Neurocomputing 113 (2013) 105–121

Contents lists available at SciVerse ScienceDirect

Neurocomputing

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

Effective search for genetic-based machine learning systems via estimation of distribution algorithms and embedded feature reduction techniques

Jiadong Yang [a], Hua Xu [b,*], Peifa Jia [b]

[a] Jike.com, Beijing 100020, China
[b] 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 11 July 2011
Received in revised form
28 December 2012
Accepted 1 January 2013
Communicated by M. Chetty
Available online 26 February 2013

Keywords:
Genetic-based machine learning systems
Estimation of distribution algorithms
Features reduction
Evolutionary computation

ABSTRACT

Genetic-based machine learning (GBML) systems, which employ evolutionary algorithms (EAs) as search mechanisms, evolve rule-based classification models to represent target concepts. Compared to Michigan-style GBML, Pittsburgh-style GBML is expected to achieve more compact solutions. It has been shown that standard recombination operators in EAs do not assure an effective evolutionary search to solve sophisticated problems that contain strong interactions between features. On the other hand, when dealing with real-world classification tasks, irrelevant features not only complicate the problem but also incur unnecessary matchings in GBML systems, which increase the computational cost a lot. To handle the two problems mentioned above in an integrated manner, a new Pittsburgh-style GBML system is proposed. In the proposed method, classifiers are generated and recombined at two levels. At the high level, classifiers are recombined by rule-wise uniform crossover operators since each classifier consists of a variable-size rule set. At the low level, single rules contained in classifiers are reproduced via sampling Bayesian networks that characterize the global statistical information extracted from promising rules found so far. Furthermore, according to the statistical information in the rule population, an embedded approach is presented to detect and remove redundant features incrementally following the evolution of rule population. Results of empirical evaluation show that the proposed method outperforms the original Pittsburgh-style GBML system in terms of classification accuracy while reducing the computational cost. Furthermore, the proposed method is also competitive to other non-evolutionary, highly used machine learning methods. With respect to the performance of feature reduction, the proposed embedded approach is able to deliver solutions with higher classification accuracy when removing the same number of features as other feature reduction techniques do.

1. Introduction

Genetic-based machine learning (GBML) systems [1], also called learning classifier systems (LCSs) [2], are machine learning paradigms in which target concepts are represented as explicit rules formulated in terms of tests for certain values of features. In order to deliver rules that correctly discriminate instances belonging to the target concept or not, GBML systems leverage evolutionary algorithms (EAs) [3] with the aim of searching over complex spaces and discovering potentially useful rules. It has been reported that GBML systems are able to yield competitive results in a wide variety of real-world classification problems compared with other non-evolutionary, highly used machine learning paradigms [4,5].

Since rules in GBML are improved iteratively by a search mechanism based on GAs, the performance of GAs plays a decisive role in the efficiency of the whole system. As the building block hypothesis suggests, GAs work by means of mixing pieces of multiple over-average solutions of sub-problems (called building blocks, BBs) and combining BBs to compose solutions. However, traditional recombination operator, such as crossover, often breaks BBs rather than mixes them effectively. And when handling problems with BBs spreading all over the solutions, GAs perform very poorly [6]. For Michigan GBML systems, it has been pointed out that standard recombination operators perform fixed and problem-independent recombination; hence they do not assure an effective evolutionary search in many sophisticated problems that contain strong interactions between BBs. More specifically, standard crossover operators can frequently disrupt important combinations of features, which often results in poor performance [7]. In the new generation of Pittsburgh GBML, called GAssist [14], crossover point can be set to any position in

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

0925-2312/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.neucom.2013.01.014

the binary string of the classifier during the classifiers recombination. Since a classifier in GAssist [14] contains several rules and each rule represents an entire solution for the classification problem, recombination in a single rule or between two rules occurs indiscriminately when crossover is executed. Therefore, classifiers evolution in Pittsburgh GBML should be divided into rule-wise and rule set-wise two levels. In order to protect interactions between features from being broken, recombinations in each level should be guided according to the problem at hand. Otherwise, the evolutionary search may be misled away from the global optimum and the performance of the whole system is degraded.

On the other hand, in many real-world complicated classification tasks, all features that characterize a data point may not have the same impact with regard to its classification. Some of the features may be redundant while some others may cause confusion or noise during the learning phase. For GBML systems, redundant features incur unnecessary matchings between classifiers and instances, and thus increasing the computational cost of the systems without any benefits, since matching can occupy up to the 65–85% of the overall time [8]. More specifically, GAssist has to check values of all features in the classification problem to judge whether each rule matches with one instance. It means that the number of features directly impacts the computational cost. Therefore, with respect to GBML systems, removing redundant features during the evolution of classifier not only improves the prediction accuracy but also decreases the computational cost of the learning.

In this paper, we propose a method to deal with the two problems mentioned above in an integrated manner by means of estimation of distribution algorithms (EDAs) [9,10] and an embedded feature reduction technique. In the proposed method (called BOA based Pittsburgh learning classifier systems, pLCS_BOA), we first divide the recombination of classifiers into two levels: rule-wise level and rule set-wise level. At rule set-wise level, classifiers are recombined by exchanging all bits in single rules to keep their semantics, since each classifier consists of a variable number of rules to encode an entire candidate solution. At rule-wise level, single rules contained in each classifier are produced by sampling Bayesian networks which characterizes global statistical information extracted from promising rules found so far in the search space. According to the global statistical information provided at rule-wise level, an embedded feature reduction approach (called pLCS_BOA_FR) is proposed to remove redundant features incrementally following the evolution of rule population. More specifically, our method first detects candidate features to be removed according to the proportion of rules that are independent to the feature in the rule population. Then, Markov blankets of each feature is calculated in the context of Bayesian networks built in the process of rule evolution. After calculating Markov blankets, we identify the relevancies between features and determine features to be actually removed or not.

Experiments are carried out on artificial problems and real world binary classification problems to evaluate the proposed method. According to the results, pLCS_BOA and pLCS_BOA_FR are able to deliver solutions with higher classification accuracy compared with GAssist, which is the new generation of Pittsburgh-style GBML systems [14], while reducing the computational cost in terms of the number of generations and match operators totally executed. Furthermore, solutions achieved by pLCS_BOA and pLCS_BOA_FR are also competitive to other non-evolutionary machine learning methods with respect to the classification accuracy. And when removing the same number of features as other feature reduction techniques do, the proposed pLCS_BOA_FR method produces classifiers with higher classification accuracy.

The remainder of this paper is organized as follows: in the next section, background materials are introduced. Section 3 describes the evolution of rule and rule-set in the proposed method. And the proposed embedded features reduction method is presented in Section 4. The experimental setup is illustrated in Section 5, followed by Section 6, which shows the experimental results on artificial and real world problems, respectively. Section 7 discusses the merit of the proposed method and analyzes the parameter sensitivity. Finally, the paper is summarized in Section 8.

2. Background

2.1. Genetic-based machine learning systems

Genetic-based machine learning (GBML) systems [1], which were first invented by Holland and called learning classifier systems (LCSs) in [2], have emerged as an appealing alternative for real world classification problems [4]. In GBML systems, the target concept is presented as a set of rules called classifiers, and EAs are employed as search mechanisms in order to evolve the population of classifiers. According to the chromosome codification, the research on GBML systems has been classically conducted from two perspectives: Michigan-style and Pittsburgh-style.

Michigan-style GBML systems, initially defined as cognitive systems [15], combine reinforcement learning [16] with GAs to evolve populations of accurate rules. The population in Michigan-style GBML systems is made up of a single rule-set (see Fig. 1(a)), in which each rule is encoded in a single chromosome and acts in

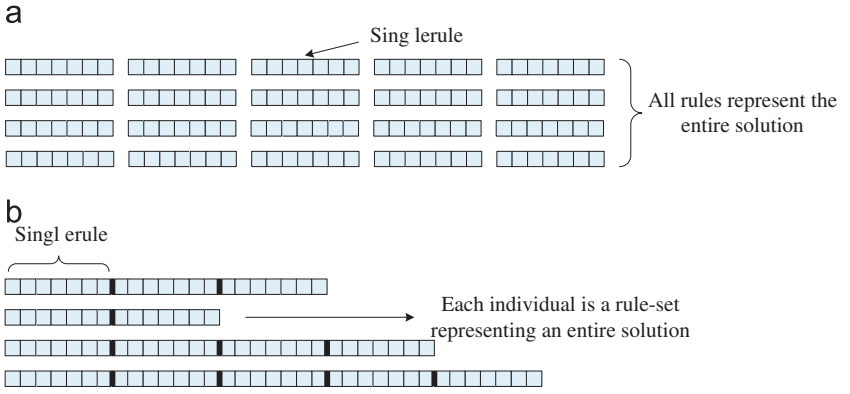

Fig. 1. Population in Michigan and Pittsburgh style GBML systems: (a) in Michigan style GBML systems, all rules represent the entire solution and (b) in Pittsburgh style GBML systems, each individual is a rule set representing an entire solution.

some parts of the problem domain. In order to achieve an ideal population, the two separate components in Michigan-style GBML systems work with different goals. The reinforcement component exploits the incoming reward to estimate the action values in each subproblem so as to identify the best classifiers in the population. And GAs improve the current solution by means of exploring new rules. In this family of GBML systems, XCS [17], which was originally introduced by Wilson in [18], is regarded as the most prominent Michigan-style GBML systems to-date.

Pittsburgh-style GBML systems 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. 1(b)). Individuals in the population compete among themselves according to fitness functions, which consider different aspects such as the prediction accuracy and the generality of the rule-set, 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 main procedure of Pittsburgh-style GBML systems is listed in Algorithm 1. The first successful developments of Pittsburgh-style GBML systems for supervised learning were GABIL [19] and GIL [20]. A new generation Pittsburgh-style GBML derived from GABIL can be found in GAssist system [14,21], which improves the generalization capacity of the model and proposes representation for real-valued attributes.

Algorithm 1. Basic procedure of Pittsburgh style GBML systems.

```
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
```

2.2. Estimation of distribution algorithms

Estimation of distribution algorithms (EDAs) [9,10], also called probabilistic model building genetic algorithms (PMBGAs) [6], have been recently identified as a promising paradigm in the field of EAs. Unlike traditional EAs that rely on traditional genetic operators (e.g., mutation or crossover), EDAs create offspring via sampling a probabilistic model that has been 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 is served as the distribution of promising solutions. In the latter, conditional dependency chains or networks are taken into consideration so as to model multivariate interactions.

As one of state of the art multivariate EDAs, the BOA [11,13] uses Bayesian networks [12] to capture the dependencies between decision variables of the problem with the aim of generating 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 [13].

2.3. Classifiers recombination in GBML systems via EDAs

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

In Michigan GBML systems, Butz et al. [7] combined XCS with two EDAs paradigms, including the Extended Compact Genetic Algorithm (ECGA) [22] and BOA [13], to ensure effective exploration for problems in which features strongly interact and standard variation operators lead to poor performance. The method can detect and propagate lower-level dependency structures effectively without any information about these structures given in advance.

The Compact Classifier System (CCS) [23] and its extension, called χeCCS [24], are integrations of EDAs within the framework of Pittsburgh GBML systems. In CCS, sets of perturbed probability vectors, which represent rules population, evolve by means of the compact Genetic Algorithm (cGA) [25]. The latter evolves a population of rules by means of the model building and recombination mechanisms of ECGA. Both methods aim at determining the minimum set of rules that creates a maximally general solution.

The works mentioned above are the ones most related to ours. However, there are obvious differences. First, our method focuses on Pittsburgh GBML systems, while works proposed in [7] paid attention to Michigan GBML systems. On the other hand, methods in [23,24] adopted cGA and ECGA to guide the combination of rules, while our method leverages BOA. More essentially, our method explicitly divides the recombination of classifiers into two levels to improve the efficiency for exploring optimal solutions. Furthermore, our method is evaluated based on complicated problems in real world, instead of only multiplexer problems handled in [23,24].

2.4. Feature reduction techniques and its application in GBMLs

Since real-world data always contain redundant and noisy features, many feature reduction techniques have been proposed to provide better understandings of the underlying structure of data sets. According to the their interaction with the classifier construction, feature reduction techniques can be classified into three categories: filter methods, wrapper methods and embedded methods [26]. Filters methods select features by means of scoring them according to the relevance and perform independently of the chosen classifier. Wrappers utilize the learning machine of interest as a black box to rank subsets of features according to their predictive power. And the embedded methods perform feature selection in the process of training and are usually specific to given learning machines. The most related ones to the proposed method include [27,28], etc., which leverage the Markov blankets in the context of Bayesian networks as well.

