Automated Global Structure Extraction For Effective Local Building Block Processing in XCS

Martin V. Butz mbutz@psychologie.uni-wuerzburg.de
Department of Cognitive Psychologie, University of Würzburg, Würzburg, 97070, Germany

Martin Pelikan pelikan@cs.umsl.edu
Department of Math and Computer Science, University of Missouri at St. Louis, St. Louis, Missouri 63043, USA

Xavier Llorà xllora@illigal.ge.uiuc.edu
Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, Urbana-Champaign, Illinois 61801, USA

David E. Goldberg deg@illigal.ge.uiuc.edu
Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, Urbana-Champaign, Illinois 61801, USA

Abstract

Learning Classifier Systems (LCSs), such as the accuracy-based XCS, evolve distributed problem solutions represented by a population of rules. During evolution, features are specialized, propagated, and recombined to provide increasingly accurate subsolutions. Recently, it was shown that, as in conventional genetic algorithms (GAs), some problems require efficient processing of subsets of features to find problem solutions efficiently. In such problems, standard variation operators of genetic and evolutionary algorithms used in LCSs suffer from potential disruption of groups of interacting features, resulting in poor performance. This paper introduces efficient crossover operators to XCS by incorporating techniques derived from competent GAs: the extended compact GA (ECGA) and the Bayesian optimization algorithm (BOA). Instead of simple crossover operators such as uniform crossover or one-point crossover, ECGA or BOA-derived mechanisms are used to build a probabilistic model of the global population and to generate offspring classifiers locally using the model. Several offspring generation variations are introduced and evaluated. The results show that it is possible to achieve performance similar to runs with an informed crossover operator that is specifically designed to yield ideal problem-dependent exploration, exploiting provided problem structure information. Thus, we create the first competent LCSs, XCS/ECGA and XCS/BOA, that detect dependency structures online and propagate corresponding lower-level dependency structures effectively without any information about these structures given in advance.

Keywords

Learning Classifier Systems, XCS, Online Learning, Building Block Processing, Decomposable Classification Problems, Estimation of Distribution Algorithms, Extended Compact GA, Bayesian Optimization Algorithm

1 Introduction

Genetic and evolutionary algorithms are stochastic optimization methods inspired by principles of Darwinian evolution. Despite many successful applications to diverse and challenging optimization problems, standard variation operators of genetic and evolutionary algorithms, such as crossover and mutation, can fail at ensuring effective exploration of the search space. Especially if the problem is *decomposable*, that is, if the problem can be decomposed into (potentially partially overlapping) subproblems of bounded order, it was shown that simple crossover tends to be disruptive while simple mutation will take a very long time to generate the optimal solution (Holland, 1975; Goldberg, 2002). A block of interacting attributes is often referred to as a *building block* (BB). Genetic search in decomposable problems improves when BBs are identified and processed effectively, that is, when the GA identifies BBs, prevents their disruption, and propagates their structure as a whole.

Since decomposable problems appear to occur highly frequently in nature as well as in engineering (Simon, 1969; Gibson, 1979; Goldberg, 2002), one of the biggest challenges in genetic and evolutionary computation is the design of techniques that adapt to the problem at hand, identify BBs, and consequently improve the effectiveness of variation operators. Estimation of distribution algorithms (EDAs) (Mühlenbein & Paas, 1996; Larrañaga & Lozano, 2002; Pelikan, Goldberg, & Lobo, 2002) address this challenge by using probabilistic variation operators based on building and sampling probabilistic models. The extended compact genetic algorithm (ECGA) (Harik, 1999) and the Bayesian optimization algorithm (BOA) (Pelikan, Goldberg, & Cantu-Paz, 1999; Pelikan, 2004) are among the most powerful EDAs. Using advanced variation operators such as those in ECGA and BOA provides scalable solutions for problems decomposable into subproblems of bounded order (Goldberg, 2002), which are intractable using standard variation operators.

Since most learning classifier systems (LCS) use variation operators of genetic and evolutionary algorithms to ensure exploration, the lessons learned in genetic and evolutionary algorithms should carry over to LCSs. Despite that, the LCS literature has somewhat ignored the importance of applying effective crossover operators and only few results exist that discuss this topic (Butz, Goldberg, & Tharakunnel, 2003). The same situation applies to mutation. Usually, a simple bit-flip mutation operator is applied that changes each attribute or action with a given probability. As a result, the performance of standard LCSs can often suffer from inefficient learning because of the ineffectiveness of commonly used variation operators of simple genetic algorithms (GAs).

Here we focus on the XCS classifier system (Wilson, 1995), which may be considered one of the most prominent and well-understood LCSs to date. In XCS, the accuracy-based fitness results in an evolutionary pressure towards higher accuracy and thus higher specificity starting from the over-general classifiers. Meanwhile, a continuous generalization pressure assures that a maximally accurate, maximally general problem solution evolves (Butz, Kovacs, Lanzi, & Wilson, 2004). It was shown that for problems where recombination of individual problem features improves accuracy, uniform crossover can ensure successful learning (Butz, Goldberg, & Tharakunnel, 2003). However, in problems where recombination of individual problem features is inefficient but recombination of blocks of features is necessary for effective exploration, XCS with standard crossover operators cannot assure an efficient problem solution (Butz,

2004a). Analogically to GAs, blocks of interacting features that should be processed together to ensure effective learning in LCSs will be referred to as building blocks (BBs).

This paper introduces a general characterization of BB-hard problems in LCSs and XCS in particular. We present several hierarchical problems that can be used to test whether an LCS is capable of efficient exploration on problems where features interact so that blocks of features must be processed efficiently. We then combine XCS with statistics-based model-building mechanisms adopted from the extended compact GA (ECGA) (Harik, 1999) and the Bayesian optimization algorithm (BOA) (Pelikan, Goldberg, & Cantu-Paz, 1999; Pelikan, 2004). A probabilistic model is first built for the *global* population of classifiers. The model is then updated to reflect the *local* probability distribution in the current action set, in which XCS reproduces classifiers. The updated model is then used to generate offspring classifiers. The experimental evaluations show that the resulting XCS search mechanism is able to automatically identify and propagate BB structures effectively, reaching performance comparable to a hand-coded BB processing mechanism, which is given information about important BBs in advance (BB-wise uniform crossover).

The remainder of this paper is structured as follows. The following section summarizes XCS and some important XCS theory. Next, Section 3 introduces several hierarchical problems, shows the poor performance of the basic XCS mechanism on these problems, and indicates the need for more competent recombination operators. Section 4 combines XCS with probabilistic variation operators adopted from ECGA and BOA, which are used to automatically identify and process important feature subsets or BBs in XCS. Section 5 presents experimental results, which confirm the validity of the approach. Section 6 concludes the paper and outlines interesting directions for future research in this area.

2 XCS in a Nutshell

Like other Michigan-style learning classifier systems (LCSs), XCS evolves a set of condition-action rules, the so-called *population of classifiers*. Classifiers evolve using a steady-state, niched GA. In contrast to the original LCS framework (Holland & Reitman, 1978; Booker, Goldberg, & Holland, 1989), the fitness in the XCS classifier system is based on the accuracy of reward predictions rather than on the reward predictions themselves. Thus, XCS is designed to evolve not only the representation of an optimal behavioral policy or classification, but rather to evolve a representation of the expected payoff in each possible situation-action combination.

This section starts with a short introduction to XCS. Next, we briefly overview XCS theory. Finally, we use exemplar artificial problems to show XCS's bias towards higher accuracy, leading to the BB-challenge investigated in the remainder of the paper. For more details on the XCS mechanism, the interested reader is referred to the algorithmic description of XCS (Butz & Wilson, 2001).

2.1 XCS Overview

XCS represents its knowledge using a fixed-length population $[P]$ of classifiers. Each classifier consists of five main components:

(1) **Condition.** The condition part C specifies the subspace of the input space in which the classifier is applicable, or *matches*. In the binary case, C is coded in the ternary alphabet $\{0, 1, \#\}^l$ given a problem with l attributes. The $\#$ symbol matches both 0 and 1.

(2) **Action.** The action part A specifies the advocated action or classification.

(3) **Payoff prediction.** The payoff prediction p estimates the average payoff encountered after executing action A in the situations in which the condition part matches.

(4) **Prediction error.** The prediction error ε estimates the average deviation, or error, of the payoff prediction.

(5) **Fitness.** The fitness F reflects the scaled average relative accuracy of the classifier with respect to other overlapping classifiers.

Learning usually starts with an empty population. Given the current input consisting of the values of all attributes, the set of all classifiers in $[P]$ whose conditions match this input is called the match set $[M]$. If an action is not represented in $[M]$, a covering mechanism is applied. Covering creates classifiers that match the current input and specify the uncovered actions. Given a match set, XCS estimates the payoff for each possible action and computes a prediction array $P(A)$. Essentially, $P(a)$ reflects the fitness-weighted average of all reward prediction estimates of the classifiers in $[M]$ that advocate classification a. The payoff predictions determine the appropriate classification. During learning, XCS chooses actions randomly. During testing, the action a_{max} with the highest value $P(a_{max})$ is chosen.

XCS iteratively updates its population of classifiers with respect to the successive problem instances. After the classification is selected using the prediction array and applied to the problem, scalar feedback is received. In a classification problem, classifier parameters are updated with respect to the immediate feedback in the current action set $[A]$, which comprises all classifiers in $[M]$ that advocate the chosen classification a. After rule evaluation and possible GA invocation, the next iteration starts.

The aforementioned covering mechanism ensures that all actions in a particular problem instance are represented by at least one classifier. In the binary case, a covering classifier condition is initialized by setting each attribute to a $\#$-symbol with a probability $P_\#$ and to the current input value otherwise. Thus, the average *specificity* of a covering classifier is $1 - P_\#$. XCS applies a GA for rule evolution. A GA is invoked if the average time since the last GA application upon the classifiers in $[A]$ exceeds a threshold θ_{ga}. The GA selects two parental classifiers using set-size relative tournament selection (Butz, Sastry, & Goldberg, 2003). Two offspring are generated by applying crossover and mutation to the two selected parents. Parents remain in the population, competing with their offspring.

In the insertion process, subsumption deletion may be applied (Wilson, 1998) to stress generalization. Because the effects of action-set subsumption may be too strong, we use GA subsumption, which searches for an accurate, more general classifier that may subsume the offspring. If such a more general classifier is found, the offspring is discarded and the numerosity of the subsumer is increased, where the numerosity denotes the number of identical (micro-) classifiers a usual "macro-classifier" represents. The population of classifiers $[P]$ is of fixed size N. Excess classifiers are deleted from

$[P]$ with probability proportional to an estimate of the size of the action sets that the classifiers occur in. If the classifier is sufficiently experienced and its fitness F is significantly lower than the average fitness of classifiers in $[P]$, its deletion probability is further increased.

XCS strives to evolve a complete, accurate, and maximally general problem solution represented by maximally accurate classifiers. Each classifier specifies a solution (that is, a class) for the problem subspace defined by its condition part. In combination, the population evolves a complete problem solution.

2.2 Underlying Theory

Two main theoretical components were developed to understand XCS functioning: (1) analysis of the evolutionary pressures, and (2) bounding models for learning complexity.

Several evolutionary pressures were identified in XCS (Butz, Kovacs, Lanzi, & Wilson, 2004) that together evolve the desired complete, accurate, and maximally general problem solution. The three main pressures are

(1) the *set pressure*, which causes a general generalization pressure,

(2) the *mutation pressure*, which causes diversification and pressure towards an equal distribution of symbols in condition parts, and

(3) the *fitness pressure*, which pushes the population towards more accurate classifiers.

To ensure that XCS successfully learns a problem solution, the fitness pressure must be strong enough to overcome potentially disruptive effects of mutation, crossover and the generalization effect of the set pressure.

Several bounding models were identified including (1) a covering bound (Butz, Kovacs, Lanzi, & Wilson, 2001), (2) a schema bound (Butz, Kovacs, Lanzi, & Wilson, 2001), (3) a reproductive opportunity bound (Butz, Goldberg, & Tharakunnel, 2003), (4) a niche support bound (Butz, Goldberg, Lanzi, & Sastry, 2004), and (5) a learning time bound (Butz, Goldberg, & Lanzi, 2004). The covering bound ensures that incoming examples are covered by the classifiers, requiring a sufficient generality in the population. The schema bound requires a sufficient specificity to ensure a sufficient initial BB supply. The reproductive opportunity bound requires a sufficiently large population size to be able to propagate better classifiers. The niche support bound requires a sufficiently large population to ensure that the final solution is not disrupted. Finally, the time bound estimates learning time assuming the domino convergence model (Thierens, Goldberg, & Pereira, 1998a), which provides an upper bound on time to convergence for GAs on decomposable problems of bounded difficulty.

2.3 Exemplar Problems

A typically easy problem for the XCS mechanism is the count ones problem (Butz, Goldberg, & Tharakunnel, 2003), in which a majority of ones (or zeros) in the relevant attributes decides the class. The accuracy structure in the count ones problem is very similar to the fitness structure of a one-max problem used often in the design and verification of GA theory. Each relevant bit increases accuracy and thus is progressively

Table 1: Expected reward prediction and reward prediction error measures for exemplar condition parts in several common Boolean function problems for classifiers with action part $A = 1$.

