7

Effective XCS Search: Building Block Processing

The facetwise approach to GA theory stresses effective mixing and decision making among BBs. The last chapter showed that in XCS, BBs are subsets of specified attributes that increase accuracy. The reproductive opportunity bound additionally ensures that BBs are able to grow in the population making time for the identification and reproduction of schema representatives. Until now, we assumed that mutation is sufficient to generate better classifiers as investigated in the time bound. However, the GA literature suggests that effective crossover operators are mandatory to solve boundedly difficulty optimization problems in which small, lower-level BBs may mislead the population to a local optimum.

Thus, this section investigates problems that pose a similar BB-challenge to the XCS system. We create hierarchical classification problems that demand the effective processing of lower level BB structures. In effect, we face the third part of the proposed facetwise problem decomposition for LCS systems, that is, the necessity to enable optimal solution search: (1) Search via mutation needs to be effective; (2) Search via recombination needs to be effective; (3) Local problem solution structure may be different from global structure and thus needs to be taken into account when designing effective recombinatory search operators.

Search via mutation was investigated in several of the previous chapters. We showed its influence on specificity as expressed in the specificity equation (Equation 5.8) as well as its influence in generating schema representatives and finding an optimal problem solution (time extension of schema supply bound, time bound, Chapter 6). We also noted slight disruptive effects affecting the sustenance of problem solutions.

This chapter focuses on recombination as well as differences between local and global problem structure. To investigate the effectiveness of recombination, we identify BB-hard problems in classification problems. XCS is not able to solve these problems due to disruption caused by crossover. Mutation alone may solve the problem, but may take a long time. To solve the problems effectively, a competent crossover operator is necessary that recombines BBs

without disrupting them. Experiments with an informed crossover operator confirm this hypothesis but are unsatisfactory since BB structures cannot be assumed to be known beforehand.

Thus, we integrate structure extraction mechanisms previously successfully applied in the GA literature. However, since XCS reproduces in action sets and thus in problem subspaces, the methods need to be modified for the XCS system—respecting the difference in local problem solution structure in comparison to global problem solution structure as suggested in the eighth point of our facetwise LCS theory approach.

In particular, we introduce the formation of *marginal product models* used in the *extended compact GA* (ECGA) (Harik, 1999). The technique is able to identify non-overlapping dependency structures in a problem. Since the marginal product model can only model non-overlapping BBs, we also utilize dependency structures in the form of Bayesian decision trees as used in the Bayesian optimization algorithm (BOA) (Pelikan, Goldberg, & Cantu-Paz, 1999). The Bayesian model can also detect overlapping dependency structures.

We integrate the methods in XCS by extracting a dependency structure from the *global* population and using the gained structural knowledge to generate *local* offspring. The resulting enhanced XCS system is able to solve the identified BB-hard problems.

The remainder of this chapter first derives BB-hard problems for classification. The evaluations show that only an informed crossover operator can solve the problems reliably. Next, we introduce competent crossover operators derived from mechanisms used in ECGA and BOA to solve the BB-challenge without any prior structural information. Summary and conclusions put the results into a broader LCS perspective.

7.1 Building Block Hard Problems

As we have seen in the previous chapter, XCS relies on the supply of minimal order schemata that increase classification accuracy—the BBs in XCS. In the previous chapter, we evaluated the schema bound and reproductive opportunity bound in a problem in which one minimal order schema had to be present. The solution was found once the block that specified all k_m attributes correctly was detected and reproduced.

The question now is if XCS is able to identify and process several of those BBs, represented by schemata of order k_m, effectively. Thus, we first revisit fitness guidance to understand BB processing in XCS even better. Next, we create hierarchical classification problems which consist of several BB structures. In order to solve the problems efficiently, it is necessary to identify, reproduce, *and* recombine the blocks appropriately. Thus, fitness guidance needs to be exploited to successfully grow blocks. Effective recombination operators need to be available as well to successfully combine blocks.

This section first provides further exemplar problems and the consequent fitness guidance in the problem. Next, we introduce hierarchical classification problems showing that a proper BB propagation algorithm is mandatory to solve these types of problems effectively. Section 7.2 introduces explicit BB-identification and propagation mechanisms to XCS.

7.1.1 Fitness Guidance Exhibited

As noted before, the strength of the fitness pressure in XCS depends on the problem at hand. A typical easy problem for the XCS mechanism is the count ones problem (Butz, Goldberg, & Tharakunnel, 2003), in which the 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 in the GA literature. Each relevant bit raises accuracy. Thus, each relevant bit is progressively more specialized in the condition parts of the classifiers in XCS.

Table 7.1 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 specialization of progressively more ones or more zeroes decreases error and consequently fitness. Thus, fitness progressively pushes towards the specification of more ones (zeros) in the problem. Butz, Goldberg, and Tharakunnel (2003) showed that uniform crossover can assure and improve successful learning of the count ones problem with many additional irrelevant bits due to its effective uniform recombination.

Table 7.1. Expected reward prediction and reward prediction error estimates for exemplar condition parts in several typically-used Boolean function problems for classifiers with action part $A = 1$.

5-Count-Ones Problem			Hidden 4-Parity Problem			6-Multiplexer Problem		
C	R	ε	C	R	ε	C	R	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

In comparison with the count-ones problem, the hidden parity problem (Kovacs, 1999) is harder because the specialization of one attribute of the parity bits does not raise accuracy. Only once all relevant attributes are specialized, accuracy raises, effectively directly solving the problem. Thus, supply of classifiers that specialize all parity bits is necessary, as also shown in the previous chapter. Table 7.1 shows the four hidden parity problem (the fifth bit is irrelevant). Error only drops to zero once all four attributes are correctly specified. In the next section we show that a hierarchical parity, multiplexer problem forces XCS to propagate the lower level parity blocks effectively.

Finally, we again show the widely studied multiplexer problem (Wilson, 1995; Wilson, 1998), in which accuracy somewhat guides towards the correct specializations. Initially, though, only the specialization of the value bits raises accuracy. It is only once some value bits are specified in a classifier condition that specialization of the address bits decreases accuracy further. Table 7.1 clarifies the property in the 6-multiplexer case. 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 does specificity in the address attributes take over.