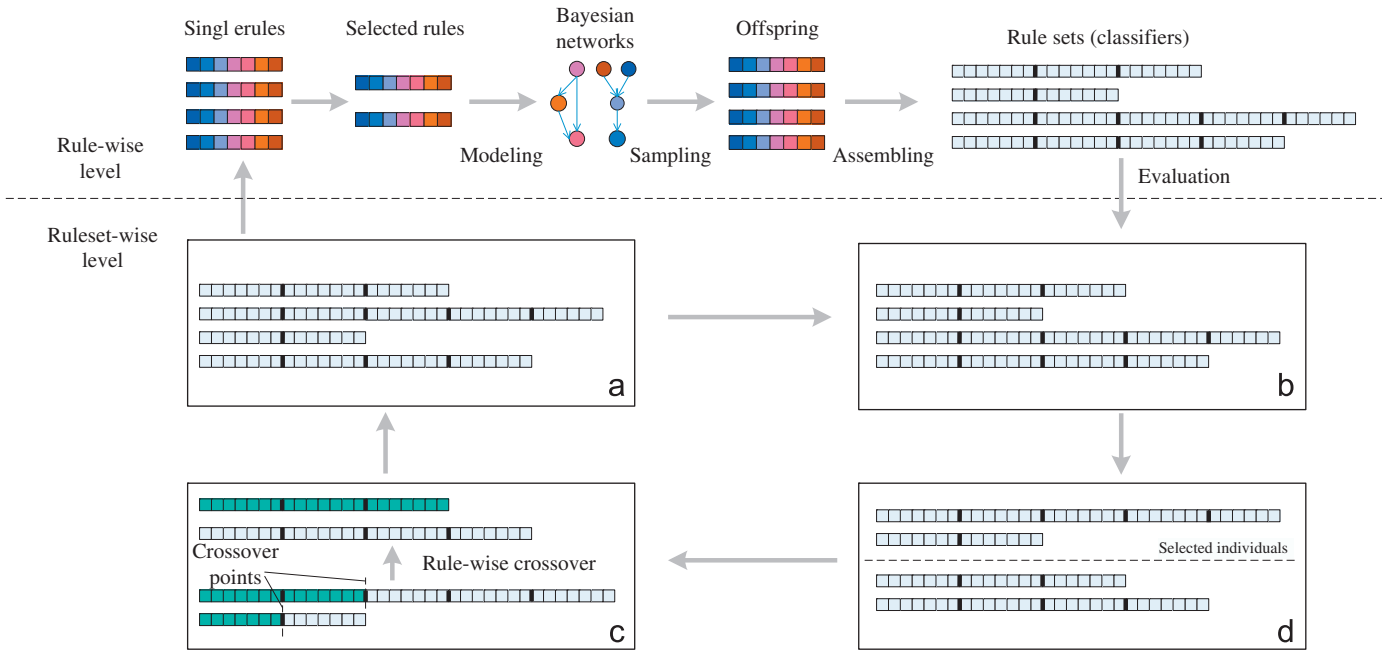

Fig. 2. In the proposed method, the procedure of classifier evolution includes two levels. The main steps at the rule-wise level and rule set-wise level are represented above and below the dashed line, respectively.

However, those methods employ and view feature reduction as a preprocessing step, which means redundant features are removed independently of the later processing. Consequently, they are categorized into filters [26]. While in our method, the component of feature reduction is embedded in the process of classifier learning; hence, redundant features are removed gradually following the evolution of rule population. Given the discussion above, the proposed method is totally different from those related works.

The removal of features, which are insufficiently correlated to the given task, is also tackled within GBML systems. With respect to Michigan GBML systems, a co-evolutionary multi-population XCS, called CoXCS, was proposed in [29,30]. In CoXCS, features contained in the data set are divided into several subsets. The classifiers are also partitioned into several isolated sub-populations, each of which accumulates and specializes its expertise in the corresponding feature subspace. In the family of Pittsburgh GBML systems, [19] represented a biased mutation operators, which altered the logical relationship between chosen features with the aim of ignoring the redundant features. The proposed method is inspired by such a work. However, since the rule-wise evolution is executed by means of EDAs, there are no mutation operators in the proposed method. In [31] irrelevant features removal and learning are performed in a concurrent and integrated manner, which is the same as our method does. However, methods proposed in [31] removes irrelevant features for each rule, which means rules use different sets of features to represent the knowledge. While, all rules in our method use the same set of features during the evolution. Furthermore, we leverage the probabilistic models built during rule-wise evolution to remove redundant features. Consequently, there are essential distinguishing characteristics between our work and the method proposed in [31].

3. Classifiers evolution in the proposed method

In this section, the procedure of classifier evolution in our method is described. In order to improve the efficiency of rule recombination, the proposed method integrates EDAs into the basic frame of Pittsburgh GBML systems, which is listed in Algorithm 1. As noted in Section 2.1, each individual (i.e., classifier) is represented as a rule set to encode an entire solution, and the number of rules in each set is not identical to each other. 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 of classifiers is performed at two levels based on EDAs. The first one, called rule-wise level, deals with rules in each individual. And the other, called rule-set wise level, focuses on rule sets (i.e., individuals). Fig. 2 represents the framework of our method. The main steps at the two levels are represented just 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 evolution of classifiers at two levels will be discussed respectively.

3.1. Knowledge representation

In this paper, we use the method in GABIL [19] and GAssist [14] to represent candidate solution of the problem. More specifically, each solution, also called individual, is encoded as a variable-length disjunctive set of possibly overlapping classification rules, i.e., in disjunctive normal form (DNF) to represent the unidentified concept, which is defined as

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

where each rule R_i includes two part: the condition part and the action part. The first part, defining the region of the feature space where the rule could be applied, consists of one or more terms of tests involving feature values. And the action part specifies the concept (i.e., the label of classification) to be assigned to the example which is covered by the condition part. With respect to the condition part, it is defined as a conjunctive normal form of several sub-conditions

$$R = (C_1 \wedge C_2 \wedge \cdots \wedge C_n) \rightarrow action \tag{2}$$

where the sub-condition C_i, which is used to check the corresponding feature f_i, is a subset of all possible values of f_i, namely

$$C_i \subseteq S_{f_i} \tag{3}$$

where S_{f_i} denotes the set containing all possible values of f_i. An example is covered by C_i for f_i if the value of f_i in the example is a member of C_i. Within nominal features, the sub-condition could be encoded as a fixed-length string of k bits: $b_1^i b_2^i \ldots b_k^i$, where $k = |S_{f_i}|$ denotes the number of possible values f_i could take. b_j^i is equal to 1 when it is included in C_i, and b_j^i is set to 0, otherwise. When dealing with numerical features, they could be transformed into nominal ones via certain discretization algorithm such as Chi2 [32]. Taking all features together, each rule is encoded as a binary string with the length equal to l, which is defined as

$$l = \sum_{i=1}^{n} |S_{f_i}| \tag{4}$$

For example, suppose we have three features $\{X,Y,Z\}$, and the value of each could be taken from $\{X_1,X_2\}$, $\{Y_1,Y_2,Y_3\}$, and $\{Z_1,Z_2,Z_3,Z_4\}$, respectively. Then the rule $R_{example}$ $11|001|1001 \rightarrow 1$ means that "if the value of first feature is equal to X_1 or X_2, the value of feature Y is set to Y_3, and the value of the last feature is equal to Z_1 or Z_4, then the rule predicts class 1".

From the discussion above, it has been known that each individual in Pittsburgh GBML systems consists of variable number of rules. Therefore, the evolution of classifiers may be executed at two levels, the one with the rules contained in individuals and the one with the rule sets (i.e., classifiers). We will describe them in the following two subsections, respectively.

3.2. Rule-wise evolution

In the following two subsections, we will first describe the fitness function used to evaluate each single rule, and then present the procedure in which the BOA is responsible for rule evolution.

3.2.1. Fitness of single rules

It is already known that each rule has its own domain defined by its condition part. Therefore, the accuracy should be defined within such a domain. A straightforward way to measure such a metric 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, R_j outperforms R_i. Therefore, another metric, called negative accuracy, has to be taken into consideration to complement the positive accuracy. Such a metric reflects the accuracy of the rule with respect to problems outside the domain.

Consequently, in order to achieve a competent classifier system, rules evolve towards maximal positive and negative accuracy simultaneously. And the fitness of each rule in our method needs to describe both the metrics, which is defined as below

$$f(R) = 0.5 \cdot \alpha_p(R) + 0.5 \cdot \alpha_n(R) \tag{5}$$

where the $\alpha_p(r)$ and $\alpha_n(r)$ represent positive and negative accuracy respectively. Their definitions are described as

$$\alpha_p(R) = N_p(R)/N_c(R) \tag{6}$$

$$\alpha_n(r) = N_n(r)/(N - N_c(r)) \tag{7}$$

where $N_p(R)$ and $N_n(R)$ denote the number of positive instances correctly classified 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. The fitness of a single rule is the weighted sum of its positive accuracy and negative accuracy, in which $\alpha_p(R)$ and $\alpha_n(R)$ produces bias toward the accuracy inside and outside the domain respectively.

The fitness of a rule is computed during the evaluation of the rule set containing the rule More specifically, for a certain training instance, all rules in the rule set are compared to the instance. Consequently, it is known whether a rule in the rule set gives a right or wrong prediction for the training instance. And the fitness of each rule is computed.

3.2.2. Evolution of single rules via BOA

The first issue in applying BOA is the choice of promising rules to build the Bayesian network. One of the straightforward strategies is selecting rules from individuals with higher fitness. However, such an approach has three drawbacks. First, if more than one rule match the instance, the individual has to choose only one to determine its action (discussed in Section 3.3.2). In such a procedure, the fitnesses of rules not selected are lower than those of the selected ones. Therefore, not all rules in individuals with higher fitness actually perform well. Next, rules in these individuals may not match any training instances. Including these rules may perturb the Bayesian network. Additionally, rules with higher fitness may be contained in individuals with lower fitness. Only because the performances 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. (5), regardless the individual they are contained in, to build Bayesian networks.

According to the description in Section 3.1, the condition part of each rule could be represented as a binary code of fixed length. With a set of binary codes to represent promising single rules, which are selected from all individuals, a data set with dimension equal to the length of the code (i.e., l, see Eq. (4)) is collected. Then Bayesian networks, in which each node denotes a corresponding bit in rules, are built based on the data set. With the aim of learning the 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* [13] is performed to add dependencies gradually into the network. More specifically, the Bayesian–Dirichlet (BD) metric [12] 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 the dataset. Then, new rules are generated just by sampling the Bayesian networks. In this paper, we choose the forward sample technique [12].

Algorithm 2. Rules evolution via BOA.

```
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

Another problem during rule evolution is to maintain diversity in the population. As noted in Section 3.1, each rule has its own functional region, and niche technique should be taken into 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 [33], such as crowding, fitness sharing and clearing, etc. In our method, the Restricted Tournament Replacement (RTR) [34] is adopted and the size of random subset in RTR is set to the length of the bit representing the rule according to the suggestion in [13]. Given the discussion above, the whole procedure of rule evolution via BOA is described in Algorithm 2.

3.2.3. Rules assembling

According to the discussion in Section 3.1, it is known that new rules have to be assembled in individuals to represent an entire solution in Pittsburgh GBML systems. In the proposed method, we first create a new population of individual (called P'_{rs}), whose size is equal to the current one, say P_{rs}. Each individual in the new population consists of rules that are randomly picked from the offspring produced by BOA. The size of each individual in P'_{rs} is set to the value of the corresponding one in P_{rs}. Then, its the fitness is computed according to Eq. (8). Next, individuals in P'_{rs} and P_{rs} are ranked together according to their fitness. Finally, a new rule set population is achieved based on simple elitist replacement. The new rule set population 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     Delete all rules in I
6     for i=1 to size of I do
7       R ← Random one in P_r
8       Add R into I
9     end for
10  end for
11  Evaluate P'_rs
12  Replace P_rs with P'_rs
```

3.3. Rule set-wise evolution

3.3.1. Evaluation of rule set

Given the description of knowledge representation in Section 3.1, it is possible that more than one rule match the instance when the rule set do the prediction. In such a situation, the rule set has to choose the classification of only one rule as its proposed solution, if their classifications conflict with each other. Compared with GAssist [14], a different way is adopted in the proposed method. More specifically, for each rule set in GAssist, match operators are executed until the first matched rule is found, and it is adopted as the solution. Then the procedure of prediction terminates. In the proposed method, all rules are compared to the instance and their corresponding fitnesses are computed. Then, the rule with the highest positive accuracy (see Eq. (6)) is chosen as the prediction in the proposed method, because the instance is located in the region of rules and the competition between them has to be restricted in their regions. Given the definite prediction when matching with instances, the fitness of each individuals is defined as

$$f(I) = N_r(I)/N \tag{8}$$

where $N_r(I)$ and N specify the number of instances the individual correctly classifies and the number of all the training instances, respectively.

