Knowledge Extraction and Problem Structure Identification in XCS

Martin V. Butz[1], Pier Luca Lanzi[2], Xavier Llorà[1], and David E. Goldberg[1]

[1] Illinois Genetic Algorithms Laboratory (IlliGAL)
University of Illinois at Urbana-Champaign
Urbana, IL, 61801
{butz,xllora,deg}@illigal.ge.uiuc.edu
http://www-illigal.ge.uiuc.edu
[2] Artificial Intelligence and Robotics Laboratory
Dipartimento di Elettronica e Informazione
Politecnico di Milano
Milano 20133, Italy
pierluca.lanzi@polimi.it
http://www.elet.polimi.it/index.jsp

Abstract. XCS has been shown to solve hard problems in a machine-learning competitive way. Recent theoretical advancements show that the system can scale-up polynomially in the problem complexity and problem size given the problem is a k-DNF with certain properties. This paper addresses two major issues in XCS: (1) knowledge extraction and (2) structure identification. Knowledge extraction addresses the issue of mining problem knowledge from the final solution developed by XCS. The goal is to identify most important features in the problem and the dependencies among those features. The extracted knowledge may not only be used for further data mining, but may actually be re-fed into the system giving it further competence in solving problems in which dependent features, that is, building blocks, need to be processed effectively. This paper proposes to extract a feature dependency tree out of the developed rule-based problem representation of XCS. The investigations herein focus on Boolean function problems. The extension to nominal and real-valued features is discussed.

1 Introduction

Despite the original proposal of the schema notion by John Holland about three decades ago [1], learning classifier system (LCS) research has hardly addressed the importance of effective schemata processing nor was the effectiveness of the crossover operator addressed extensively. More recent research on genetic algorithms (GAs) suggests that in many problems—namely deceptive problems of bounded difficulty—identification and effective processing of interdependent substructures or schemata, the so called *building blocks* (BBs), is crucial for the success of the applied genetic algorithm [2, 3].

The XCS classifier system, proposed by Wilson in 1995 [4], may be the most well-studied LCS to date. It has been shown that the system is able to solve typical data mining problems machine learning competitively [5–7]. Several theoretical advancements have been made recently bounding population size to ensure learning success with high probability [8, 9]. Additionally, Wilson's original proposition that XCS may scale polynomially in problem size and problem complexity [10] was confirmed for boundedly difficult $k - DNF$ problems [11]. However, the analysis models learning time as a step-by-step process solely dependent on mutation. Also in XCS, effective BB identification or processing has hardly been addressed. This paper takes a step towards BB identification in XCS using a tree-construction mechanisms that identifies feature importance and interdependencies.

On the other hand, knowledge extraction was recently addressed in the XCS classifier system. Wilson [12] showed that most relevant rules can be extracted from the integer-based XCS system XCSI reducing the rule set by over 95% only marginally degrading performance. Dixon et. al. [13] further analyzed and enhanced the algorithm improving the complexity requirements. Both systems are data-driven in that they use the available training data set to reduce the final problem representation in XCS.

The algorithm proposed herein is not data driven—as the ones mentioned above—but solely considers the classifier population of XCS. Similar approaches for knowledge extraction in populations have already been used successfully for function optimization using genetic algorithms [14]. The algorithm extracts problem knowledge using a heuristic, specificity-based approach. For now, the proposed algorithm is restricted to the binary problem domain. We show that the algorithm does not reduce the accuracy of the final problem knowledge significantly. The resulting tree structure is easily readable and emphasizes the significance and dependency structure of each feature of the problem instances. Additionally, we show that the system clearly identifies the relevant interdependent features even early in a run making the algorithm a valuable candidate for BB identification and processing.

The paper is structured as follows. The next section provides a quick overview over XCS problem representation and learning. Next, we introduce the tree generation algorithm. The algorithm is applied on several typical Boolean function problems revealing its knowledge extraction capabilities as well as its BB identification capabilities. Finally, summary and conclusions are provided.

2 XCS Overview

The XCS classifier system was introduced in [4]. XCS is designed to solve classification problems as well as reinforcement learning problems. The system is learning while interacting online with an unknown problem.