7.1.2 Building Blocks in Classification Problems

The above problems consist of BB structure that either consist of 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 initial fitness guidance leads to the less important value bits and only later, fitness guides towards the specialization of the address bits. Thus, BB processing is somewhat easy in the count ones problem in that only one specialization needs to be identified at a time. This can be accomplished by mutation. More challenging is the hidden parity problem in which classifiers that specialize all relevant bits need to be available from the beginning.

What if we combine the output of multiple hidden-parity problems to a higher-level problem input? The hierarchical dependency between the subproblems would require that the hidden parity blocks need to be identified and then recombined effectively. This described hierarchical problem structure consequently requires effective BB processing.

We construct such a problem structure using a two-level hierarchy. On the lower level, small Boolean functions are evaluated which provide the input to the higher level. Thus, the function evaluation is pursued in two stages. The evaluation of the functions on the lower level serve as input to the higher level. For example, we mainly use a parity, multiplexer combination in which the small lower-level blocks are evaluated by the parity function. The results are then fed into the higher-level multiplexer function deriving the overall class of the problem instance. Further information and visualizations on the hierarchical problem class are provided in Appendix C.

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). Thus, we believe that the introduced hierarchical problem is an important problem to solve with a general machine learning system.

How can XCS solve this problem? Clearly, the lower level parity blocks need to be identified first to enable the discovery of the higher level function. Table 7.2 shows exemplar conditions with 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 it is only if the detected blocks are not disrupted that XCS is able to solve the problem. Additionally, it is only if the BBs are recombined effectively that XCS can solve the problem efficiently.

Table 7.2. 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.

C	R	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

Note that we focus in the remainder on XCS's 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 challenges with respect to the facetwise theory for LCSs.

7.1.3 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.

Hierarchical Three-Parity, Six-Multiplexer Problem

Performance of XCS in the hierarchical 3-parity, 6-multiplexer problem is shown in Figure 7.1.[1] 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, but rather disrupts them.

In addition to the usual crossover operators of uniform, one-point, and two-point crossover, we applied an informed crossover operator to investigate the potential of more competent recombination operators. The informed crossover operator is informed about the BB structure in the problem applying a BB-wise uniform crossover operator. BB-wise uniform crossover exchanges only complete BBs uniformly randomly similar to uniform crossover, which exchanges attributes uniformly randomly.

XCS with BB-wise crossover solves the problem effectively and nearly independently of the mutation type used. Thus, a mechanism in XCS that is able to identify the lower-level BB structure is necessary. Once identification is successful, effective BB processing and recombination can be applied. Uniform crossover strongly disrupts BB propagation, preventing learning (Figure 7.1a,c). The continuously high population size indicates that uniform crossover causes a high diversity in the population. However, BB-disruption prevents the growth of higher-order BBs (Figure 7.1b,d). Mutation alone is able to solve the problem but learning takes about three times as long as in the case of BB-wise uniform crossover operator (Figure 7.1a,c). The population sizes indicate that diversity stays much lower resulting in a lower macro-population size (Figure 7.1a,c). If the BBs are tightly coded, one-point and two-point crossover are able to recombine BBs effectively (Figure 7.1a). However, if the attributes are randomly distributed, the potential recombinatory benefit of one-point or two-point crossover is overshadowed by their disruptive effect delaying learning and population convergence (Figure 7.1c,d). Thus, one-point and two-point crossover show beneficial effects in the case of tightly-coded BBs, but disruptive effects in the loosely-coded case. Since the

[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 $P_{\#} = 0.6$.

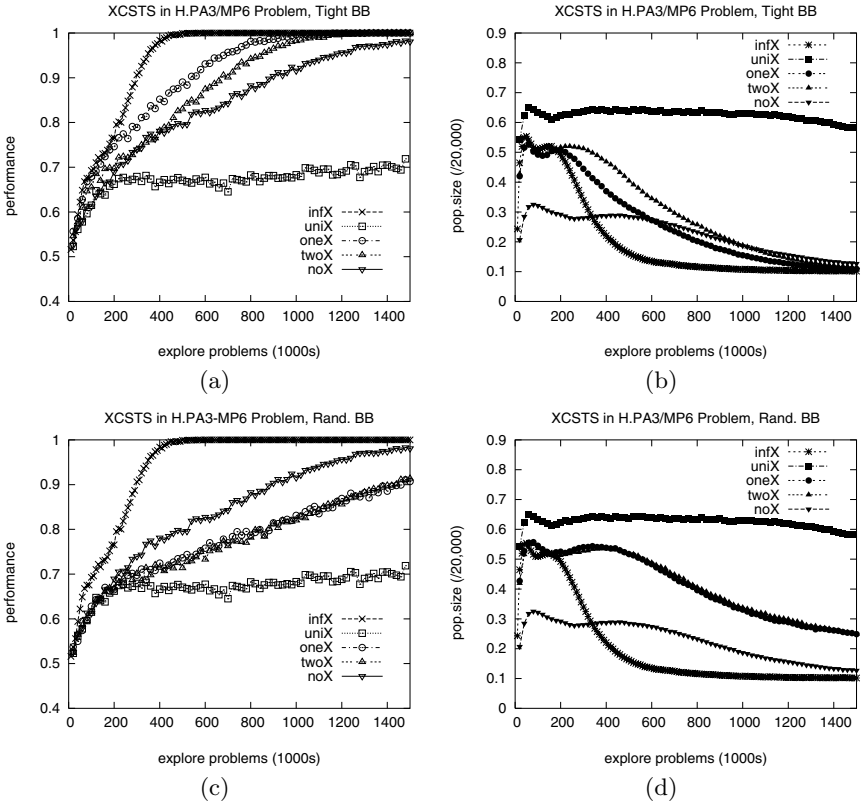

Fig. 7.1. Performance (a,c) and population sizes (b,d) of XCS ($N = 20k$) in the hierarchical 3-parity, 6-multiplexer problem (infX = informed (i.e. BB-wise uniform) crossover, uniX = uniform crossover, oneX = one-point crossover, twoX = two-point crossover, noX = mutation only). Efficient BB recombination 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.

dependency structures cannot be expected to be tightly coded in general, competent crossover operators are mandatory.