Although the fitness of the rule and rule set in the proposed method are different, the fitness of a rule is computed during the evaluation of the rule set containing the rule. For a certain training instance, all rules in the rule set are compared to the instance. The fitness of each rule is computed according to Eqs. (5)–(7) when the rule set is evaluated via predicting for all training instances in each iteration.

3.3.2. Evolution of rule set

According to the description in Section 3.1, each individual consists of a variable-size rule set. Since all rules have the same length l (see Eq. (4)), 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 the proposed method, since BOA guides 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 frame (d) in Fig. 2). Because the rule-wise uninform crossover prevents rule disruption but maximizes rule mixing or exchange, it represents an ideal crossover operator to use. It is necessary to point out that any information about the structure of the problem are not have to be given in advance to execute the rule-wise uniform crossover, because the crossover points could only be set to the position between ilth bit and $(il+1)$th bit, where i is an integer smaller than n.

3.4. Integration

Since building Bayesian networks is computationally expensive, we do not perform rule evolution in each generation. Instead, the proposed method uses a sporadic model building strategy; that is, the model is updated once in every few Pittsburgh model iterations. The strategy is demonstrated as one of the efficiency enhancement techniques for EDAs [10]. Consequently, rules are updated only once in several generations. In contrast, the rule-wise crossover on individuals is executed in every generation. In summary, the whole procedure of the proposed pLCS_BOA method is listed in Algorithm 4. Lines 5–15 represent the procedure of rule-wise combination via BOA and the parameter *inte* denotes the interval of rule-wise recombination, while lines 16–20 describe the procedure of rule set-wise recombination.

Algorithm 4. The procedure of the pLCS_BOA.

```
1   t ← 0
2   P_t ← Generate a random population of rule sets
3   Evaluate rule sets in P_t
4   repeat
5     if mod(t,inter) = 0 then
6       R ← all single rules in P_t
7       S_r ← Select promising rules from R
8       B ← Build Bayesian network from S_r
9       O_r ← Sample from B
10      Evaluate O_r
11      R' ← Replace R with O_r by RTR
12      P' ← Assemble R' into rule sets
13      Evaluate P'
```

$P_t \leftarrow$ Replace P_t with P'
end if
$S_{rs} \leftarrow$ Select promising rule sets
$O_{rs} \leftarrow$ Rule-wise crossover on S_{rs}
Evaluate O_{rs}
$P_{t+1} \leftarrow$ Replace P_t with O_{rs}
$t \leftarrow t+1$
until Stopping criterion is met
The best rule set is treated as the final solution

4. Embedded features reduction technique

In order to remove features that are insufficiently correlated to the classification, an embedded feature selection method is presented in this section. Rather than relying on the property of the original data, the proposed pLCS_BOA_FR method leverages the statistical information contained in the rule population. More specifically, pLCS_BOA_FR first detects candidate features to be removed according to the ratio of the number of rules, which are independent to the feature, to the size of rule population. Then, Markov blankets of each feature is calculated in the context of Bayesian networks to identify the relevancy between features and determine which features are actually removed. In the following, we will describe the two parts in details respectively below.

4.1. Feature ranking

Generally, a feature is said redundant in a rule if its corresponding value in the rule could not make much contribute to the classification. Given the description about the knowledge representation in Section 3.1, we have known that the sub-condition C on feature f in a rule R involves the matching with a subset S_f of S, where S includes all the possible values of f. If a certain value is a member of S_f, its corresponding bit is encoded as 1 s. Otherwise, it is encoded as 0. Therefore, all bits in the sub-condition are equal to 1 when $S_f = S$ and the sub-condition covers any value of f. It just means that taking any value of f satisfies the sub-condition. Consequently, any value of f gives the same contribution to the classification. In such a situation, the feature f is called redundant for rule R and we call R is independent of f. See the example in the last paragraph in Section 3.1, feature A is redundant for the rule $R_{example}$.

Only a single rule that is independent of a certain feature f does not make any sense to the whole systems since rules in Pittsburgh GBML systems are arranged as a population. If feature f is redundant in most rules, f may make few contributions for the system to deliver the final solution although it is rare that all bits corresponding to f are equal to 1 in all single rules. Consequently, the redundancy of feature f, say r_f, could be measured with respect to the rule population, which is defined as

$$r_f = \alpha_f / N \tag{9}$$

where N and α_f denote the number of promising rules and the ones that are independent of the feature f in those promising rules, respectively. Here, we define feature redundancy regarding the promising rules rather than all the rules. Obviously, its computational cost is $O(N)$.

After defining the redundancy, features could be ranked accordingly. When the redundancy is above some threshold (i.e., $0 \le \tau \le 1$), the corresponding feature could be viewed as the candidate to be dropped. Consequently, the list L of candidate features to be removed is achieved (i.e., $L = \{f \mid r_f > \tau\}$), in which the features are sorted according to their redundancy decreasingly.

4.2. Feature relevance computation

The aim of feature reduction is to improve the performance of the system (e.g., the accuracy of classification) via removing irrelevant, redundant, or noisy features. In another word, features could not be removed at the cost of the performance of the system. In order to prevent the degradation of the classification, the dependencies between features have to be identified. And we want to find the most relevant features to the ones removed, and delete them from the candidate list in order to decrease the adverse effect of feature reduction as much as possible. Consequently, for each feature, its actual removal or not is determined in this stage.

To detect the relevancies between features, one of the most popular methods is to compute the Markov blanket (MB). The Markov blanket of a feature f is a minimal feature subset conditioned on which all other features are probabilistically independent of f. In the context of Bayesian networks [12], the Markov blanket of f includes its direct causes, direct effects, and spouses (i.e., other direct causes of its direct effects). Therefore, we could find the Markov blanket of each feature after the Bayesian network is learned.

Given the description about the rule representation, each single rule encodes all the possible value of each feature and the length of each single rule is equal to l (see Eq. (4)). In consequence, it is straightforward to build the Bayesian network which contains l nodes and each node denotes one possible value of the corresponding feature according to promising single rules. For example, assume that the structure of the Bayesian network is depicted in Fig. 3. Then, the Markov blanket of value A_1 includes B_2, C_1, E_1 and E_3 and the Markov blanket of value A_2 includes B_3, C_4 and F_3.

Here, another problem rises: the Bayesian network built only represents the dependencies between possible values of each feature rather than the one between features themselves. Therefore, dependencies between possible values belonging to the same feature have to been gathered. To group the Markov blankets of all possible values belonging to the same feature, the most straightforward way is to joint all corresponding Markov blankets together. However, the result of is still a set of possible values. In consequence, the final Markov blanket in our approach consists of features whose possible values are included in the joint. Also see Fig. 3, since the Markov blanket of value A_1 includes

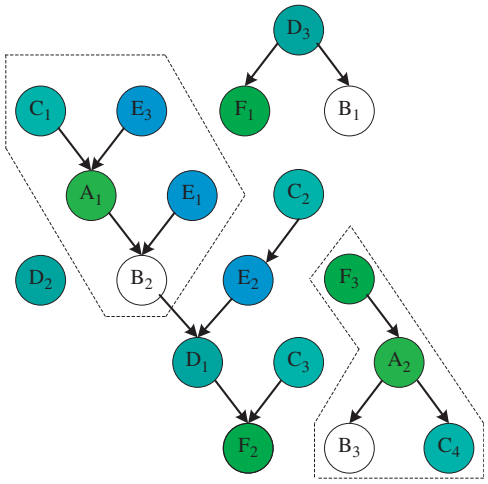

Fig. 3. Nodes with the same capital letter and different subscripts denote different values of the same features, e.g., feature A includes two possible values A_1 and A_2. Their corresponding Markov blankets are embraced in the dashed frames, respectively. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this article.)

B_2, C_1, E_1 and E_3 and Markov blanket of value A_2 contains B_3, C_4, F_3, the Markov blanket of feature A consists of B, C, E and F. More formally, the Markov blanket of feature f is defined as

$$MB(f) = \{f' \mid S_{f'} \cap MB_{joint}(f) \neq \emptyset\} \tag{10}$$

where S_f refers to the set including all possible values of feature f, and $MB_{joint}(f) = \bigcup_{v_i \in S_f} MB(v_i)$ denotes the union including values in each Markov blanket of v_i in S_f.

According to the description above, the basic route of feature reduction component is listed in Algorithm 5. Lines 1–2 represent the step in which features are ranked according to their redundancies, and the other lines describe the procedure how redundant features are actually removed.

Algorithm 5. Procedure of embedded feature reduction component.

```
1   Compute r_{f_i} for each feature f_i
2   L ← {f | r_f > τ})
3   if L ≠ ∅ then
4     Sort L descendingly according to r_f
5     B ← the Bayesian network modeling features
6     repeat
7       f' ← the first element in L
8       Compute MB_{f'} according to B
9       Remove f'
10      Delete all features in the MB_{f'} from L
11    until L = ∅
12  end if
```

4.3. Integrating feature reduction component into GBML systems

Given the main principle discussed above, the component of feature reduction has to be integrated into the original GBML system. There are three problems to solve.

First, we have to determine when to perform the component of feature reduction. The component of feature reduction includes two phases: (1) computing the redundancy, in which candidate features to be removed are chosen, and (2) computing the relevancies, in which redundant features to be removed are finally determined. The computational cost of the first phase is $O(N)$, where N denotes the size of population. It is just the same as the cost of rule-set wise evolution in each generation. With respect to the second phase, the cost is remarkable if building Bayesian networks exclusively for the feature reduction component. Consequently, we make use of Bayesian networks built in the process of the rule-wise evolution to calculate the Markov blanket. And the feature reduction component is carried out just following the rule-wise evolution. Since rule-wise evolution is performed once in several generation according to the description in Section 3.2, the feature reduction component is executed periodically as well.

The next issue is to determine the time when actually removing redundant features after they are detected. The first choice is to remove them just after they are detected. In such a situation, features are removed before new rules are generated by BOA. It is a feasible strategy because the Bayesian network has been built and Markov blankets could be calculated accordingly at that time. And in such a way, the cost of the rule-wise evolution is reduced greatest for redundant features do not involve in any subsequent steps in rule-wise evolution. However, the strategy may disturb the procedure of new rules generation. The reason lies in that new rules are created by just forward sampling the Bayesian network which models the promising rules (see the discussion in Section 3.2). If the redundant features are ignored at this time, the topological ordering implied in the Bayesian network may be destroyed and the conditional probabilities in the model may be broken, which degrades the performance of new rules obtained by sampling the model. Therefore, in the proposed method, redundant features are involved in the procedure of new rules generation, during which the Bayesian network is not changed. Then, they are removed just by deleting the corresponding bits in all rules.

Additionally, when redundant features are detected and actually removed, rules may become premature and converge to suboptimums in a certain degree. If the corresponding bits are removing immediately and rules evolve subsequently, the performance of rules may not be improved, or even degrades in some worse cases. To prevent the adverse effect mentioned above, some rules have to be added to maintain the diversity. With such an aim, additional rules are generated randomly with a uniform distribution over features remained so far. And these additional rules are incorporated with rules generated by sampling Bayesian networks.

In summary, the basic procedure of pLCS_BOA_FR is listed in Algorithm 6. The difference between the procedures of pLCS_BOA and pLCS_BOA_FR (shown in Algorithm 4) lies in that the latter incorporates the embedded feature reduction component, which is shown in line 5 and the details have been described in Algorithm 5. And the other parts are the same. The details of lines 4 and 7 in Algorithm 6 have been represented in lines 5–15 and lines 16–20 in Algorithm 4, respectively.

Algorithm 6. The main procedure of pLCS_BOA_FR.

```
1   t ← 0
2   repeat
3     if mod(t,inte) = 0 then
4       Rule evolution via BOA
5       Feature reduction according to promising rules
6     end if
7     Rule set evolution
8     t ← t+1
9   until Stopping criterion is met
10  The best rule set is treated as the final solution
```

5. Experimental setup

5.1. Artificial problems

To evaluate the performance of our method, we first build artificial problems by means of adding redundant features into the multiplexer problem, which has been the subject of study for a long time in learning classifier system research [35]. In general, the input to the Boolean multiplexer function is a binary string of length $k+2^k$ of the form

$$A_{k-1} \ldots A_1 A_0 D_{2^k-1} \ldots D_1 D_0 \tag{11}$$

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, $f_m(100010) = 1$ and $f_m(000111) = 0$. In the rest of the paper, n bits multiplexer problem is denoted as MP-n.

With the aim of evaluating the performance of feature reduction of pLCS_BOA_FR, we construct redundant multiplexer problem (RMP) via appending some bits into the original multiplexer problem. Consequently, the input of RMP with r irrelevant bits is listed as below

$$A_{k-1} \ldots A_1 A_0 D_{2^k-1} \ldots D_1 D_0 R_{r-1} R_{r-2} \ldots R_0 \tag{12}$$

Table 1
Properties of the redundant multiplexer problems (RMPs).

