Information Sciences 198 (2012) 100–117

Contents lists available at SciVerse ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Effective search for Pittsburgh learning classifier systems via estimation of distribution algorithms

Jiadong Yang, Hua Xu *, Peifa Jia

State Key Laboratory of Intelligent Technology and Systems, Tsinghua National Laboratory for Information Science and Technology, Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China

ARTICLE INFO

Article history:
Received 13 October 2010
Received in revised form 10 January 2012
Accepted 25 February 2012
Available online 7 March 2012

Keywords:
Learning classifier system
Genetics-based machine learning
Estimation of distribution algorithm
Bayesian optimization algorithm
Evolutionary computation

ABSTRACT

Pittsburgh-style learning classifier systems (LCSs), in which an entire candidate solution is represented as a set of variable number of rules, combine supervised learning with genetic algorithms (GAs) to evolve rule-based classification models. It has been shown that standard crossover operators in GAs do not guarantee an effective evolutionary search in many sophisticated problems that contain strong interactions between features. In this paper, we propose a Pittsburgh-style learning classifier system based on the Bayesian optimization algorithm with the aim of improving the effectiveness and efficiency of the rule structure exploration. In the proposed method, classifiers are generated and recombined at two levels. At the lower level, single rules contained in classifiers are produced by sampling Bayesian networks which characterize the global statistical information extracted from the current promising rules in the search space. At the higher level, classifiers are recombined by rule-wise uniform crossover operators to keep the semantics of rules in each classifier. Experimental studies on both artificial and real world binary classification problems show that the proposed method converges faster while achieving solutions with the same or even higher accuracy compared with the original Pittsburgh-style LCSs.

1. Introduction

Learning classifier systems (LCSs) [18], which are a subset of genetics-based machine learning (GBML) systems, are machine learning paradigms that combine reinforcement learning [33] or supervised learning [6] with genetic algorithms (GAs) [18]. In LCSs, knowledge is represented as a set of condition-action-prediction rules (called the population of classifiers). Reinforcement learning or supervised learning works on the classifiers' predictions to evaluate the classifiers, and GAs are responsible for discovering potentially useful rules. Compared with the most common machine learning algorithms, it has been reported that LCSs can provide competitive results in a wide variety of real-world classification problems [28,5].

Since the rules in LCSs are improved iteratively by a search mechanism based on GAs, the performance of GAs is critical to the efficiency of the whole system. As the building block hypothesis suggests, GAs guide the evolution to better solutions and even the global optimum by means of variation operators. Those operators recombine low order highly-fit schemata, called building blocks (BBs), contained in the global optimum to form higher order ones. However, if BBs which are not contained in the global optimum, increase in frequency more rapidly than those are, GAs will suffer from higher computational cost before converging to the global optimum. Even worse, GAs will be misled away from the global optimum, instead of towards it

* Corresponding author.
E-mail address: xuhua@mail.tsinghua.edu.cn (H. Xu).

0020-0255/$ - see front matter © 2012 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.ins.2012.02.059

[14]. In the context of LCSs, it has also been pointed out that standard variation operators do not guarantee an effective evolutionary search in many sophisticated problems that contain strong interactions between BBs. Particularly, standard crossover operators can frequently disrupt important combinations of features and therefore give poor performance [9].

In order to identify BBs adaptively, a class of evolutionary algorithms, called estimation of distribution algorithms (EDAs), has been proposed recently [23,11]. Instead of relying on traditional variation operators of GAs (such as crossover and mutation) to evolve the population, EDAs build probabilistic models from promising individuals in order to identify key sub-structures of the underlying search problem, and sample the model to generate new candidate solutions. As one of the state-of-the-art EDAs, Bayesian optimization algorithm (BOA) [30,27] uses Bayesian networks [21] to capture the dependencies between the decision variables. It can solve the problems of bounded difficulty with nearly decomposable structure or overlapping dependency structure in a robust and scalable manner, which cannot be solved by GAs [29].

In this paper, we integrate BOA into Pittsburgh LCSs with the aim of improving the effectiveness and efficiency of the rule structure exploration. In the proposed method (called BOA-based Pittsburgh learning classifier system, pLCS_BOA), classifiers are generated and recombined at two levels. At the lower level, single rules contained in each classifier are produced by sampling Bayesian networks which characterizes the global statistical information extracted from the current promising rules in the search space. At the higher level, classifiers are recombined by rule-wise uniform crossover, which keeps the semantics of all rules in each classifier. We implement our method based on the GAssist system [2], which is the new generation of Pittsburgh-style LCSs. The proposed method is evaluated on both artificial and real world binary classification problems, respectively. The experimental results show that with the reduced number of both generations and executed match operators as well as less running time, the proposed method can generate equivalent or better solutions in performance compared with ones produces by the original Pittsburgh-style LCSs.

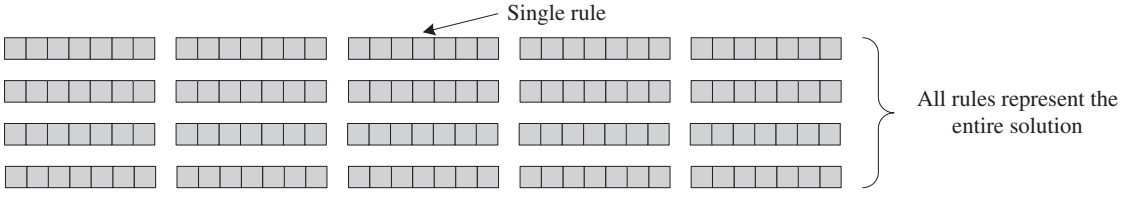
The remainder of this paper is organized as follows: in the next Section, background material is introduced. Section 3 describes the proposed method in detail. The experimental setup is illustrated in Section 4, followed by Section 5, which shows the experimental results on artificial and real world problems, respectively. Section 6 analyzes the experimental results and discusses the merit and potential of the proposed method. Finally, the paper is summarized in Section 7.

2. Related work

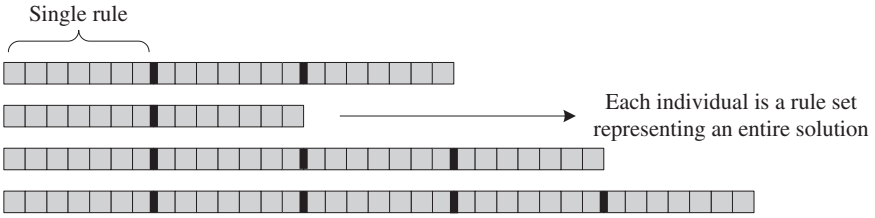
2.1. Learning classifier systems

As a subset of genetics-based machine learning systems (GBML) [13], learning classifier systems (LCSs) were first invented by Holland [18] in order to model the emergence of cognition based on adaptive mechanisms. LCSs consist of a set of rules called classifiers combined with adaptive mechanisms in charge of evolving the population of rules. Since Holland presented the first scheme of LCSs, research has been conducted from two perspectives: the Michigan style and the Pittsburgh style.

Michigan-style LCSs methods, initially defined as cognitive systems [19], combine reinforcement learning [33] with GAs to evolve populations of accurate rules. The population in Michigan-style LCSs is made up of a single rule set (see Fig. 1a), in which each rule acts in some parts of the problem domain. In order to achieve an ideal population, the two separate components in Michigan-style LCSs work with different goals. The reinforcement component exploits the incoming reward to



(a) In Michigan-style LCSs, all rules represent the entire solution.



(b) In Pittsburgh-style LCSs, each individual is a rule set representing an entire solution.

Fig. 1. Population in Michigan and Pittsburgh style LCSs.

estimate the action values in each subproblem so as to identify the best classifiers in the population. At the same time, GAs improve the current solution by means of exploring promising rules. In this family of LCSs, the XCS classifier system [7], originally introduced by Wilson in [34], is currently considered as the most popular Michigan-style LCSs.

Pittsburgh-style LCSs view learning as a form of optimization problem. Hence, it is a result directly extending GAs to deal with supervised learning problems. In this paradigm, each individual (i.e., classifier) is just a rule set, which consists of a variable number of rules to encode an entire candidate solution (see Fig. 1b). According to the fitness that considers different aspects such as the prediction accuracy and the generality, individuals in the population compete among each other and evolve following the typical cycle of GAs. At the end of the evolutionary process, the best individual found is treated as the final solution and used to predict the class of unknown examples. The first successful developments of Pittsburgh-style LCSs for supervised learning were GABIL [10] and GIL [20]. A new generation Pittsburgh-style LCS derived from GABIL can be found in GAssist system [2,4], which improves the generalization capacity of the model and designs representations for real-valued features.

2.2. Estimation of distribution algorithms

Estimation of distribution algorithms (EDAs) [23], also called probabilistic model building genetic algorithms (PMBGAs) [31], have been recently identified as a new paradigm in the field of evolutionary algorithms (EAs). Unlike traditional EAs that rely on traditional genetic operators (e.g., mutation or crossover), EDAs create offspring through sampling a probabilistic model learned from the set of promising solutions found so far. According to the statistical information they exploit, EDAs can be classified into the following two categories: univariate EDAs and multivariate EDAs. The former assumes that variables in a problem are independent from each other and, hence, only a product of independent univariate probabilities serves as the distribution of promising solutions. In the latter, conditional dependency chains or networks are taken into consideration with the aim of modeling multivariate interactions.

As one of the state-of-the-art multivariate EDAs, BOA [30,29] uses Bayesian networks [21] to capture the dependencies between the decision variables of the problem in order to generate a population of candidate solutions. After the initialization of a population at random with a uniform distribution over all possible solutions, the population is then updated for a number of generations. In each generation, BOA works as follows: (1) promising solutions are selected from the current population by means of a selection method, such as tournament or truncation selection; (2) a Bayesian network that models the population of promising solutions is constructed; (3) the offspring population is generated by sampling the built Bayesian network; (4) replacements are executed to incorporate the new solutions into the original population. The above four steps are repeated until certain termination criteria are satisfied. It has been pointed out that BOA can solve a broad class of nearly decomposable and hierarchical problems in a robust and scalable manner [29].