Although mutation can be tuned to solve the hierarchical 3-parity, 6-multiplexer problem nearly as well as the informed crossover operator does (Figure 7.2b), the behavior is unsatisfactory: larger problems or the same problem with additional irrelevant attributes would make it impossible to set the mutation rate high enough due to the reproductive opportunity bound introduced in the last chapter. However, a small mutation rate strongly delays learning if only mutation is applied. The BB-wise uniform crossover operator stays nearly independent from the mutation operator (Figure 7.2a,b). It only relies on the supply of lower level BBs, which is usually ensured by the initial

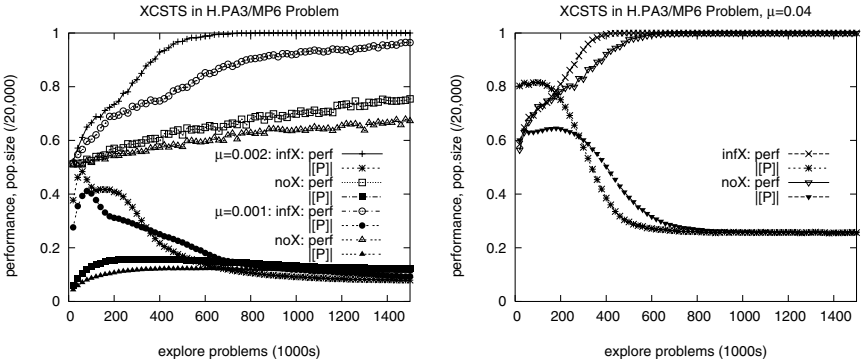

Fig. 7.2. A low mutation rate strongly delays learning if effective recombination is not applied. With a mutation rate of $\mu = 0.001$, however, certain specialized attributes might get lost so that performance is delayed even with effective recombination (a). Higher mutation rates alleviate the problem (b) but may not be applicable in problems with more attributes.

sufficiently large specificity. Thus, a competent crossover operator that detects BBs on the fly is highly desirable.

Hierarchical Parity, Count-Ones Problem

Figure 7.3 confirms similar results in the hierarchical 3-parity, 5-count ones problem. In the runs, 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 very 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 a very long time to evolve an accurate solution. The performance of BB-wise crossover is not reached by any of the other settings.

7.2 Building Block Identification and Processing

Facing the BB-challenge within XCS it is necessary to develop a mechanism that learns effective recombination online. Most appropriate for this seems to be an estimation of distribution algorithm (EDA) approach, modeling dependency structures and recombining them appropriately (Pelikan, Goldberg, & Lobo, 2002; Larrañaga, 2002).

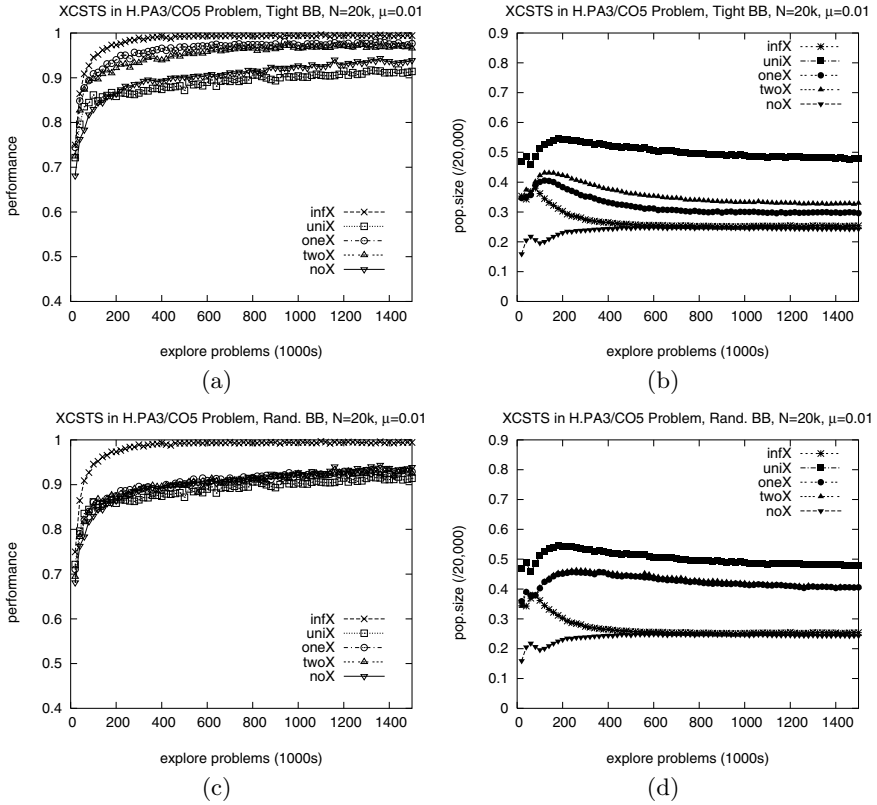

Fig. 7.3. Performance (a,c) and population sizes (b,d) of XCS ($N = 20k$) in the hierarchical 3-parity, 5-count ones problem. Again, efficient BB recombination strongly improves XCS's performance. One-point and two-point crossover are only beneficial if the BBs are tightly coded. Mutation alone gradually improves performance but is much less effective that BB-wise crossover.

However, the evolutionary component in XCS differs from the usual GA application in several respects. Due to XCS's niche reproduction in action sets and since action sets are generally rather small compared to the whole population, structure extraction is hard to apply successfully in an action set alone. On the other hand, extracting global structure results in global offspring generation, which may not reflect the local problem structure appropriately. Thus, the inclusion of an EDA mechanism in XCS is not straight-forward.