XCS (as all other LCSs) represents the problem solution by a *population* of *classifiers*. At each time step, XCS receives a problem instance. Based on its current knowledge, XCS proposes a solution for the instance. Depending

on the problem instance and the solution proposed, XCS receives numerical reward characterizing the goodness of the proposed solution. XCS exploits the incoming reward signal applying an accuracy-based genetic algorithm (GA) to evolve a complete, maximally accurate, and maximally general representation of the optimal problem solution. Accordingly, the learning is biased towards learning a complete representation of the action-value function underlying a particular problem described according to the typical reinforcement learning framework. For a complete introduction to the XCS system, the interested reader is referred to [4, 10, 15]. The remainder of this section gives a short introduction to the mechanisms crucial to ensure understanding of the rest of this paper.

This paper addresses XCS's performance on Boolean functions. A problem instance is coded by a string of l bits and belongs to one of two classes. Knowledge in XCS is represented by a population of rules—very similar to a disjunctive normal form—where each rule specifies one conjunctive term. Essentially, the sought accurate, maximally general problem solution can be represented in disjunctive normal form. Additional to the conjunctive term C (that is, the condition), a rule, or *classifier*, specifies the corresponding binary class A, a prediction of the consequent reward p, the mean absolute deviation of this prediction ϵ, and the fitness F, which estimates the mean relative accuracy of the classifier.

Given a problem instance S, a *match set* $[M]$ is formed consisting of all classifiers in $[P]$ whose conditions match S. The match set $[M]$ essentially represents the knowledge about the current problem instance. $[M]$ is used to decide on the classification on the current problem forming fitness-weighted reward predictions of each possible classification effectively generating a prediction array with entries

$$P(a_i) = \frac{\sum_{cl_k \in [M]|_{a_i}} p_k \times F_k}{\sum_{cl_k \in [M]|_{a_i}} F_k}, \tag{1}$$

where $[M]|_{a_i}$ refers to the classifiers in $[M]$ that specify action a_i, p_k refers to the reward prediction, and F_k refers to the fitness of the k's classifier in $[M]|_{a_i}$. After the execution of the chosen classification A, and the resulting reward R, an *action set* $[A]$ is formed consisting of all classifiers in $[M]$ that specify the chosen action A. The reinforcement learning component, which updates classifier parameters, is applied in accordance to R. The genetic algorithms reproduces, mutates, and crosses high-fitness rules in $[A]$ and deletes inaccurate and well-supported rules in the population. Thus, the genetic algorithm is designed to evolve maximally accurate, maximally general classifiers.

3 Tree Generation of XCS Knowledge

As mentioned above, XCS is designed to evolve a complete, accurate, and maximally general solution to the provided problem. The solution is represented by a population of classifiers. Previous approaches tried to derive problem knowledge directly out of this rule-based solution representation. However, due to the continuous application of mutation and crossover, despite the generalization

Table 1. Description of the recursive tree generation algorithm.

```
GENERATE_TREE([S], Features):
 1 PA ← GENERATE_PREDICTION_ARRAY([S])
 2 Class ← arg_max(PA)
 3 Feature ← GET_NEXT_FEATURE_TO_SPLIT([S], Features)
 4 if(Feature is null)
 5    return Leaf(Class)
 6 Features ← Features ∪ Feature
 7 SubTree(0) ← GENERATE_TREE([S.Feature=0], Features)
 8 SubTree(1) ← GENERATE_TREE([S.Feature=1], Features)
 9 SubTree(2) ← GENERATE_TREE([S.Feature=#], Features)
10 return Node(Class, [S], Feature, SubTree(0), SubTree(1), SubTree(2))
```

pressure in XCS [4, 8], the final population of XCS may still be rather large. In his original work, Wilson [4] proposed a condensation mechanism that reduces mutation and crossover rates once accuracy is reached. However, it is often difficult to assess when maximal accuracy is reached (especially in noisy problems). Another approach focuses on extracting rules based on data coverage [12]. The shown results showed promising compression of knowledge and the possibility of generating rules of thumb to express the knowledge. However, the general inter-dependency of the features in the problem remains somewhat obscured. Additionally, the data-dependence of the approaches seems somewhat unsatisfactory.

Our approach focuses on an explicit modeling of the feature dependencies using the current population of classifiers—the same approach used in evolutionary algorithms such as SI3E [14]. The algorithm recursively chooses the most specialized attribute of the available features in the classifier conditions and generates a node specifying the feature. Thus, specificity is used as an indicator of problem- or classification-significance of an attribute. The classifier set is then split into three subsets: the subsets of classifiers that code the chosen attribute as *zero*, *one*, and *don't care*. Next, the algorithm is called recursively for each of those subsets. If there is only one type of classification left in a subset of classifiers, or there is no further node found (because all nodes were split or because all non-split nodes have only don't care symbols), a leaf is created that specifies the class of the subtree choosing the maximum classification in the prediction array of the current subset as the class.