ID	#Instances	#Features	#Redundant features
MP11	2048	11	0
RMP11-11	2048	22	11
RMP11-22	2048	33	22
RMP11-44	2048	55	44
RMP11-88	2048	99	88
RMP11-176	2048	187	176

Table 2
Properties of UCI data sets.

ID	Name	#Instances	#Features		
			#Categorical	#Numerical	#Total
aus	Australian credit	690	8	6	14
che	Chess	3196	36	0	36
cre	Credit approval	690	9	6	15
dia	Diabetes	768	0	8	8
ger	German	1000	0	24	24
hab	Haberman	306	0	3	3
heart	Heart	270	8	5	13
hep	Hepatities	165	13	6	19
hill	Hill-Valley	1212	0	100	100
ion	Ionosphere	351	0	34	34
liv	Liver disorders	345	0	6	6
monk	Monk's problems	432	6	0	6
mush	Mushroom	8124	22	0	22
musk1	Musk Version 1	476	0	166	166
musk2	Musk Version 2	6598	0	166	166
par	Parkinsons	197	0	23	23
tic	Tic-Tac-Toe Endgame	958	9	0	9
tra	Transfusion	748	0	4	4
vote	Voting records	425	16	0	16
wis91	Wisconsin (January 1991)	699	0	9	9
wis95	Wisconsin (November 1995)	569	0	30	30

where bit R_i ($i=0,1,\ldots,r-1$) denotes the redundant bits. The r redundant bits are set as 0 or 1 at random, but their values do not affect the output. In other word, the output of RMP is determined by the "address" and "data" bits. In the reminder of the section, we add $r=\{1,2,4,8,16\}\times 11$ bits to the 11-bit multiplexer function to build RMPs, respectively. For the sake of brevity, the 11-bit multiplexer with n redundant bits is denoted as RMP-11-n. The properties of data sets to evaluate RMPs are listed in Table 1.

5.2. UCI data sets

Addition to artificial problems, 21 real world binary classification problems from the UCI repository [36] are used as the benchmark data sets as well. The properties of these data sets are listed in Table 2. All experiments on artificial problems and UCI data sets are conducted following the 5-fold cross validation.

5.3. Configurations

The proposed method is implemented based on the code of classical GAssist,[1] which was developed by its authors. The corresponding parameters are presented in Table 3, which are set according to [21]. Other parameter specifications for pLCS_BOA and pLCS_BOA_FR are listed in Table 4. To deal with problem containing numerical features, we use the Chi2 [32] algorithm with the significant level equal to 0.1, 0.01, and 0.001, and the solution with the highest accuracy is treated as the final result. If not stated specifically, the result of each experimental case is the average 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.

To evaluate the proposed method, we compare it with GAssist using with the code mentioned at the footnote. Its configuration is identical to that of the proposed method (see Table 3). When handling problems with numerical features, adaptive discretization intervals (ADI) [14] with two strategies is adopted, in addition to the Chi2 algorithm used in our methods. More specifically, the ADI employs the uniform-width and uniform-frequency intervals both with 5, 10, 15, 20, and 25 bins. And the final result of each data set for GAssist is processed and selected the same as the proposed methods do.

Additionally, we compare pLCS_BOA and pLCS_BOA_FR with another four non-evolutionary machine learning methods, including Naive Bayes [37], C4.5 [38], PART [39] and SVM [40] with polynomial kernels. To run them, we use Weka platform [41] with the configuration recommended in it.

Table 3
General parameters to configure pLCS_BOA and pLCS_BOA_FR.

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 and pLCS_BOA_FR.

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
Threshold on redundancy	0.9

Table 5
Comparison of classification accuracy on RMPs.

Data	GAssist	pLCS_BOA	pLCS_BOA_FR
MP11	**100.00 ± 0.00**	**100.00 ± 0.00**	**100.00 ± 0.00**
RMP11-11	98.23 ± 1.42	98.75 ± 1.97	**98.89 ± 2.35**
RMP11-22	94.03 ± 1.39	94.20 ± 1.26	**95.29 ± 1.54**
RMP11-44	89.64 ± 1.68	89.42 ± 2.09	**91.02 ± 1.84**
RMP11-88	85.05 ± 2.01	84.93 ± 1.89	**87.79 ± 2.31**
RMP11-176	82.17 ± 2.22	82.28 ± 2.30	**85.92 ± 1.55**

[1] http://www.asap.cs.nott.ac.uk/~jqb/PSP/GAssist-Java.tar.gz.

6. Experimental results

6.1. Comparison on classification accuracy

In this subsection, the classification accuracy of the proposed method is evaluated. We first compare the proposed method with GAssist on the redundant multiplexer problems. The results are listed in Table 5, in which the best results are shown in bold font. It is obvious that pLCS_BOA_FR delivers the best classifier for each problem. And as the number of redundant bits in RMP increases, the advantage of pLCS_BOA_FR over GAssist and pLCS_BOA becomes more remarkable. Furthermore, we statistically analyze the results by means of detecting significant differences between these approaches regarding the classification accuracy for a level of significance $\alpha = 0.05$. The first comparative descriptive statistic is the p-value obtained by an upper tailed paired t-test, which is a parametric test. According to the recommendations made in [42,43], two nonparametric tests, including the sign test and the Wilcoxon paired signed ranks test [44], are employed since the results may present neither normal distribution nor homogeneity of variance. From Table 6, we could see that pLCS_BOA_FR outperforms pLCS_BOA and GAssist statistically. And there are no significant differences between pLCS_BOA and GAssist in terms of classification accuracy.

Next, we use UCI data sets as the test beds to compare the classification accuracy of the proposed methods with GAssist and four non-evolutionary machine learning methods mentioned in Section 5.3. The experimental results are shown in Table 7. We also investigate the significant differences on these results, which are listed in Table 8.

According to the significance tests, we could see that the solutions delivered by pLCS_BOA and pLCS_BOA_FR are better than the ones achieved by GAssist, since the corresponding metrics of three significance tests are all below 0.05. With respect to other four non-evolutionary methods, we could see that the advantages of pLCS_BOA and pLCS_BOA_FR over NB and PART are remarkable for all three tests present the significance. And the performance of pLCS_BOA_FR is better that C4.5 and SVM

Table 8
Significance tests of classification accuracy on UCI data sets. The level of significance is set to 0.05.

Comparison	t-Test	Sign test		Wilcoxon test		
	p-Value	Win	p-Value	W^+	W^-	p-Value
GAssist v.s. NB	0.0056	14	0.0945	182	49	0.0099
GAssist v.s. C4.5	0.0473	11	0.4999	143	88	0.1653
GAssist v.s. PART	0.0096	12	0.3320	172.5	58.5	0.0228
GAssist v.s. SVM	0.1958	11	0.4999	130	101	0.3012
pLCS_BOA v.s. NB	0.0035	16	0.0133	192	39	0.0037
pLCS_BOA v.s. C4.5	0.0211	14	0.0945	170.5	60.5	0.0269
pLCS_BOA v.s. PART	0.0012	16	0.0133	199	32	0.0018
pLCS_BOA v.s. SVM	0.1010	11	0.4999	139.5	91.5	0.1971
pLCS_BOA v.s. GAssist	0.0309	16	0.0133	184	47	0.0082
pLCS_BOA_FR v.s. NB	0.0009	16	0.0133	208	23	0.0006
pLCS_BOA_FR v.s. C4.5	0.0016	16	0.0133	187	44	0.0011
pLCS_BOA_FR v.s. PART	0.0001	17	0.0036	211	20	0.0004
pLCS_BOA_FR v.s. SVM	0.0131	15	0.0392	186	45	0.0125
pLCS_BOA_FR v.s. GAssist	0.0013	17	0.0036	204	27	0.0010
pLCS_BOA_FR v.s. pLCS_BOA	0.0094	11	0.4999	164	67	0.0443

Table 6
Significance tests of classification accuracy on RMPs. The level of significance is set to 0.05.

Comparison	t-Test	Sign test		Wilcoxon test		
	p-Value	Win	p-Value	W^+	W^-	p-Value
pLCS_BOA v.s. GAssist	0.2515	3	0.4999	9	6	0.2949
pLCS_BOA_FR v.s. GAssist	0.0170	5	0.0313	15	0	0.0155
pLCS_BOA_FR v.s. pLCS_BOA	0.0240	5	0.0313	15	0	0.0155

Table 7
Comparison of classification accuracy on UCI data sets.

Data	GAssist	pLCS_BOA	pLCS_BOA_FR	NB	C4.5	PART	SVM
aus	85.80 ± 1.42	86.23 ± 1.97	**86.59 ± 2.35**	81.59 ± 2.44	85.65 ± 2.48	80.87 ± 3.92	71.30 ± 3.75
che	96.85 ± 1.39	96.32 ± 1.26	96.29 ± 1.54	87.98 ± 1.10	99.34 ± 0.39	99.00 ± 0.55	**99.41 ± 0.46**
cre	86.43 ± 1.68	**87.42 ± 2.09**	87.32 ± 1.84	81.30 ± 2.83	87.25 ± 1.50	85.22 ± 2.09	85.65 ± 1.65
dia	73.05 ± 2.01	73.73 ± 1.89	**77.79 ± 2.31**	75.38 ± 4.77	72.39 ± 4.01	72.52 ± 4.05	77.34 ± 2.89
ger	73.17 ± 2.22	70.28 ± 2.30	71.92 ± 1.55	**74.20 ± 2.25**	71.80 ± 4.25	68.40 ± 5.07	69.00 ± 1.77
hab	74.54 ± 2.72	75.41 ± 1.84	**78.25 ± 1.82**	73.52 ± 9.11	71.25 ± 5.82	72.23 ± 7.28	71.90 ± 7.59
heart	79.32 ± 3.58	80.80 ± 4.36	80.49 ± 4.15	**83.70 ± 6.60**	73.33 ± 3.36	76.30 ± 4.79	75.56 ± 4.79
hep	83.66 ± 5.22	**85.70 ± 3.13**	84.41 ± 2.82	84.52 ± 6.99	76.77 ± 10.05	80.65 ± 7.57	77.42 ± 14.96
hill	51.54 ± 1.72	53.01 ± 0.86	**56.96 ± 1.43**	51.40 ± 0.57	51.64 ± 1.04	51.65 ± 1.04	55.67 ± 2.64
ion	89.62 ± 3.48	91.46 ± 2.79	**91.69 ± 3.28**	90.73 ± 4.09	90.41 ± 4.77	90.05 ± 3.16	85.96 ± 4.44
liv	66.96 ± 4.54	65.99 ± 2.18	68.45 ± 2.46	65.80 ± 5.29	68.12 ± 4.35	63.48 ± 3.61	**69.28 ± 5.16**
monk	98.32 ± 3.37	99.83 ± 1.19	**100.00 ± 0.00**	75.00 ± 0.68	**100.00 ± 0.00**	99.53 ± 1.04	**100.00 ± 0.00**
mush	99.76 ± 0.34	99.99 ± 0.05	99.97 ± 0.02	95.58 ± 0.89	**100.00 ± 0.00**	**100.00 ± 0.00**	**100.00 ± 0.00**
musk 1	84.25 ± 2.77	85.36 ± 2.44	**88.65 ± 2.47**	74.16 ± 1.29	84.67 ± 3.16	83.20 ± 2.70	88.26 ± 2.47
musk 2	97.32 ± 0.61	97.79 ± 0.41	**98.87 ± 0.10**	84.15 ± 0.32	96.68 ± 0.70	97.38 ± 0.38	98.83 ± 0.11
par	90.94 ± 4.55	91.20 ± 2.67	90.09 ± 3.67	71.28 ± 5.56	86.15 ± 6.93	83.59 ± 5.90	**92.31 ± 2.92**
tic	96.58 ± 1.78	**96.92 ± 1.11**	96.72 ± 2.13	70.77 ± 3.08	83.93 ± 3.43	94.78 ± 2.74	96.03 ± 1.31
tra	77.27 ± 3.55	77.85 ± 2.04	77.84 ± 1.98	75.80 ± 4.88	**78.75 ± 3.59**	78.35 ± 3.38	75.17 ± 2.04
vote	96.24 ± 1.98	96.13 ± 1.92	**97.32 ± 2.42**	90.34 ± 4.85	96.32 ± 2.74	96.32 ± 2.74	94.48 ± 1.50
wis91	95.11 ± 1.34	95.18 ± 1.10	96.91 ± 1.17	**97.57 ± 1.09**	93.85 ± 1.64	94.85 ± 1.84	92.09 ± 1.81
wis95	93.68 ± 1.31	93.65 ± 1.54	93.16 ± 1.89	94.73 ± 1.65	93.32 ± 2.20	94.20 ± 1.59	**96.84 ± 1.59**

statistically. On the other hand, GAssist does not outperform any of the four non-evolutionary machine learning methods, since at least one significance test does not show the significance. It means that the propose methods enhance the model rather than simply inherit the performance from GAssist. To summarize, pLCS_BOA and pLCS_BOA_FR outperform GAssist statistically. And compared with other non-evolutionary machine learning methods, the proposed two methods are able to deliver competent results as well.