5-Count-Ones Problem			Hidden 4-Parity Problem			6-Multiplexer Problem		
C	p	ϵ	C	p	ϵ	C	p	ϵ
#####	500.0	500.0	#####	500.0	500.0	######	500.0	500.0
1####	687.5	429.7	1####	500.0	500.0	1#####	500.0	500.0
##1##	687.5	429.7	0####	500.0	500.0	0#####	500.0	500.0
0####	312.5	429.7	11###	500.0	500.0	##1###	625.0	468.8
####0	312.5	429.7	1##1#	500.0	500.0	##0###	375.0	468.8
11###	875.0	218.8	00###	500.0	500.0	##11##	750.0	375.0
##1#1	875.0	218.8	111##	500.0	500.0	##00##	250.0	375.0
00###	125.0	218.8	000##	500.0	500.0	0#1###	750.0	375.0
#0#0#	125.0	218.8	101##	500.0	500.0	0#0###	250.0	375.0
110##	750.0	375.0	1110#	1000.0	0.0	0#11##	1000.0	0.0
111##	1000.0	0.0	0100#	1000.0	0.0	001###	1000.0	0.0
11##1	1000.0	0.0	0000#	0.0	0.0	10##1#	1000.0	0.0
000##	0.0	0.0	1010#	0.0	0.0	000###	0.0	0.0
0##00	0.0	0.0	1111#	0.0	0.0	01#0##	0.0	0.0

more specialized in the condition parts of the classifiers in XCS. Table 1 shows some exemplar classifier condition parts and the corresponding average reward prediction and reward prediction error estimates for classifiers with action part 1 in the count ones problem. It can be seen that the specification of progressively more ones or more zeroes decreases error and consequently increases fitness. Thus, fitness progressively pushes towards the specification of more ones (zeros) in the problem. In Butz, Goldberg, and Tharakunnel (2003) it was shown that uniform crossover can assure and improve successful learning of the count ones problem with many additional irrelevant bits due to its effective mixing capabilities. This result is in agreement with the good performance of GAs with uniform crossover in the one-max problem (Harik et al., 1997; Mühlenbein & Schlierkamp-Voosen, 1993).

The hidden parity problem (Kovacs, 1999) is harder than the count ones problem because the specialization of one attribute of the relevant parity positions does not increase accuracy. Only once all relevant attributes are specialized, accuracy increases. Thus, we need to ensure a sufficient initial supply of BBs, which in this case consist of classifiers that specialize all parity bits (Butz, Goldberg, & Tharakunnel, 2003). This is similar to the needle-in-the-haystack problem studied in optimization where all solutions are equal except for the global optimum and thus there is no signal to guide an optimizer to the optimum until the optimum is found; that is why one cannot do better than to use a population that contains all possible solutions. Table 1 shows the four hidden parity problem (the fifth bit is irrelevant). The error only drops to zero once all four attributes are specified.

Another interesting problem is the widely studied multiplexer problem (De Jong & Spears, 1991; Wilson, 1995; Wilson, 1998). The multiplexer problem codes b address bits that reference one of the 2^b value bits. The referenced bit denotes the solution. Al-

though accuracy somewhat guides towards the correct specializations, initially, only the specialization of the value bits increases accuracy. Once some value bits are specialized in a condition, specialization of the address bits decreases accuracy further. Table 1 clarifies the property in the 6-multiplexer case ($b = 2$). When starting with complete generality ($P_{\#} = 1.0$), relying on mutation for the first specializations, specificity initially raises more in the value attributes of the classifiers. Only later, specificity in the address attributes takes over (Butz, Goldberg, & Tharakunnel, 2003).

Specificity is one factor that guides towards an accurate problem solution. In the simplest case, as exhibited in the count-ones problem, a step-by-step increase in specificity in the relevant attributes results in a step-by-step increase in accuracy. Given such a problem structure, a domino-convergence model (Rudnick, 1992; Thierens, Goldberg, & Pereira, 1998b) can be derived based solely on mutation. The model estimates the time until the optimal solution to the problem is found. It was shown, that this time depends linearly on the number of irrelevant bits in the problem and exponentially on the number of relevant bits in a solution (Butz, Goldberg, & Lanzi, 2004). Since solution complexity also increases exponentially with the number of relevant bits, given the domino-convergence property of the problem, the algorithm learns the problem in polynomial time with respect to problem solution complexity.

What if domino-convergence cannot be assured? As exhibited in the hidden-parity problem, sometimes it might not be sufficient to specify one attribute to gain accuracy but several attributes might need to be specified. In this case, learning time becomes polynomial in the number of irrelevant attributes where the order of the polynomial equals to the number of attributes that need to be specified at once (Butz, Goldberg, & Tharakunnel, 2003). Thus, learning may be significantly delayed. Even worse, what if several of those accuracy blocks need to be combined to reach optimal performance? In this case, many subsolutions need to be maintained in parallel. Mutation alone, however, would not be able to combine subsolutions but would need to detect the accuracy blocks rather independently for each subsolution. Thus, mutation can be expected to take a very long time until the complete solution is found. Only an effective recombination operator that combines substructures effectively can improve learning speed in such problems. This situation is in agreement with the comparison of crossover and mutation for decomposable problems of bounded order. While the number of evaluations until convergence of a GA with effective crossover grows subquadratically with problem size (Mühlenbein & Schlierkamp-Voosen, 1993; Harik et al., 1997), the complexity of a GA based only on mutation will grow proportionally to $n^k \log k$ where n is the size of the problem and k is the order of decomposition (Mühlenbein, 1992a).

The next section discusses problems where the processing of blocks of features is necessary for efficient exploration using XCS. Once XCS is able to tackle these problems, it should be capable of solving other problems where blocks of features need to be automatically identified and processed. However, using standard crossover operators, such as one-point or uniform crossover, cannot ensure effective processing of these blocks of features. That is why XCS with standard crossover operators fails to solve these problems efficiently.

3 Building Block Hard Problems

In the GA literature, BB hard problems are rather easily characterized by a composition of *trap problems*, in which the fitness gradient leads towards a local optimum away from the global optimum (Goldberg, 2002). Since fitness is defined by accuracy in XCS, a trap problem is not constructible that easily. However, we can create a needle in the haystack problem, in which all solutions are equally good or bad and only one solution is better than all the others. This problem is the hidden parity problem, discussed in the previous section. In this problem, accuracy stays low as long as not all relevant bits are specialized. Thus, the pendant to a needle in the haystack problem in GAs is the parity problem in XCS (and all other accuracy-driven LCSs).

This section investigates problems that can be used to test XCS's capability of identifying and processing BB structures effectively using parity problems as a BB. The idea is to design a general set of decomposable problems in such a way that if the proposed XCS performs well on this set of problems, it will also work well on all other decomposable problems. We design several hierarchical classification problems that demand the effective processing of lower level BB structures.

XCS is shown to be unable to solve the proposed problems due to frequent BB disruption and ineffective BB mixing with standard crossover operators. Mutation alone may solve the problem but the consequent local, syntactic-neighborhood search cannot ensure efficient exploration and may be disruptive due to its implicit specialization effect in LCSs. To solve decomposable problems efficiently and scalably, a competent crossover operator is necessary that maximizes the mixing of BBs and minimizes their disruption.

Experiments with an informed crossover operator, which is informed about the BB structure, confirm this hypothesis and show that if the crossover operator considers an appropriate feature-set decomposition, XCS can find a solution efficiently. However, these positive results are impractical because in practice the BB structure of the problem cannot be assumed to be known in advance. Section 4 then shows that statistics-based structure detection and propagation mechanisms can be incorporated into XCS to detect and propagate BB structures on the fly without any prior structural information.

The remainder of this section first derives BB-hard problems for classification. The experiments show that standard recombination operators, such as one-point, two-point and uniform crossover, do not provide scalable solutions for decomposable problems of bounded difficulty, but an informed crossover operator can solve the problems efficiently, reliably and accurately.

3.1 Building Block Hard Problems in Classification

The problems introduced above consist of BB structures that contain either only one BB, as in the hidden parity problem, or many single-attribute BBs, as in the count ones problem. The multiplexer problem is somewhat a hybrid since the initial fitness guidance leads to the less important value bits and only later, fitness guides towards the specialization of the address bits. In the count ones problem, BB processing is easy because only one specialization needs to be identified at a time, which can be accomplished by mutation. The hidden parity problem is more challenging because classifiers that specialize all relevant bits need to be available from the beginning.

However, what happens if we combine multiple hidden-parity problems? If there is a hierarchical dependency between the subproblems, first, the hidden parity blocks need to be identified, and next, they need to be recombined effectively. This section describes a hierarchical problem structure that makes a problem hard for simple crossover operators.

We construct such a problem structure using a two-level hierarchy. On the lower level, small-order parity functions are evaluated to provide the input to the higher level. The higher level is evaluated based on the results of these functions and the classification problem under consideration. Thus, the function evaluation is pursued in two stages. As an example, we can consider a parity, multiplexer combination in which the lower-level blocks are evaluated by the parity function. The results of the parity functions are then fed into the higher-level multiplexer function, which determines the overall class of the problem instance.

Note that we are not interested in creating a problem to force BB processing for its own sake. In fact, many indications in nature and engineering suggest that typical natural problems are structured in a hierarchical, decomposable structure (Simon, 1969; Gibson, 1979; Goldberg, 2002). Similarly, in the world of classification and prediction, interacting factors can be expected to determine the result where the factors should be expected to be modularized as well. Thus, we believe that the introduced class of problems is an important problem class and should be solvable by a rather general machine learning system such as the XCS system.

How can XCS solve the proposed hierarchical problem? Clearly, the lower level parity blocks need to be identified first to enable the discovery of the higher level function. Table 2 shows several examples of condition codes and their corresponding average reward predictions and prediction errors for the hierarchical 3-parity, 6-multiplexer problem. In contrast to the plain multiplexer problem or count ones problem, in these hierarchical problems, the lower-level BBs (for example, parity blocks) need to be identified and then processed effectively. The next section shows that only if the detected blocks are not disrupted, XCS is able to solve the problem. Additionally, only if the BBs are recombined effectively, XCS can solve the problem efficiently.

In the remainder of this paper we focus on XCS performance in the parity, multiplexer and parity, count-ones combination. Nonetheless, any other type of Boolean function combination in the proposed hierarchical manner is possible. Additionally, it is not necessary that all BBs on the lower level are evaluated by the same Boolean function nor do they need to be of equal length. Certainly, though, all these potential manipulations may lead to different population size and learning-time requirements as outlined in Butz (2004b).

3.2 The Need for Effective BB Processing

We tested XCS on the proposed hierarchical problem combining parity and multiplexer problem as well as parity and count ones problem. Results confirm that the parity, multiplexer combination is particularly challenging.

Table 2: Expected reward prediction and reward prediction error measures for exemplar condition parts in the hierarchical 3-parity, 6-multiplexer problem for classifiers with action part $A = 1$. For readability reasons, the lower level 3-parities are tightly coded and separated by spaces. For example, the last line encodes that if the first two parity blocks are ones (defined here as an odd number of ones), the last parity block is relevant. Since the last parity block also codes a 1, the result is 1000 and the error in this prediction goes to zero.

C						p	ε
###	###	###	###	###	###	500.0	500.0
111	###	###	###	###	###	500.0	500.0
#1#	###	#11	#1#	#11	###	500.0	500.0
###	###	111	###	###	###	625.0	468.8
###	#1#	###	100	##1	###	625.0	468.8
###	0##	###	###	000	###	375.0	468.8
###	111	###	010	###	###	750.0	375.0
##1	111	##0	100	#0#	###	750.0	375.0
101	###	111	###	###	###	750.0	375.0
###	000	###	###	000	###	250.0	375.0
101	111	###	100	###	###	1000.0	0.0
101	000	111	###	###	###	1000.0	0.0
001	000	###	###	001	###	1000.0	0.0
100	001	###	###	###	010	1000.0	0.0

3.2.1 Hierarchical three-parity, six-multiplexer problem

Performance of XCS in the hierarchical 3-parity, 6-multiplexer problem is shown in Figure 1.[1] Note that we chose quite a large population size. However, the optimal solution $[O]$ (Kovacs, 1996; Kovacs, 1997) in the problem is of size 2^{10} so that an approximately 20 times larger population size seems reasonable. It can be seen that XCS is not able to solve the problem if uniform crossover is applied. Due to the disruptive effects of uniform crossover—as already suggested in Holland's original schema theory (Holland, 1975)—XCS is not able to process the lower level BBs due to their frequent disruption.

In addition to standard crossover operators including uniform, one-point, and two-point crossover, we applied an informed crossover operator called *BB-wise uniform crossover*, to investigate the potential of more competent recombination operators. BB-wise uniform crossover is designed using the information about BBs so that it creates new classifiers by exchanging all attributes in each BB partition between the two parents with probability 50%. The BB-wise uniform crossover operator thus proceeds similarly to uniform crossover, but instead of exchanging individual features, it exchanges groups of features according to the BB decomposition of the condition. Since the BB-wise uniform crossover prevents BB disruption but maximizes BB mixing or exchange (with respect to other two-parent recombination operators), this crossover operator represents an ideal crossover operator to use for this problem.

[1] If not stated differently, all results in this chapter are averaged over ten experiments. Performance is assessed by test trials in which no learning takes place and the better classification is chosen. During learning, classifications are chosen at random. If not stated differently, parameters were set as follows: $N = 20000$, $\beta = 0.2$, $\alpha = 1$, $\varepsilon_0 = 10$, $\nu = 5$, $\theta_{GA} = 25$, $\chi = 1.0$, $\mu = 0.01$, $\gamma = 0.9$, $\theta_{del} = 20$, $\delta = 0.1$, $\theta_{sub} = 20$, and $P_{\#} = 0.6$.