A formal algorithmic description of the recursive algorithm is provided in Table 1. `[S]` refers to the current subset of classifiers, `Features` is a set that includes all features which were already used in this subtree. The procedure `GENERATE_PREDICTION_ARRAY` returns the prediction array as specified in Equation 1 and the procedure `GET_NEXT_FEATURE_TO_SPLIT` returns the next most specific feature. A node contains information about its class (i.e., the class with the highest prediction array), its support set of classifiers, the feature it splits the support set on, and the three sub-trees which might be nodes and/or leafs.

To avoid the influence of young or unreliable classifiers, we filter the classifier population. Specifically, we generate a filtered classifier set that only contains

classifiers that are experienced (they were evaluated more than θ_{exp} times), have a support of at least two ($num \geq 2$ consequently requiring at least one successful selection and reproduction), and have error smaller than 10_0. Additionally, we revert the class of all classifiers that satisfy the above constraints and have a reward prediction below 500. This is possible since there are only two problem classes and a classifier that essentially predicts zero reward predicts the other class (closed-world assumption).

As can be inferred from the algorithm, the complexity is quadratic in the number of features and linear in the population size $O(l^2N)$. The maximum depth of the tree equals l (when splitting on all attributes), and on each level in the tree the whole population is searched for the current most specific attributes (excluding the attributes already split on). The size of the tree is bounded by $O(lN)$ since each level in the tree needs to maximally store the whole population once (each node in a level stores a part of the whole population). Since Wilson's and Dixon's algorithms depend on the data size, the comparison is difficult. However, note that N itself grows in l as well as in the complexity of the problem [11] and is generally much larger than l and thus dominates the factor l^2. Since also learning time grows in l as well as in problem complexity [16], tree generation is a minor computational effort compared to the learning process.

The resulting tree can also be used as a classifier by itself. However, a slight problem shows up with respect to the don't care branch. Given a problem instance and a node that splits on a particular attribute, should the classification algorithm descent the specified attribute path or the don't care path? Our algorithm makes this decision by counting the support of classifiers in both sub-paths. The branch is chosen in which more classifiers of the corresponding support set match the given problem instance. If a node has less than five (micro-) classifier representatives, the class of this node is chosen as the classification.

The next section will elaborate on the proposed algorithm investigating structure and accuracy of the resulting tree.

4 Structure Extraction

This section investigates how well the generated tree can be used to visualize problem structure as well as how compact the resulting representation is. To do this, we investigate three Boolean function problems: (1) the multiplexer problem [4], (2) a carry problem, and (3) a hierarchical parity-multiplexer problem. The three problems are non-overlapping (accurate, maximally general sub-solutions are non-overlapping), overlapping, unbalanced (accurate, maximally general sub-solutions have a different number of attributes specified), and hierarchically structured, respectively.

4.1 Multiplexer Problem

The multiplexer is a Boolean function problem widely studied in LCS research [4, 17, 5]. The problem is defined for binary strings of length $k + 2^k$. The output

is determined by the bit located at the position referred to by the binary value of k *position bits*.

Figure 2(a) shows the resulting tree of the 11-multiplexer problem running the problem for 50,000 problem instances to assure complete convergence. It can be inferred that the tree represents the problem structure perfectly when descending the zero and one branches only. The don't care branches may be regarded as irrelevant but actually convey additional information. Descending down the don't care path on the most left `2=0` node, we can see that XCS also "knows" that if attribute two is ignored but attribute four and six are zero, the output will be zero.

A statistical analysis of 100 runs in the multiplexer problem showed that the conversion to the tree representation does not decrease the accuracy of the classification (all 100 runs showed equal 100% performance). Even when decreasing the population size to $N = 1500$ and the number of learning steps to $30,000$, at which point the accuracy of the knowledge is at a level of about 90%, no significant statistical difference was found between the filtered population and the induced tree—paired t-test and Wilcoxon rank sum test present confidence values greater than 99%. Moreover, results show that the tree removes noisy spurious interactions, providing a stable description of the evolved knowledge in the population.

4.2 Carry Problem

The carry problem is defined by two equally long sub-strings which are added together. If the result has a carry, then the output is one otherwise the output is zero. The different conjunctions that form the solution are (1) overlapping, and (2) do not have the same order. More general conjunctions will match more often and consequently are expected to be expressed by a larger number of classifiers.