6.2. Comparison on computational cost

The computational costs of classical GAssist, pLCS_BOA and pLCS_BOA_FR are investigated in this part. In order to measure the computational cost, three metrics are taken into consideration. (1) Iteration (Iter.): it is just the number of generations needed before convergence. (2) The number of match operators (MO) totally executed. Here, match operator is considered to be executed once when the condition part of one single rule is compared with an instance (see Section 3.3.2). (3) The number of micro-Match operators (mMO) executed. (4) Running time: it is the total time consumed to train the classifiers. It is necessary to point out that the third metric (i.e., mMO) is similar with the second one (i.e., MO). The differences lie in that micro-Match operator is executed once when bits representing one feature in the condition part of one rule is compared with the value of the corresponding feature in an instance. Consequently, the granularity of three metrics (i.e., iteration, MO, mMO) becomes gradually fined, and mMO is the most decisive one of the three. For the GAssist and pLCS_BOA, mMO is just equal to the product of MO and the number of features involved. In consequence, the comparison on mMO between GAssist and pLCS_BOA is identical to the one on MO While mMO in pLCS_BOA_FR does not satisfy the relation since the number of features decreases during the process of evolution. We use the number of MO and mMO executed to measure the computational cost because matching is the most computationally demanding part of the overall process in learning classifier systems [8].

We first investigate the computational cost between GAssist and the proposed methods when dealing with RMPs. The experimental results and the corresponding significant tests are list in Tables 9 and 10, respectively. From each data set, we could see that pLCS_BOA employs the lowest number of iterations and MOs to deliver the final classifier, and pLCS_BOA_FR relies the fewest mMOs before convergence. The reason behind the phenomenon is that pLCS_BOA_FR needs more iterations to remove the redundant features; however, the as the number of features decreases, the increase of mMO is slower than those of GAssist and pLCS_BOA. Consequently, the more features removed in pLCS_BOA_FR, the fewer mMOs it costs. Consequently, the computational cost of pLCS_BOA_FR is the lowest in the three methods, since the metric of mMO reflects the computational cost at the finest level. With respective to running time, the differences between the proposed two methods and GAssist are not statistically obvious.

The results on UCI data sets are similar to those on RMPs. From the results in Table 11, we could see that for each problem, the lowest iteration is achieved either by pLCS_BOA or by pLCS_BOA_FR. And in general, pLCS_BOA and pLCS_BOA_FR could reduce more than 35% and 20% iterations compared with GAssist. The situation in the comparison on the number of MOs executed is similar. The proposed two methods deliver the lowest MOs for all data sets except only one data set (*ger*). The reduction is equal to 15% and 5%. With respect to the number of mMO executed, it is obvious that pLCS_BOA_FR relies on the lowest before convergence for most data sets.

Table 9
Comparison of the number of iterations, match operators and micro-Match operators executed on RMPs.

Data	Iteration ($\times 10^1$)			Match operators ($\times 10^9$)			Micro-match operators ($\times 10^{10}$)			Running time		
	GAssist	pLCS_BOA	pLCS_BOA_FR	GAssist	pLCS_BOA	pLCS_BOA_FR	GAssist	pLCS_BOA	pLCS_BOA_FR	GAssist	pLCS_BOA	pLCS_BOA_FR
MP11	22.7 ± 6.41	**16.4 ± 4.41**	18.6 ± 5.17	0.04 ± 0.02	**0.03 ± 0.02**	**0.03 ± 0.01**	0.04 ± 0.02	**0.03 ± 0.02**	**0.03 ± 0.01**	**5.37 ± 3.07**	7.02 ± 0.63	7.14 ± 0.96
RMP11-11	38.2 ± 6.11	**30.7 ± 6.73**	38.8 ± 6.28	0.13 ± 0.02	**0.12 ± 0.01**	0.13 ± 0.02	0.29 ± 0.02	0.26 ± 0.02	**0.23 ± 0.02**	12.2 ± 1.46	12.3 ± 1.37	**12.0 ± 1.24**
RMP11-22	84.4 ± 10.4	**74.6 ± 11.4**	91.8 ± 12.8	0.64 ± 0.11	**0.59 ± 0.10**	0.70 ± 0.13	2.12 ± 0.24	1.94 ± 0.18	**1.75 ± 0.20**	69.6 ± 7.95	65.5 ± 8.17	**63.3 ± 8.07**
RMP11-44	167 ± 35.3	**138 ± 34.7**	195 ± 33.9	3.24 ± 1.22	**2.91 ± 1.21**	3.75 ± 1.90	28.8 ± 6.70	26.5 ± 5.90	**19.5 ± 4.50**	788 ± 87.5	803 ± 78.6	**766 ± 87.5**
RMP11-88	495 ± 73.7	**383 ± 64.7**	598 ± 65.9	13.4 ± 2.39	**11.4 ± 2.39**	15.1 ± 3.65	132 ± 23.7	113 ± 39.1	**88.6 ± 14.1**	1182 ± 97.5	1107 ± 102	**1050 ± 97.6**
RMP11-176	911 ± 124	**738 ± 96.4**	1092 ± 94.3	137 ± 10.9	**117 ± 9.25**	158.1 ± 16.5	2567 ± 204	2194 ± 173	**1513 ± 132**	20 371 ± 1457	19 479 ± 1347	**18 148 ± 1598**

Table 10
Significance tests of the number of iterations, match operators and micro-Match operators executed on RMPs. The level of significance is set to 0.05.

Metric	Comparison	*t*-Test	Sign test		Wilcoxon test		
		p-Value	Win	*p*-Value	W^+	W^-	*p*-Value
Iteration	pLCS_BOA v.s. GAssist	0.0001	6	0.0157	0	21	0.0106
	pLCS_BOA_FR v.s. GAssist	0.1151	1	0.1095	17	4	0.0711
	pLCS_BOA_FR v.s. pLCS_BOA	0.0016	0	0.0157	21	0	0.0106
Match operator	pLCS_BOA v.s. GAssist	0.0017	6	0.0157	0	21	0.0106
	pLCS_BOA_FR v.s. GAssist	0.2476	1	0.1095	10	5	0.2093
	pLCS_BOA_FR v.s. pLCS_BOA	0.0048	5	0.1095	15	0	0.0155
micro-Match operator	pLCS_BOA v.s. GAssist	0.0017	6	0.0157	0	21	0.0106
	pLCS_BOA_FR v.s. GAssist	0.0003	6	0.0157	0	21	0.0106
	pLCS_BOA_FR v.s. pLCS_BOA	0.0080	5	0.1095	0	15	0.0155
Running time	pLCS_BOA v.s. GAssist	0.3271	3	0.4999	9	12	0.3376
	pLCS_BOA_FR v.s. GAssist	0.4759	5	0.1095	6	15	0.1473
	pLCS_BOA_FR v.s. pLCS_BOA	0.0196	5	0.1095	1	20	0.0180

We also use the paired *t*-test, the sign test, and the Wilcoxon paired signed ranks test to investigate the significant differences on Iter., M.O, and mMO. According to the results in Table 12, it can be concluded that the pLCS_BOA converges to the near-optimal solution faster than original GAssist in terms of number of iterations and total match operators, although the number of match operators executed in each iteration in pLCS_BOA is higher than that in GAssist. With respect to pLCS_BOA_FR, it also relies on less computational cost than GAssist in terms of the number of iterations and micro-Match operators. Compared with pLCS_BOA, although the advantages of pLCS_BOA_FR on iteration and match operator are not obvious, there are statistical differences in the number of micro-Match operator. Therefore, pLCS_BOA_FR is better than pLCS_BOA regarding computational efficiency. Regarding to running time, although the difference between GAssist and pLCS_BOA is not statically obvious, the running time of pLCS_BOA_FR is statically lower that those of pLCS_BOA and GAssist. The reason behind the phenomenon lies in that although learning Bayesian network introduces an additional computational cost, the embedded feature reduction approach, which removes the redundant features during classifiers training, reduces the number of micro-Match operators (mMO) executed when pLCS_BOA_FR compares its rules with instances. The decrease in mMO also cuts down the total running time of training.

Additionally, the numbers of rules contained in the final classifier delivered by GAssist and the proposed method are also compared. Tables 13 and 14 show the results and the corresponding significant tests, respectively. From these results, we can see that there are no significant differences among these methods on the metric. It means that the complexities of the final solution are generally the same.

6.3. Comparison on feature reduction

In this part, we present the performance of pLCS_BOA_FR regarding feature reduction. The reduction rate is listed in Tables 15 and 16 when pLCS_BOA_FR dealing with RMPs and UCI data sets. Given the experimental results, it is obvious that the more features involved in the data set, the larger the feature reduction rate achieved by pLCS_BOA_FR.

In order to evaluate the feature reduction performance of pLCS_BOA_FR, we adopt UCI data sets to compare it with eight combinations, each of which consists of a classifier and a feature selection. Four classifiers are taken, including C4.5, SVM, GAssist and pLCS_BOA. The feature selection techniques include Information Gain (IG) [45] and ReliefF (RF) [46,47]. These combinations are presented as *C_F*, in which *C* denotes the classifier and *F* is the feature selection techniques.

In the experiments, the data sets in which features are removed more than 20% by pLCS_BOA_FR are chosen as the test beds. Here, these data sets are called redundant data sets. Consequently, 16 data sets from the originals are picked up. Since IG and RF have to determine the threshold on the strength of dependencies between features and classification, and such a threshold significantly affects their performances, we set the threshold to the value under which IG and ReliefF remove the same number of features as pLCS_BOA_FR does on redundant data sets. The experimental results are described in Table 17. And Table 18 presents the significant differences on the results.

According to the results in Table 17, we could see that pLCS_BOA_FR achieves the highest accuracy on most redundant data sets. And given the results of the significance test in Table 18, it could be safely concluded that the proposed pLCS_BOA_FR is statistically better than C4.5, SVM, GAssist and pLCS_BOA integrated with IG or RF in terms of the classification accuracy, when removing the same number of features.

7. Discussion

According to experimental results presented in Section 6, pLCS_BOA and pLCS_BOA_FR deliver solution with higher classification accuracy compared with GAssist while reducing the computational cost. The reasons behind the phenomenon are listed as below.

First, the proposed method divides the evolution of classifiers into two levels explicitly. At the low level, single rules contained in individuals evolve by means of building and sampling Bayesian networks of promising rules of all individuals, aiming at identification of dependencies between variables and detection of substructure lying in single rules. And at the rule set level, the recombination of individuals is performed via rule-wise uniform crossover operators to make sure that rules are not disrupted in the individuals. The strategies generating offspring at both levels are able to assure the effective recombination.

Next, both the specification and the generalization are considered into the evaluation of rules. With respect to specification, the positive accuracy, which measures its performance inside the domain, is adopted as the metrics. And, the negative accuracy, which reflects the performance of the rules 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.

Table 11
Comparison of the number of iterations, match operators and micro-Match operators executed on UCI data sets.