This section integrates the BB-identification mechanism in the ECGA (Harik, 1999) to identify and process lower-level dependency structures. Alternatively, we also show how to integrate the more powerful Bayesian learning mechanism used in BOA (Pelikan, Goldberg, & Cantu-Paz, 1999). We show that both mechanisms are suitable to learn the *global* lower-level problem

structure and can be used to generate or improve *local* classifier offspring. The generation and improvement of the local offspring depends on the current action set, just as the original XCS crossover operation does.

We first give an overview of the learning algorithms used in the ECGA as well as in BOA to learn the respective dependency structures. Next, we show how these mechanisms may be integrated into the XCS classifier system. We learn the dependency structures from the filtered and converted XCS population and then use the learned structures to sample and/or optimize offspring in local problem niches.

7.2.1 Structure Identification Mechanisms

Our investigations show that at least two structure identification mechanisms are suitable for competent BB processing in XCS: (1) *marginal product models* used for example in the ECGA mechanism (Harik, 1999), and (2) Bayesian decision tree structures used in BOA (Pelikan, Goldberg, & Cantu-Paz, 1999; Pelikan, 2002). The former is easier to understand and to apply but is limited to the identification of non-overlapping BBs only. The latter is more complicated but is able to model overlapping dependency structures as well.

Any structure extraction mechanism, however, faces the problem of accuracy vs. generality. That is, the generated model is intended to identify relevant dependencies but ignore spurious, irrelevant dependencies. Hereby, we rely on *Occam's razor* in that we want to find the model that codes the data structure most compactly. The usual approach to balance the two optimization factors is to apply the minimum description length principle (MDL) (Mitchell, 1997). Essentially, the MDL principle weighs accuracy with model complexity by combining the cost of describing the derived model with the resulting cost of encoding the modeled data using the model. Using information theoretic principles, the two influences can be appropriately balanced using entropy as the basic measure.

7.2.2 BB-Identification in the ECGA

As mentioned above, the ECGA mechanism learns a non-overlapping BB-structure, termed a marginal product model. ECGA considers the best individuals in its current population (selected by any suitable selection mechanism, e.g. tournament selection) and builds the model from these individuals, that is, the data. For example, consider the simple population shown in Table 7.3. ECGA finds dependency structures in terms of feature subsets (that is, the BBs). A block is essentially formed if the representation as a block, albeit more complex to express as a model, results in a sufficient reduction in the resulting data description complexity when using the model.

ECGA expresses these two complexity measures in terms of model complexity MC, which favors more compact models, and the resulting compressed

population complexity CPC, which favors a more compact (accurate) population representation with respect to the used model. The MDL measure is simply the sum of MC and CPC. The two complexity measures are determined by

$$MC = \log N \sum_{I} 2^{S[I]} - 1, \tag{7.1}$$

$$CPC = N \sum_{I} E(M_I), \tag{7.2}$$

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 $E(M_I)$ the entropy of a probability distribution. The measure MC exploits the fact that $\log N 2^{S[I]}$ bits are necessary to describe the probability distributions over each subset $S[I]$ ($2^{S[I]}$) probability entries. The measure CPC then determines the complexity of coding all N individuals with respect to the subsets, which is determined by the sum of the entropies over all subsets. Table 7.3 shows several potential model structures and the resulting MC and CPC measures. It can be seen how the MDL principle balances the model complexity with the resulting population description complexity.

Table 7.3. The illustrated example shows how the marginal product model learning mechanism detects structural properties in a population. While MC measures the model complexity, CPC measures the compressed population complexity potentially gained due to a more complex model representation.

problem instances	
11000	11111
11001	11110
11000	11110
00111	00001

marginal product model	MC	CPC	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]	48	22.49	70.49
[1 2 3] [4 5]	30	30.49	60.49

The ECGA mechanism learns the marginal product model, greedily minimizing the sum of MC and CPC. That is, if the scaled entropy decrease and thus the decrease of CPC due to a merge of two sets is larger than the consequent MC increase, the merge is performed. Subsets are greedily merged until no more merge is able to decrease the MDL measure. In the ECGA, the model is learned every GA iteration. The offspring population is generated out of the derived dependency structure probabilistically sampling from the dependency structure. That is, each BB is considered independently when generating an offspring individual choosing the corresponding code probabilistically with respect to the determined probability distribution. The MDL mechanisms used to grow the dependency structure in XCS similar to the ECGA are taken from the available ECGA implementation (Lobo & Harik, 1999).

The ECGA mechanism has shown to be able to effectively solve previously BB-hard problems, such as the typically used deceptive trap problems (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.

7.2.3 BB-Identification in BOA

BOA uses the more powerful representation of Bayesian networks in order to represent BB-structures. The overall learning mechanism is similar to the one applied in the ECGA, learning a Bayesian network from a selected subset of individuals and sampling from the Bayesian network. Due to the potentially much more complex Bayesian network structure, the generation and sampling mechanisms are not as straight-forward as in the ECGA.

Bayesian networks (BNs) (Howard & Matheson, 1981; Pearl, 1988; Buntine, 1991; Mitchell, 1997) combine statistics with graph theory generating a modular graphical model of the analyzed data. BNs can be used to estimate probability distributions as well as to do inference. A Bayesian network is defined by its structure and its (conditional) probabilities. The structure is usually encoded by a directed acyclic graph with the nodes corresponding to the 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.

The BN as a whole encodes a joint probability distribution given by

$$p(x) = \prod_{i=1}^{n} p(x_i|\Pi_i), \tag{7.3}$$

where $X = (X_0, \ldots, x_{n-1})$ 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 then codes the probability of the values of x_i given the parental values. A directed edge relates the variables so that in the encoded distribution, a variable corresponding to a terminal node is conditioned on the parental variables. More incoming edges into a node result in a conditional probability of the variable with a condition containing all its parents.

As the ECGA structure assumes the independence of the blocks, also a Bayesian network encodes a set of (implicit) independence assumptions. Variables are assumed to be independent of each other given the values of the variables of all of their parents and none of their common descendants. The exact independence assumptions resulting from the BN structure can be found in the literature (Mitchell, 1997).