Figures 1a,c show that simple uniform crossover strongly disrupts BBs and cannot process BBs effectively, resulting in its poor performance and low accuracy of the learned solution. The continuously high population size indicates that uniform crossover causes a high diversity in the population. BB-disruption prevents the growth of higher-order BBs (Figure 1b,d). Mutation alone is able to solve the problem but learning takes about three times as long as in the case of the BB-wise uniform crossover operator (Figure 1a,c). The population sizes indicate that diversity stays much lower resulting in a lower macro-population size (Figure 1a,c). If the BBs are tightly coded (as in the Example shown in Table 2), one-point and two-point crossover are able to recombine BBs effectively (Figure 1a). However, if the attributes in each BB are not located close to each other in the chosen representation for conditions but are randomly distributed over the bit string, the potential recombinatory benefit of one-point or two-point crossover is overshadowed by their disruptive effect, which delays learning and population convergence (Figure 1c,d). Thus, one-point and two-point crossover show beneficial effects in the case of tightly-coded building blocks but disruptive effects in the loosely-coded case. Since the BBs cannot be expected to be tightly coded in general, competent crossover operators are necessary.

Mutation can also be tuned to solve the hierarchical 3-parity, 6-multiplexer problem nearly as well as BB-wise uniform crossover (Figure 2b). Nonetheless, using mutation only yields unsatisfactory results in the general case. More specifically, mutation performs poorly on large problems and problems with additional irrelevant attributes where the mutation rate cannot be sufficiently high due to the reproductive opportunity bound (Butz, Goldberg, & Tharakunnel, 2003). However, a small mutation rate strongly delays learning if only mutation is applied. Figure 2c,d shows the dependence of XCS on the mutation rate in further detail.

Despite the poor performance of mutation only, BB-wise uniform crossover strongly alleviates XCS dependence on the mutation rate (Figure 2a,b), which confirms that effective crossover results in sufficiently strong exploration even without mutation. The success of BB-wise crossover relies only on the initial supply of lower-order BB structures. In our settings, each attribute in the condition part of the initially generated classifiers is specified to zero or one with a probability of $(1 - P_{\#})$. Thus, the probability that a parity block of order three is specialized is $(1 - .6)^3 = .064$, so that the initial population can be expected to have $.064N = 1280$ classifiers that supply a relevant lower-order BB structure. This initial BB supply appears to be nearly always sufficient to recombine the BBs effectively before loosing them due to genetic drift and the inherent generalization pressure in XCS (Butz, Kovacs, Lanzi, & Wilson, 2004).

The observations confirm that the success of BB-wise uniform crossover relies mostly on the initial supply of lower-order BBs; this indicates that using a competent crossover operator, which processes blocks of features efficiently, provides tractable solutions for problems intractable using standard GA variation operators. An analogical conclusion regarding the efficiency of standard and informed variation operators was reached for GAs on decomposable problems of bounded difficulty, where standard recombination and mutation operators make the search for the optimum intractable (Thierens & Goldberg, 1993; Mühlenbein, 1992b; Goldberg, 2002).

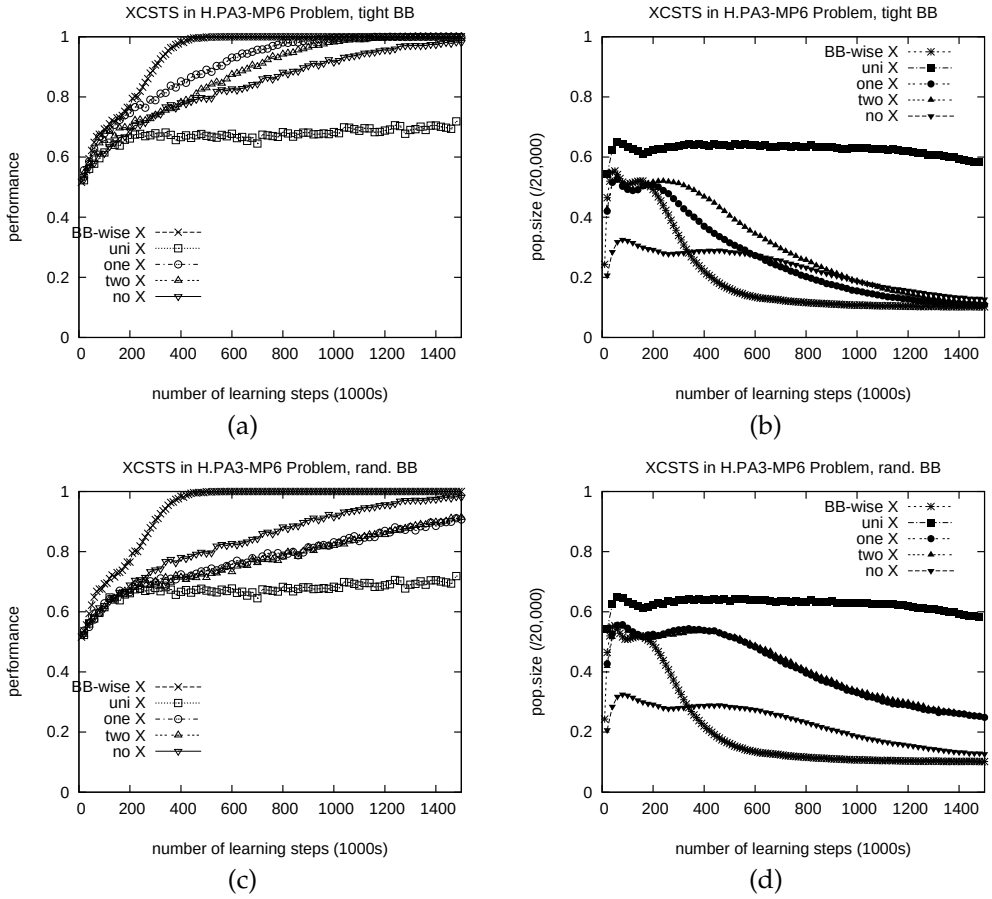


Figure 1: Performance (a,c) and population sizes (b,d) of XCS ($N = 20k$) in the hierarchical 3-parity, 6-multiplexer problem (BB-wise X = BB-wise uniform crossover, uni X = uniform crossover, one X = one-point crossover, two X = two-point crossover, no X = mutation only). Efficient BB recombination strongly improves XCS performance. One-point and two-point crossover are only beneficial if the BBs are tightly coded. Although mutation alone is able to solve the problem, the time until the solution is found is much larger.

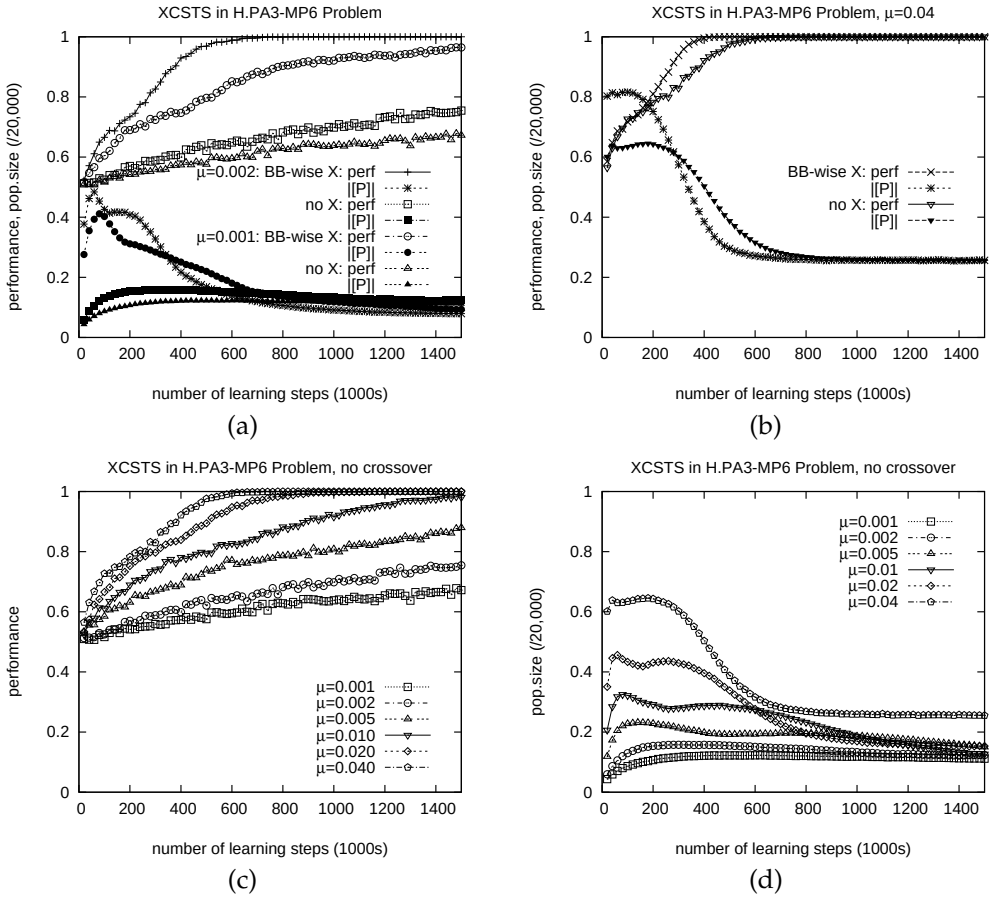


Figure 2: A low mutation rate strongly delays learning if effective recombination is not applied. With a mutation rate of $\mu = 0.001$, however, certain specialized attributes might get lost so that performance is delayed even with effective recombination (a). Higher mutation rates alleviate the problem (b) but may not be applicable in problems with more attributes. The comparison of different mutation rates exhibits XCS dependence on a sufficiently high mutation rate for successful learning (c). Lower mutation rates cause lower diversity, lower specificity, and thus smaller population sizes (d).

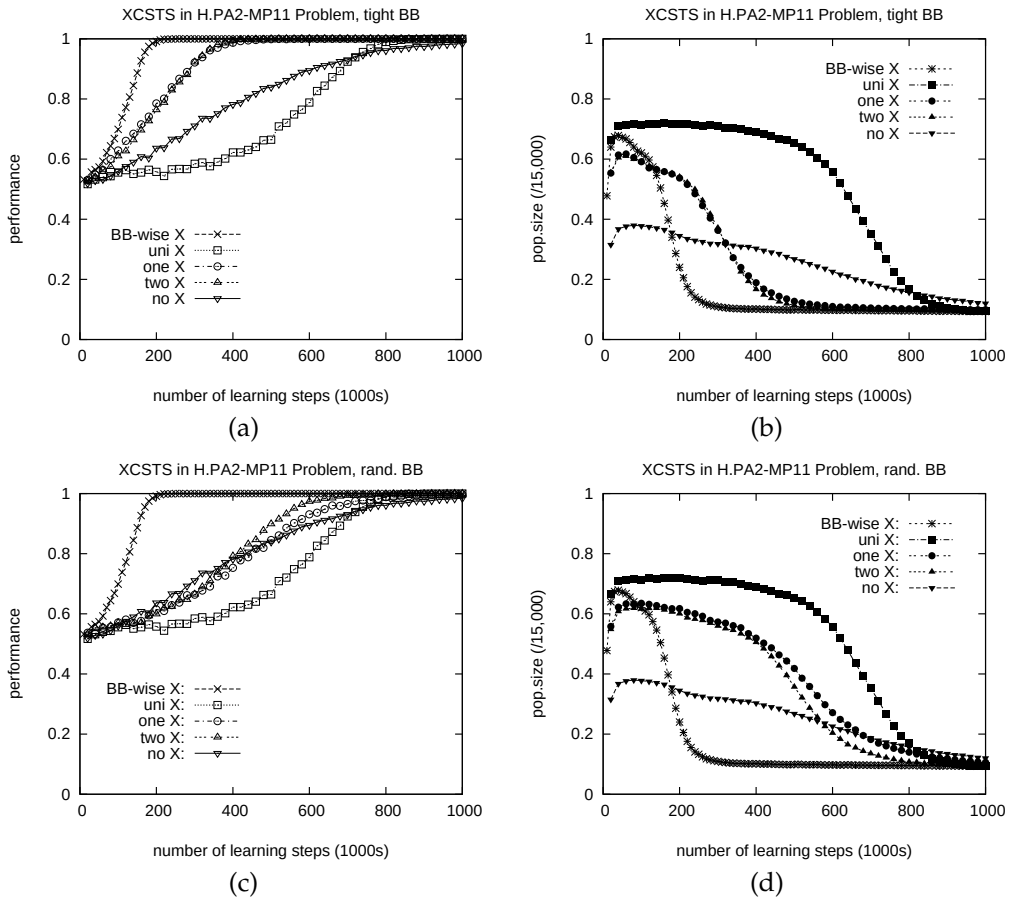


Figure 3: Performance (a,c) and population sizes (b,d) of XCS ($N = 15k$) in the hierarchical 2-parity, 11-multiplexer problem. Efficient BB recombination strongly improves XCS's performance. One-point and two-point crossover are only beneficial if the BBs are tightly coded. Although mutation alone is able to solve the problem, the time until the solution is found is much larger.