2.3. Block building process for LCSs via EDAs

Recently, one of many important advances in LCSs has been achieved by leveraging EDAs to extract important substructures of the problem to be solved. With machine learning or statistics techniques, LCSs can explore the search space more efficiently by adopting variation operators based on probabilistic models learned from the current population.

Within Michigan-style LCSs, Butz et al. [9] combined XCS with two EDAs paradigms, including the Extended Compact Genetic Algorithm (ECGA) [17] and BOA [29], to guarantee effective exploration for problems in which features strongly interact and standard variation operators lead to poor performance. Those methods can detect and propagate lower-level dependency structures effectively without any information about these structures given in advance.

The Compact Classifier System (CCS) [25] and its extension, called χeCCS [26], integrate EDAs into the framework of Pittsburgh LCSs. In the former, sets of perturbed probability vectors, which represent rules population, evolve through the Compact Genetic Algorithm (cGA) [15]. The latter evolves a population of rules via the model building and sampling mechanisms used in ECGA. Both methods aim at determining the minimum set of rules that creates a maximally general solution.

The works mentioned in this subsection are the ones most related to ours. However, there are obvious differences. First, our method focuses on Pittsburgh-style LCSs, while works proposed in [9] paid attention to Michigan-style LCSs. On the other hand, methods in [25,26] adopted cGA and ECGA to guide the combination of rules, while our method leverages BOA. More essentially, our method divides the combination of classifiers into two levels to improve the efficiency for exploring optimal solutions. Furthermore, our method is evaluated on both artificial problems with hierarchical building blocks and binary classification problems in real world, instead of only multiplexer problems handled in [25,26].

2.4. Rule recombination by means of memetic algorithms

In addition to relying on EDAs to improve the efficiency of exploration, another way to recombine rules more intelligently for LCSs is by employing memetic algorithms (MAs) [24,22], which are considered as hybridization of local and global search in EAs for solving difficult optimization problems efficiently. The rationale is that, the hybrid incorporates domain-specific information to augment the population-based search such as evolutionary algorithms with refinement components.

With respect to Michigan-style LCSs, Wyatt and Bull [35] showed that the application of MAs within rule learning can improve the performance of the system. In the family of Pittsburgh-style LCSs, a local search based smart crossover operator

for GAssist was presented in [3]. Such an operator takes more than two rules as parents and heuristically selects the minimum subset of rules with maximum accuracy over the training set. Furthermore, Bacardit and Krasnogor [4] proposed Memetic Pittsburgh Learning Classifier System (MPLCS). The authors systematically designed and evaluated several local search mechanisms in order to intelligently edit classification rules and recombine rule sets. Several strategies for integrating the local search mechanisms into the evolutionary cycle were investigated as well in [4]. It has been demonstrated that the local search mechanisms in MPLCS remarkably improve the classification accuracy and enable the system converge much faster.

3. The proposed method

The proposed algorithm is described in detail in this section. The basic procedure of Pittsburgh-style LCSs is listed in Algorithm 1. As discussed in Section 2.1, each individual (i.e., classifier) is represented as a rule set to encode an entire solution, in which the number of rules are not identical to each other. (In the remainder of this paper, the terms *individual* and *classifier* are used interchangeably.) Therefore, there is no cooperation among individuals and only competition is performed in the typical cycle of GAs. The main peculiarity of our method lies in lines 6 and 7 in Algorithm 1, in which evolution and evaluation of classifiers are performed at two levels based on BOA. One level, called rule-wise level, deals with rules in classifiers. And the other, called rule set-wise level, handles rule sets (i.e., classifiers). Fig. 2 shows the framework of our method. The main steps at the two levels are represented above and below the dashed line, respectively. In the remainder of this section, we first represent the design of classifiers in our method, then the evaluation and evolution of classifiers at the two levels will be discussed, respectively.

Algorithm 1. Basic procedure of Pittsburgh-style LCSs

```
1   t ← 0
2   P_t ← Generate a random population of rule sets
3   Evaluate rule sets in P_t
4   repeat
5     P′ ← Select promising rule sets in P_t
6     O ← Variation operator on P′
7     Evaluate the rule sets in O
8     P_{t+1} ← Replace P_t with O
9     t ← t + 1
10  until Stopping criterion is met
11  The best rule set is treated as the final solution
```

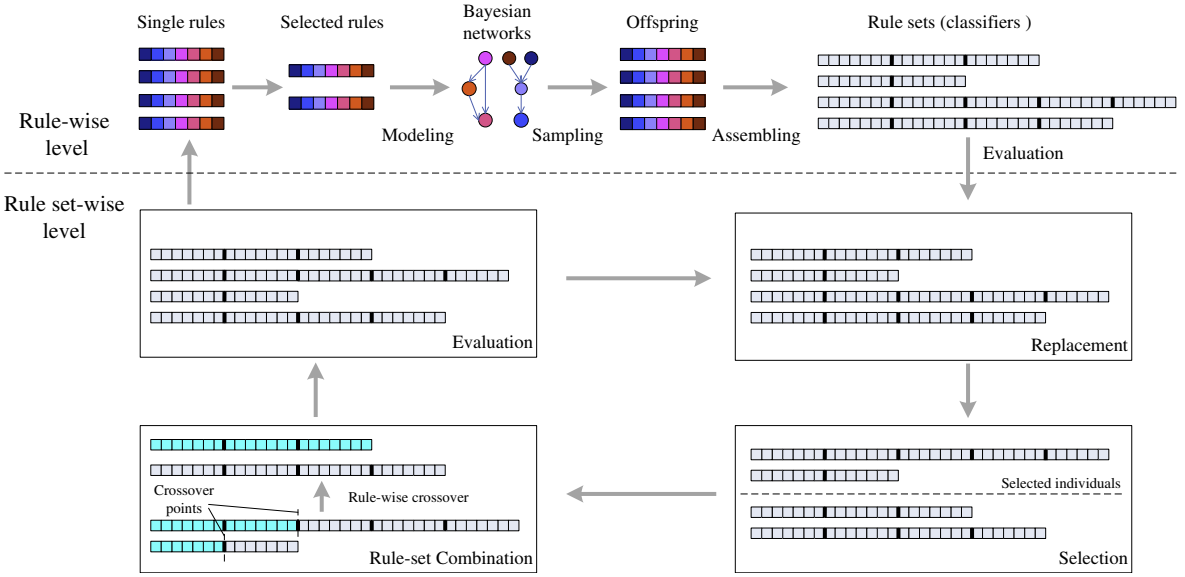


Fig. 2. In the proposed method, classifiers are recombined at two levels. The main steps at the rule-wise level are represented above the dashed line, where single rules contained in classifiers are produced by sampling Bayesian networks that characterize the global statistical information extracted from the current promising single rules in the search space, and then single rules are assembled together to form the rule sets in order to do prediction. The procedures at the rule set-wise level are depicted below the dashed line, which are similar to the classic GA cycle. The main peculiarity lies in the crossover operator, in which classifiers are recombined by rule-wise uniform crossover operators to keep the semantics of rules in each classifier.

3.1. Classifiers design

In this paper, we use the method in GABIL [10] and GAssist [2] to represent an entire solution of the problem. In the method each entire solution is embedded in a single individual. Formally, each individual is a variable-length disjunctive normal form of rules:

$$I = R_1 \vee R_2 \vee \cdots \vee R_n \tag{1}$$

where each rule, e.g., R_i, includes a condition part and a corresponding action. The former identifies to which input states the rule is applicable, and the latter specifies the proposed solution (e.g., classification) when its condition is satisfied. The single rule R_i is defined as:

$$\left(V_1^1 \vee \cdots \vee V_{k_1}^1\right) \wedge \left(V_1^2 \vee \cdots \vee V_{k_2}^2\right) \wedge \cdots \wedge \left(V_1^n \vee \cdots \vee V_{k_n}^n\right) \rightarrow \text{classification} \tag{2}$$

where the condition part of each rule is a conjunctive normal form, and V_k^j denotes the kth value of the jth feature. The rule is triggered when the value of the jth feature in the input is equal to one of elements in $\left\{V_1^j, \cdots, V_k^j\right\}$. Consequently, the length of each single rule, l, is defined as:

$$l = \sum_{i=1}^{n} k_i \tag{3}$$

where k_i denotes the number of possible values feature V^i could take.

To represent the condition part of rules in binary strings, one bit is associated to each available value of each feature. If a value is contained in the condition, the corresponding bit is set to one. Otherwise it is set to zero. For example, suppose we have three features $\{A, B, C\}$, and the value of each could be taken from $\{A_1, A_2\}$, $\{B_1, B_2, B_3\}$ and $\{C_1, C_2, C_3, C_4\}$, respectively. The rule $11|001|1001 \rightarrow 1$ means that "if the first feature has value A_1 or A_2, the second one has value B_3 and the third one has value C_1 or C_4, then the rule predicts class 1". If an individual I contains n single rules, I is encoded as a binary string with length equal to nl.

Given the description above, it is obvious that the knowledge in Pittsburgh-style LCSs is represented at two levels. At the lower level, each rule works according to the domain its condition part specifies (see Eq. (2)). At the higher level, individuals combine rules to form the entire solution for the problem (see Eq. (1)). In consequence, LCSs have to achieve rules with high performance first. Then, it assembles rules to form individuals to compete with each other aiming at optimum solution. The evaluation and evolution at the two levels should be discussed separately from each other.

3.2. Classifiers evaluation

3.2.1. Rule-wise evaluation

From the discussion above, it is already known that each rule has its own domain defined by the condition part. Therefore, the fitness should be defined in the domain. A straightforward way to measure the fitness is to compute the positive accuracy, which is equal to the ratio of the number of instances correctly classified to the number of instances covered by the rule. However, only positive accuracy could not fully reflect the performance. For example, two rules r_i and r_j have the same positive accuracy, but more instances, which lie outside the domain of both rules, belong to the classification of r_i than that of r_j. In such a situation, the performance of r_j is better than that of r_i. Therefore, another metric, called negative accuracy, which reflects the accuracy of the rule with respect to problem space outside the domain, has to be taken into consideration to complement the positive accuracy.