The tree generated out of the final population of XCS in the (2,2)-carry problem accurately identified classes of all sub-problems (not shown). In this case, no spurious nodes can be found. Statistical analysis—using the previously introduced t-test and Wilcoxon—confirmed the robustness of the tree generation. Averaged over 100 experiments, there were no statistically significant differences detected between the classification accuracy of the filtered population and the tree with confidence values greater than 99.5%

4.3 Building Block Identification

As the final problem, we construct another Boolean function problem that requires a competent crossover operators to learn efficiently [18]. The problem is structured hierarchically in that the evaluation is pursued in two stages. We combine a parity problem on the lower level with a multiplexer problem on the higher level. The result of the lower level parity function results in a shorter bit string which is then evaluated by the higher level function multiplexer. For example, consider the hierarchical 2-parity, 3-multiplexer problem. The problem is six bits long. On the lower level, blocks of two bits are evaluated by the parity

function, that is, if there are an even number of ones, the result will be zero and one otherwise. The result is a string of three bits that is then evaluated by the 3-multiplexer function. The result is the class of the problem instance. For example, the hierarchical 2-parity, 3-multiplexer problem can be written in disjunctive normal form as follows:

$$\texttt{2-PA,3-MP}(x_1, x_2, x_3, x_4, x_5, x_6) = x_1x_2\neg x_3x_4 \vee x_1x_2x_3\neg x_4 \vee \neg x_1\neg x_2\neg x_3x_4 \vee \neg x_1\neg x_2x_3\neg x_4 \vee \neg x_1x_2\neg x_5x_6 \vee \neg x_1x_2x_5\neg x_6 \vee x_1\neg x_2\neg x_5x_6 \vee x_1\neg x_2x_5\neg x_6 \quad (2)$$

Figure 1 show a graphical example of how this problem is extended to a hierarchical 3-parity 6-multiplexer. In [18], we showed that larger instances of this hierarchical problem are very difficult to solve for XCS. Crossover seems to continuously disrupt the found lower-level building blocks so that effective processing of the blocks is impossible. However, we also showed that when applying building-block wise uniform crossover (effectively preventing building block disruption), XCS is able to solve the problem. Thus, for larger hierarchical problems, an operator is necessary to identify and process building blocks.

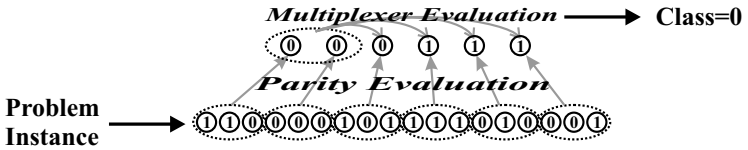


Fig. 1. Combined 3-Parity / 6-Multiplexer Problem

Figure 2(b) shows the tree generated out of the filtered, converged population in the hierarchical 2-parity, 3-multiplexer problem (that is still solvable with XCS and uniform crossover). Clearly, the tree identifies the dependency of the two parity blocks. Note that the *don't care* branch considers the other two parities and correctly specifies the function outcome in most of those cases. However, since the coverage of the left branches is higher, the hash-branch won't be used for classification. There is also a spurious node (node `3=0` on the left lower side of the tree) that does not influence performance because all branches correctly identify the class as zero.

Finally, we generated a tree in the hierarchical 3-parity, 6-multiplexer problem, which XCS is not able to solve in 500,000 steps even with a population size of 20,000 (performance is at around 65% at that point) when uniform crossover is applied or low mutation rate used. However, XCS with a BB-wise uniform crossover (explicitly preventing the disruption of the parity blocks, exchanging BBs uniformly randomly) is able to solve the problem with the provided resources. The question is if we are able to identify the parity blocks by the means of the tree generation mechanism. Figure 2(c) shows the tree (restricted to depth four and suppressing the hash branches on all levels except for at the root) generated out of the filtered population after 80,000 steps. Whereas XCS's performance at this point is at only 59%, parity blocks (13,14,15) and (7,8,9)

are identified by the tree generation mechanism. The tree generation mechanism seems to be able to extract lower-level problem structure that XCS learned albeit XCS is not able to process (that is, propagate and recombine) the lower level structure effectively. Thus, we hope to be able to extract the dependencies detected by the tree generation mechanism and propagate building blocks more effectively similar to the competent genetic algorithms such as the extended compact GA [19] or probabilistic model building GAs [20].