3.2.2 Hierarchical two-parity, eleven-multiplexer problem

Figure 3 confirms similar results in the hierarchical 2-parity, 11-multiplexer problem. In these experiments, population size is set to $N = 15,000$. The problem is slightly easier since the lower-level BBs are only of order two and the size of the optimal set $|[O]|$ is 2^9 instead of 2^{10}. Thus, uniform crossover is less disruptive and the problem remains tractable even with uniform crossover. One- and two-point crossover are less disruptive but still show the increased disruptive effect when the attribute locations are randomly distributed over the input string. Despite the decreased disruption, the beneficial effects of BB-wise crossover remain tremendous.

Figure 4 exhibits the strong advantage of effective BB processing further. Performance with BB-wise uniform crossover stays even more independent of mutation rate than in the case of the 3-parity, 6-multiplexer function (Figure 4a,b). This can be ex-

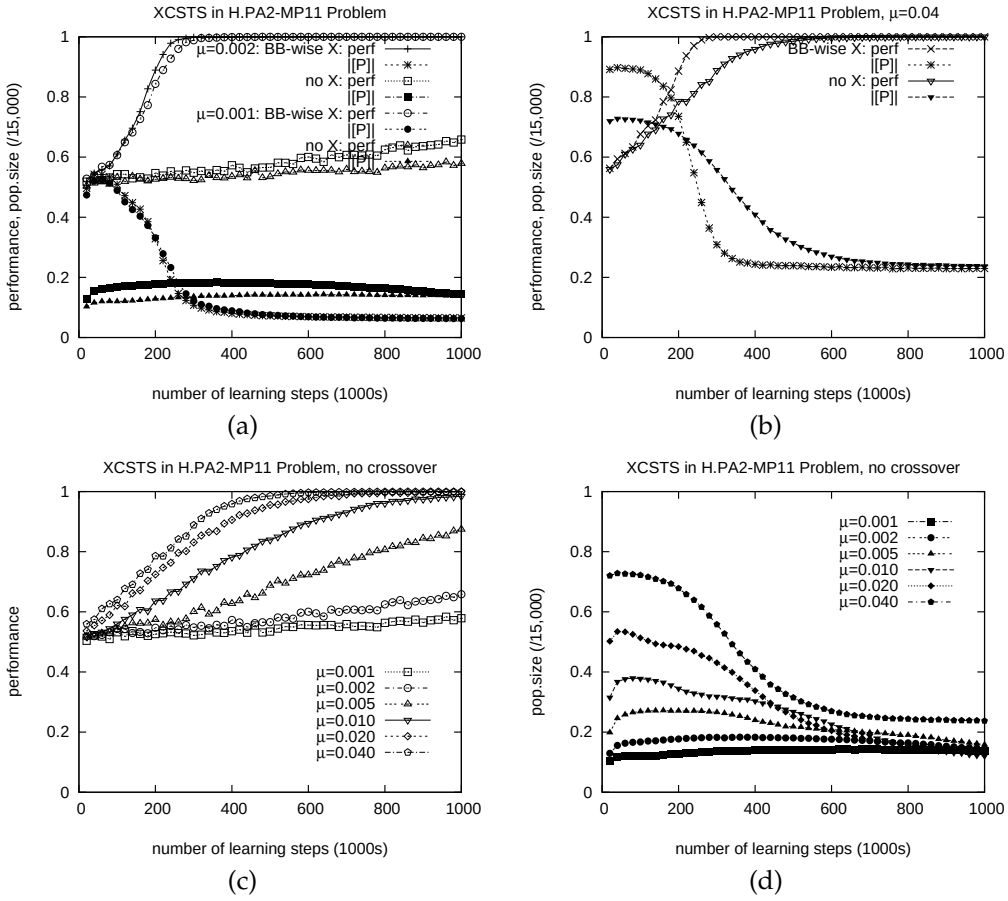


Figure 4: Similarly to the hierarchal 3-parity, 6-multiplexer problem, low mutation rates strongly delay learning in the 2-parity, 11-multiplexer problem when only mutation is applied. BB-wise uniform crossover, on the other hand, stays nearly independent of the mutation operator. Gradually increasing mutation rate show the direct influence of mutation on performance and population size if no crossover is applied (c,d).

plained by the increased number of classifiers that are expected to provide lower order BBs in the initial population, which can be determined as above by $N(1-P_{\#})^2 = 15,000 * (1-.6)^2 = 2400$ classifiers. Thus, mutation is nearly unnecessary because BB-wise uniform crossover takes care of the recombination of lower order BBs and, consequently, creates the final maximally accurate and maximally general classifiers without mutation. The much stronger initial increase in population size when BB-wise uniform crossover is applied indicates also a much stronger initial exploration of the search space. The rapid decrease in population size then indicates the identification and effective processing of relevant BB structures. (Figure 4a). As in the hierarchical 3-parity, 6-multiplexer problem, without crossover, performance strongly depends on an appropriate setting of mutation rate (Figure 4c,d).

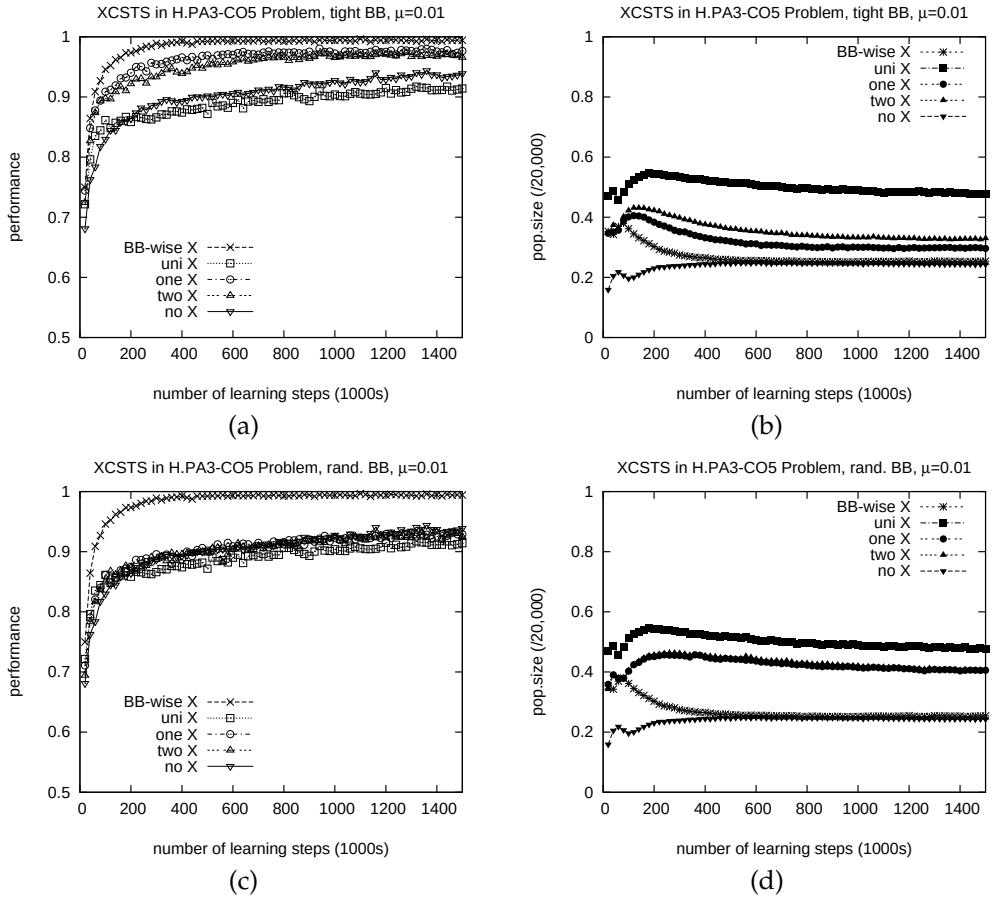


Figure 5: Performance (a,c) and population sizes (b,d) of XCS in the hierarchical 3-parity, 5-count ones problem. Again, efficient BB recombination strongly improves XCS's performance. One-point and two-point crossover are only beneficial if the BBs are tightly coded. Mutation alone gradually improves performance but is much less effective that BB-wise crossover.

3.2.3 Hierarchical parity, count-ones problem

Figure 5 confirms similar results in the hierarchical 3-parity, 5-count ones problem. In these experiments, population size is set to $N = 20,000$. Note that the problem has as many niches as the 3-parity, 6-multiplexer problem. However, the smaller population size as well as the overlapping niches in the problem make it hard for XCS to solve the problem completely optimally. Nonetheless, effective recombination significantly improves performance. As before, one-point and two-point crossover are only effective if the blocks are tightly coded; otherwise, the operators are nearly as disruptive as uniform crossover. Mutation alone slowly improves performance but takes very long time to evolve an accurate solution. The performance of BB-wise crossover is not reached by any of the other settings.

To sum up, XCS with BB-wise uniform crossover solves the created decomposable

problem efficiently and nearly independently of the mutation type used. The independence depends on the supply of lower order BBs similar to the necessary supply or lower order BBs in GAs (Goldberg, 2002). This confirms that processing subsets of attributes as opposed to individual attributes enables the solution of problems intractable with standard recombination operators.

Since many complex real-world systems can be expected to be composed of smaller sub-systems, many real-world problems can be expected to be decomposable. Decomposition exploited in XCS with standard crossover operators is severely limited and does not yield efficient solutions for problems where blocks of features must be processed, as illustrated in the investigated hierarchical problems. Thus, a mechanism in XCS is necessary that is able to identify the lower-level BB structure. Once identification is successful, effective BB processing and recombination can be applied. We introduce two alternative mechanisms that accomplish this task in the successive sections.

4 Building Block Identification and Processing

To meet the challenge of effective exploration in XCS by identifying and processing important BBs, it is necessary to develop a mechanism that can do this online. Estimation of distribution algorithms (Mühlenbein & Paas, 1996; Pelikan, Goldberg, & Lobo, 2002; Larrañaga, 2002) seem to be most appropriate for this purpose for three basic reasons: (1) EDAs enable automated identification of BBs given only a limited sample of candidate individuals, (2) EDAs enable the use of prior information to make BB identification more efficient, and (3) EDAs provide information about BBs in the form of a probabilistic model, which can be utilized and adjusted in several different ways to ensure effective exploration of the solution space.

The evolutionary component in XCS differs from the usual GA application in several ways. First of all, XCS applies recombination inside a niche defined by the current action set; since usually the niches are relatively small, learning an accurate probabilistic model from these samples will most likely fail. On the other hand, identifying BBs using the whole population results in a model that may not accurately encode dependencies in an action set. Thus, the inclusion of an EDA mechanism in XCS is not completely straight-forward.

This section integrates the BB-identification mechanism used in ECGA (Harik, 1999) to identify and process BBs. Next, the section integrates a more powerful class of models called Bayesian networks, which are used in BOA (Pelikan, Goldberg, & Cantu-Paz, 1999). We show that both mechanisms can learn *global* dependency structures and can then be adjusted to generate or optimize *local* action-set classifiers. The generation and improvement of the local offspring depends on the current action set similarly to the original XCS offspring generation mechanism.

We first give an overview of the classes of models and the learning algorithms used in ECGA and BOA to identify and encode BBs. Next, we show how these mechanisms may be integrated into XCS.

4.1 Structure Identification Mechanisms

This paper considers two structure identification mechanisms suitable for competent BB processing in XCS: (1) ECGA BB-identification mechanism, which generates *marginal product models* (Harik, 1999), and (2) BOA BB-identification mechanism, which generates *Bayesian networks with decision graphs* (Pelikan, 2001; Pelikan, 2004). The former model is easier to understand and to apply but it is limited to the identification of non-overlapping BBs. The latter is more complex but is able to model overlapping dependency structures as well.

To ensure that the learned model is of appropriate complexity and that it encodes only relevant dependencies, measures based on Occam's razor and penalized marginal likelihood are used in model identification to achieve a balance between model accuracy and model complexity. The BB identification mechanisms are explained in detail in the following sections.

4.1.1 BB-identification in the ECGA

To identify and combine BBs, ECGA uses marginal product models (MPMs), which encode non-overlapping BBs. Considering the binary-string representation of individuals, an MPM clusters all string positions into groups of positions so that dependencies within each group are much stronger than the dependencies between the groups. Table 3 shows examples of several MPMs for a bit string of length five. To learn the model, the best individuals in the current population are used where the best individuals can be selected by any standard GA selection method. The model is then learned to fit the dependencies identified in the selected set of individuals and the resulting model is used to generate offspring.

The model building in ECGA is guided by a minimum description length (MDL) metric, in which models that allow higher compression of data (including the description of the model) are favored. More specifically, ECGA uses two complexity measures: (1) model complexity MC, which favors more compact models by measuring the complexity of the model representation, and (2) the compressed population complexity CPC, which favors more accurate models by measuring the entropy of the probability distributions over the subsets of attributes defined by the model. The MDL measure is simply the sum of MC and CPC. The two complexity measures are determined by

$$MC = \log_2 N \sum_I (2^{S[I]} - 1), \tag{1}$$

$$CPC = N \sum_I -H(M_I), \tag{2}$$

where N specifies the population size, I a dependency subset, $S[I]$ the number of attributes in subset I, M_I the probability distribution of all possible values in subset I, and $H(M_I)$ the entropy of a probability distribution. The measure MC is based on the fact that $\log_2 N$ bits are necessary to describe one frequency in a set of size N and that there are $(2^{S[i]} - 1)$ independent probabilities for a set of size $S[i]$. The measure CPC then computes the complexity of coding all N individuals with respect to the subsets, which is determined by the sum of the entropies over all subsets; the smaller the overall entropy, the more accurate the model.