Consequently, in order to achieve a competent classifier system, rules should evolve towards maximal positive and negative accuracy simultaneously. In the proposed method, the fitness of each rule is the weighted sum of its positive and negative accuracy. It is calculated as:

$$f(r) = 0.5 \cdot \alpha_p(r) + 0.5 \cdot \alpha_n(r); \tag{4}$$

where $\alpha_p(r)$ and $\alpha_n(r)$ produce bias toward the accuracy inside and outside the domain, respectively. They are computed as:

$$\alpha_p(r) = N_p(r)/N_c(r); \tag{5}$$

$$\alpha_n(r) = N_n(r)/(N - N_c(r)); \tag{6}$$

where $N_p(r)$ and $N_n(r)$ denote the number of positive and negative instances correctly classified respectively, $N_c(r)$ specifies the number of examples covered by the rule, and N is the number of all the training instances.

It should be noted that the fitness in our method is not the same as that in CCS [25] and χECCS [26]. The differences lie in that the fitness in the proposed method measures the performance both inside and outside the domain specified by the condition part. While, the fitness in CCS/χECCS consists of an accuracy term and an error term. Although the former is defined on positive and negative examples together, the latter only takes account of the positive examples.

3.2.2. Rule set-wise evaluation

Before evaluating each individual, there is another issue: when more than one rule matches the instance, the individual has to choose the classification of only one rule as its proposed solution, if their classifications conflict with each other. Because the instance is located in the region of rules and the competition between them has to be restricted in their regions, our method chooses the one with higher positive accuracy (see Eq. (5)) in such a situation. Given the definite prediction for each matched instanced, the evaluation of individuals is straightforward. According to all the training data, the fitness of each individuals is defined as:

$$f(i) = N_r(i)/N; \tag{7}$$

where N and $N_r(i)$ specify the number of instances used for training and instances correctly classified by the individual, respectively.

3.3. Classifiers evolution

From the discussion in Section 3.1, it has been known that each individual in Pittsburgh LCSs consists of variable number of rules. Therefore, the evolution of classifiers may be executed at two levels, the one with rules contained in rule sets and the one with rule sets themselves. We will describe them in the following two subsections respectively, and then the integration of two levels is discussed.

3.3.1. Rule-wise evolution

The evolution of rules relies on classical variation operators (e.g., crossover and mutation) in the traditional Pittsburgh LCSs. However, as pointed out in [9], these operators may frequently disrupt important combinations of features, which often results in poor performance. Therefore, we replace those operators with probabilistic modeling and sampling mechanisms used in BOA, which relies on Bayesian networks to model dependencies between variables, to generate offspring rules.

The first issue in probabilistic modelings is the selection of promising rules to build the Bayesian network. One of strategies is to select rules from individuals with higher fitness. However, such an approach has three drawbacks. First, if the condition part of more than one rule matches one instance, the individual has to choose only one rule to determine its action according to the principle discussed in Section 3.2.2. In that procedure, the fitness of rules not selected is lower than that of the ones selected. Therefore, in individuals with higher fitness, not all rules actually have higher fitness. Next, rules in these individuals may not match any instances in the training data. Including these rules may produce perturbation in the Bayesian network. Additionally, rules with higher fitness may be contained in individuals with lower fitness. Only because the performance of other rules in the same individual are not good enough, the whole performance of the individual is not competent. Given the discussion above, we select promising rules according to the fitness defined in Eq. (4) to build Bayesian networks, regardless of the individual in which rules are contained.

According to the description in Section 3.1, the condition part of each rule is represented as a binary string of fixed length. With a set of binary codes to represent promising single rules, which are selected from all individuals, a dataset with dimension equal to the length of single rule (i.e., l, see Eq. (3)) is collected. Then Bayesian networks, in which each node represents a bit in the rule, are built according to the dataset. With the aim of learning Bayesian networks, both the structure representing the dependencies between variables and the conditional probabilities for each variable have to be identified. In order to learning the structure, the strategy called *score + search* [29] is performed to add dependencies gradually into the network. More specifically, the Bayesian–Dirichlet (BD) metric [21] is employed as the score in the paper. The procedure terminates when the metric could not be improved. After the structure has been identified, conditional probabilities for each node are calculated just by counting the number of times the value 0 or 1 appears in the corresponding bit in all promising single rules. After Bayesian networks are learned, new rules are generated just by sampling the networks. In this paper, we choose the forward sample technology [21].

Another problem during rule evolution is to maintain diversity in the population. As described in Section 3.1, each rule has a condition part to specify when it is applicable. Consequently, niche technique should be taken into to consideration to execute local competition: similar rules compete with each other, whereas dissimilar ones compete only rarely or never. There are many methods to localize competition, such as crowding, fitness sharing, and clearing. [36]. In our method, the restricted tournament replacement (RTR) [16] is adopted. In RTR, each new individual I is incorporated in the original population as follows. First, a random subset of individuals S with size σ is selected from the original population. Next, the individual I', which is most similar to I in S, is detected. In our method, the similarity between rules is measured according to the Hamming distance between their condition parts. Finally, I' is replaced with I, if I is better, otherwise I is discarded. The size of random subset σ is set to the length of the bit, i.e., $\sigma = l$. The reasons for the setting, as suggested in [29], are described as below. The number of niches (i.e., n) that can be maintained in a population of candidate solutions is equal to some fraction of the population size N:$n = O(N)$. On the other hand, RTR with size of σ can maintain the number of niches that is equal to some fraction of size of random subset: $n = O(\sigma)$. Because the population in BOA is expected to grow approximately linearly with the size of the problem: $N = O(l)$, the size of RTR is set proportionally to the size of the problem (i.e., $\sigma = O(l)$) in order to maximize the number of niches maintained by BOA without affecting the population sizing. Given the discussion above, the whole procedure of rule evolution via probabilistic modeling and sampling mechanisms used in BOA is described in Algorithm 2.

Algorithm 2. Rules evolution via probabilistic modeling and sampling mechanisms

```
1  g ← 0
2  P_g ← Rules in all individuals
3  Evaluate P_g
4  repeat
5    S ← Select promising rules in P_g
6    B ← Build Bayesian Network from S
7    O ← Sample from B
8    Evaluate O
9    P_{g+1} ← Replace P_g with O by RTR
10   g ← g + 1
11 until Stopping criterion is met
```

According to the discussion in Section 3.1, it is known that rules have to be assembled in individuals to represent entire solutions in Pittsburgh LCSs. In the proposed method, we first create a new population (i.e., P'_{rs}) with size equal to the current one (i.e., P_{rs}). Each individual in P'_{rs} consists of rules that are randomly picked from the rule offspring, which are produced by sampling the Bayesian networks. The size of each individual in P'_{rs} is set to the value of the corresponding one in P_{rs}. Then the fitness of each individual in P'_{rs} is computed according to Eq. (7). Next, individuals in P_{rs} and P'_{rs} are ranked together according to the fitness. Finally, a new rule-set population is achieved, which consists of the best half of all individuals in P_{rs} and P'_{rs}. The procedure described above is listed in Algorithm 3.

Algorithm 3. The procedure of assembling rules

```
1  P_rs ← Individual (i.e., rule set) population before BOA is performed
2  P'_rs ← P_rs
3  P_r ← Rule population generated by BOA
4  for all Individual I in P'_rs do
5    n ← Size of I
6    Delete all rules in I
7    for i = 1 to n do
8      R ← Random one in P_r
9      Add R into I
10   end for
11 end for
12 Evaluate P'_rs
13 P_rs ← The best half of all individuals in P_rs and P'_rs.
```

3.3.2. Rule set-wise evolution

According to the description in Section 3.1, each individual consists of a variable number of rules. Since all rules have the same length l (see Eq. (3)), the individual is represented by a binary string with length equal to nl, where n is the number of rules contained in the individual. In GABIL and GAssist, crossover point can be set to any position of the string. Just because of the mechanism mentioned above, recombination in a single rule or just between two rules occurs indiscriminately when the operator is fired.

In the proposed method, however, situations are different. Since probabilistic modeling and sampling mechanisms guide the recombination at rule-wise level, the semantics of each rule in individuals has to be kept. With such an aim, the recombination of individuals in our method is performed by means of rule-wise uniform crossover, in which new individuals are generated by exchanging all bits in rules between the two parents with certain probability instead of exchanging single bits (See the frame in the lower left corner in Fig. 2). Because the rule-wise uniform crossover prevents rule disruption but maximizes rule mixing or exchange, it represents an ideal crossover operator to use. Note that any information about the structure of the problem do not have to be given in advance to execute the rule-wise uniform crossover. The crossover points could only be set to the position between ilth bit and $(il + 1)$th bit, where i is an integer lower than n, and l is already known when rules are encoded in binary strings.

3.3.3. Integration

In this subsection, we discuss how to integrate the evolution of rules and rule sets together. Since probabilistic modeling mechanisms used in BOA are computationally expensive, we do not perform rule evolution in each generation. Instead, rules

are updated only once in several generations. In contrast, the rule-wise crossover on rule set is executed in each generation according to the description in subsubsection 3.3.2. Given the discussion above, the whole procedure of the proposed method is listed in Algorithm 4. Lines 5–14 represent the procedure of rule-wise combination via probabilistic modeling and sampling mechanisms used in BOA. The parameter *inte* denotes the interval of rule-wise combination. Lines 15–19 describe the procedure of rule set-wise combination.

Algorithm 4. The procedure of the proposed method

```
1   t ← 0
2   P_t ← Generate a random population of rule sets
    // Each rule set represents a classifier
3   Evaluate rule sets in P_t
4   repeat
5     if mod(t,inte) = 0 then
6         S_r ← Select promising rules in all population
7         B ← Build Bayesian network from S_r
8         O_r ← Sample from B
9         Evaluate O_r
10        R' ← Replace S_r with O_r by RTR
11        P' ← Assemble R' into rule sets
12        Evaluate P'
13        P_t ← Replace P_t with P'
14    end if
15    S_rs ← Select promising rule sets
16    O_rs ← Rule-wise crossover on S_rs
17    Evaluate O_rs
18    P_{t+1} ← Replace P_t with O_rs
19    t ← t + 1
20  until Stopping criterion is met
21  The best rule set is treated as the final solution
```

4. Experimental setup

4.1. Artificial problems

To evaluate the performance of our method, the experiments on four Boolean functions, including the multiplexer problem, the count ones problem, the parity problem, the ladder problem, and their combinations are performed.