Data	Iteration ($\times 10^1$)			Match operators ($\times 10^7$)			Micro-Match operators ($\times 10^8$)			Running time (s) ($\times 10^1$)		
	GAssist	pLCS_BOA	pLCS_BOA_FR	GAssist	pLCS_BOA	pLCS_BOA_FR	GAssist	pLCS_BOA	pLCS_BOA_FR	GAssist	pLCS_BOA	pLCS_BOA_FR
aus	107 ± 13.5	75.6 ± 12.3	**48.2 ± 27.4**	19.6 ± 2.19	28.1 ± 7.11	**16.9 ± 8.72**	27.5 ± 3.07	39.3 ± 9.95	**14.1 ± 7.21**	**3.00 ± 0.33**	3.45 ± 0.53	3.01 ± 0.46
che	38.8 ± 10.9	**33.6 ± 13.5**	41.0 ± 22.2	39.1 ± 12.6	**33.4 ± 13.5**	49.4 ± 3.45	141 ± 45.3	120 ± 48.7	**101 ± 8.78**	2.31 ± 0.40	2.26 ± 0.19	**2.19 ± 0.18**
cre	115 ± 22.5	88.3 ± 121.6	**60.6 ± 28.1**	30.5 ± 7.95	43.1 ± 1369	**20.5 ± 887**	45.8 ± 11.9	64.7 ± 20.5	**18.7 ± 8.44**	**3.56 ± 0.32**	4.20 ± 0.53	3.61 ± 0.41
dia	63.7 ± 12.9	**25.7 ± 7.22**	49.2 ± 20.7	13.0 ± 1.19	**10.87 ± 4.71**	11.8 ± 1.73	10.4 ± 0.95	**8.62 ± 3.77**	9.12 ± 0.93	1.61 ± 0.03	**1.49 ± 0.03**	1.59 ± 0.03
ger	83.5 ± 11.3	**48.0 ± 7.21**	50.5 ± 27.5	**29.8 ± 3.13**	34.8 ± 806	49.3 ± 10.5	71.5 ± 7.51	83.5 ± 19.3	**71.0 ± 21.7**	4.66 ± 0.28	4.87 ± 0.68	**4.73 ± 0.51**
hab	64.9 ± 8.46	12.1 ± 41.8	**10.8 ± 4.02**	4.60 ± 1.52	2.94 ± 1.11	**2.89 ± 0.95**	1.38 ± 0.46	0.88 ± 0.33	**0.68 ± 0.23**	0.44 ± 0.13	0.37 ± 0.11	**0.36 ± 0.12**
heart	86.3 ± 35.4	**45.6 ± 18.6**	55.1 ± 32.4	21.4 ± 8.49	**7.80 ± 2.60**	8.50 ± 0.71	27.8 ± 11.0	10.1 ± 3.38	**8.06 ± 4.44**	3.99 ± 0.31	2.41 ± 0.28	**2.31 ± 0.25**
hep	32.6 ± 12.3	**7.80 ± 3.74**	19.7 ± 13.8	2.76 ± 0.72	**0.94 ± 0.51**	1.66 ± 0.90	5.24 ± 1.37	1.79 ± 0.97	**1.49 ± 0.62**	1.27 ± 0.13	1.00 ± 0.10	**0.97 ± 0.10**
hill	597 ± 112	**512 ± 98.1**	574 ± 105	232 ± 66.8	**206 ± 59.6**	242 ± 61.2	2316 ± 668	2059 ± 596	**1597 ± 478**	159 ± 12.3	197 ± 15.7	**148 ± 14.4**
ion	32.9 ± 9.01	**8.66 ± 2.15**	16.1 ± 5.98	3.23 ± 1.22	**2.74 ± 1.14**	4.27 ± 0.41	11.0 ± 4.15	9.32 ± 3.88	**8.71 ± 1.50**	1.86 ± 0.16	1.54 ± 0.13	**1.52 ± 0.13**
liv	78.4 ± 7.54	49.7 ± 223.4	**32.3 ± 18.3**	7.81 ± 3.50	**4.96 ± 2.70**	5.31 ± 0.07	4.69 ± 2.10	**2.98 ± 1.62**	3.12 ± 0.15	1.09 ± 0.05	**0.85 ± 0.04**	0.86 ± 0.03
monk	7.80 ± 3.44	**3.74 ± 2.41**	9.10 ± 4.27	1.31 ± 0.79	**1.21 ± 0.18**	2.12 ± 0.07	0.79 ± 0.47	**0.73 ± 0.11**	0.87 ± 0.08	0.17 ± 0.03	**0.16 ± 0.01**	0.17 ± 0.01
mush	261 ± 12.3	18.3 ± 124.6	**18.2 ± 10.7**	49.2 ± 2.46	**4.90 ± 13.0**	96.3 ± 13.3	108 ± 54.1	**108 ± 28.6**	168 ± 27.6	**4.21 ± 0.62**	4.31 ± 0.47	4.79 ± 0.57
musk 1	196 ± 43.0	**176 ± 27.6**	189 ± 39.8	39.7 ± 3.46	**37.5 ± 3.63**	46.1 ± 34.6	659 ± 23.4	623 ± 25.9	**465 ± 38.6**	15.8 ± 1.48	17.9 ± 1.57	**15.0 ± 1.33**
musk 2	976 ± 229	**897 ± 203**	1028 ± 218	23 165 ± 3233	**21 652 ± 2933**	25 482 ± 1233	384 539 ± 53 667	359 423 ± 65 231	**308 975 ± 46 523**	2878 ± 198	3172 ± 210	**2683 ± 176**
par	27.4 ± 9.49	**19.5 ± 9.80**	47.8 ± 33.3	1.60 ± 0.58	**1.31 ± 0.74**	2.14 ± 0.16	3.68 ± 1.33	3.01 ± 1.70	**2.91 ± 0.30**	0.84 ± 0.04	0.67 ± 0.02	**0.67 ± 0.02**
tic	42.6 ± 6.48	32.9 ± 13.0	**23.3 ± 251.8**	44.4 ± 8.59	34.5 ± 690	**33.0 ± 4.53**	39.9 ± 7.73	31.0 ± 0.62	**29.6 ± 4.16**	19.99 ± 3.43	**17.41 ± 2.28**	18.25 ± 2.69
tra	53.8 ± 13.3	7.64 ± 1.42	**7.35 ± 3.71**	7.79 ± 1.29	2.93 ± 57	**2.71 ± 1.47**	3.12 ± 0.52	1.17 ± 0.23	**1.03 ± 0.55**	0.85 ± 0.09	**0.70 ± 0.05**	0.72 ± 0.06
vote	21.5 ± 12.2	**17.3 ± 10.1**	45.2 ± 28.0	2.83 ± 0.78	**1.86 ± 0.25**	2.92 ± 0.98	4.53 ± 1.25	2.98 ± 0.40	**2.78 ± 0.71**	0.10 ± 0.03	0.10 ± 0.02	**0.09 ± 0.01**
wis91	46.5 ± 7.91	35.4 ± 15.9	**24.5 ± 26.3**	7.39 ± 0.94	**5.52 ± 0.52**	5.91 ± 0.11	6.65 ± 0.85	4.97 ± 0.47	**4.54 ± 0.20**	**1.19 ± 0.05**	1.40 ± 0.07	1.37 ± 0.08
wis95	56.0 ± 36.3	**15.6 ± 11.1**	25.7 ± 5.67	31.6 ± 4.42	**4.76 ± 1.94**	7.76 ± 3.53	94.8 ± 13.3	14.3 ± 0.58	**9.84 ± 2.91**	8.81 ± 0.59	4.40 ± 0.41	**4.20 ± 0.57**

Additionally, the prediction of individuals in the proposed method is different from GAssist. More specifically, the prediction of an individual in GAssist terminates when the first matched rule is found, while all rules in an individual have to be compared with examples so as to do the prediction in the proposed method. Consequently, rules in the proposed method are evaluated sufficiently in each generation, which gives more information to do selection and recombination. Although the number of match operators executed in each generation in the proposed method is larger than that in GAssist, the total number during the whole procedure of evolution in our method is lower. The reason behind the phenomenon is that the proposed method relies on fewer generations before convergence, and the reduction in generation decreases the total number of match operators executed.

With respect to performance of the features reduction, pLCS_BOA_FR embeds a feature reduction component into pLCS_BOA to remove redundant features incrementally with the rule evolution. Unlike other feature reduction techniques, which employ the information contained in the original data, our method leverages the statistical information in the rule population. And since Bayesian networks are built in the process of rule-wise evolution to model promising rules, the component of features reduction does not increase the computational costs of the whole system.

7.1. Parameter sensitivity analysis

In this subsection, the influence of two parameters in the algorithm, including the interval of structural learning on Bayesian networks and the threshold on the redundancy, is discussed.

Since learning Bayesian network is computationally expensive, the proposed method uses a sporadic model building strategy; that is, the model is updated once in every few Pittsburgh model iterations. The number of iterations between Bayesian network learning is denoted as *inter*. We use the problems of MP-11 as the test beds to discuss the changes in the running time and the number of match operators as *inter* increases. Fig. 4(a) shows the performance sensitivity to *inter*. If the value of inter is set too low, which means BOA is performed over-frequently, the running time grows significantly. The running time decreases as the value of inter become larger. This phenomenon demonstrates the rationality of the sporadic model building strategy. On the other hand, as the number of Pittsburgh model iterations between model buildings becomes larger, which means that as rule-wise combination via BOA is performed less frequently, the number of match operators grows. The reason lies in that large inter means rule-wise combination via BOA is performed occasionally, which degrades the efficiency of rule-wise evolution. Consequently, the parameter inter has to balance the evolutionary efficiency and computational cost. According to Fig. 4(a), the balance is achieved when the interval is set around 10. With the configuration, the efficiency of evolution is improved while the additional cost of BOA is not remarkable.

To discuss the performance sensitivity to threshold on the redundancy (denoted as τ), RMP-11-88 is adopted as the test bed. Because the same value of *tau* leads to different performance when the probability of value 1 in classifier initialization (denoted as P_1) is different, we discuss the two parameter together. From the results depicted in Fig. 4(b), we could see that as the value of P_1 decreases, the performance of the proposed method degrades. The reason behind the phenomena lies in that low value of P_1 leads to many 0 in the initial classifier. These classifiers mismatch most instances in training data sets, which introduces negative effect to the evaluation of the system. Regarding the influence of parameter τ, when its value is lower than that of P_1, the performance of the pLCS_BOA_FR degrades under the same

Table 12
Significance tests of the number of iterations, match operators and micro-Match operators executed on UCI data sets. The level of significance is set to 0.05.

Metric	Comparison	*t*-Test	Sign test		Wilcoxon test		
		p-Value	Win	*p*-Value	W^+	W^-	*p*-Value
Iteration	pLCS_BOA v.s. GAssist	<0.0001	22	<0.0001	0	231	<0.0001
	pLCS_BOA_FR v.s. GAssist	0.0125	16	0.0133	50	181	0.0109
	pLCS_BOA_FR v.s. pLCS_BOA	0.0346	9	0.2615	157	74	0.0721
Match operator	pLCS_BOA v.s. GAssist	0.0024	19	0.0004	39	192	0.0037
	pLCS_BOA_FR v.s. GAssist	0.3821	12	0.4160	103	128	0.3256
	pLCS_BOA_FR v.s. pLCS_BOA	0.0319	5	0.0133	186	45	0.0068
micro-Match operator	pLCS_BOA v.s. GAssist	0.0024	19	0.0004	39	192	0.0037
	pLCS_BOA_FR v.s. GAssist	<0.0001	19	0.0004	18	231	0.0003
	pLCS_BOA_FR v.s. pLCS_BOA	0.0479	18	0.0022	44	187	0.0062
Running time	pLCS_BOA v.s. GAssist	0.0856	8	0.1915	81	150	0.1118
	pLCS_BOA_FR v.s. GAssist	0.0043	6	0.0392	37.5	193.5	0.0032
	pLCS_BOA_FR v.s. pLCS_BOA	0.0272	6	0.0392	65	166	0.0381

Table 13
Comparison of the number of rules contained in the final classifier on UCI data sets.

Data	Number of rules		
	GAssist	pLCS_BOA	pLCS_BOA_FR
aus	6.32 ± 0.34	8.56 ± 0.26	7.62 ± 0.27
che	7.70 ± 0.22	7.12 ± 0.33	9.74 ± 0.67
cre	4.96 ± 0.22	8.13 ± 0.17	5.29 ± 0.26
dia	7.35 ± 0.28	5.96 ± 0.22	5.42 ± 0.13
ger	4.35 ± 0.15	18.1 ± 0.12	6.15 ± 0.45
hab	3.91 ± 0.11	2.48 ± 0.11	2.39 ± 0.10
heart	7.13 ± 0.34	8.55 ± 0.18	7.05 ± 0.12
hep	3.56 ± 0.16	3.17 ± 0.17	3.49 ± 0.13
hill	12.1 ± 2.52	12.4 ± 2.16	12.48 ± 2.72
ion	2.94 ± 0.19	4.88 ± 0.22	6.23 ± 0.59
liv	5.24 ± 0.17	5.14 ± 0.23	4.83 ± 0.18
monk	4.56 ± 0.24	4.74 ± 0.24	4.95 ± 0.14
mush	4.72 ± 0.23	4.33 ± 0.12	10.9 ± 0.72
musk 1	11.2 ± 2.18	12.3 ± 2.61	11.4 ± 2.37
musk 2	47.2 ± 6.23	44.3 ± 6.12	43.0 ± 6.72
par	4.02 ± 0.09	4.25 ± 0.19	5.46 ± 0.22
tic	22.3 ± 1.38	18.9 ± 0.95	14.1 ± 0.83
tra	4.12 ± 0.13	2.74 ± 0.13	2.35 ± 0.15
vote	5.14 ± 0.16	4.17 ± 0.08	3.62 ± 0.12
wis91	4.06 ± 0.11	4.33 ± 0.19	4.12 ± 0.13
wis95	4.59 ± 0.22	5.32 ± 0.13	7.33 ± 0.57

Table 14
Significance tests of the number of rules contained in the final classifier on UCI data sets. The level of significance is set to 0.05.