Table 3 shows several potential model structures and the resulting MC and CPC

Table 3: This table shows an example that illustrates how ECGA detects structural properties in a population. Problem instances used are shown on the left-hand side while the right-hand side table shows different MPMs and the MDL metric values determined with respect to the eight problem instances. While MC measures model structure complexity, CPC measures the compression, which usually improves for more complex models. Thus, MC increases with model complexity whereas CPC mostly decreases with model complexity. MDL metric is defined as the sum of MC and CPC in order to combine the pressure toward simple models with the pressure toward accurate models.

used individuals	
11000	11111
11001	11110
11000	11110
00111	00001

marginal product model	MC	CPC	$=$
[1][2][3][4][5]	15	36.98	51.98
[1 2] [3] [4] [5]	18	30.49	48.49
[1 2] [3 4] [5]	21	22.49	43.49
[1 2 3] [4 5]	30	30.49	60.49
[1 2 3 4] [5]	48	22.49	70.49

measures for an example population. The table illustrates how the MDL principle balances model complexity and population compression complexity (model accuracy). The table shows models of varying complexity and the resulting encoding gain measured by CPC. For example, when grouping the attributes in two groups (attributes 1,2,3, and attributes 4 and 5), as indicated in the second last example in Table 3, using the problem instances denoted on the left-hand side of Table 3, MC and CPC are determined by $MC = \log_2 8(7+3) = 30$ and $CPC = -8((2*3/8\log_2 3/8+2*1/8\log_2 1/8)+(4*2/8\log_2 2/8)) = 14.49+16 = 30.49$. The sum $30+30.49 = 60.49$ yields the MDL metric measure. From the table, it can be seen that the third MPM yields the lowest value of the metric and thus specifies the model that encodes the eight problem instances most effectively according to the MDL metric.

A greedy algorithm is used to build the model in ECGA. The greedy algorithm starts with each string position forming a separate group. In each iteration two groups are merged that yield highest increase in model quality measured by the sum of MC and CPC. If no two groups can be merged to improve the model quality, the learning is terminated. ECGA uses the learned model to sample new candidate solutions by sampling each group of positions independently according to the distribution of instances of this group in the selected population. The probabilities of different instances of each group are thus expected to remain the same as in the selected population. For the integration of these mechanisms into XCS, we made use of the relevant parts of the available ECGA implementation (Lobo & Harik, 1999).

ECGA was shown to be able to identify BBs and solve problems with BBs of bounded order (such as deceptive trap problems) in a scalable manner (Harik, 1999; Sastry & Goldberg, 2000). Due to its rather straight-forward approach and the various successful applications, it appears a valuable candidate for integration into XCS.

4.1.2 BB-identification in BOA

BOA uses a more powerful class of models called Bayesian networks in order to identify, encode, and recombine BBs. The overall learning mechanism in BOA is similar to

the one applied in ECGA. A Bayesian network is first built from a selected population of individuals. The built network is then sampled to generate offspring.

Bayesian networks (BNs) (Howard & Matheson, 1981; Pearl, 1988; Buntine, 1991; Mitchell, 1997) combine statistics with graph theory, providing a modular graphical model of the analyzed data. BNs can be used to estimate and sample probability distributions as well as to do inference. A Bayesian network is defined by its structure and parameters. The structure is usually encoded by a directed acyclic graph with the nodes corresponding to features and the edges corresponding to conditional dependencies. The parameters are represented by a set of conditional probability tables (CPTs) specifying a conditional probability for each variable given any instance of the variables that the variable depends on.

A BN encodes a joint probability distribution given by

$$p(x) = \prod_{i=1}^{n} p(x_i|\Pi_i), \tag{3}$$

where $x = (x_1, \ldots, x_n)$ is a vector of all the variables in the problem, Π_i is the set of parents of x_i (the set of nodes from which there exists an edge to x_i), and $p(x_i|\Pi_i)$ is the conditional probability of x_i given its parents Π_i. A CPT for x_i encodes the probability of the values of x_i given the values of its parents Π_i. A directed edge relates the variables so that in the encoded distribution, a variable corresponding to the terminal node is conditioned on the initial node. More incoming edges into a node result in a conditional probability of the variable with a condition containing all its parents.

Conditional probability tables (CPTs) store conditional probabilities $p(x_i|\Pi_i)$ for each variable x_i. The number of conditional probabilities for a variable that is conditioned on k parents grows exponentially with k. For binary variables, for instance, the number of conditional probabilities is 2^k, because there are 2^k instances of k parents and it is sufficient to store the probability of the variable being 1 for each such instance. Figure 6a shows an example set of individuals and the part (b) of the figure shows their corresponding CPT representation for $p(x_0|x_1, x_2, x_2)$.

To alleviate the exponential growth of the number of parameters with the order of dependencies, local structures can be incorporated to represent local dependencies for each variable (Chickering, Heckerman, & Meek, 1997; Friedman & Goldszmidt, 1999). Decision trees and graphs are among the most flexible local structures that can be used to represent the parameters of a BN.

In BNs with decision trees, the conditional probabilities of each variable are stored in a separate decision tree. Each internal (non-leaf) node in the decision tree for $p(x_i|\Pi_i)$ has a variable from Π_i associated with it and the edges connecting the node to its children stand for different values of the variable. For binary variables, there are two edges coming out of each internal node; one edge corresponds to 0 and the other edge corresponds to 1. For more than two values, either one edge can be used for each value, or the values may be classified into several categories and each category creates an edge. Figure 6c shows a decision tree for variable x_0 that encodes the probability distribution of the individuals shown in Figure 6a.

Each path in the decision tree for $p(x_i|\Pi_i)$ that starts in the root of the tree and ends in a leaf encodes a set of constraints on the values of variables in Π_i. Each leaf stores the value of a conditional probability of $x_i = 1$ given the condition specified

by the path from the root of the tree to the leaf. A decision tree can encode the full conditional probability table for a variable with k parents if it splits into 2^k leaves, each corresponding to a unique condition. However, a decision tree enables a more efficient and flexible representation of local conditional distributions as illustrated in Figure 6c.

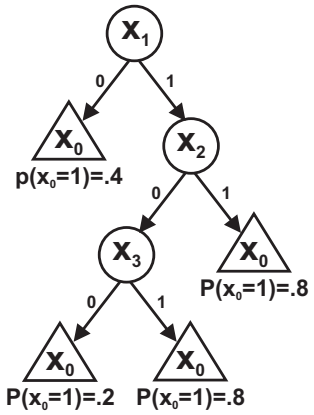
Pelikan (2001) proposed also BOA with (acyclic) decision graphs (Chickering et al., 1997; Friedman & Goldszmidt, 1999), which allow more edges to terminate in a single node, enabling multiple parents per node. This makes the representation even more flexible and allows even more compact representation of BN parameters. Figure 6d shows an exemplar decision graph encoding the same constrains as the CPT and the decision tree in figures 6b,c.

used individuals									
1000	1000	0000	0000	0000	1001	1001	0001	0001	0001
1010	1010	0010	0010	0010	1011	1011	0011	0011	0011
1100	0100	0100	0100	0100	1101	1101	1101	1101	0101
1110	1110	1110	1110	0110	1111	1111	1111	1111	0111

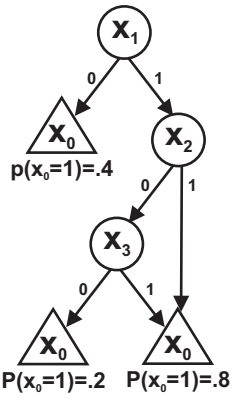
(a)

Π			$p(x_0=1\|\Pi)$
x_1	x_2	x_3	
0	0	0	.40
0	0	1	.40
0	1	0	.40
0	1	1	.40
1	0	0	.20
1	0	1	.80
1	1	0	.80
1	1	1	.80

(b)



(c)



(d)

Figure 6: The individuals shown in (a) are used to generate the full-blown conditional probability table for $p(x_0|x_1, x_2, x_3)$ shown in (b) as well as to generate the corresponding decision tree (c) and decision graph (d).

To estimate model quality, a Bayesian metric called the Bayesian Dirichlet metric with likelihood equivalence (BDe) (Cooper & Herskovits, 1992; Heckerman, Geiger, & Chickering, 1994; Chickering, Heckerman, & Meek, 1997) is used. To provide additional pressure toward simple models, the prior probability of each network decreases exponentially with the description length of its parameters.

To learn Bayesian networks with decision trees, a decision tree for each variable x_i is initialized to an empty tree with a univariate probability of $x_i = 1$. Each iteration starts by comparing the quality of the resulting models after applying splits on all variables in all leaves of all decision trees, where only splits that do not create a cycle in the Bayesian network are considered. Then, the split that yields the network of highest

quality is applied. Learning is terminated when no potential split is able to improve the current network. For decision graphs, a merge operation is introduced to allow merging two leaves in any (single) decision graph. The Bayesian decision tree mechanism applied to XCS is taken from Pelikan's BOA implementation available on the net (Pelikan, 2001).

The sampling of a Bayesian network can be done using probabilistic logic sampling (PLS) (Henrion, 1988). In PLS the variables are first ordered topologically so that every variable is preceded by its parents. The variables are then generated according to the topological ordering. As a result, once the value of a variable x_i is to be generated, its parents Π_i are assured to have been generated already. Thus, the probabilities of different values of x_i can be directly extracted from the CPT for x_i using the known values of Π_i.

As the ECGA structure assumes the independence of blocks of attributes, also a Bayesian network encodes a set of implicit independence assumptions. Each variable in a BN is independent of all its non-descendants given the values of this variable's parents (Mitchell, 1997). To encode an MPM using Bayesian networks, a network can be used that contains an edge between every pair of nodes in one cluster of variables and that does not contain an edge between any pair of nodes from two different clusters of variables. In general, every MPM can be written as a BN but many BNs cannot be expressed as an MPM without modifying the model.

4.2 Learning Dependency Structures in XCS

With two BB-identification mechanisms in hand, we now proceed to integrate these mechanisms in the XCS learning mechanisms by replacing the simple crossover operator with the model-based offspring generation mechanisms derived from ECGA and BOA. To accomplish this, it needs to clarified (1) which classifiers will be used to build a probabilistic model and (2) how the model is used to generate or recombine offspring classifiers. These two topics are discussed in this section and the next one.

The first important design task is to find a set of classifiers to build the probabilistic model. There are two sources of classifiers available at each iteration: the action set and the global population. Classifiers could be selected from the current action sets and a model could be built from those classifiers. However, there are three arguments against such an approach. First, the action sets are usually rather small; consequently, the built model is likely to yield highly noisy dependency structures. Second, model building is computationally expensive. Since XCS applies a steady-state niched GA, it seems inappropriate to build a new model each time a reproductive event of two classifiers occurs in XCS. Finally, an action set represents XCS's knowledge restricted to the current problem instance. However, it can be expected that BB structure does not differ significantly throughout many problem instances (that is, BB-specific subsets of problem instances); as a result, relevant BB information is more likely to be found in the whole classifier population instead of action sets. Thus, we build models by extracting classifiers from the overall classifier population.

When selecting from the whole classifier population, it is still necessary to decide on the selection criterion. In ECGA and BOA, fitness is the direct criterion to select the individuals used for model building. Although also each classifier in XCS has a fitness value, fitness alone may not be the appropriate measure for selection since different

problem niches may be in different learning stages. Consequently, the fitness may be misleading, potentially favoring already converged subspaces.

Thus, we decided to require a certain classifier confidence for selection, similar to the thresholded application of subsumption. We use a filtering mechanism that extracts the most accurate classifiers out of the current population. The mechanism extracts those classifiers that have a minimum experience θ_{be} and a minimum error $\theta_{b\varepsilon}$. The parameters were set to $\theta_{be} = 20$ and $\theta_{b\varepsilon} = 400$ throughout the subsequent experiments to filter out young and high error classifiers. Certainly, other constraints could be used, including constraints based on classifier parameters (such as fitness); nonetheless, experiments with other constraints showed similar learning behavior as the experiments shown herein. However, further investigations on the optimal constraints for a particular problem may be worthwhile. Since predictions below the average reward of 500 can be considered as predictions of the opposite class with higher reward, we switch the class of those classifiers that predict a reward of less than 500. Note that this method can only be applied in classification problem in which only two types of reward (e.g. $1000/0$) are possible. This method gives an additional boost to the information available for the model building (effectively doubling the available information) but is not necessary for the success of the algorithm.

In order to use the available code for learning and sampling probabilistic models, classifiers need to be transformed into a binary string representation of fixed length. Since specificity is of high importance in XCS, an explicit distinction between a specific (zero or one) and a general (don't care) attribute was expected to provide relevant structural information. Thus, we decided to code each conditional attribute by two bits: The first bit determines whether the attribute is general (that is, don't care) or specific. The second bit encodes the value of the attribute. If the attribute is a don't care symbol, we choose zero or one uniformly randomly for the second bit (assuring that there is no bias present in the second bit). If a classifier has a numerosity above one (representing several identical classifiers), several (as many as the numerosity specifies) bit codes are created where the second bit is chosen independently uniformly randomly in each second bit that codes a don't care symbol. The classification part may yet play a special role and future work may build models for each classification separately. For now, we simply code the classification part as another bit. Table 4 shows a set of classifiers and the corresponding encoding that is used to learn marginal product models of ECGA or Bayesian networks with decision graphs of BOA.