Conditional probability tables (CPTs) store the 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 7.4 shows an example CPT for $p(x_1|x_2, x_3, x_4)$.

A greedy algorithm is usually used to learn a BN. The greedy algorithm starts with an empty BN. Each iteration, an edge is added to the network that improves quality of the network maximally. Network quality can be measured by any popular scoring metric for Bayesian networks, such as the Bayesian Dirichlet metric with likelihood equivalence (BDe) (Cooper & Herskovits, 1992; Heckerman, Geiger, & Chickering, 1994) or the Bayesian information criterion (BIC) (Schwarz, 1978). Learning terminates when no more improvement is possible.

The sampling of a Bayesian network can be done using probabilistic logic sampling (PLS) (Henrion, 1988). In PLS the variables are ordered topologically so that it is assured that every variable is preceded by all parental variables it depends on. Variable values are then generated iteratively according to the topological ordering. As a result, once the value of a variable x_i is to be generated, the values of 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.

Despite the encoded independence assumptions in a Bayesian network, identified dependencies may also contain regularities. Furthermore, the exponential growth of full CPTs with respect to the number of parents often obstructs the creation of models that are both accurate and efficient. Thus, often local structures are used in Bayesian networks to represent local conditional probabilities more efficiently than traditional full BNs (Chickering, Heckerman, & Meek, 1997; Friedman & Goldszmidt, 1999).

Pelikan (2002) uses decision trees to store the conditional probabilities of each variable in a separate 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.

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 to 2^k leaves, each corresponding to a unique condition. However, a decision tree enables the more efficient and flexible representation of local conditional distributions. See Figure 7.4b for an example decision tree modeling the conditional probability table presented earlier.

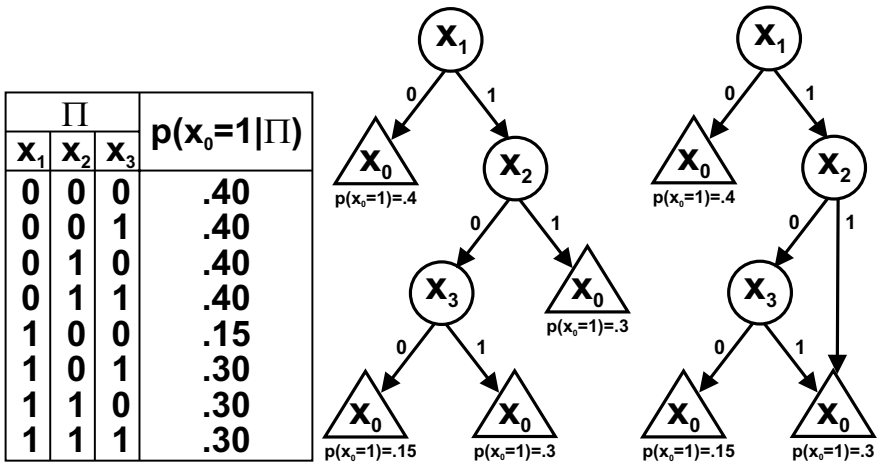

Fig. 7.4. A conditional probability distribution representation for $p(x_0|x_1, x_2, x_3)$ using a full-blown conditional probability table (a), as well as a decision tree (b) and a decision graph (c).

Pelikan (2002) uses also the (acyclic) decision graph feature allowing more edges to terminate in a single node, enabling the sharing of children by several internal nodes. This makes the representation even more flexible and allows even more compact dependency structure representations. Figure 7.4c shows an exemplar decision graph.

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 $x_i = 1$. In each iteration, each leaf of each decision tree is split (as long as a topological ordering remains possible) determining the quality change of the current network measured by the applied metric. The best split is performed. Learning stops when no potential split is able to improve the current network.

To estimate model quality, a combination of the BDe (Cooper & Herskovits, 1992; Heckerman, Geiger, & Chickering, 1994) and BIC (Schwarz, 1978) metrics is used, where the BDe score is penalized with the number of bits required to encode parameters (Pelikan, 2002). For decision graphs, a merge operation is introduced to allow merging two leaves in any (single) decision graph. The Bayesian decision graph mechanism applied to XCS is taken from Pelikan's BOA implementation available on the net (Pelikan, 2001)

Similar to the ECGA approach, Bayesian networks model dependencies and independencies where a dependency is defined as a *non-linearity*. This non-linearity is identified by an entropy decrease as measured in the applied metric. Applying the introduced decision graph structure focuses the modeled

dependencies in one decision graph on one feature modeling local, lower-level dependency structures, that is, the expected BBs in the problem.

7.2.4 Learning Dependency Structures in XCS

Similar to the BB-identification mechanisms in ECGA and BOA, which search BBs in the current best subset of individuals, it is possible to learn dependency structures from the current population in XCS. However, two aspects need to be considered. (1) Selection from the global population is not straight-forward. (2) Classifiers need to be suitably transferred into binary.

As in ECGA and BOA, the dependency structure needs to be built from the current best individuals in the population of XCS. Despite the fitness sharing in XCS, relative accuracy may not be the appropriate measure since different problem niches may currently exhibit different learning stages so that the fitness may be misleading, potentially favoring already converged subspaces. However, it is possible 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}, a minimum numerosity θ_{bn}, and a maximum error $\theta_{b\varepsilon}$. The parameters were set to $\theta_{be} = 20$, $\theta_{bn} = 1$, $\theta_{b\varepsilon} = 400$ throughout the subsequent experiments filtering out the young and high-error classifiers. 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.

Given a filtered population, how to translate the classifier population into a suitable representation to build the model needs to be clarified. Don't care symbols may be simply coded by a third symbol in a ternary alphabet. However, don't care symbols do have a special meaning in that they match zero or one. Thus, to simplify model-building, we decided to code each condition attribute by two bits: The first bit encodes if the condition 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. Finally, 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 7.4 shows a set of classifiers and the corresponding encoding that is used to learn the Bayesian network with decision graphs.