Comparison	*t*-Test	Sign test		Wilcoxon test		
	p-Value	Win	*p*-Value	W^+	W^-	*p*-Value
pLCS_BOA v.s. GAssist	0.2181	10	0.0133	121	111	0.4172
pLCS_BOA_FR v.s. GAssist	0.226	9	0.3320	127	104	0.0010
pLCS_BOA_FR v.s. pLCS_BOA	0.4865	13	0.1915	89.5	141.5	0.1785

settings of P_1. Otherwise, the classification accuracy is improved. The reason is listed as below. The redundancy of a feature (see definition in Section 4.1) is lower than P_1 after several iterations means that some bits corresponding to the feature changes from 1 to 0 generally. In such a situation, the possible values of the feature make different contributions to the classification. Therefore, the value of *tau* set below P_1 may remove some non-redundant features and the accuracy decreases correspondingly. Given the discussion above, we configure pLCS_BOA_FR with $P_1=0.9$ and $\tau=0.9$. And Fig. 4(b) shows that the proposed methods performs best under such configurations.

Table 15
Feature reduction rates on RMPs.

Data	#Features	#Redundant features	#Reduced features	Reduction ratio (%)
RMP11-11	22	11	7.48 ± 1.09	34.00
RMP11-22	33	22	13.04 ± 2.17	39.52
RMP11-44	55	44	25.57 ± 4.45	46.49
RMP11-88	99	88	48.72 ± 9.30	49.21
RMP11-176	187	176	96.94 ± 21.54	51.84

Table 16
Feature reduction rates on UCI data sets.

Data	#Features	#Reduced features	Reduction ratio (%)
aus	14	6.23 ± 1.09	44.5
che	36	25.07 ± 3.57	69.6
cre	15	6.64 ± 1.40	44.3
dia	8	0.93 ± 0.43	11.7
ger	24	10.17 ± 2.34	42.4
hab	3	0.37 ± 0.29	12.3
heart	13	4.53 ± 1.59	34.9
hep	19	13.71 ± 2.02	72.2
hill	100	76.45 ± 8.34	76.2
ion	34	22.50 ± 1.61	66.2
liv	6	0.90 ± 1.09	15.0
monk	6	2.26 ± 0.77	37.7
mush	22	10.05 ± 2.01	45.7
musk 1	166	130.05 ± 14.43	78.3
musk 2	166	114.31 ± 12.71	68.7
par	23	122.69 ± 1.19	54.9
tic	9	0.07 ± 0.34	0.80
tra	4	0.47 ± 0.51	11.8
vote	16	6.13 ± 1.04	38.3
wis91	9	2.17 ± 1.84	24.1
wis95	30	22.17 ± 1.05	73.9

7.2. Performance on noisy data

This section compares the performance between GAssist, pLCS_BOA and pLCS_BOA_FR when dealing with noisy data. We adopt noisy 11-bit multiplexer problems, where the actual classification of multiplexer problems is flipped with a certain probability p_n, as the test bed. Here, four levels of noise are considered: $p_n=\{0.05,0.10,0.15,0.20\}$.

The experimental results of classification accuracy and computational cost are listed in Tables 19 and 20, respectively. According to the results, we can see that both the gaps between final accuracy and 1, and the total computational costs (i.e., the number of total match operators) are proposition to the noisy

Table 17
Comparison of classification accuracy on redundant data sets when removing the same number of features.

Data	pLCS_BOA_FR	GAssist_IG	GAssist_RF	pLCS_BOA_IG	pLCS_BOA_RF	C4.5_IG	C4.5_RF	SVM_IG	SVM_RF
aus	**86.59 ± 2.35**	84.59 ± 1.02	84.47 ± 1.83	83.79 ± 1.68	82.46 ± 2.13	84.78 ± 2.01	85.23 ± 1.94	83.33 ± 1.79	84.64 ± 1.87
che	96.29 ± 1.54	96.02 ± 0.17	96.00 ± 0.12	96.26 ± 0.27	**96.36 ± 0.26**	94.84 ± 0.43	94.98 ± 0.24	94.21 ± 0.13	95.18 ± 0.29
cre	**87.32 ± 1.84**	86.74 ± 1.24	86.04 ± 0.95	84.69 ± 0.85	86.91 ± 1.69	84.78 ± 0.89	86.23 ± 1.07	85.51 ± 0.97	86.49 ± 1.29
ger	71.92 ± 1.55	71.75 ± 1.33	71.83 ± 1.30	69.87 ± 1.84	70.35 ± 1.81	68.50 ± 1.97	71.85 ± 1.68	**77.50 ± 1.65**	75.50 ± 1.82
heart	80.49 ± 4.15	83.33 ± 2.48	82.21 ± 3.40	81.60 ± 2.48	78.70 ± 3.03	82.90 ± 3.46	82.56 ± 2.54	**85.19 ± 2.36**	83.33 ± 3.47
hep	**84.41 ± 2.82**	82.24 ± 2.64	78.75 ± 1.32	83.49 ± 1.63	83.76 ± 1.33	83.10 ± 2.18	83.87 ± 1.53	83.87 ± 2.01	82.74 ± 2.35
hill	**56.96 ± 1.43**	51.25 ± 1.17	51.93 ± 1.35	52.91 ± 1.14	53.17 ± 1.03	50.73 ± 0.98	51.04 ± 0.85	52.07 ± 1.16	51.37 ± 1.42
ion	**91.19 ± 3.28**	90.00 ± 2.26	90.12 ± 3.46	87.25 ± 1.45	91.12 ± 2.35	90.98 ± 1.68	90.49 ± 1.47	82.46 ± 2.47	85.96 ± 2.19
monk	**100.00 ± 0.00**	95.38 ± 3.41	96.59 ± 3.04	**100.00 ± 0.00**	**100.00 ± 0.00**	**100.00 ± 0.00**	**100.00 ± 0.00**	**100.00 ± 0.00**	**100.00 ± 0.00**
mush	99.97 ± 0.02	99.56 ± 0.35	99.40 ± 0.40	**100.00 ± 0.00**	**100.00 ± 0.00**	**100.00 ± 0.00**	**100.00 ± 0.00**	**100.00 ± 0.00**	**100.00 ± 0.00**
musk 1	**88.65 ± 2.47**	79.15 ± 3.70	77.82 ± 3.75	82.47 ± 3.07	83.43 ± 2.98	78.95 ± 2.69	72.63 ± 3.12	75.79 ± 2.47	71.58 ± 2.99
musk 2	**98.87 ± 0.10**	94.31 ± 0.51	94.87 ± 0.54	94.93 ± 0.48	95.14 ± 0.39	96.36 ± 0.48	94.77 ± 0.54	92.95 ± 0.36	92.34 ± 0.41
par	90.29 ± 3.67	90.17 ± 3.23	**91.54 ± 4.42**	87.18 ± 1.33	89.40 ± 1.75	87.18 ± 1.65	88.29 ± 2.05	82.05 ± 2.47	79.49 ± 2.91
vote	97.32 ± 2.42	96.95 ± 0.79	95.97 ± 1.00	96.75 ± 0.96	95.27 ± 1.04	**100.00 ± 0.00**	**100.00 ± 0.00**	97.08 ± 0.49	96.85 ± 0.24
wis91	**96.91 ± 1.17**	95.48 ± 1.63	96.86 ± 1.10	95.74 ± 0.44	93.02 ± 0.77	94.96 ± 1.24	96.40 ± 1.37	96.12 ± 0.87	96.40 ± 1.02
wis95	93.16 ± 1.89	90.18 ± 1.01	93.30 ± 1.47	90.56 ± 0.50	92.02 ± 0.48	92.98 ± 0.76	92.11 ± 0.61	91.39 ± 0.37	**93.61 ± 0.42**

Table 18
Significance tests of classification accuracy on redundant data sets when removing the same number of features. The level of significance is set to 0.05.

Comparison	*t*-Test	Sign test		Wilcoxon test		
	p-Value	Win	*p*-Value	W^+	W^-	*p*-Value
pLCS_BOA_FR v.s. GAssist_IG	0.0059	15	0.0002	125	11	0.0016
pLCS_BOA_FR v.s. GAssist_RF	0.0105	13	0.0107	116	20	0.0065
pLCS_BOA_FR v.s. pLCS_BOA_IG	0.0004	14	0.0021	113.5	6.5	0.0012
pLCS_BOA_FR v.s. pLCS_BOA_RF	0.0004	14	0.0021	116.5	3.5	0.0007
pLCS_BOA_FR v.s. C4.5_IG	0.0143	12	0.0176	100	20	0.0124
pLCS_BOA_FR v.s. C4.5_RF	0.0457	12	0.0176	96	24	0.0176
pLCS_BOA_FR v.s. SVM_IG	0.0244	12	0.0176	99	21	0.0144
pLCS_BOA_FR v.s. SVM_RF	0.0232	12	0.0176	100	20	0.0124

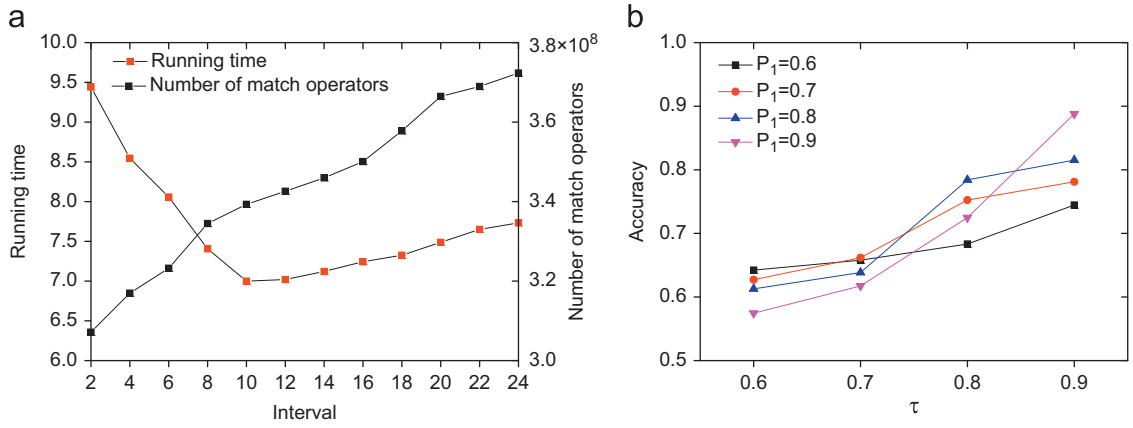

Fig. 4. Performance sensitivity to parameters of *inte* and τ: (a) performance sensitivity to *inte* and (b) performance sensitivity to τ.

Table 19
Comparison of classification accuracy on noisy MP-11.

Data	GAssist	pLCS_BOA	pLCS_BOA_FR
MP-11, $p_n = 0.05$	**94.42 ± 0.54**	94.17 ± 0.47	94.47 ± 0.50
MP-11, $p_n = 0.10$	**89.95 ± 0.74**	89.31 ± 0.46	89.75 ± 0.69
MP-11, $p_n = 0.15$	**86.94 ± 0.53**	86.43 ± 0.98	86.10 ± 0.78
MP-11, $p_n = 0.20$	**81.38 ± 0.89**	80.59 ± 1.43	81.01 ± 0.94

degree. The reason for the phenomenon lies in that the level of noise reflects the complexity of the problem. In another word, the larger noisy degree, the more sophisticated the problem. Furthermore, the classification accuracy of three methods degrades almost in a similar manner when the level of noise increases. With respect to the computational cost, the numbers of iterations, match operators, micro-Match operators and running time in the three methods rise as the level of noise increases. Therefore, performances of GAssist and the proposed methods change almost the same when adding noisy data into standard multiplexer problems.

8. Conclusion and future work

In order to improve the performance when dealing with real world classification problems that contain strong interactions between features or redundant features, we incorporate two

Table 20
Comparison of the number of generations, match operators and micro-Match operators executed on noisy MP-11.