With a binary coded set of individuals at hand, we are able to identify building blocks by applying the learning algorithms used in ECGA or BOA to the filtered population of binary-string classifiers. Since the changes in the population of classifiers are small (only two classifiers are changed) in each iteration, it is more efficient to re-build the network after a fixed number of time steps θ_{bs}. To set θ_{bs}, it seems reasonable to ensure that a new model is built after N classifiers have been changed, yielding $\theta_{bs} \approx N/2$. In our experiments, θ_{bs} is usually set to $10,000$. In general, the lower the threshold, the more often the model is rebuilt, potentially adjusting the model to newly detected dependencies faster but also potentially wasting computational resources for rebuilding the same dependency structure.

Assuming BBs of order k are detected by the model building procedure, we know that model building is accomplished in time $O(kl^2N)$ (Pelikan, 2002). On the other hand, matching takes time $O(lN)$ and is executed each time step. Since model building

Table 4: Sample classifiers (from the multiplexer problem) and their corresponding binary encoding for the structure learning mechanism. Spaces are added for clarity. If an attribute is a don't care symbol, the second bit in the corresponding binary code is chosen randomly. The class bit is flipped if the reward prediction is below 500.

C	A	p	ε	binary encoding
##11##	1	750.0	375.0	10 11 01 01 11 10 1
##00##	1	250.0	375.0	11 11 00 00 10 11 0
0#1###	0	250.0	375.0	00 11 01 11 11 10 1
0#0###	0	750.0	375.0	00 10 00 11 10 10 0
0#11##	1	1000.0	0.0	00 11 01 01 11 10 1
0#11##	0	1000.0	0.0	00 11 01 01 11 11 0
001###	1	1000.0	0.0	00 00 01 10 10 10 1
10##0#	0	1000.0	0.0	01 00 11 11 00 10 0
000###	1	0.0	0.0	00 00 00 11 10 10 0
01#1##	0	0.0	0.0	00 01 11 01 11 11 1

should be executed approximately every $N/2$ iterations, each interaction requires an additional computational effort that grows in $O(kl^2)$. Considering that $k << l$ and N needs to grow in l^k (Butz, Goldberg, & Tharakunnel, 2003), to ensure supply and growth of BBs, the additional computational effort is rather small.

4.3 Sampling from the Learned Dependency Structures

Once the model is built, it needs to be sampled to generate new offspring. However, XCS generates offspring from parental classifiers selected from the current action set; that means that XCS reproduces classifiers that encode solutions with respect to the current problem instance. Consequently, the probabilistic model built from the global population should be adjusted to the current action set to reflect the local probability distribution. Otherwise, newly generated classifiers cannot be expected to reflect the structure of the current niche. While the BB structure can be expected to be similar in different niches, each niche specifies the parameters of the network encoding these structures in a different way.

We investigate three approaches to sampling offspring classifiers from the learned probabilistic model: (1) Sampling classifiers using probabilistic logic sampling (PLS) of the probabilistic model in which the probabilities (but not the structure) of the model are set to the local probability distribution, (2) probabilistically improving the selected parent classifier using the model with global probabilities, and (3) probabilistically improving the selected parent classifier using the model with local probabilities. The fourth option would be to sample classifiers using the global model directly. However, as discussed above, this would produce classifiers whose condition structure would reflect the global condition structure distribution and not the required local distribution with respect to the current problem instance. Experiments confirmed that this offspring generation method is inappropriate (not shown). Similarly, if local offspring is optimized using the model with global probabilities (case two above), the more the classifier is optimized, the less it will reflect the local attribute distribution. Thus, over-optimization is expected when improving offspring classifiers using the model with global probabilities. Figure 7 shows the different potential methods for offspring gen-

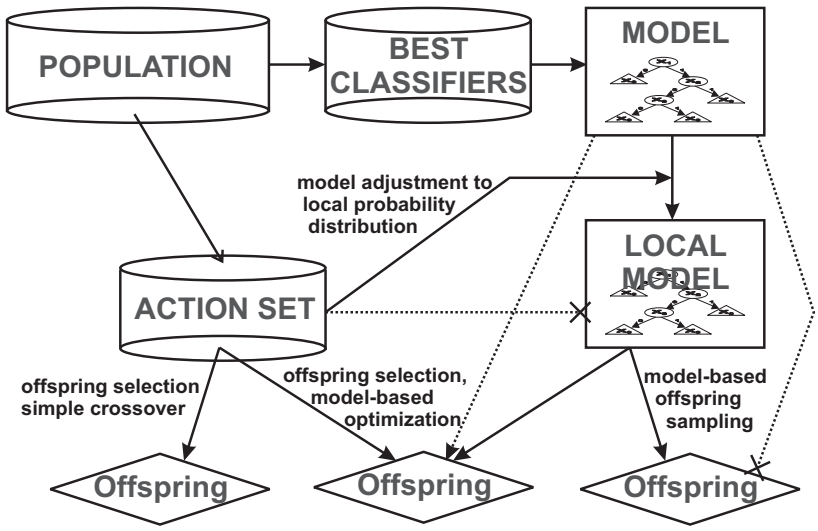


Figure 7: A probabilistic model of the problem structure can be built and used for offspring generation in many ways. Since action sets are usually too small to gather reliable statistics, a selected classifier subset from the global population should reflect problem structure more effectively. Once the model is formed, offspring may either be generated optimizing a parental classifier or sampling directly from the model. Sampling from the global model is inappropriate since the global distribution does not reflect the local solution structure. Setting the model distribution probabilities to the local probability distribution results in a model that encodes global dependency structures with respect to the local probability distribution. Thus, optimizing offspring by the means of the local model can be expected to be most cautious and most robust.

eration using a probabilistic model.

The offspring generation methods are introduced in turn. The methods with probabilistic improvement are used only for XCS/BOA but could also be implemented for XCS/ECGA.

4.3.1 Sampling using local probabilities

Considering the effect on the specificity of reproducing classifiers in an action set shows that offspring classifiers are expected to exhibit specificity equal to the average specificity of the action set. Fitness may increase this average specificity due its pressure towards higher accuracy, which often leads to an implicit specialization pressure (Butz & Pelikan, 2001).

Thus, to sample offspring from the probabilistic model, the probabilities in the model need to reflect the local specificity distribution. To achieve this, we learn the *structure* of the probabilistic model from the global population but update the *probabilities* of the model with respect to the best classifiers in the current action set. First, we select a subset of classifiers from the action set using the usual tournament selection mechanism. The selected subset (which may contain identical classifiers several times) is used to update model parameters (probabilities). The time complexity of the proba-

bility update step is $O(klM)$ (Pelikan, 2002) where k is the maximum size of detected BBs and M is the number of selected classifiers. Since $M << N$, this additional overhead is neglectable compared to the matching process, the time complexity of which is $O(lN) = O(l^{k+1})$, since N needs to grow in $O(l^k)$ (Butz, Goldberg, & Tharakunnel, 2003). After updating the probabilities, the model reflects the detected global dependencies but at the same time it mimics the local probability distribution. The globally detected dependencies are combined with the local probabilities resulting in an offspring sampling mechanism that combines global and local problem knowledge.

The resulting model (with updated parameters) is then sampled using the same sampling algorithms as in ECGA or BOA. Effectively, we use the globally detected dependency structures to sample local offspring. While the globally extracted structure biases recombination, the current local probability distribution is used to sample local offspring from the global structure. As a result, the sampling ensures effective recombination of important BBs related to the current problem instance as long as the global dependency structure applies in the local problem niche.

4.3.2 Structure optimization

In the former case, the parameters of the model built for the global population were updated according to the local niche. In the case of structure optimization we can use the global model structure with its parameters as is to sample new classifiers in the current niche. Hereby we have to be cautious to not over-commit to the model of the global population and to ensure that we bias the sampling to the local niche sufficiently. We applied structure optimization only for Bayesian networks with decision graphs, but an analogical procedure can be designed for marginal product models.

To use the global model with a bias toward the local niche, we use Markov Chain Monte Carlo (MCMC) (Neal, 1993) to sample new classifiers. MCMC proceeds by applying small changes to a particular classifier, accepting the changes with a probability equal to the ratio of the likelihood of the classifier after the modification and the sum of its likelihood before and after the modification. To determine a likelihood of a particular instance, we use the joint probability distribution as defined in equation 3, which computes a product of probabilities obtained by traversing all decision graphs in the network.

More specifically, XCS chooses an offspring via tournament selection in the current action set. Instead of crossover, XCS then applies the MCMC mechanism to probabilistically optimize the classifier structure. Bits of the binary code of a classifier are chosen at random and the likelihood of the classifier before and after flipping the chosen bit is computed. The ratio of the likelihood of the classifier after the flip and the sum of the likelihoods before and after the flip determines the probability of accepting the change. If the likelihood increases by flipping the particular bit, the probability of accepting the change is higher than the probability of accepting a flip that decreases the likelihood. To avoid zero likelihoods, all conditional probabilities are linearly normalized to values ranging from 0.05 to 0.95.

In effect, the MCMC pushes the selected local classifier towards the global probability distribution. The aim is to combine local *and* global structural information in the offspring classifier. Too many update iterations can be expected to be ineffective because the resulting classifier will reflect the global model structure. On the other

hand, too few updates will result in no exploration at all. The subsequent experimental evaluations confirm these expectations.

To avoid using the global probability distribution, it is also possible to combine the two mechanisms above; specifically, one can first adjust the dependency structure to reflect the local probability distribution and then use the resulting model to probabilistically optimize a selected local offspring classifier using MCMC. This has the advantage of avoiding the problem of over-optimization towards the global structure. Additionally, the actual parent classifier biases the sampling, which proceeds for a limited number of steps. In effect, the combination might be the most cautious but also the most robust one with respect to model inaccuracy.

5 Experiments

We evaluate and compare XCS performance on the proposed hierarchical problems, considering both offspring generation methods in several different settings. XCS with Bayesian networks with decision graphs of BOA will be denoted by XCS/BOA, whereas XCS with marginal product models of ECGA will be denoted by XCS/ECGA. XCS is set to build a new model structure every 10,000 learning steps ($\theta_{bs} = 10,000$). The model is built from the filtered population as explained above. If the filtered population is empty, no model is learned. As long as no model is learned, XCS applies uniform crossover instead of the model-based crossover. Mutation is applied to the offspring classifiers generated by the model. The results are averaged over 10 experiments. Other parameters are set as in the experiments presented earlier except for the population size, which is set to $N = 20,000$ in all runs. Mutation is set to $\mu = .001$ in the runs with the ECGA or BOA mechanism, unless indicated otherwise, and to $\mu = 0.01$ in the runs with informed or simple crossover operators, because of the stronger dependence on mutation of the latter.

5.1 Hierarchical parity, multiplexer problems

In Section 3 we saw that the evolution of a successful solution in the 3-parity 6-multiplexer problem strongly depends on the choice of mutation rate and crossover type. If a small mutation rate is used, effective BB recombination is necessary, which was achieved by BB-wise uniform crossover. If the mutation rate is large, XCS with mutation could solve the problem but it took longer than with BB-wise uniform crossover. However, we know that a large mutation rate is not an option in larger problems in which a smaller mutation rate is necessary to satisfy the covering bound and the reproductive opportunity bound (Butz, Goldberg, & Tharakunnel, 2003). Thus, to solve the addressed problem, the mutation rate needs to be small and crossover needs to be effective.

