Thus, the number of accurate, maximally general classifiers $|[O]| = 2^{k+1}$. Unlike in the count ones problem, the accurate, maximally general classifiers do not overlap.

References

Baker, J. (1985). Adaptive selection methods for genetic algorithms. In Grefenstette, J. (Ed.), *Proceedings of the International Conference on Genetic Algorithms and Their Applications* (pp. 14–21). Hillsdale, NJ: Lawrence Erlbaum Associates.

Bernadó, E., Llorà, X., & Garrell, J. M. (2002). XCS and GALE: A comparative study of two learning classifier systems and six other learning algorithms on classification tasks. In Lanzi, P. L., Stolzmann, W., & Wilson, S. W. (Eds.), *Advances in Learning Classifier Systems: 4th International Workshop, IWLCS 2001* (pp. 115–132). Berlin Heidelberg: Springer-Verlag.

Bull, L., & Hurst, J. (2002). ZCS redux. *Evolutionary Computation*, *10*(2), 185–205.

Butz, M. V., Kovacs, T., Lanzi, P. L., & Wilson, S. W. (2001). How XCS evolves accurate classifiers. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2001)*, 927–934.

Butz, M. V., Kovacs, T., Lanzi, P. L., & Wilson, S. W. (submitted, 2003). Theory of generalization and learning in XCS. *IEEE Transaction on Evolutionary Computation*. also available as IlliGAL technical report 2002011 at http://www-illigal.ge.uiuc.edu/techreps.php3.

Butz, M. V., & Pelikan, M. (2001). Analyzing the evolutionary pressures in XCS. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2001)*, 935–942.

Butz, M. V., Sastry, K., & Goldberg, D. G. (in press, 2003). Tournament selection in XCS. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2003)*.

Butz, M. V., & Wilson, S. W. (2001). An algorithmic description of XCS. In Lanzi, P. L., Stolzmann, W., & Wilson, S. W. (Eds.), *Advances in Learning Classifier Systems: Third International Workshop, IWLCS 2000* (pp. 253–272). Berlin Heidelberg: Springer-Verlag.

De Jong, K. A., & Spears, W. M. (1991). Learning concept classification rules using genetic algorithms. *IJCAI-91 Proceedings of the Twelfth International Conference on Artificial Intelligence*, 651–656.

Dixon, P. W., Corne, D. W., & Oates, M. J. (2002). A preliminary investigation of modified XCS as a generic data mining tool. In Lanzi, P. L., Stolzmann, W., & Wilson, S. W. (Eds.), *Advances in Learning Classifier Systems: 4th International Workshop, IWLCS 2001* (pp. 133–150). Berlin Heidelberg: Springer-Verlag.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization and machine learning*. Reading, MA: Addison-Wesley.

Goldberg, D. E. (2002). *The design of innovation: Lessons from and for competent genetic algorithms*. Boston, MA: Kluwer Academic Publishers.

Goldberg, D. E., & Deb, K. (1991). A comparative analysis of selection schemes used in genetic algorithms. *Foundations of Genetic Algorithms*, 69–93.

Goldberg, D. E., & Sastry, K. (2001). A practical schema theorem for genetic algorithm design and tuning. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2001)*, 328–335.

Holland, J. H. (1975). *Adaptation in natural and artificial systems*. Ann Arbor, MI: Universtiy of Michigan Press. second edition 1992.

Holland, J. H. (1976). Adaptation. In Rosen, R., & Snell, F. (Eds.), *Progress in Theoretical Biology*, Volume 4 (pp. 263–293). New York: Academic Press.

Holland, J. H., & Reitman, J. S. (1978). Cognitive systems based on adaptive algorithms. In Waterman, D. A., & Hayes-Roth, F. (Eds.), *Pattern directed inference systems* (pp. 313–329). New York: Academic Press.

Kovacs, T. (1997). XCS classifier system reliably evolves accurate, complete, and minimal representations for boolean functions. In Roy, Chawdhry, & Pant (Eds.), *Soft computing in engineering design and manufacturing* (pp. 59–68). Springer-Verlag, London.

Kovacs, T. (1999). Deletion schemes for classifier systems. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99)*, 329–336.

Kovacs, T. (2000). Strength or Accuracy? Fitness calculation in learning classifier systems. In Lanzi, P. L., Stolzmann, W., & Wilson, S. W. (Eds.), *Learning Classifier Systems: From Foundations to Applications* (pp. 143–160). Berlin Heidelberg: Springer-Verlag.

Kovacs, T. (2001). Towards a theory of strong overgeneral classifiers. *Foundations of Genetic Algorithms 6*, 165–184.

Kovacs, T. (2002). XCS's strength based twin: Part I. In *Fifth International Workshop on Learning Classifier Systems (IWLCS-2002), Workshop Working Notes* (pp. 59–79). Granada, Spain.

Kovacs, T., & Kerber, M. (2001). What makes a problem hard for XCS? In Lanzi, P. L., Stolzmann, W., & Wilson, S. W. (Eds.), *Advances in Learning Classifier Systems: Third International Workshop, IWLCS 2000* (pp. 80–99). Berlin Heidelberg: Springer-Verlag.

Lanzi, P. L. (1999). An analysis of generalization in the XCS classifier system. *Evolutionary Computation*, *7*(2), 125–149.

Lanzi, P. L. (2000). Adaptive agents with reinforcemen learning and internal memory. *From Animals to Animats 6: Proceedings of the Sixth International Conference on Simulation of Adaptive Behavior*, 333–342.

Lanzi, P. L., & Colombetti, M. (1999). An extension to the XCS classifier system for stochastic environments. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99)*, 353–360.

Pelikan, M., Goldberg, D. E., & Lobo, F. (2002). A survey of optimization by building and using probabilistic models. *Computational Optimization and Applications*, *21*(1), 5–20.

Quinlan, J. R. (1993). *C4.5: Programs for machine learning*. San Francisco, CA: Morgan Kaufmann.

Venturini, G. (1994). Adaptation in dynamic environments through a minimal probability of exploration. *From Animals to Animats 3: Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, 371–381.

Wilson, S. W. (1995). Classifier fitness based on accuracy. *Evolutionary Computation*, *3*(2), 149–175.

Wilson, S. W. (1998). Generalization in the XCS classifier system. *Genetic Programming 1998: Proceedings of the Third Annual Conference*, 665–674.

Wilson, S. W. (2000). Get real! XCS with continuous-valued inputs. In Lanzi, P. L., Stolzmann, W., & Wilson, S. W. (Eds.), *Learning Classifier Systems: From Foundations to Applications* (pp. 209–219). Berlin Heidelberg: Springer-Verlag.

Wilson, S. W. (2001). Mining oblique data with XCS. In Lanzi, P. L., Stolzmann, W., & Wilson, S. W. (Eds.), *Advances in Learning Classifier Systems: Third International Workshop, IWLCS 2000* (pp. 158–174). Berlin Heidelberg: Springer-Verlag.