5 Conclusions

This paper has addressed two major issues in XCS: (1) knowledge extraction and (2) structure identification. Knowledge extraction addresses the issue of mining problem knowledge from the final solution developed by XCS. The goal was to identify most important features in the problem, as well as the dependencies among those features. Such behavior has two promising applications: (1) obtain a compact representation of the evolved knowledge, and (2) identify the *building blocks* of the problem. One main contribution of our approach is being population-driven instead of data-driven.

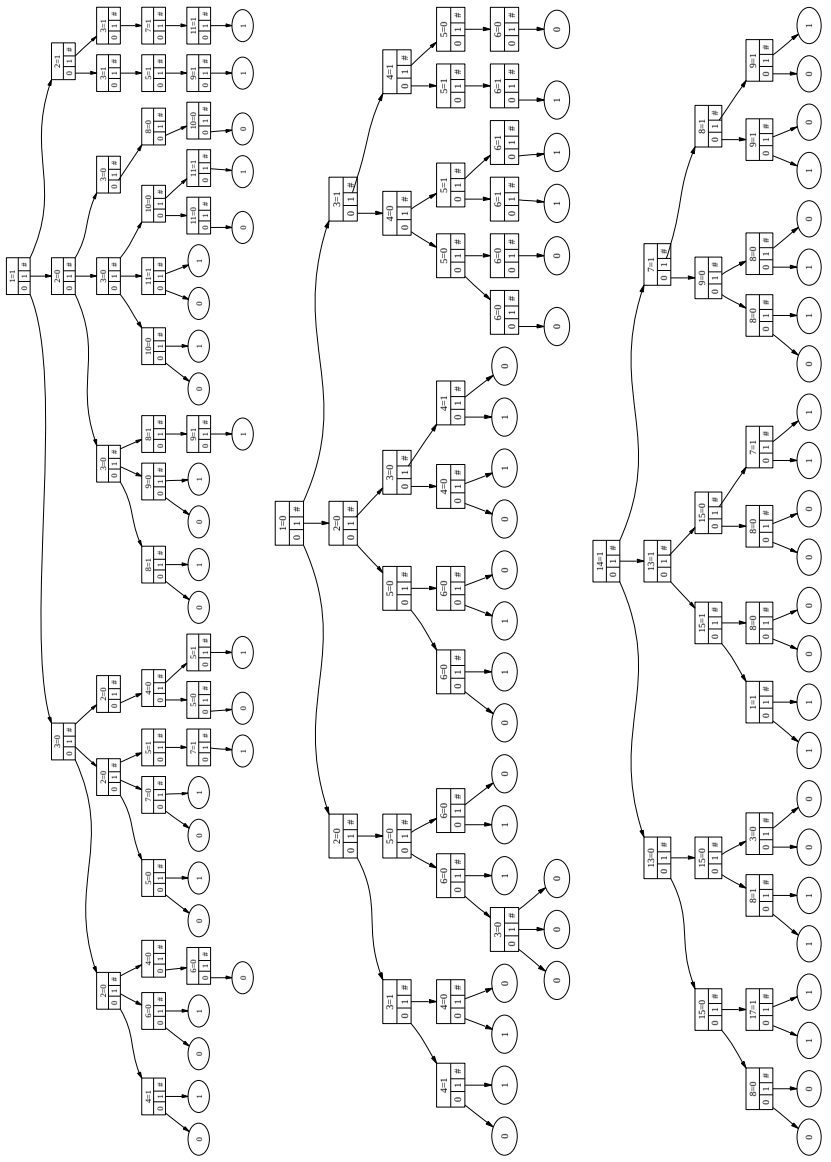
We have mainly focused on extracting a compact representation based on the knowledge evolved by XCS. Results show a remarkable ability to obtain compact descriptions out of the final population of classifiers without any significant degradations of the overall accuracy. However, our tree-based approach aims a much broader application: the creation of a first *building block* processing LCS. In order to achieve such purpose our current work is focusing on using the identified *building blocks* in XCS to create a first competent LCS.

Acknowledgments

This work was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF (F49620-03-1-0129), and by the Technology Research, Education, and Commercialization Center (TRECC), at University of Illinois at Urbana-Champaign, administered by the National Center for Supercomputing Applications (NCSA) and funded by the Office of Naval Research (N00014-01-1-0175).