Figure 8 shows XCS performance in the hierarchical 3-parity, 6-multiplexer problem. The different settings refer to the number of selected classifiers used to adjust the model to the local probability distribution and to the number of MCMC steps where applicable. In the ECGA comparison (Figure 8), we see that although BB-wise uniform crossover performs better than XCS/ECGA, XCS/ECGA performance remains competitive. This result is not surprising because XCS/ECGA must identify building blocks by itself whereas XCS with BB-wise uniform crossover is given information about BBs in advance. Nonetheless, both variants of XCS perform similarly. Higher mutation rates

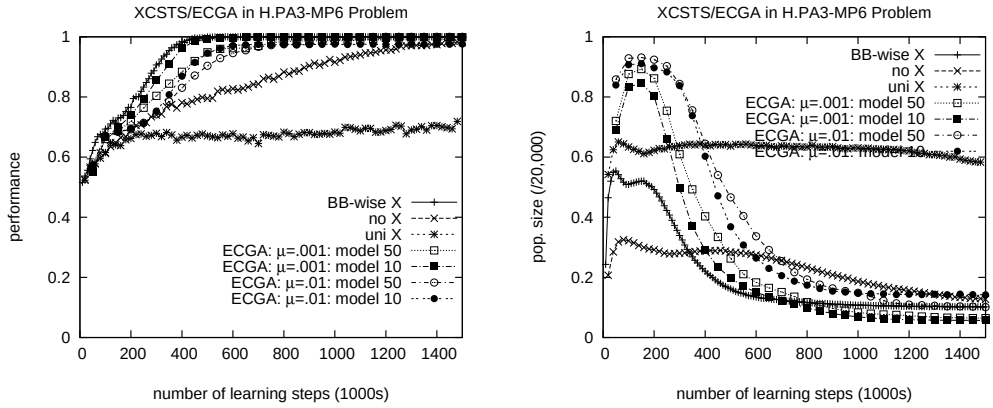
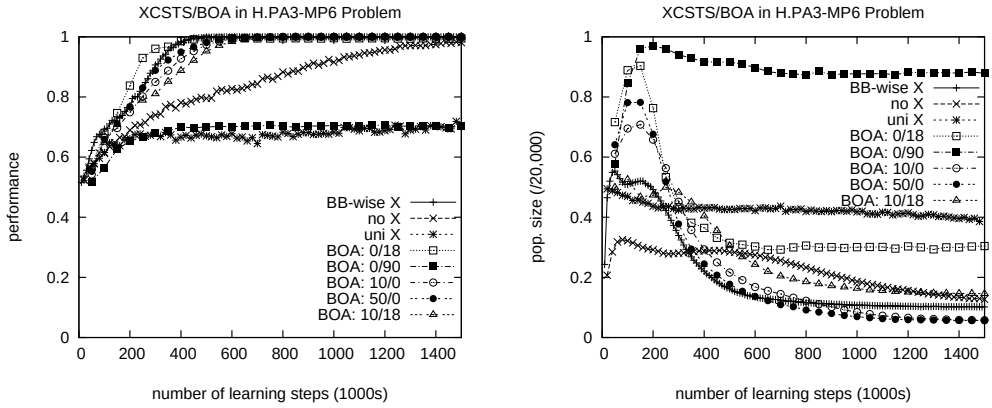


Figure 8: When applying XCS/ECGA to the hierarchical 3-parity, 6-multiplexer problem, recombination becomes effective and XCS is able to effectively learn a complete solution comparably fast to the BB-wise crossover. The 50 or 10 variation refers to the number of selected classifiers used to set the probabilities to the local probability distribution.

are actually somewhat disruptive as also indicated by the resulting higher population sizes.

An additional specializing effect is observable that is partially a result of the binary re-coding of the population for the model building and model-based offspring generation. Due to the random choice of the second bit of a don't care attribute, actual offspring may be generated that does not match the current action set. For example, consider the two simple classifiers `1#→1` `##→1`. The resulting binary codes may be `0111` and `1010`. Thus, dependent on the model dependencies, offspring may be generated with the code `0010`, which translates into `0#→1`. Although the average specificity is maintained, the offspring may not match the current action set leading to an increasing diversity in the population. This additional diversity may be slightly disruptive as observed in the XCS/ECGA results. Note that we also ran experiments applying uniform crossover on the transferred binary code. The result was that the population was filled-up with apparently meaningless classifiers and no learning was observable.

XCS/BOA results in a similarly successful solution of the 3-parity, 6-multiplexer problem (Figure 9). In fact, XCS/BOA with MCMC sampling reaches higher performance slightly faster than BB-wise uniform crossover (curve BOA: 0/18). This is due to the extra specialization pressure induced by the classifier optimization, which uses the probabilistic model of the global population. However, if too many optimization steps are used with the global model probabilities, (setting BOA: 0/90), the mechanism over-optimizes the offspring towards the global probability distribution and consequently over-specializes the population with respect to the current global model as also indicated in the large population size. This problem does not occur when the offspring is sampled using the local probability distribution or if the selected offspring is optimized using a model with parameters updated to fit the local niche. All settings exhibit similar performance, which is nearly as good as that with BB-wise uniform crossover. In contrast to XCS/ECGA, XCS/BOA does not suffer from any over-specializations and



(a) (b)

Figure 9: When applying XCS/BOA to the hierarchical 3-parity, 6-multiplexer problem, recombination becomes effective. The model learning mechanisms shows fast and accurate learning in various settings. The first number in the XCS/BOA runs refers to the number of selected classifiers used to set the probabilities to the local distribution (0 indicates that the global probability distribution is used). The second number refers to the probabilistic optimization steps applied to a selected parental classifier (0 indicates that offspring was sampled directly from the model using PLS).

all runs reach 100% performance reliably.

The hierarchical 2-parity, 11-multiplexer combination is slightly easier to solve, despite that the problem length is longer ($l = 22$ as opposed to $l = 18$ in the 3-parity, 6-multiplexer). XCS/ECGA solves the problem reliably in all settings. If more classifiers are used to derive the current local probabilities, the consequent larger diversity appears to slightly delay convergence. XCS/BOA converges to a complete problem solution even with MCMC and 90 probabilistic optimization steps towards the global model ($0/90$). However, note the large population size that strongly decreases the learning speed. XCS/BOA with MCMC sampling and local probabilities learns the solution slightly slower than XCS/BOA with MCMC with global probabilities or standard PLS sampling with local probabilities. This is the case due to the more cautious offspring generation causing less diversity in the population and thus a slightly slower but more cautious learning progress.

5.2 Hierarchical parity, count ones problem

The hierarchical 3-parity, 5-count ones problem requires a final optimal solution of $2^{10} * 10$ classifiers. Subsolutions are overlapping in that three out of five parity blocks need to be specified correctly to predict class zero or one accurately. XCS/ECGA does not show any difficulties in solving the problem (Figure 11a,b). All runs converge to the near-optimal solution nearly as fast as XCS with BB-wise uniform crossover. A lower mutation rate hardly influences performance.

Similarly, XCS/BOA succeeds in solving the problem (Figure 11c,d). In this case, XCS/BOA is slightly slower than XCS/ECGA. We believe that the reason for this is

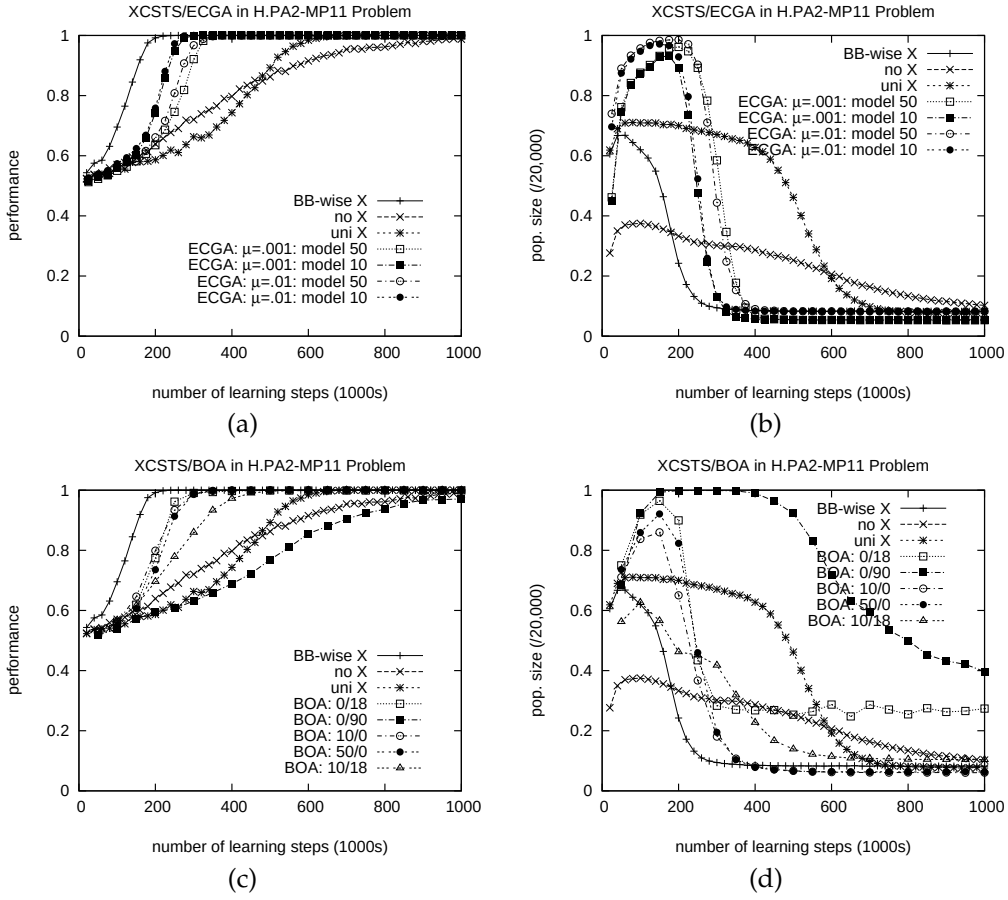


Figure 10: In the hierarchical 2-parity, 11-multiplexer problem, XCS/ECGA and XCS/BOA nearly achieve the performance of the BB-wise uniform crossover.

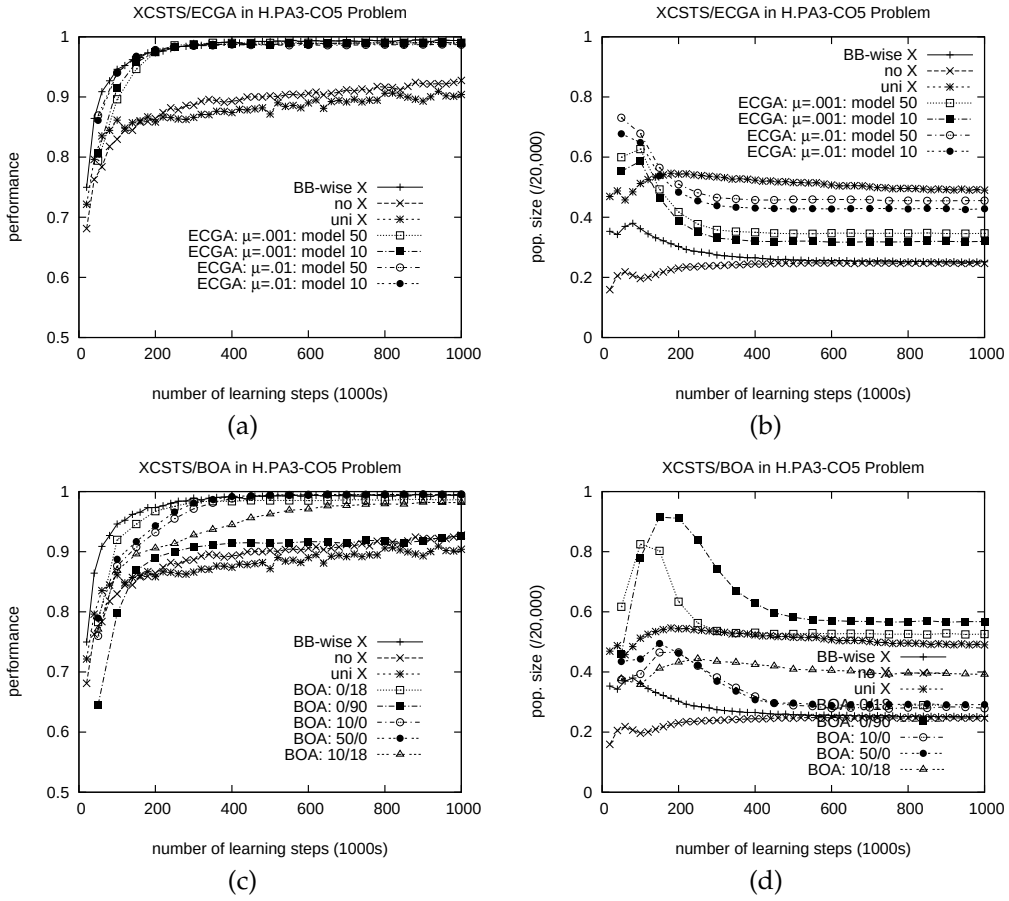


Figure 11: Also the hierarchical 3-parity, 5-count ones problem is effectively solvable with either model-based offspring generation method. The ECGA combination appears slightly more robust in this case, indicating that the Bayesian-net might model unnecessary, spurious dependencies that delay convergence.

that for this problem, XCS/BOA detects spurious dependencies that may slow down the overall learning process. Since in this problem the propagation of all three BBs independently is nearly most effective, the Bayesian learning algorithm appears to overmodel and thus delay the learning. Nonetheless, a near-optimal performance level is still reached very fast clearly outperforming XCS without crossover as well as XCS with uniform crossover. Similarly as in other experiments, using MCMC sampling with 90 steps and global probabilities degrades XCS performance, indicating an overly strong bias towards the global population of classifiers.

To summarize, the results confirm that XCS can be successfully combined with a number of structural learners to improve offspring generation. The implemented XCS/ECGA and XCS/BOA combinations achieve performance similar to the performance with BB-wise uniform crossover, which relies on explicit problem knowledge. XCS/ECGA as well as XCS/BOA do not require any global problem knowledge and thus allow XCS to flexibly adjust its recombination mechanism to the encountered

problem.

6 Discussion

One of the most important lessons from the design of competent genetic and evolutionary algorithms is that standard variation operators are severely limited in typically challenging problems and that variation operators must be capable of adapting to the problem under consideration (Goldberg, 2002; Pelikan, 2004) to provide robust and scalable solutions for challenging real-world problems. Despite that, learning classifier system (LCS) research has adhered to simple crossover and mutation operators. Moreover, the utility and necessity of the crossover operator in XCS remained hardly understood.

In this paper we have shown that also in LCSs, and in the XCS classifier system in particular, effective building block (BB) processing is necessary for an efficient and reliable evolution of a complete problem solution. If there are strong dependencies between different attributes, it is necessary to identify these dependencies and consequently ensure effective BB processing. Mutation alone is strongly handicapped, taking a very long time to find the optimal problem solution. Uninformed, simple crossover operators are prone to disrupt the evolutionary search and often prevent the evolution of the optimal solution. However, if XCS uses competent recombination operators that ensure effective processing of important feature subsets, it becomes capable of identifying and processing important BBs.