With a binary coded set of individuals at hand, we are able to learn the BB structure via the MDL-metric of the ECGA or the Bayesian decision graph structure via the Bayesian-network learning algorithm. Note that since XCS applies a steady-state niche GA, the dependency structure does not need to be rebuilt every time step. We rebuild the network after a fixed number of

Table 7.4. 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	R	ε	binary encoding
##11##	1	750	375	10 11 01 01 11 10 1
##00##	1	250	375	11 11 00 00 10 11 0
0#1###	0	250	375	00 01 11 11 11 10 1
0#0###	0	750	375	00 00 10 11 10 10 0
0#11##	1	1000	0	00 11 01 01 11 10 1
0#11##	0	1000	0	00 11 01 01 11 11 0
001###	1	1000	0	00 00 01 10 10 10 1
10##0#	0	1000	0	01 00 11 11 00 10 0
000###	1	0	0	00 00 00 11 10 10 0
01#1##	0	0	0	00 01 11 01 11 11 1

time steps θ_{bs}, usually set to $10,000$ in our experiments. The threshold is only slightly problem dependent and does not appear to have a strong impact on performance. 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.

7.2.5 Sampling from the Learned Dependency Structures

As shown in Section 7.1, the recombination of the parents using common crossover operators may lead to disruptive effects potentially destroying important BB-structure. Once the dependency model is learned, XCS may use the model to recombine or directly generate offspring classifiers more effectively. As long as the learned model reflects important dependency structures, it can be expected that the resulting recombination is less disruptive and much more directed towards generating offspring that combines already successfully learned substructures effectively searching in the neighborhoods defined by the substructures.

As investigated in detail in the previous chapters, XCS generates offspring from parental classifiers selected from the current action set. This means that XCS reproduces classifiers that encode solutions with respect to the current problem instance. When using the globally learned dependency structures to generate offspring, we consequently need to adjust the structures to be able to generate local offspring. We investigate the following two options: (1) sampling classifiers using the model updated to the local probability distribution; and (2) probabilistically improving the selected parental offspring classifier using the model with global or local probabilities.

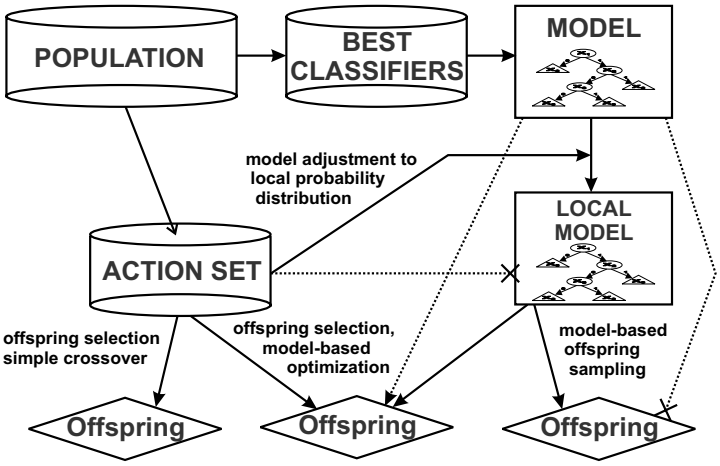

Fig. 7.5. A probabilistic model of the problem structure can be built and used for offspring generation in many ways. Since action sets are usually too small to gather reliable statistics, a selected classifier subset from the global population should reflect problem structure most effectively. Once 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.

Figure 7.5 shows the different potential methods for offspring generation by the means of a dependency model structure. Since XCS generates offspring in local niches, reproducing classifiers simply sampling from the global model is impossible since the classifier cannot be expected to reflect the solution structure in the current niche. Similarly, optimizing classifier structure by the means of the global model with global probabilities is expected to be disruptive as well since the optimization biased on the global probability structure again generates a classifier that reflects the global probability distribution. Even if the global model represents the dependencies in the population optimally, it may not be used to directly sample local offspring since it can only be expected to code lower-level BB information. Higher-level BB dependencies depend on the problem niche under investigation. Thus, it appears ineffective and very hard to grasp these higher-level dependencies in the global model.

Both offspring generation methods are introduced next. The latter is used only in conjunction with the learned Bayesian networks.

Sampling using Local Probabilities

As shown in Chapter 5, reproducing classifiers in action sets yields classifier offspring that has an average specificity that corresponds to the average specificity distribution in the action sets. Fitness may increase the average specificity due its pressure towards higher accuracy, which often leads to an implicit specialization pressure.

Thus, to sample offspring using the learned dependency structure, the model probabilities need to reflect the local specificity distribution. Consequently, we update the probabilities in the applied model with respect to the best classifiers in the current action set. To achieve this, we select a subset of classifiers from the action set using the tournament selection mechanism. The selected subset (which may contain identical classifiers several times selecting with replacement) is used to update the (conditional) probabilities. The updated dependency structure consequently reflects the detected global dependency structure but mimics the local probability distribution. The globally detected dependencies are thus combined with the local probabilities resulting in an offspring sampling mechanism that combines global with local problem knowledge.

The sampling is then achieved using the sampling mechanisms in the dependency structures explained above. Effectively, we use the globally detected dependency structures to sample local offspring. Hereby, the globally extracted structure biases the recombination. The current local probability distribution is used to sample offspring locally using the global structure. As a result, we recombine the locally important BBs effectively as long as the global dependency structure applies in the local problem niche.

Structure Optimization

In the former case, we combined global model information about dependencies with the local probability distribution. Vice versa, it is also possible to use the global dependency structure and probability distribution to optimize the local probability structure. Hereby we have to be cautious to not overwrite the local information completely by the global information.

Essentially we apply a Markov Chain Monte Carlo (MCMC) approach (Neal, 1993) first introduced in the statistical physics literature in the 1950s in the so-called *Metropolis Algorithm*. An MCMC essentially iteratively and probabilistically changes a current probability distribution to an equilibrium distribution. MCMC iteratively evaluates potential changes of single attributes and decides probabilistically if the change should be made. Which probabilities are chosen to confirm an update is application dependent. Our method uses the likelihood of the change with respect to the global model.