References

1. Holland, J.H.: Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor, MI (1975) second edition 1992.
2. Goldberg, D.E.: The race, the hurdle and the sweet spot: Lessons from genetic algorithms for the automation of innovation and creativity. In Bentley, P., ed.: Evolutionary design by computers. Morgan Kaufmann, San Francisco, CA (1999) 105–118
3. Goldberg, D.E.: The Design of Innovation: Lessons from and for Competent Genetic Algorithms. Kluwer Academic Publishers, Boston, MA (2002)
4. Wilson, S.W.: Classifier fitness based on accuracy. Evolutionary Computation **3** (1995) 149–175



(a) 11-MP (b) H. 2-P./3-MP (c) H. 3-P./6-MP

Fig. 2. Each node in a tree specifies the attribute it splits on and the class of the node (attribute=class). The leafs specify the resulting class. **(a)** Tree for the 11-multiplexer problem. **(b)** Tree for the hierarchical 2-parity/3-multiplexer problem. **(c)** Upper part of the tree (plotting up to depth four, suppressing the don't care branches except for at the root) in the hierarchical 3-parity/6-multiplexer problem.

5. Bernadó, E., Llorà, X., Garrell, J.M.: XCS and GALE: A comparative study of two learning classifier systems and six other learning algorithms on classification tasks. In Lanzi, P.L., Stolzmann, W., Wilson, S.W., eds.: Advances in learning classifier systems: Fourth international workshop, IWLCS 2001 (LNAI 2321). Springer-Verlag, Berlin Heidelberg (2002) 115–132
6. Bernadó-Mansilla, E., Garrell-Guiu, J.M.: Accuracy-based learning classifier systems: Models, analysis, and applications to classification tasks. Evolutionary Computation **11** (2003) 209–238
7. Dixon, P.W., Corne, D.W., Oates, M.J.: A preliminary investigation of modified XCS as a generic data mining tool. In Lanzi, P.L., Stolzmann, W., Wilson, S.W., eds.: Advances in learning classifier systems: Fourth international workshop, IWLCS 2001 (LNAI 2321). Springer-Verlag, Berlin Heidelberg (2002) 133–150
8. Butz, M.V., Kovacs, T., Lanzi, P.L., Wilson, S.W.: Toward a theory of generalization and learning in XCS. IEEE Transactions on Evolutionary Computation **8** (2004) 28–46
9. Butz, M.V., Goldberg, D.E., Tharakunnel, K.: Analysis and improvement of fitness exploitation in XCS: Bounding models, tournament selection, and bilateral accuracy. Evolutionary Computation **11** (2003) 239–277
10. Wilson, S.W.: Generalization in the XCS classifier system. Genetic Programming 1998: Proceedings of the Third Annual Conference (1998) 665–674
11. Butz, M.V., Goldberg, D.E., Lanzi, P.L.: PAC Learning in XCS. IlliGAL report 2004011, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign (2004)
12. Wilson, S.W.: Compact rulesets from XCSI. In Lanzi, P.L., Stolzmann, W., Wilson, S.W., eds.: Advances in learning classifier systems: Fourth international workshop, IWLCS 2001 (LNAI 2321). Springer-Verlag, Berlin Heidelberg (2002) 196–208
13. Dixon, P.W., Corne, D.W., Oates, M.J.: A ruleset reduction algorithm for the xcs learning classifier system. In Lanzi, P.L., Stolzmann, W., Wilson, S.W., eds.: Proceedings of the Fifth International Workshop on Learning Classifier Systems, IWLCS 2002. Springer-Verlag, Berlin Heidelberg (in press)
14. Llorà, X., Goldberg, D.E.: Wise breeding ga via machine learning techniques for function optimization. Proceedings of the Fifth Genetic and Evolutionary Computation Conference (GECCO-2003) (2003) 1172–1183
15. Butz, M.V., Wilson, S.W.: An algorithmic description of XCS. Soft Computing **6** (2002) 144–153
16. Butz, M.V., Goldberg, D.E., Lanzi, P.L.: Bounding Learning Time in XCS. IlliGAL report 2004003, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign (2004)
17. De Jong, K.A., Spears, W.M.: Learning concept classification rules using genetic algorithms. IJCAI-91 Proceedings of the Twelfth International Conference on Artificial Intelligence (1991) 651–656
18. Butz, M.V., Goldberg, D.E.: Hierarchical Classification Problems Demand Effective Building Block Identification and Processing in LCSs. IlliGAL report 2004017, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign (2004)
19. Harik, G.: Linkage learning via probabilistic modeling in the ecga. IlliGAL report 99010, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign (1999)
20. Pelikan, M., Goldberg, D.E., Lobo, F.: A survey of optimization by building and using probabilistic models. Computational Optimization and Applications **21** (2002) 5–20