To ensure effective processing of BBs, the model-building and offspring generation mechanisms from the extended compact GA (ECGA) and the Bayesian optimization algorithm (BOA) were incorporated into XCS. The proposed extensions of XCS were named XCS/ECGA and XCS/BOA. Since XCS applies a steady-state, niched GA, the probabilistic model does not have to be built in every iteration but instead it can be built at predefined points in time.

We pointed out that in difference to GAs and optimization problems, in LCSs and classification problems, problem structure may differ dependent on the problem subspace under consideration. In essence, different attributes may be relevant in different problem subspaces so that recombination operators need to be adjusted to the current problem subspace. That is why the built probabilistic dependency model is modified to reflect the local probability distribution in an action set at each time step to generate offspring according to the local probability distribution with the bias to the global dependency structure.

To investigate the recombinatory capabilities of the XCS system, we introduced hierarchical binary classification problems. In the investigated examples, parity blocks on the lower level were combined with the multiplexer or count ones function on the higher level. XCS with simple crossover is not able to solve the resulting problems reliably. We observed the expected disruptive effects of simple recombination when applying uniform crossover as well as when applying one-point or two-point crossover with loosely coded blocks (attributes are randomly distributed over the input). Mutation alone is able to solve the parity, multiplexer problem—albeit much less efficiently—but it is not able to solve the parity, count ones problem satisfactorily given a reasonable number of iterations.

On the other hand, XCS/ECGA and XCS/BOA learned complete solutions to all hierarchical problems proposed in this paper efficiently and reliably. Further investigations in Butz (2004b) show that XCS/ECGA and XCS/BOA are able to solve other binary classification problems more reliably and as fast or faster than XCS with standard crossover operators.

In conclusion, XCS/ECGA and XCS/BOA may be termed *competent LCSs*, that is, LCSs that can solve decomposable problems of bounded order scalably and reliably. The experiments presented in this paper considered lower order BBs of order three or less. In general, the larger the lower order BBs, the more difficult the problem. In the extreme case of only one building block (such as one large parity problem), we know that XCS scales exponentially in the size of the BB because of necessary initial supply and sufficient reproductive opportunities (Butz, Goldberg, & Tharakunnel, 2003; Butz, Kovacs, Lanzi, & Wilson, 2004). Note, however, that the solution complexity also grows exponentially in BB size and thus XCS still solves the problem polynomially in the size of the problem solution (Butz, 2006). Thus, as long as the population size is sufficiently large and sufficiently specific to ensure the supply and growth of lower order BB structures, XCS is guaranteed to detect and process those structures effectively and reliably.

XCS/ECGA and XCS/BOA are currently applicable to problems in the binary domain and for other fixed-size alphabets. Analogically to GAs (Bosman & Thierens, 2000; Larrañaga et al., 2000; Ahn et al., 2004; Pelikan et al., 2003), real-valued extensions of the real-valued XCSR system (Wilson, 2000; Wilson, 2001) are expected to be possible and await future research.

In addition to extending XCS/ECGA and XCS/BOA to real-valued classification problems, the developed competent XCS system should be tested in challenging real-world scenarios. XCS may be applied to more challenging predictive tasks such as the prediction of actual changes in the world, similarly to the ACS system (Butz, Goldberg, & Stolzmann, 2002), the modular MACS system (Gérard & Sigaud, 2003), or other state transformations in the environment. The current XCS system is ready to be applied to such online learning, predictive tasks and is expected to work particularly well in problems in which different predictions are required in different problem subspaces. The more the structural properties between the different subspaces are related, the easier will XCS evolve a complete problem solution. Future research needs to evaluate these conjectures in different abstract and real-world problem domains.

References

Ahn, C. W., Ramakrishna, R. S., & Goldberg, G. (2004). Real-coded Bayesian optimization algorithm: Bringing the strength of BOA into the continuous world. *Genetic and Evolutionary Computation Conference (GECCO-2004), 3102*, 840–851.

Booker, L. B., Goldberg, D. E., & Holland, J. H. (1989). Classifier systems and genetic algorithms. *Artificial Intelligence*, *40*, 235–282.

Bosman, P. A., & Thierens, D. (2000). *Mixed IDEAs* (Utrecht University Technical Report UU-CS-2000-45). Utrecht, Netherlands: Utrecht University.

Buntine, W. L. (1991). Theory refinement of Bayesian networks. *Proceedings of the Uncertainty in Artificial Intelligence (UAI-91)*, 52–60.

Butz, M. V. (2004a). *Hierarchical classification problems demand effective building block identification and processing in lcss* (IlliGAL report 2004017). University of Illinois at Urbana-Champaign: Illinois Genetic Algorithms Laboratory.

Butz, M. V. (2004b). *Rule-based evolutionary online learning systems: Learning bounds, classification, and prediction*. Doctoral dissertation, University of Illinois at Urbana-Champaign, Urbana, IL.

Butz, M. V. (2006). *Rule-based evolutionary online learning systems: A principled approach to LCS analysis and design*. Studies in Fuzziness and Soft Computing. Berlin Heidelberg: Springer-Verlag.

Butz, M. V., Goldberg, D. E., & Lanzi, P. L. (2004). Bounding learning time in XCS. *Proceedings of the Sixth Genetic and Evolutionary Computation Conference (GECCO-2004): Part II*, 739–750.

Butz, M. V., Goldberg, D. E., Lanzi, P. L., & Sastry, K. (2004). *Bounding the population size to ensure niche support in XCS* (IlliGAL report 2004033). University of Illinois at Urbana-Champaign: Illinois Genetic Algorithms Laboratory.

Butz, M. V., Goldberg, D. E., & Stolzmann, W. (2002). The anticipatory classifier system and genetic generalization. *Natural Computing*, *1*, 427–467.

Butz, M. V., Goldberg, D. E., & Tharakunnel, K. (2003). Analysis and improvement of fitness exploitation in XCS: Bounding models, tournament selection, and bilateral accuracy. *Evolutionary Computation*, *11*, 239–277.

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

Butz, M. V., Kovacs, T., Lanzi, P. L., & Wilson, S. W. (2004). Toward a theory of generalization and learning in XCS. *IEEE Transactions on Evolutionary Computation*, *8*, 28– 46.

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

Butz, M. V., Sastry, K., & Goldberg, D. E. (2003). Tournament selection in XCS. *Proceedings of the Fifth Genetic and Evolutionary Computation Conference (GECCO-2003)*, 1857–1869.

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 (LNAI 1996)* (pp. 253–272). Berlin Heidelberg: Springer-Verlag.

Chickering, D. M., Heckerman, D., & Meek, C. (1997). *A Bayesian approach to learning Bayesian networks with local structure* (Technical Report MSR-TR-97-07). Redmond, WA: Microsoft Research.

Cooper, G. F., & Herskovits, E. H. (1992). A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, *9*, 309–347.

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.

Friedman, N., & Goldszmidt, M. (1999). Learning Bayesian networks with local structure. In Jordan, M. I. (Ed.), *Graphical models* (pp. 421–459). Cambridge, MA: MIT Press.

Gérard, P., & Sigaud, O. (2003). Designing efficient exploration with MACS: Modules and function approximation. *Proceedings of the Fifth Genetic and Evolutionary Computation Conference (GECCO-2003)*, 1882–1893.

Gibson, J. J. (1979). *The ecological approach to visual perception*. Mahwah, NJ: Lawrence Erlbaum Associates.

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

Harik, G. (1999). *Linkage learning via probabilistic modeling in the ECGA* (IlliGAL report 99010). University of Illinois at Urbana-Champaign: Illinois Genetic Algorithms Laboratory.

Harik, G. R., Cantú-Paz, E., Goldberg, D. E., & Miller, B. L. (1997). The gambler's ruin problem, genetic algorithms, and the sizing of populations. *Proceedings of the International Conference on Evolutionary Computation (ICEC-97)*, 7–12. Also IlliGAL Report No. 96004.

Heckerman, D., Geiger, D., & Chickering, D. M. (1994). *Learning Bayesian networks: The combination of knowledge and statistical data* (Technical Report MSR-TR-94-09). Redmond, WA: Microsoft Research.

Henrion, M. (1988). Propagating uncertainty in Bayesian networks by probabilistic logic sampling. In Lemmer, J. F., & Kanal, L. N. (Eds.), *Uncertainty in Artificial Intelligence* (pp. 149–163). Amsterdam, London, New York: Elsevier.

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

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.

Howard, R. A., & Matheson, J. E. (1981). Influence diagrams. In Howard, R. A., & Matheson, J. E. (Eds.), *Readings on the principles and applications of decision analysis*, Volume II (pp. 721–762). Menlo Park, CA: Strategic Decisions Group.

Kovacs, T. (1996). *Evolving optimal populations with XCS classifier systems*. Master's thesis, School of Computer Science, University of Birmingham, Birmingham, U.K.

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.

Larrañaga, P. (2002). A review on estimation of distribution algorithms. In Larrañaga, P., & Lozano, J. A. (Eds.), *Estimation of Distribution Algorithms* (Chapter 3, pp. 57–100). Boston, MA: Kluwer Academic Publishers.

Larrañaga, P., Etxeberria, R., Lozano, J. A., & Pena, J. M. (2000). Optimization in continuous domains by learning and simulation of Gaussian networks. In Wu, A. (Ed.), *Workshop Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)* (pp. 201–204). San Fransisco, CA: Morgan Kaufmann.

Larrañaga, P., & Lozano, J. A. (Eds.) (2002). *Estimation of distribution algorithms: A new tool for evolutionary computation*. Boston, MA: Kluwer.

Lobo, F., & Harik, G. (1999). *Extended compact genetic algorithm in C++* (IlliGAL report 99016). University of Illinois at Urbana-Champaign: Illinois Genetic Algorithms Laboratory.

Mitchell, T. M. (1997). *Machine learning*. Boston, MA: McGraw-Hill.

Mühlenbein, H. (1992a). How genetic algorithms really work: I.Mutation and Hillclimbing. *Parallel Problem Solving from Nature*, 15–25.

Mühlenbein, H. (1992b). How genetic algorithms really work: I.Mutation and Hillclimbing. In Männer, R., & Manderick, B. (Eds.), *Parallel Problem Solving from Nature- PPSN II* (pp. 15–25).

Mühlenbein, H., & Schlierkamp-Voosen, D. (1993). Predictive models for the breeder genetic algorithm: I. Continuous parameter optimization. *Evolutionary Computation*, *1*(1), 25–49.

Mühlenbein, T. M., & Paas, G. (1996). From recombination of genes to the estimation of distributions: Binary parameters. In *???* (pp. 178–187).

Neal, R. M. (1993). *Probabilistic inference using Markov chain Monte Carlo methods* (Technical Report CRG-TR-93-1). Dept. of Computer Science, University of Toronto.

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: Networks of plausible inference*. San Mateo, CA: Morgan Kaufmann.

Pelikan, M. (2001). Bayesian optimization algorithm, decision graphs, and Occam's razor. *Proceedings of the Third Genetic and Evolutionary Computation Conference (GECCO-2001)*, 511–518.

Pelikan, M. (2002). *Bayesian optimization algorithm: From single level to hierarchy*. Doctoral dissertation, University of Illinois at Urbana-Champaign, Urbana, IL. Also IlliGAL Report No. 2002023.

Pelikan, M. (2004). *Hierarchical Bayesian optimization algorithm: Toward a new generation of evolutionary algorithms*. Springer-Verlag. In press.

Pelikan, M., Goldberg, D. E., & Cantu-Paz, E. (1999). BOA: The Bayesian optimization algorithm. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99)*, 525–532.

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.

Pelikan, M., Goldberg, D. E., & Tsutsui, S. (2003). Getting the best of both worlds: Discrete and continuous genetic and evolutionary algorithms in concert. *Information Sciences*, *156*(3–4), 36–45.

Rudnick, M. W. (1992). *Genetic algorithms and fitness variance with application to automated design of artificial neural networks*. Doctoral dissertation, Oregon Graduate Institute of Science & Technology, Beaverton, OR.

Sastry, K., & Goldberg, D. E. (2000). *On extended compact genetic algorithm* (IlliGAL report 2000026). University of Illinois at Urbana-Champaign: Illinois Genetic Algorithms Laboratory.

Simon, H. A. (1969). *Sciences of the artificial*. Cambridge, MA: MIT Press.

Thierens, D., & Goldberg, D. E. (1993). Mixing in genetic algorithms. *Proceedings of the Fifth International Conference on Genetic Algorithms*, 38–45.

Thierens, D., Goldberg, D. E., & Pereira, A. G. (1998a). Domino convergence, drift, and the temporal-salience structure of problems. In *Proceedings of the 1998 IEEE World Congress on Computational Intelligence* (pp. 535–540). New York, NY: IEEE Press.

Thierens, D., Goldberg, D. E., & Pereira, A. G. (1998b). Domino convergence, drift, and the temporal-salience structure of problems. *Proceedings of the International Conference on Evolutionary Computation (ICEC-98)*, 535–540.

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 (LNAI 1813)* (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 (LNAI 1996)* (pp. 158–174). Berlin Heidelberg: Springer-Verlag.