Particularly, XCS chooses an offspring via tournament selection in the current action set. Instead of simple 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 determining the likelihood of the structure before and after the change. To avoid zero likelihoods, all conditional probabilities are linearly normalized to values ranging from 0.05 to 0.95. Normalizing the likelihood before and after the change to one, the change is then committed with the probability of the normalized likelihood.

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 not be useful since the resulting classifier will reflect the global model structure. On the other hand, too few updates will have no effect 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, adjusting the dependency structure to reflect the local probability distribution using the consequent structure to probabilistically optimize a selected local offspring classifier. This has the advantage of avoiding the problem of over-optimization towards the global structure. Additionally, the freedom of sampling local offspring is further constrained since a parental offspring classifier is optimized, constraining the search to an actual parental classifier. In effect, the combination might be the most cautious but also the most robust offspring optimization mechanism overall.

7.2.6 Experimental Evaluation

We evaluate XCS's performance on the above introduced hierarchical problems evaluating and comparing both offspring generation methods applying several different settings. XCS is set to learn a Bayesian network every 10,000 learning steps ($\theta_{bs} = 10,000$). The population XCS learns from is 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, as before when simple crossover was applied. The results are averaged over ten experiments. Other parameters are set as above except for the population size which is set to $N = 20,000$ in all runs as well as mutation which is set to $\mu = 0.01$ in the runs with normal crossover operators, to $\mu = 0.001$ in the XCS/BOA combination, and to either value as indicated in the subsequent figures in the XCS/ECGA combination. This population size seems large but the investigated problems are huge as well. For example, the hierarchical 3-parity, 6-multiplexer problem requires a final solution of 2^{10} classifiers so that the only twenty times larger population size appears reasonable.

Hierarchical Parity, Multiplexer Problems

In Section 7.1 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 is chosen, effective BB recombination is mandatory and was achieved by an informed BB-wise crossover operator. If mutation is large, the problem was solvable but took longer than with the application of the informed crossover operator. 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 challenge as well as the reproductive opportunity bound (see Chapter 6). Thus, to solve the addressed problem, mutation needs to be set low and crossover needs to be effective.

Figure 7.6 shows XCS's performance in the hierarchical 3-parity, 6-multiplexer problem. In the ECGA comparison (Figure 7.6 a,b), we see that while BB-wise crossover learns the model slightly faster, ECGA reaches similar performance. The different settings refer to the number of selected classifiers used to adjust the model to the local probability distribution. Higher mutation rates are actually somewhat disruptive as also indicated by the resulting higher population sizes.

An additional specializing effect is observable, which is partially a result of the binary recoding 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, consequently increasing diversity in the population. This additional diversity may be slightly disruptive as observed in the ECGA graphs. 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. No learning was observable in this case.

The application of the Bayesian model results in a similarly successful solution of the 3-parity, 6-multiplexer problem. The probabilistic optimization method even reaches higher performance slightly faster than the informed BB-wise crossover application (curve BOA: 0/18). However, if too many optimization steps are applied (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. This problem does not occur when offspring is sampled using the local probability distribution or if selected offspring is optimized using the local probability distribution. All settings exhibit similar performance nearly as good as the informed BB-wise crossover technique. In contrast to the ECGA combination, the BOA combination does not suffer from any over-specializations and all runs reach 100% performance reliably.

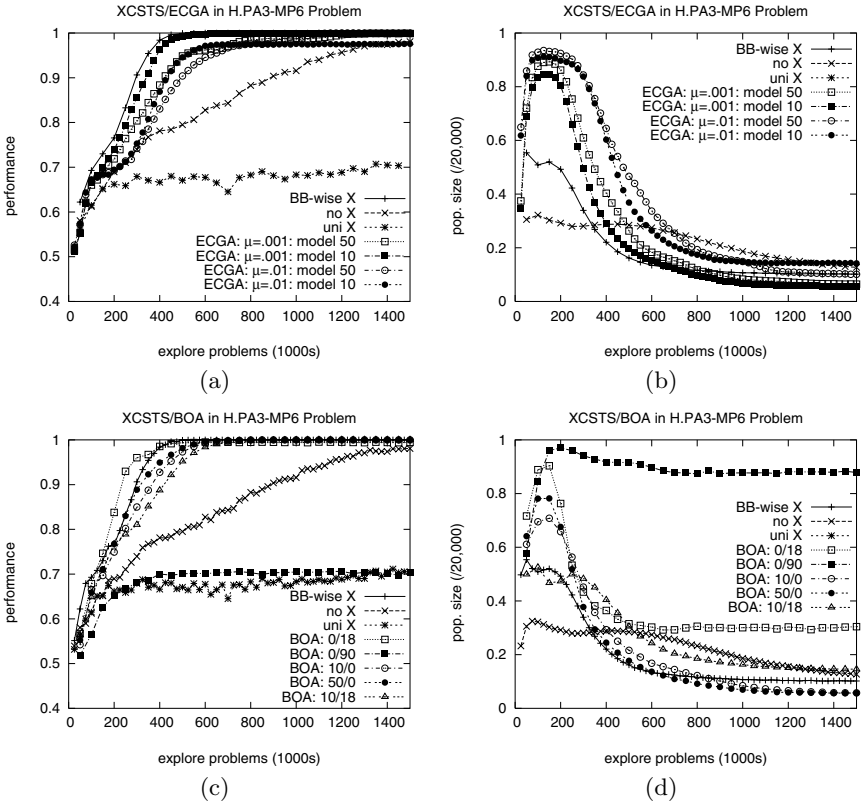

Fig. 7.6. When applying ECGA or BOA to the hierarchical 3-parity, 6-multiplexer problem, recombination becomes effective and XCS is able to effectively learn a complete solution comparatively fast to the runs with BB-wise crossover. The application of the ECGA-based model learning mechanism (a,b) shows competent performance. The 50/10 variation refers to the number of selected classifiers used to set the probabilities to the local probability distribution. In the BOA-based model learning (c,d), the first number again 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).