The multiplexer problem has been the subject of study for a long time in learning classifier system research [8]. In general, the input to the Boolean multiplexer function is a binary string of length $k + 2^k$ of the form

$$A_{k-1} \cdots A_1 A_0 D_{2^k-1} \cdots D_1 D_0 \tag{8}$$

where the former k bits A_i and latter 2^k bits D_i are called "address" bits and "data" bits, respectively. The output of the multiplexer function is determined by the data bit located at the position referred by the binary value of the address bits. For example, in the six bits multiplexer problem, $f_m(100010) = 1$ and $f_m(000111) = 0$. Obviously, the search space for the multiplexer problem with $k + 2^k$ arguments is of size 2^{k+2^k}. In the rest of the paper, n bits multiplexer problem is denoted as MP-n. We also construct noisy multiplexer problems by flipping the actual classification with a certain probability p_n. In this paper, we consider four levels of noise: $p_n = \{0.05, 0.10, 0.15, 0.20\}$.

The count ones problem [8] is also a kind of Boolean functions, in which the output is defined by the number of ones in input bits. If this number is greater than half length of the input, the output is one and otherwise the output is zero. For example, within the input of length equal to five, $f_o(00001) = 0$ and $f_o(10111) = 1$. For the sake of brevity, count ones problem with n-bit input is abbreviated as One-n below.

The parity problem [8] is similar to the count ones problem, in which the output is also determined according to the number of ones in input bits. The difference lies in that the output is equal to the number modulo two. For example, in the six multiplexer $f_p(00001) = 1$ and $f_p(10111) = 0$. In the remainder of the paper, n bits parity problem is denoted as Parity-n.

The ladder problem also considers the number of ones in input bits. The problem denoted as Ladder-n-l means that output is set to one, when more than n bits in the l-length input are ones, and otherwise the output is zero. In fact, the count ones problem is a specific case of the ladder problem. More specifically, the counter one problem One-n is just identical to the ladder problem Ladder-n/2-n, which means that output is set to one, when more than n/2 bits in the n-length input are ones.

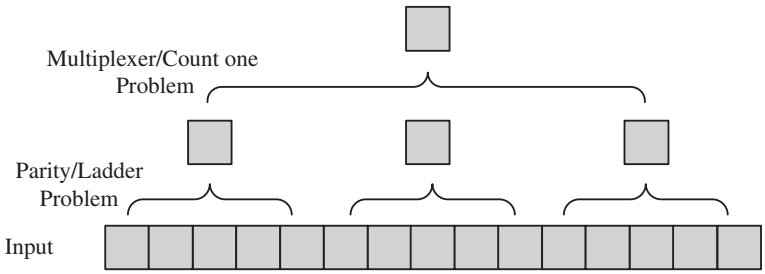


Fig. 3. The hierarchy of artificial problems.

Table 1
Properties of the artificial problems.

Name	#Instance	#Feature	Rule length
MP-11 (with or without noise)	2048	11	22
MP-20	1048576	20	40
MP-37	1.37E11	37	74
MP-70	1.18E21	70	140
ParityMP-2-6	4096	12	24
ParityMP-3-6	262144	18	36
ParityOne-2-5	1024	10	20
ParityOne-3-5	32768	15	30
LadderMP-2-2-6	4096	12	24
LadderMP-1-2-6	4096	12	24
LadderMP-3-3-6	262144	18	36
LadderMP-2-3-6	262144	18	36

To investigate the performance of the proposed method, we combine the four problems above to construct more challenging ones. In these hierarchical problems, bits gathered in disjoint groups first apply parity or ladder problems, then results of all groups are viewed as the input of multiplexer or count ones problem (see Fig. 3). For the sake of brevity, the parity multiplexer, parity count ones and ladder multiplexer problems are noted as ParityMP, ParityOne and LadderMP, respectively. More specifically, ParityMP-*n*-*m* and ParityOne-*n*-*m* mean that every *n* concatenated bits applies Parity problems, then *m* bit multiplexer or counter one problems are applied to the results. And LadderMP-*n*-*l*-*m* means that every *n* bits applies Ladder-*n*-*l*, then MP-*m* is applied on the results.

Within the problems above, the total instances are used to train and test the proposed pLCS_BOA and GAssist except on the problems of MP-37, MP-70, ParityMP-3-6 and ParityOne-3-5 which contain huge instances that cannot be stored in RAM memory. Instead, only a random subset of the full dataset of the problem is used in each generation. The sizes of training subset used in each generation for MP-37, MP-70, ParityMP-3-6 and ParityOne-3-5 are equal to 1373, 1574, 1000, and 1000.[1] In summary, the properties of these artificial problems are listed in Table 1, in which the number of features is equal to the length of bits representing the condition part in each rule, and the rule length is computed according to Eq. (3).

4.2. Real world problems

In additional to artificial problems, we also consider five real world binary classification problems with categorical features. These data sets were obtained from the UCI repository [1]. We choose binary classification problems because of the convenience for the binary coding. The properties of these data sets are listed in Table 2. The meanings of feature and rule length are identical to those in Table 1.

4.3. Configurations

For GAssist, we use the open source code[2] developed by the authors. The proposed method is implemented based on the code of GAssist mentioned above. The parameters for GAssist are presented in Table 3, which are set according to [4].

For pLCS_BOA, the corresponding parameter specifications are identical to those for GAssist except the probability of mutation, since there is no mutation in our method. And settings different from GAssist and other parameters specifically designed for pLCS_BOA are listed in Table 4. For the artificial and real world problems mentioned above, 5-fold cross validation is used to compare the performance of GAssist and pLCS_BOA. For each experimental case, the result is the average

[1] The sizes of training subset for MP-37 and MP-70 are set according to [4].
[2] http://www.asap.cs.nott.ac.uk/~jqb/PSP/GAssist-Java.tar.gz.

Table 2
Properties of the real world problems.

Name	#Instance	#Features	Rule length
House-vote	425	16	32
KR-KP	3196	36	72
Mushroom	8124	22	127
TicTacToe	958	9	27
Monks-1	432	6	17

Table 3
General parameters to configure GAssist and pLCS_BOA.

Parameter	Value
Crossover prob.	0.6
Selection algorithm	Tournament selection
Selection tournament size	3
Population size	200
Individual-wise mutation prob.	0.6
Initial number of rules per individual	20
Default class policy	Major
Probability of value 1 in initialization	0.90
Terminal criterion	Convergence
Generation of activation for rule deletion	5
Minimum number of rules	6
Generation of activation for MDL-based fitness function	25
Initial theory length ratio	0.075
Weight relax factor	0.90

Table 4
Other parameters to configure pLCS_BOA.

Parameter	Value
Interval to execute rule evolution via BOA	10
Probability of rule-wise crossover	0.6
Selection algorithm in BOA	Truncation selection
Proportion for selection in BOA	0.5
Replacement algorithm in rules assembling	Elitist replacement
Proportion for replacement in rules assembling	0.5
Use MDL	False
Minimum number of rules	50 (ParityMP, ParityOne, and LadderMP-2-3-6)

over thirty runs. If not stated differently, the experiments are run on a desktop with Intel Core 2 3.00 GHz processor and 2 GB RAM.

5. Experimental results

5.1. Artificial problems

The comparison between GAssist and pLCS_BOA on the multiplexer problems is presented in Fig. 4. From those results, we can see that the proposed method outperforms GAssist obviously with respect to the total number of match operators. Here, the match operator is referred to be executed once when the condition part of one single rule is compared with an instance.

Experimental results on multiplexer problems with different levels of noise are presented in Fig. 5, which show that pLCS_BOA converges faster than GAssist in terms of total number of match operators executed. Additionally, results in Fig. 5 demonstrates that the gaps between prediction accuracy and 1 as well as the total computational costs (i.e., number of total match operators) are proposition to the noisy degree. The reason for the phenomenon lies in that the level of noise reflects the complexity of the problem. In another word, larger noisy degree, more sophisticated the problem.

The performance comparison between GAssist and pLCS_BOA on the hierarchical problems is presented in Figs. 6 and 7, respectively. With parity hierarchical problems, the proposed algorithm converges to the near-optimal solution faster than GAssist (see Fig. 6). The same trend can be also observed in ladder multiplexer problems (see Fig. 7). More specifically, when problems in the lower level are more sophisticated (e.g., LadderMP-1-2-6 and LadderMP-2-3-6), the gap in performance between the two methods is more remarkable.