Data-set	Iteration			Match operators ($\times 10^8$)			Micro-Match operators ($\times 10^9$)			Running time		
	GAssist	pLCS_BOA	pLCS_BOA_FR	GAssist	pLCS_BOA	pLCS_BOA_FR	GAssist	pLCS_BOA	pLCS_BOA_FR	GAssist	pLCS_BOA	pLCS_BOA_FR
MP-11, $p_n = 0.05$	290 ± 74.5	**172 ± 41.6**	180 ± 39.8	4.16 ± 0.82	**3.49 ± 0.69**	3.55 ± 0.56	4.58 ± 0.90	**3.84 ± 0.76**	3.89 ± 0.82	**5.86 ± 3.08**	7.66 ± 0.93	7.92 ± 0.89
MP-11, $p_n = 0.10$	446 ± 61.2	238 ± 43.6	**231 ± 40.1**	4.66 ± 0.80	4.26 ± 0.93	**4.24 ± 1.13**	5.13 ± 0.88	4.69 ± 1.02	**4.63 ± 1.26**	11.4 ± 7.72	10.8 ± 0.85	**10.6 ± 0.90**
MP-11, $p_n = 0.15$	603 ± 52.8	295 ± 77.5	**284 ± 79.0**	6.22 ± 1.59	5.29 ± 1.42	**5.15 ± 1.53**	6.84 ± 1.75	5.82 ± 1.56	**5.70 ± 1.45**	15.1 ± 10.3	**12.5 ± 0.72**	12.8 ± 0.87
MP-11, $p_n = 0.20$	696 ± 27.8	**364 ± 28.7**	379 ± 31.7	7.46 ± 1.04	**6.80 ± 1.36**	6.98 ± 1.60	8.21 ± 1.14	**7.48 ± 1.50**	7.55 ± 1.62	16.9 ± 11.5	**15.7 ± 0.63**	16.3 ± 0.87

techniques from different aspects into the frame of Pittsburgh GBML systems. First, we integrate BOA into the frame with the aim of enhancing the effectiveness and efficiency of rule structure exploration. In the proposed method, classifiers are generated and recombined at two levels. At the low level, single rules contained in each classifier are produced by sampling Bayesian networks which characterize the distribution of promising rules found so far in the search space. And at the high level, classifiers are recombined by rule-wise uniform crossover, which keeps the semantics of all rules in each classifier. Then, an embedded feature reduction approach for pLCS_BOA (called pLCS_BOA_FR) is proposed to remove redundant features during the process of rule evolution. More specifically, the proposed approach is executed including two phases. First, candidate features to be removed are detected according to their redundancies, which is equal to the ratio of the number of rules that is independent of the feature to the size of rule population. Then, Markov blanket of each feature is calculated in the context of Bayesian networks, which are built in the process of rule evolution, in order to find relevancies between features and determine which features are actually removed.

Experiments are carried out on real world binary classification problems to evaluate the proposed method. According to the experimental results, pLCS_BOA and pLCS_BOA_FR are able to deliver solutions with higher classification accuracy compared with classical GAssist while reducing the computational cost in terms of the number of generations and total match operators executed. Furthermore, solutions achieved by the proposed methods are also competitive to other non-evolutionary machine learning methods regarding classification accuracy. With respect to the performance of feature reduction, the proposed embedded approach is able to deliver solutions with higher classification accuracy when removing the same number of features as information gain and ReliefF do.

In this paper, binary classification problems are used as the test bed to evaluate the proposed method. Experimental results presented in Tables 5–8 show that probabilistic modeling mechanisms used in BOA are competent of evolving rules with higher classification accuracy. However, when dealing with multiclass problems, there are no significant differences between the proposed methods and GAssist in terms of the classification accuracy. The reasons behind this phenomenon are listed below. First, in our proposed method, we make use of Restricted Tournament Replacement (RTR) to deal with the global optima, and hence maintaining the diversity of the population. However, as the number of the classes increases, the number of the global optima becomes larger during the rule wise evolution. Consequently, the adverse effects caused by such a multimodal problems is not easy to handle. Second, one single rule could only deal with one class in our proposed method. When there are many classes in the classification problem, the number of positive samples for some classes is much lower than those of other classes. Hence, when single rules compete with each other during the rule wise evolution, rules that handle the class with fewer samples could not be evaluated well. Therefore, the performance of the proposed method degrades as the number of the classes increases. Consequently, in our future works, the multiclass problems will be coped with by transforming the target problem into several binary classification problems, and then solved by pLCS_BOA and pLCS_BOA_FR.

Furthermore, future areas for research will incorporate certain local search mechanisms into the frame of BOA, which may improve the search efficiency in Pittsburgh-style GBML systems.

Acknowledgments

The authors would like to thank anonymous reviewers for their valuable suggestions. This work was supported by National

Natural Science Foundation of China (Grant no. 60875073, 61175110), National Science & Technology Major Projects of China (Grant no. 2009ZX02001, 2011ZX02101-004) and the National Basic Research Program of China (973 Program) (Grant no. 2012CB316305).

References

[1] A. Fernández, S. García, J. Luengo, E. Bernadó-Mansilla, F. Herrera, Genetics-based machine learning for rule induction: state of the art, taxonomy, and comparative study, IEEE Trans. Evolut. Comput. 14 (6) (2010) 913–941.
[2] 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.
[3] A.E. Eiben, J.E. Smith, Introduction to Evolutionary Computing, Springer, Verlag, 2003.
[4] A. Orriols-Puig, J. Casillas, E. Bernad-Mansilla, Genetic-based machine learning systems are competitive for pattern recognition, Evolut. Intell. 1 (3) (2008) 209–232.
[5] E. Bernadó-Mansilla, J.M. Garrell-Guiu, Accuracy-based learning classifier systems: models, analysis and applications to classification tasks, Evolut. Comput. 11 (3) (2003) 209–238.
[6] M. Pelikan, D.E. Goldberg, F.G. Lobo, A survey of optimization by building and using probabilistic models, Comput. Optim. Appl. 21 (1) (2002) 5–20.
[7] M.V. Butz, M. Pelikan, X. Llorà, D.E. Goldberg, Automated global structure extraction for effective local building block processing in XCS, Evolut. Comput. 14 (3) (2006) 345–380.
[8] X. Llorà, K. Sastry, Fast rule matching for learning classifier systems via vector instructions, in: Proceedings of the Eighth Annual Conference on Genetic and Evolutionary Computation, GECCO '06, 2006, pp. 1513–1520.
[9] P. Larranaga, J. Lozano, Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation, Kluwer Academic Publishers, 2002.
[10] M. Hauschild, M. Pelikan, An introduction and survey of estimation of distribution algorithms, Swarm Evolut. Comput. 1 (3) (2011) 111–128.
[11] M. Pelikan, D.E. Goldberg, E. Cantá-Paz, BOA: the Bayesian optimization algorithm, in: Proceedings of the 1999 Genetic and Evolutionary Computation Conference, GECCO'99, 1999, pp. 525–532.
[12] D. Koller, N. Friedman, Probabilistic Graphical Models: Principles and Techniques, MIT Press, 2009.
[13] M. Pelikan, Hierarchical Bayesian Optimization Algorithm: Toward a New Generation of Evolutionary Algorithms, Springer, Berlin, 2005.
[14] J. Bacardit, Pittsburgh Genetics-Based Machine Learning in the Data Mining Era: Representations, Generalization, and Run-time, Ph.D. Thesis, Ramon Llull University, 2004.
[15] J.H. Holland, J.S. Reitman, Cognitive systems based on adaptive algorithms, in: W. DA, H.-R. F (Eds.), Pattern-directed Inference Systems, Academic Press, San Diego, 1978, pp. 313–329.
[16] R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction, MIT Press, 1998.
[17] M.V. Butz, Rule-Based Evolutionary Online Learning Systems, Springer, 2006.
[18] S.W. Wilson, Classifier fitness based on accuracy, Evolut. Comput. 3 (2) (1995) 149–175.
[19] K.A. De Jong, W.M. Spears, D.F. Gordon, Using genetic algorithms for concept learning, Mach. Learn. 13 (2–3) (1993) 161–188.
[20] C.Z. Janikow, A knowledge-intensive genetic algorithm for supervised learning, Mach. Learn. 13 (2–3) (1993) 189–228.
[21] J. Bacardit, N. Krasnogor, Performance and efficiency of memetic Pittsburgh learning classifier systems, Evolut. Comput. 17 (3) (2009) 307–342.
[22] 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.
[23] X. Llora, K. Sastry, D.E. Goldberg, The compact classifier system: scalability analysis and first results, in: Proceedings of the Congress on Evolutionary Computation 2005, 2005, pp. 596–603.
[24] 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 Ninth International Workshop on Learning Classifier Systems, 2006.
[25] G. Harik, F. Lobo, D.E. Goldberg, The compact genetic algorithm, IEEE Trans. Evolut. Comput. 3 (4) (1999) 287–297.
[26] I. Guyon, A. Elisseeff, An introduction to variable and feature selection, J. Mach. Learn. Res. 3 (2003) 1157–1182.
[27] D. Koller, M. Sahami, Toward optimal feature selection, in: Proceedings of the 13th International Conference on Machine Learning, 1996, pp. 284–292.
[28] M. Drugan, M. Wiering, Local causal and Markov blanket induction for causal discovery and feature selection for classification part I: algorithms and empirical evaluation, J. Mach. Learn. Res. 11 (2010) 235–284.
[29] M. Abedini, M. Kirley, A multiple population XCS: evolving condition-action rules based on feature space partitions, in: Proceedings of 2010 IEEE Congress on Evolutionary Computation, 2010, pp. 1–8.
[30] M. Abedini, M. Kirley, CoXCS: a coevolutionary learning classifier based on feature space partitioning, in: A. Nicholson, X. Li (Eds.), AI 2009: Advances in Artificial Intelligence, Springer, Berlin, Heidelberg, 2009, pp. 360–369.
[31] J. Bacardit, E. Burke, N. Krasnogor, Improving the scalability of rule-based evolutionary learning, Memetic Comput. 1 (1) (2009) 55–67.
[32] C.-T. Su, J.-H. Hsu, An extended Chi2 algorithm for discretization of real value attributes, IEEE Trans. Knowledge Data Eng. 17 (2005) 437–441.
[33] E. Yu, P. Suganthan, Ensemble of niching algorithms, Inf. Sci. 180 (15) (2010) 2815–2833.
[34] G.R. Harik, Finding multimodal solutions using restricted tournament selection, in: Proceedings of the Sixth International Conference on Genetic Algorithms, 1995, pp. 24–31.
[35] M.V. Butz, D.E. Goldberg, K. Tharakunnel, Analysis and improvement of fitness exploitation in XCS: bounding models, tournament selection, and bilateral accuracy, Evolut. Comput. 11 (3) (2003) 239–277.
[36] A. Asuncion, D. Newman, UCI Machine Learning Repository, 2007. URL 〈http://www.ics.uci.edu/~mlearn/MLRepository.html〉.
[37] G. John, P. Langley, Estimating continuous distributions in Bayesian classifiers, in: Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence, 1995, pp. 338–345.
[38] J.R. Quinlan, C45: Programs for Machine Learning, Morgan Kaufmann Publishers, Inc., San Francisco, CA, USA, 1993.
[39] E. Frank, I.H. Witten, Generating accurate rule sets without global optimization, in: Proceedings of the 15th International Conference on Machine Learning, 1998, pp. 144–151.
[40] N. Cristianini, J. Shawe-Taylor, An Introduction to Support Vector Machines and Other Kernel-based Learning Methods, first ed., Cambridge University Press, 2000.
[41] I.H. Witten, E. Frank, Data Mining: Practical Machine Learning Tools and Techniques, second ed., Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.
[42] S. García, A. Fernández, J. Luengo, F. Herrera, A study of statistical techniques and performance measures for genetics-based machine learning: accuracy and interpretability, Soft Comput. 13 (10) (2009) 959–977.
[43] 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, Inf. Sci. 180 (10) (2010) 2044–2064.
[44] D.J. Sheskin, Handbook of Parametric and Nonparametric Statistical Procedures, fourth ed., Chapman& Hall, CRC, 2007.
[45] T.M. Mitchell, Machine Learning, McGraw-Hill, New York, 1997.
[46] I. Kononenko, Estimating attributes: analysis and extensions of RELIEF, in: Proceedings of the 1994 European Conference on Machine Learning, 1994, pp. 171–182.
[47] M. Robnik-Šikonja, I. Kononenko, Theoretical and empirical analysis of ReliefF and RReliefF, Mach. Learn. 53 (2003) 23–69.

Dr. Jiadong Yang received the B.S. degree and the M.S. degree from Tianjin University in 2006 and 2008, respectively and the Ph.D. degree from Tsinghua University in 2012, all in computer science. Now he is working in Jike.com. His research interests include machine learning and evolutionary computation.

Prof. & Dr. Hua Xu received his B.S. from Xi'an Jiaotong University in 1998. He received his M.S. and Ph.D from Tsinghua University in 2000 and 2003, respectively. Now he is working in Department of Computer Science, Tsinghua University. His research fields include the following aspects: data mining, intelligent information processing and advanced process controllers for IC manufacturing equipments. He has published over 30 academic papers, received 10 invention patents of advanced controller and is also the copyright owner of 5 software systems.

Peifa Jia is a professor and doctorial supervisor in Department of Computer Science, Tsinghua University. He is also the former Director of State Key Laboratory on Intelligent Technology and Systems. He is the member of national expert team in the field of advanced manufacturing and automation and the member of National IC Manufacturing Technology Committee in China. His research areas include robotics, virtual reality and network behavior.