Hierarchical Parity, Count Ones Problem

Also the investigated hierarchical 3-parity, 5-count ones problem requires a final optimal solution size of 2^{10}. However, in this case the final population size is overlapping in that three out of five parity blocks need to be specified correctly to predict class zero or one accurately. XCS with ECGA model does not show any problems in solving the problem (Figure 7.7a,b). All runs

converge to the near-optimal solution nearly as fast as the informed BB-wise crossover runs. Even with a lower mutation rate, performance is hardly influenced.

Similarly, the XCS runs with Bayesian network successfully learn the problem (Figure 7.7c,d). BOA learns slightly slower than the ECGA combination early in the run but then reaches a slightly higher performance level. Apparently, BOA initially models spurious dependencies that may slow down the overall learning process. Since in this problem the propagation of all five BBs independently is nearly most effective, the Bayesian learning algorithm appears to over-model and thus delay learning early on. In the end of a run BOA decreases disruptive effects. Performance of both methods clearly outperforms the runs without crossover application as well as the runs with uniform crossover application.

In sum, 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 showed to be able to 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 operators dependent on the encountered problem. The next chapter provides further evidence for the generality of the model-building approach in XCS applying the techniques to several other typical Boolean function problems.

7.3 Summary and Conclusions

This chapter considered the last three aspects of the facetwise LCS theory approach in the XCS learning system for single-step RL problems. We showed that—as in GAs—the problem may require effective BB processing in LCSs to ensure reliable learning of a problem solution. Additionally, we highlighted the difference between LCSs and GAs in that problem structure may differ dependent on which problem subspace is currently under investigation. In essence, different attributes may be relevant in different problem structures so that different recombinatory mechanisms need to be applied dependent on the current problem subspace.

To investigate the recombinatory capabilities of the XCS system, we introduced hierarchical binary classification problems combining parity blocks on the lower level 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 (randomly distributed over the population). Mutation alone is able to solve the parity, multiplexer problem

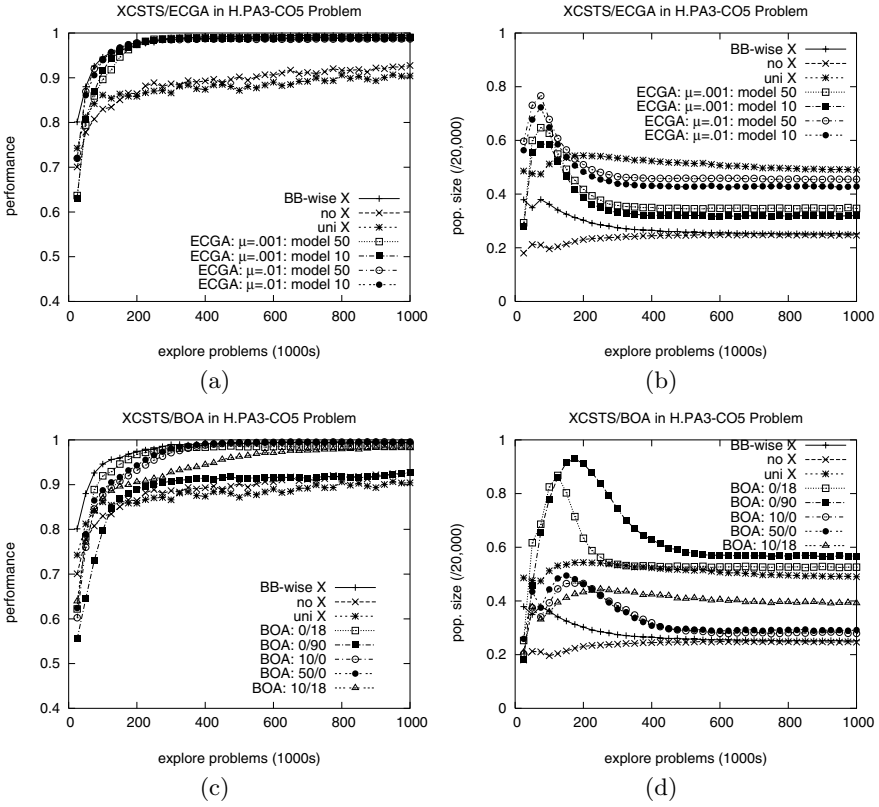

Fig. 7.7. Also the hierarchical 3-parity, 5-count ones problem is effectively solvable with either model-based offspring generation method. The ECGA combination appears slightly more robust in this case, indicating that the Bayesian-net might model unnecessary, spurious dependencies that delay convergence. Note also how over-specialization in the 0/90 setting again disrupts learning.

combination—albeit severely delayed in time—but it is not able to solve the parity, count ones problem combination satisfactorily in the available time.

The integration of the model-building and offspring generation mechanisms from the extended compact GA (ECGA) or the Bayesian model building algorithm (BOA) show that competent crossover operators can be integrated in the XCS learning structure. Since XCS applies a steady-state niche GA, the probabilistic model is not built every time step but it is built from the global model at predefined points in time. The model is then modified to reflect the local probability distribution in an action set at each time step to generate offspring respecting the local probability distribution but biasing recombination on the globally detected dependency structure. XCS combined with either model learner evolved complete solutions to the hierarchical problems.

With respect to the final points of our facetwise theory approach for LCSs we showed that XCS respects the difference between global and local problem structure by reproducing in action sets. Recombination can be made more efficient by using global BB structure information. However, since the local problem structure needs to be respected but usually cannot be coded in the global dependency structure, offspring recombination needs to be adapted to the current local problem niche. Since knowledge about the current niche is represented in the local classifier population, that is, the current action set, model-based offspring generation needs to be biased on the classifier distribution in the current action set.

In conclusion, XCS with the integration of either model learner may be termed a *competent LCS*. That is, it is able to solve boundedly difficult problems—those with a minimal order complexity of k_m—effectively. The next chapter provides further evidence for the generality of the model-learning integration, investigating XCS's performance in several Boolean function problems including noisy and very large problems.