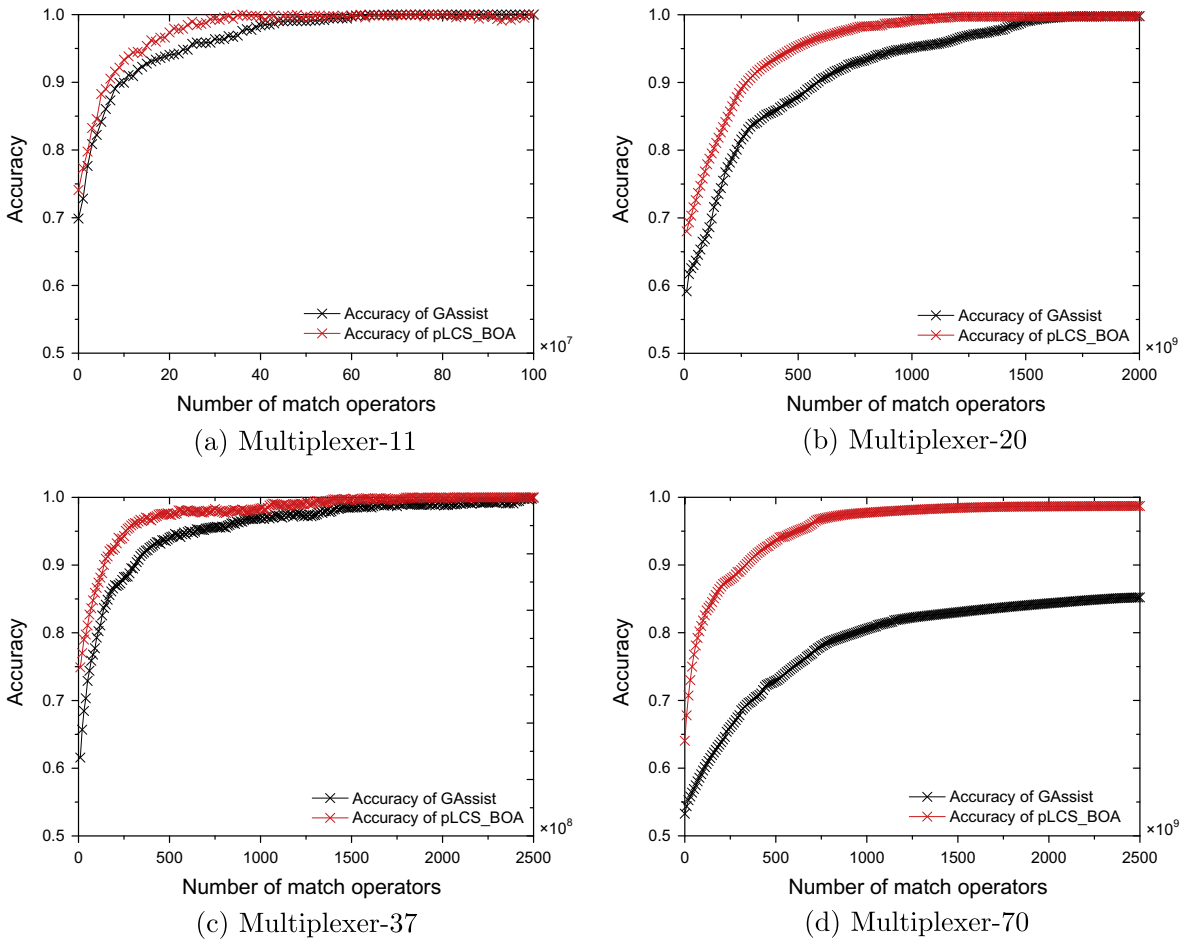


Fig. 4. Performance comparison on multiplexer problems.

Table 5 shows the classification accuracy on the those artificial problems, and Table 6 represents the number of generations, the number of match operators executed, and the running time. According to Table 5, pLCS_BOA can deliver solutions with the same or even higher classification accuracy. From Table 6, it is obvious that pLCS_BOA converges faster than GAssist in terms of both generation and total match operators for all problems except LadderMP-3-3-6. The two metrics could be generally reduced about 30% and 15%, respectively. It is also noted that the numbers of match operators executed per generation in pLCS_BOA are larger than those of GAssist for all problems. Generally, the proposed method performs twice as many match operators as GAssist does in each generation according to the results. The reason behind the phenomenon lies as below. For each rule set in GAssist, match operator terminates when the first matched rule (in the order the rules are placed in the individual) is detected, and the prediction of the rule is treated as the result of the rule set. While in the proposed method, all rules in a rule set are compared to examples with the aim of computing the fitness, and the rule set adopts the best one to solve the problems. Although the computation cost of each generation in pLCS_BOA is higher than that in GAssist, however, the total cost has been reduced since the number of generations is decreased remarkably. With respect to the running time, the proposed algorithm takes less than GAsssit to deal with most problems. More specifically, when handling problems with fewer variables and uncomplicated structures (i.e., 11 bits multiplexer problem and its nosy versions), the running time of pLCS_BOA is higher than that of GAssist. However, as the number of variables increases and the structure becomes more sophisticated, the advantage of pLCS_BOA over GAssist is obvious regarding to the running time. The reason behind the phenomena lies in that the rule evolution via BOA introduces more computational costs. And the additional costs do not improve the overall performance when the problems are easy to be solved.

Additionally, we want to detect significant differences between pLCS_BOA and classic GAssist on these artificial problems in terms of accuracy, the number of generations, the number of match operators executed totally and in each generation, as well as the running time. Since the obtained results may present neither normal distribution nor homogeneity of variance, the Wilcoxon paired signed ranks test [32], which is a nonparametric test, is executed in our experiments, according to the recommendations made in [12]. The results are presented in Table 7. For a level of significance $\alpha = 0.05$, it is obvious that

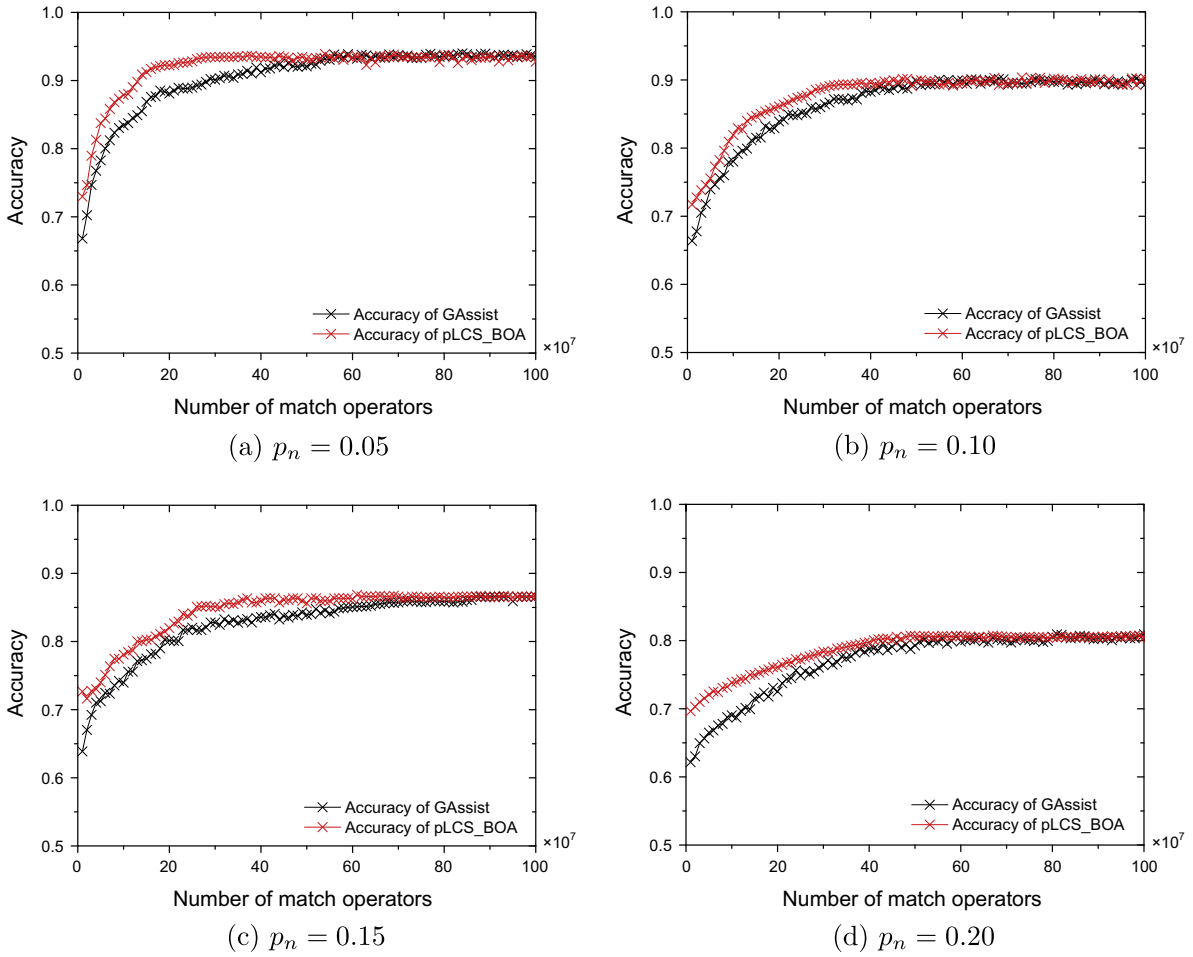


Fig. 5. Performance comparison on noisy multiplexer problems.

pLCS_BOA delivers solutions with the same accuracy in fewer generations and taking less time. Although the number of match operators executed in pLCS_BOA per generation is higher than that in GAssist, the total number is lower.

We also compare the proposed method with MPLCS [4], which integrates rule-wise and rule set-wise local search mechanisms into classic GAssist. Here, we use 20, 37 and 70 bits multiplexer problems as the benchmark problems. The parameters in MPLCS are set according to the values listed in Table 3, and the local search probability of MPLCS is equal to 0.05, according to the suggestion in [4]. The results are represented in Table 8. With respect to the number of both generations and match operators executed, the advantage of MPLCS is remarkable. It means that well designed local search mechanisms may produce higher efficiency than sophisticated global search mechanisms (e.g., EDA) do for Pittsburgh learning classifier systems.

Furthermore, an approximate comparison is performed between the the proposed method and XCS. Since there are essentially distinguishing characteristics between Michigan-style and Pittsburgh-style LCS, the number of learning steps in the former is totally different from the number of generations or match operators executed in the latter. Therefore, we compare the running time of pLCS_BOA and XCS. It has been pointed out that XCSBOA performs similarly to the standard XCS on 20 bit multiplexer [4]. Consequently, we adopt such a problem as the test bed. To run XCS, we use the open code[3] published by the authors. For pLCS_BOA, rather than employing all data for training as done above, here, the size of training set used in each generation is 420.[4] It is because data sets are generated on-the-fly in XCS, and pLCS_BOA performs in a similar way aiming at a fair comparison. The results are listed in Table 9, which is the average over thirty runs. According to the results, we could see that the running time of pLCS_BOA is a little higher than that of XCS, which means that the performance of pLCS_BOA is comparable to that of XCS.

[3] http://www.illigal.uiuc.edu/pub/src/XCS/XCS1.2.tar.z.
[4] The size of training subset is set according to [4].

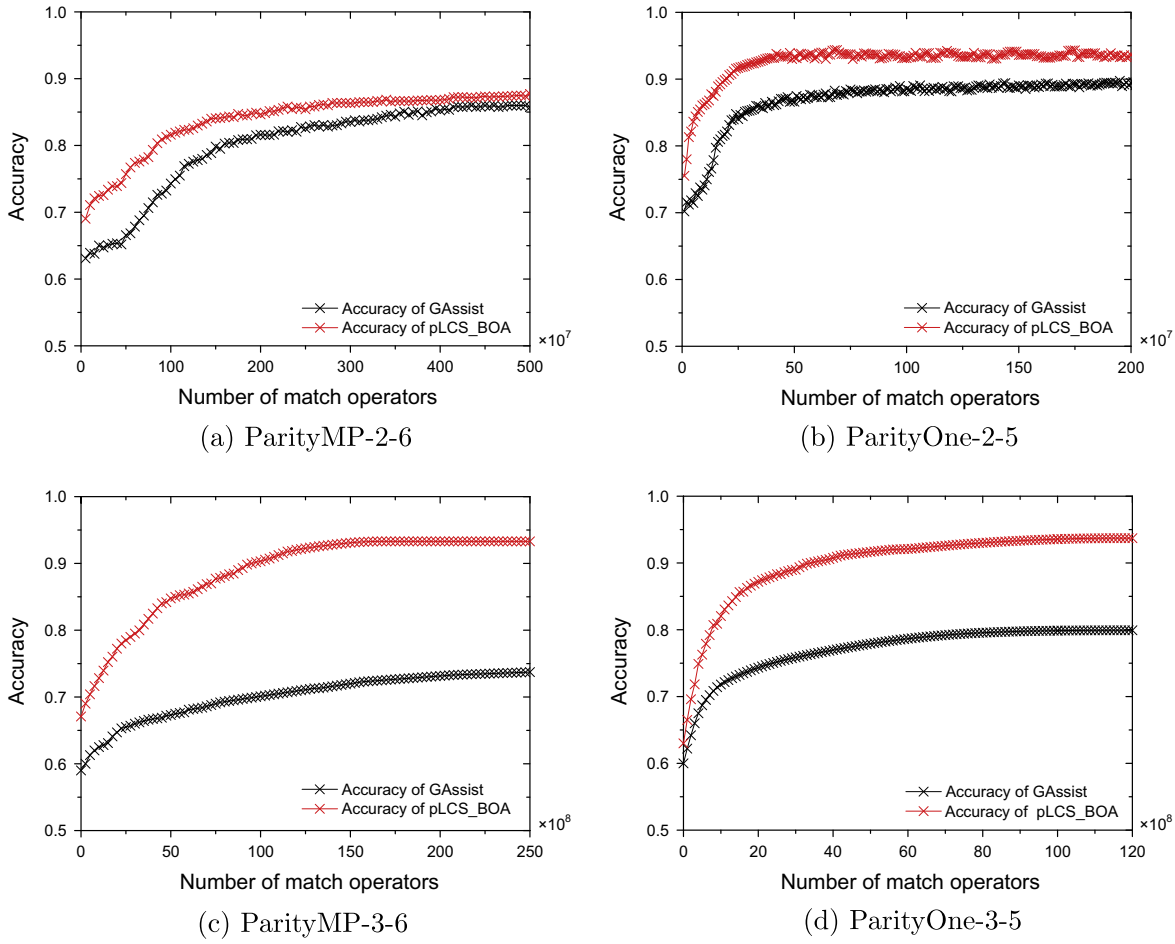


Fig. 6. Performance comparison on parity multiplexer and parity one problems.

5.2. Real world problems

This subsection describes the comparison between pLCS_BOA and GAssist on the real world problems listed in Table 2. The results are represented in Tables 11 and 10. It is obvious that pLCS_BOA can deliver solutions with almost the same accuracy in much fewer generations. The reduction in the number of generations generally reaches more than 20%. Regarding to the number of match operators executed, although pLCS_BOA perform more in each generation, it relies on fewer during the evolution. With respect to the running time, pLCS_BOA requires less to deliver solutions for all problems except mushroom.

We also use the Wilcoxon paired signed ranks test to investigate the significant differences in results listed in Tables 11 and 10. According to the results in Table 12, although the number of match operators executed in each generation in pLCS_BOA is larger than that in GAssist, it can be safely concluded that pLCS_BOA converges to the near-optimal solution faster than original GAssist in terms of the number of both generations and total match operators executed as well as the running time.

6. Discussion

According to experimental results presented in Section 5, the proposed method converges faster than GAssist while achieving solutions with the same or even higher accuracy. The reasons behind the phenomenon are listed as below.

First, classifiers in the proposed method evolve at two levels. At the lower level, single rules contained in individuals evolve through building and sampling Bayesian networks of promising rules in all individuals in order to identify dependencies between variables and detection of substructure in single rules. At the rule set level, since each individual consists of variable number of rules, the recombination of individuals is performed based on rule-wise uniform crossover operators, which exchange all bits in single rules between the two parents to make sure that rules are not disrupted in the individuals. The strategies generating offspring at both levels assure an effective combination of features.

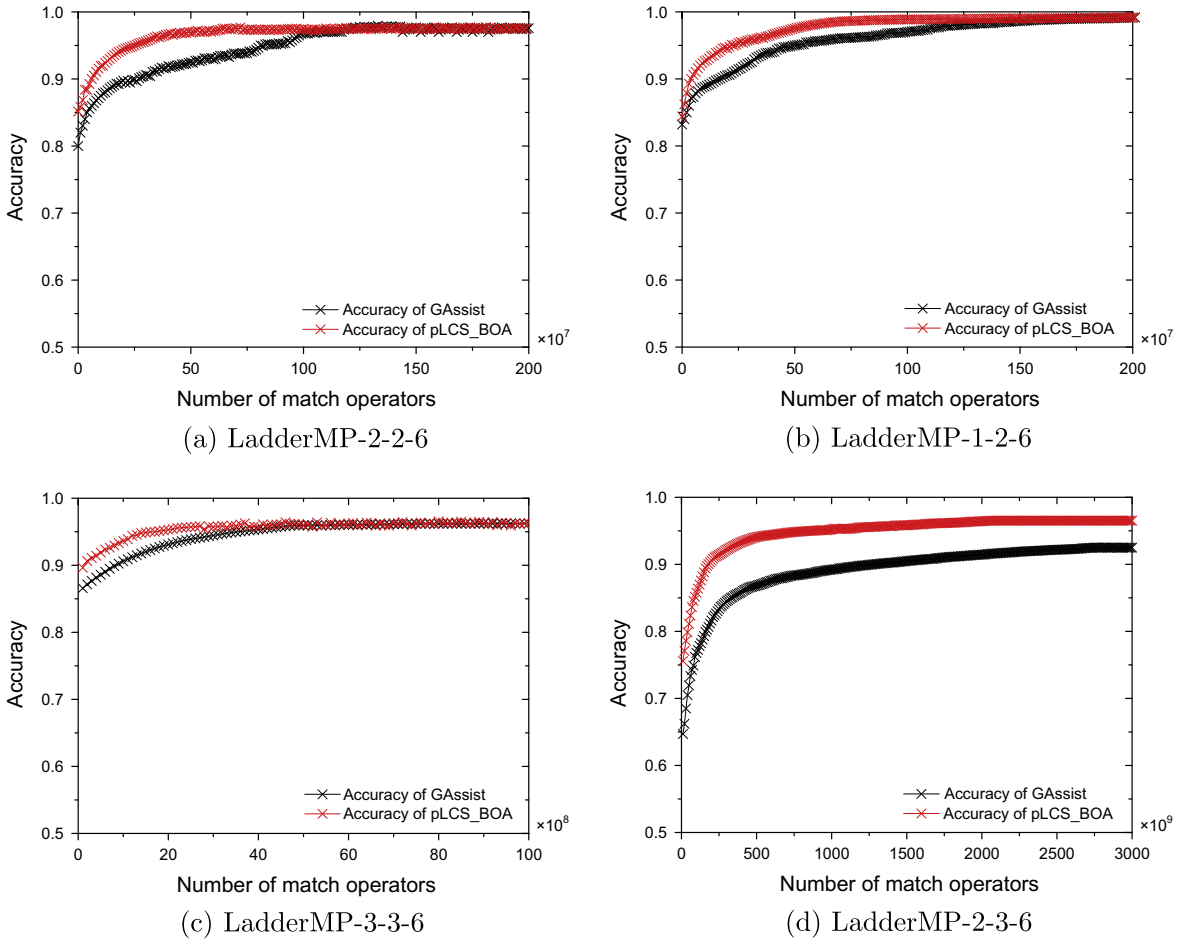


Fig. 7. Performance comparison on ladder multiplexer problems.

Table 5
Comparison of the classification accuracy between GAssist and pLCS_BOA on artificial problems.

Data-set	GAssist	pLCS_BOA
MP-11 without noises	1.0000 ± 0.0000	1.0000 ± 0.0000
MP-20	0.9989 ± 0.0089	0.9974 ± 0.0052
MP-37	0.9870 ± 0.0157	0.9914 ± 0.0035
MP-70	–	0.9867 ± 0.0114
MP-11 $p_n = 0.05$	0.9442 ± 0.0054	0.9417 ± 0.0047
MP-11 $p_n = 0.10$	0.8995 ± 0.0074	0.8931 ± 0.0046
MP-11 $p_n = 0.15$	0.8694 ± 0.0053	0.8643 ± 0.0098
MP-11 $p_n = 0.20$	0.8138 ± 0.0089	0.8059 ± 0.0143
ParityMP-2-6	0.8593 ± 0.0234	0.8702 ± 0.0214
ParityOne-2-5	0.8923 ± 0.0103	0.9387 ± 0.0187
ParityMP-3-6	0.7789 ± 0.0100	0.9302 ± 0.0247
ParityOne-3-5	0.8049 ± 0.0093	0.9368 ± 0.0275
LadderMP-2-2-6	0.9837 ± 0.0297	0.9801 ± 0.0193
LadderMP-1-2-6	0.9859 ± 0.0103	0.9874 ± 0.0097
LadderMP-3-3-6	0.9838 ± 0.0314	0.9652 ± 0.0265
LadderMP-2-3-6	0.9250 ± 0.0182	0.9652 ± 0.0201

Next, the evaluation of rules considers the balance between the specification and the generalization. With respect to specification, the fitness adopts positive accuracy inside the domain to measure the performance. Obviously, the higher the positive accuracy, the stronger the performance. However, the competent rule also give the negative prediction when instance do not match the condition part of the rule. In consequence, negative accuracy, which reflects the performance of the rules

Table 6
Comparison of the number of generations, the number of match operators executed, and the running time between GAssist and pLCS_BOA on artificial problems.

Data-set	Generations		Total match operators ($\times 10^8$)		Average match operators in each generation ($\times 10^6$)		Running time (s)	
	GAssist	pLCS_BOA	GAssist	pLCS_BOA	GAssist	pLCS_BOA	GAssist	pLCS_BOA
MP-11	226.67 ± 64.10	163.58 ± 44.09	3.913 ± 1.542	3.426 ± 0.678	1.602 ± 0.055	2.185 ± 0.247	5.37 ± 3.07	7.02 ± 0.63
MP-20	1103.7 ± 216.1	456.94 ± 186.2	14541 ± 3696	9822 ± 2215	1129 ± 345.3	2354 ± 475.1	(1.91 ± 0.35) E5	(1.66 ± 0.29) E5
MP-37	6573.8 ± 1473.5	3204.2 ± 812.6	692.5 ± 173.5	578.2 ± 174.5	9.791 ± 1.425	18.24 ± 3.251	(1.28 ± 0.39) E4	(7.94 ± 0.89) E3
MP-70	–	18403 ± 4512	–	9578.2 ± 1745	–	49.46 ± 34.56	–	(1.84 ± 0.45) E6
MP-11 $p_n = 0.05$	290.14 ± 74.48	172.29 ± 41.56	4.159 ± 0.824	3.493 ± 0.690	1.465 ± 0.044	2.207 ± 0.194	5.86 ± 3.08	7.66 ± 0.93
MP-11 $p_n = 0.10$	445.89 ± 61.17	237.75 ± 43.60	4.657 ± 0.798	4.262 ± 0.932	1.030 ± 0.067	1.827 ± 0.322	11.4 ± 7.72	10.8 ± 0.85
MP-11 $p_n = 0.15$	602.70 ± 52.76	294.63 ± 77.54	6.219 ± 1.585	5.291 ± 1.422	1.016 ± 0.075	1.841 ± 0.123	15.1 ± 10.3	12.5 ± 0.72
MP-11 $p_n = 0.20$	695.63 ± 27.84	364.25 ± 28.67	7.462 ± 1.036	6.803 ± 1.363	1.042 ± 0.061	1.897 ± 0.888	16.9 ± 11.5	15.7 ± 0.63
ParityMP-2-6	1216.0 ± 75.82	844.71 ± 128.5	46.53 ± 16.23	42.22 ± 9.654	3.807 ± 0.315	5.042 ± 0.235	193 ± 27.6	172 ± 28.4
ParityOne-2-5	896.4 ± 139.8	382.33 ± 8.802	14.19 ± 2.814	11.53 ± 1.137	1.568 ± 0.106	3.124 ± 1.130	107.6 ± 9.0	92.1 ± 19.6
ParityMP-3-6	2671.9 ± 345.0	676.41 ± 204.3	234.3 ± 57.36	121.0 ± 31.76	8.384 ± 1.210	20.32 ± 1.210	468 ± 55.6	378 ± 46.2
ParityOne-3-5	1824.4 ± 263.5	495.74 ± 163.4	76.72 ± 16.36	46.03 ± 9.261	4.009 ± 0.681	9.150 ± 0.791	374 ± 40.0	285 ± 44.7
LadderMP-2-2-6	692.50 ± 194.0	444.80 ± 22.86	10.02 ± 2.927	7.912 ± 0.836	1.423 ± 0.114	1.813 ± 0.541	69.0 ± 5.52	52.0 ± 14.1
LadderMP-1-2-6	791.67 ± 92.18	399.56 ± 82.25	9.57 ± 2.356	7.510 ± 5.567	1.189 ± 0.223	1.907 ± 1.544	95.9 ± 23.2	76.7 ± 16.7
LadderMP-3-3-6	58.346 ± 1.236	63.346 ± 1.314	34.94 ± 4.568	36.94 ± 4.410	53.44 ± 10.95	58.05 ± 18.67	542 ± 20.9	712 ± 254
LadderMP-2-3-6	2417.7 ± 268.6	946.9 ± 167.6	24734 ± 836.4	21237 ± 763.2	986.4 ± 132.6	2334 ± 517.7	(2.98 ± 0.53) E5	(2.23 ± 0.39) E5

Table 7
Significance tests between GAssist and pLCS_BOA according to the performance on artificial problems. The *p*-values are from one sided Wilcoxon paired signed-rank tests. The level of significance is set to 0.05.

	R^+	R^-	*p*-Value
Accuracy	65.5	39.5	0.1985
Number of generations	1	119	0.0004
Number of total match operators	1	119	0.0004
Number of match operators per generation	120	0	0.0003
Running time	31	89	0.0498

Table 8
Comparison of the number of generations and match operators between pLCS_BOA and MPLCS on multiplexer problems.

Data-set	Generations		Total match operators ($\times 10^8$)	
	pLCS_BOA	MPLCS	pLCS_BOA	MPLCS
MP-20	456.9 ± 186.2	59.96 ± 13.52	9822 ± 2215	992.5 ± 167.6
MP-37	3204.2 ± 812.6	197.3 ± 47.82	578.2 ± 174.5	21.01 ± 4.572
MP-70	18403 ± 4512	833.7 ± 134.9	9578 ± 1745	189.4 ± 45.73

Table 9
Comparison of the running time between pLCS_BOA and XCS on the 20 bits multiplexer problem.

	pLCS_BOA	XCS
Running time (s)	29.94 ± 4.21	25.40 ± 3.94

outside the domain, is taken into account to measure the generalization. Guided by such a fitness, rules evolve towards high accuracy both inside and outside the domain.

Additionally, the prediction of individuals in the proposed method is different from that in GAssist. More specifically, all rules in an individual have to be compared with examples in order to do the prediction in the proposed method, while in GAssist the prediction of an individual terminates when the first matched rule is found. Consequently, the computational cost of each generation in the proposed method is higher than that in GAssist. However, the total number of match operators executed in the proposed method is lower than that in GAssist. The reason behind the phenomenon is that the proposed method relies on fewer generations before convergence, which in turn decrease the total number of match operators.

Compared with GAssist, the additional computational cost of the proposed method is introduced by probabilistic modeling and sampling mechanisms in BOA. The overhead of the mechanisms is controlled by tuning parameter *inte* in Algorithm 4, which denotes the interval of rule-wise combination. Here, we use the problems of ParityMP-2-6 and ParityOne-2-5 (see

Table 10
Comparison of the number of generations, match operators and the running time between GAssist and pLCS_BOA on real world problems.

Data-set	Generations		Total match operators ($\times 10^7$)		Average match operators in each generation ($\times 10^5$)		Running time (s)	
	GAssist	pLCS_BOA	GAssist	pLCS_BOA	GAssist	pLCS_BOA	GAssist	pLCS_BOA
House-vote	207.37 ± 65.65	112.08 ± 13.84	2.83 ± 0.78	1.46 ± 0.25	1.10 ± 0.13	1.81 ± 0.21	0.99 ± 0.33	0.95 ± 0.02
KR-KP	366.54 ± 107.59	228.54 ± 27.12	39.10 ± 12.59	33.36 ± 13.52	10.76 ± 2.12	18.49 ± 3.61	23.06 ± 4.00	22.58 ± 1.99
Mushroom	201.58 ± 104.90	168.78 ± 24.69	49.17 ± 24.60	48.96 ± 12.99	24.63 ± 3.04	38.01 ± 15.22	42.06 ± 6.22	43.14 ± 4.71
TicTacToe	440.97 ± 155.34	305.24 ± 156.89	44.38 ± 8.59	34.49 ± 6.90	10.08 ± 1.56	11.38 ± 1.63	19.99 ± 3.43	17.41 ± 2.28
Monks-1	112.67 ± 68.81	54.47 ± 10.30	1.31 ± 0.79	1.21 ± 0.18	1.17 ± 0.08	3.41 ± 0.65	1.68 ± 0.27	1.64 ± 0.04

Table 11
Comparison of the classification accuracy between GAssist and pLCS_BOA on real world problems.

Data-set	GAssist	pLCS_BOA
House-vote	0.9916 ± 0.0041	0.9896 ± 0.0059
KR-KP	0.9726 ± 0.0134	0.9695 ± 0.0103
Mushroom	0.9978 ± 0.0050	0.9999 ± 0.0004
TicTacToe	0.9704 ± 0.0099	0.9697 ± 0.0156
Monks-1	0.9847 ± 0.0218	0.9994 ± 0.0143

Table 12
Significance tests between GAssist and pLCS_BOA according to the performance on real world problems. The *p*-values are from one sided Wilcoxon paired signed-rank tests. The level of significance is set to 0.05.

	R^+	R^-	*p*-Value
Accuracy	8	7	0.3936
Number of generations	0	15	0.0155
Number of total match operators	0	15	0.0155
Number of match operators per generation	15	0	0.0155
Running time	1	14	0.0295

the definition in Section 4.1) as the test beds to investigate the changes in the running time and the number of match operators as the interval increases. According to the results depicted in Fig. 8, the number of match operators grows when the interval becomes larger. The reason lies in that large interval means rule-wise combination via BOA is performed occasionally, which degrades the efficiency of evolution and increases the running time. However, since BOA is computationally expensive, when the value of interval is too low, which means BOA is performed over-frequently, the running time grows significantly. From Fig. 8, the balance is achieved when the interval is set around 10. With the configuration, the efficiency of evolution is improved while the cost of BOA is not remarkable.

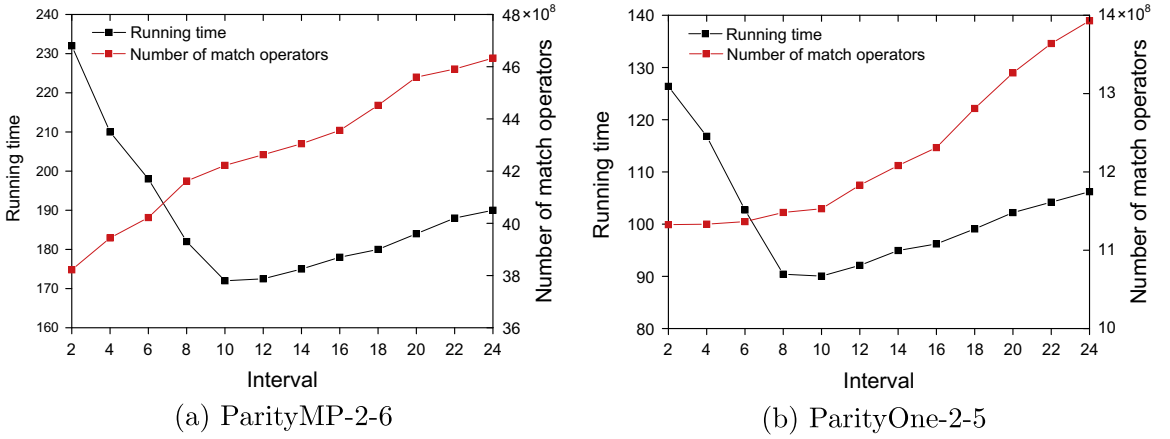


Fig. 8. Performance sensitivity to the interval of rule-wise combination via BOA.

From Table 8, MPLCS [4], which integrates local search mechanisms (rule-wise and rule set-wise) into the classic GAssist, outperforms pLCS_BOA in terms of the number of generations and match operators executed. According to the results, it seems that for Pittsburgh LCSs well designed local search mechanisms may produce higher efficiency than sophisticated global search mechanisms (e.g., EDAs) do. Consequently, combining local search mechanisms with EDAs may deliver more competent results.

7. Conclusion

In this paper, we integrate BOA into Pittsburgh LCSs with the aim of improving the effectiveness and efficiency of the rule structure exploration. In the proposed method, classifiers are generated and recombined at two levels. At the lower level, single rules contained in each classifier are produced by sampling Bayesian networks which characterize the distribution of promising rules in the search space. At the higher level, classifiers are recombined by rule-wise uniform crossover, which keeps the semantics of all rules in each classifier. We implement our method based on the GAssist system, which is the new generation of Pittsburgh-style LCSs. The proposed method is evaluated on artificial and real world binary classification problems, respectively. The experimental results show that the proposed method reduces the number of both generations and match operators executed as well as the running time while achieving solutions with the same or even higher accuracy, when compared with the original Pittsburgh-style LCSs.

Future areas for research include extending the proposed method in multiple classification problems. To deal with those problems, the action part of rules may play a special role and the system may build models for each class separately. In addition, our future work will incorporate certain local search mechanisms into the frame of BOA, which may improve the search efficiency in Pittsburgh learning classifier systems.

Acknowledgments

The authors thank anonymous reviewers for their valuable suggestions. This work was supported by National Natural Science Foundation of China (Grant Nos. 60875073 and 61175110), Important National Science & Technology Specific Projects of China (Grant Nos. 2009ZX02001 and 2011ZX02101-004) and the National Basic Research Program of China (973 Program) (Grant No. 2012CB316300).

References

[1] A. Asuncion, D. Newman, UCI machine learning repository 2007. URL <http://www.ics.uci.edu/mlearn/MLRepository.html>.
[2] J. Bacardit, Pittsburgh genetics-based machine learning in the data mining era: representations, generalization, and run-time. Ph.D. thesis, Ramon Llull University, 2004.
[3] J. Bacardit, N. Krasnogor, Smart crossover operator with multiple parents for a pittsburgh learning classifier system, in: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation, GECCO '06, 2006, pp. 1441–1448.
[4] J. Bacardit, N. Krasnogor, Performance and efficiency of memetic pittsburgh learning classifier systems, Evolutionary Computation 17 (3) (2009) 307–342.
[5] E. Bernadó-Mansilla, J.M. Garrell-Guiu, Accuracy-based learning classifier systems: models, analysis and applications to classification tasks, Evolutionary Computation 11 (3) (2003) 209–238.
[6] C. Bishop, Pattern Recognition and Machine Learning, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.
[7] M.V. Butz, Rule-Based Evolutionary Online Learning Systems, Springer, 2006.
[8] M.V. Butz, D.E. Goldberg, K. Tharakunnel, Analysis and improvement of fitness exploitation in XCS: bounding models, tournament selection, and bilateral accuracy, Evolutionary Computation 11 (3) (2003) 239–277.
[9] M.V. Butz, M. Pelikan, X. Llorà, D.E. Goldberg, Automated global structure extraction for effective local building block processing in XCS, Evolutionary Computation 14 (3) (2006) 345–380.
[10] K.A. DeJong, W.M. Spears, D.F. Gordon, Using genetic algorithms for concept learning, Machine Learning 13 (2-3) (1993) 161–188.
[11] W. Dong, X. Yao, Unified eigen analysis on multivariate Gaussian based estimation of distribution algorithms, Information Sciences 178 (15) (2008) 3000–3023.
[12] S. García, A. Fernández, J. Luengo, F. Herrera, Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: experimental analysis of power, Information Sciences 180 (10) (2010) 2044–2064.
[13] D.E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.
[14] D.E. Goldberg, The Design of Innovation: Lessons from and for Competent Genetic Algorithms, Kluwer Academic Publishers, Norwell, MA, USA, 2002.
[15] G.R. Harik, F. Lobo, D.E. Goldberg, The compact genetic algorithm, IEEE Transactions on Evolutionary Computation 3 (4) (1999) 287–297.
[16] G.R. Harik, Finding multimodal solutions using restricted tournament selection, in: Proceedings of the 6th International Conference on Genetic Algorithms, 1995, pp. 24–31.
[17] G.R. Harik, F. Lobo, K. Sastry, Linkage learning via probabilistic modeling in the extended compact genetic algorithm (ECGA), in: M. Pelikan, K. Sastry, E. Cantá-Paz (Eds.), Scalable Optimization via Probabilistic Modeling, Springer, Berlin, 2006, pp. 39–61.
[18] J.H. Holland, Adaptation in Natural and Artificial Systems: an introductory analysis with applications to biology, control, artificial intelligence, The University of Michigan Press, Ann Arbor, 1975.
[19] J.H. Holland, J.S. Reitman, Cognitive systems based on adaptive algorithms, in: D.A. Waterman, F. Hayes-Roth (Eds.), Pattern-directed inference systems, Academic Press, San Diego, 1978, pp. 313–329.
[20] C.Z. Janikow, A knowledge-intensive genetic algorithm for supervised learning, Machine Learning 13 (2-3) (1993) 189–228.
[21] D. Koller, N. Friedman, Probabilistic Graphical Models: Principles and Techniques, MIT Press, 2009.
[22] N. Krasnogor, J. Smith, A tutorial for competent memetic algorithms: model, taxonomy, and design issues, IEEE Transactions on Evolutionary Computation 9 (5) (2005) 474–488.
[23] P. Larranaga, J. Lozano, Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation, Kluwer Academic Publishers., 2002.
[24] P. Moscato, On evolution, search, optimization, GAs and martial arts: toward memetic algorithms, Tech. Rep. 826, California Institute of Technology, Caltech Concurrent Computation Program, Pasadena, CA, 1989.

[25] X. Llora, K. Sastry, D.E. Goldberg, The compact classifier system: scalability analysis and first results, in: Proceedings of the 2005 Congress on Evolutionary Computation, pp. 596–603.
[26] X. Llora, K. Sastry, D.E. Goldberg, L. delaOssa, The χ-ary extended compact classifier system: Linkage learning in Pittsburgh LCS. In: Proceedings of the 9th International Workshop on Learning Classifier Systems, 2006.
[27] M. Munetomo, N. Murao, K. Akama, Introducing assignment functions to Bayesian optimization algorithms, Information Sciences 178 (1) (2008) 152–163.
[28] A. Orriols-Puig, J. Casillas, E. Bernadó-Mansilla, Genetic-based machine learning systems are competitive for pattern recognition, Evolutionary Intelligence 1 (3) (2008) 209–232.
[29] M. Pelikan, Hierarchical Bayesian optimization algorithm: toward a new generation of evolutionary algorithms, Springer, Berlin, 2005.
[30] M. Pelikan, D.E. Goldberg, E. Cantá-Paz, BOA: the Bayesian optimization algorithm. in: Proceedings of GECCO-99, 1999, pp. 525–532.
[31] M. Pelikan, D.E. Goldberg, F.G. Lobo, A survey of optimization by building and using probabilistic models, Computational Optimization and Applications 21 (1) (2002) 5–20.
[32] D.J. Sheskin, Handbook of Parametric and Nonparametric Statistical Procedures, 4th ed., Chapman & Hall/CRC, 2007.
[33] R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction, MIT Press, 1998.
[34] S.W. Wilson, Classifier fitness based on accuracy, Evolutionary Computation 3 (2) (1995) 149–175.
[35] D. Wyatt, L. Bull, A memetic learning classifier system for describing continuous-valued problem spaces, in: W. Hart, J. Smith, N. Krasnogor (Eds.), Recent Advances in Memetic Algorithms, Springer, Berlin/Heidelberg, 2005, pp. 355–395.
[36] E. Yu, P. Suganthan, Ensemble of niching algorithms, Information Sciences 180 (15) (2010) 2815–2833.