Evolving Accurate and Compact Classification Rules With Gene Expression Programming

Chi Zhou, Weimin Xiao, Thomas M. Tirpak, *Member, IEEE*, and Peter C. Nelson

***Abstract*—Classification is one of the fundamental tasks of data mining. Most rule induction and decision tree algorithms perform local, greedy search to generate classification rules that are often more complex than necessary. Evolutionary algorithms for pattern classification have recently received increased attention because they can perform global searches. In this paper, we propose a new approach for discovering classification rules by using gene expression programming (GEP), a new technique of genetic programming (GP) with linear representation. The antecedent of discovered rules may involve many different combinations of attributes. To guide the search process, we suggest a fitness function considering both the rule consistency gain and completeness. A multiclass classification problem is formulated as multiple two-class problems by using the one-against-all learning method. The covering strategy is applied to learn multiple rules if applicable for each class. Compact rule sets are subsequently evolved using a two-phase pruning method based on the minimum description length (MDL) principle and the integration theory. Our approach is also noise tolerant and able to deal with both numeric and nominal attributes. Experiments with several benchmark data sets have shown up to 20% improvement in validation accuracy, compared with C4.5 algorithms. Furthermore, the proposed GEP approach is more efficient and tends to generate shorter solutions compared with canonical tree-based GP classifiers.**

***Index Terms*—Classification rule, data mining, gene expression programming (GEP), genetic algorithms (GAs).**

I. Introduction

RECENTLY, there has been a growing interest in the area of *data mining*, where the goal is to discover useful knowledge from observed data. Among various data mining tasks, extracting classification rules is a fundamental activity. Given a set of predetermined, disjoint target classes $\{C1, C2, \ldots, Cn\}$, a set of input attributes $\{A1, A2, \ldots Am\}$, and a set of training data S with each instance taking the form $\langle a1, a2, \ldots, am\rangle$, where $ai\,(i = 1, 2, \ldots m)$ is in the domain of attribute Ai and associated with a unique target class label, the task is to build a set of IF-THEN rules that can be used to predict the target category for new unseen data given its input attributes' values. Rule induction (e.g., CN2 [10]) and decision tree algorithms (e.g., CART [8] and C4.5 [46]) are traditionally employed to derive classification rules from data. These algorithms can quickly generate rules that are relatively accurate and understandable. The disadvantage, however, is that the generated rules are often more complex than necessary [36]. The reason is that the local, greedy search performed by traditional algorithms selects only one attribute at a time and, therefore, the feature space is approximated by a set of hypercubes. In real-world applications, the feature space is often very complex and a large set of such hypercubes might be needed to approximate the class boundaries between clusters of different classes.

Genetic classifiers, which are based on evolutionary algorithms such as genetic algorithms (GAs) [23] and genetic programming (GP) [31], have been proposed as alternative methods. Based on the principle of natural selection and "survival of the fittest," evolutionary algorithms operate by iteratively evolving a population of chromosomes, encoding candidate solutions, through genetic operators, i.e., selection, crossover, and mutation, to find an optimum solution. Unlike most traditional rule-learning algorithms, genetic classifiers perform a global search in which genetic operators can select many attributes at a time. Possible solutions, i.e., candidate rules are evaluated by the fitness function. One disadvantage of genetic classifiers, however, is that they are usually computationally intensive. Nevertheless, in the cases where off-line computation time is not a limiting factor, a genetic classifier may be more desirable.

Gene expression programming (GEP) [15] is a new technique of evolutionary algorithm for data analysis. GEP uses fixed-length, linear strings of chromosomes to represent computer programs in the form of expression trees of different shapes and sizes, and implements a GA to find the best program. Although Ferreira does not mention this relationship, GEP can be considered a specialization of GP based on linear string representation. As we will see later in this paper, GEP combines the advantages of both GA and GP, while overcoming some of their individual limitations.

In this paper, we propose a new approach for mining classification rules by using GEP technique. The discovered rules are high order in the sense that the rule antecedents can involve any logical or mathematical combination of attributes. The basic idea is to decompose a multiclass problem into multiple binary classification problems and then perform evolutionary search using GEP for better inductive learning of rules based on the covering strategy for each class. The minimum description length (MDL) principle and the integration theory are applied to avoid overfitting and remove redundant rules. The experimental results conducted on several benchmark data sets have demonstrated that our approach achieved up to 20% improvement in validation accuracy and much

Manuscript received August 30, 2002; revised July 2, 2003. This work was supported in part by Motorola's Advanced Technology Center and in part by the Manufacturing Research Center.

C. Zhou, W. Xiao, and T. M. Tirpak are with the Motorola Advanced Technology Center, Motorola, Schaumburg, IL 60196-0178 USA (e-mail: A19387@motorola.com; AWX003@motorola.com; T.Tirpak@motorola.com).

P. C. Nelson is with the Artificial Intelligence Laboratory, Department of Computer Science, University of Illinois at Chicago, Chicago, IL 60607-7053 USA (e-mail: nelson@cs.uic.edu).

Digital Object Identifier 10.1109/TEVC.2003.819261

smaller size of rule sets, compared with the C4.5 algorithms. In comparison with the canonical tree-based GP approach, GEP runs more efficiently and tends to produce shorter solutions. The remaining sections of this paper are organized as follows. In Section II, we review related work of learning classification rules through evolutionary algorithms. The following section contains a brief description of GEP. Section IV describes the proposed GEP approach for classification rule discovery. Then, we present the study of benchmark tests and comparison between GEP, C4.5, and traditional GP classifiers. The final section draws the conclusions and some directions for future work.

II. Related Work

As a robust, domain-independent mechanism for numeric and symbolic optimization, GAs have been applied to evolve a set of production rules for more than two decades, which forms a machine learning paradigm called learning classifier systems (LCS) [33]. The first LCS, called *cognitive system level one* (CS-1), was introduced in 1978 [27]. Since then many different types of classifier systems have been described in the literature. GA-based classifier systems usually fall into two basic classes: the Michigan approach and the Pittsburgh approach. The main difference between these two stems from the chromosome encoding schemes in the population of individuals. In the Michigan approach, e.g., CS-1, each individual with fixed length encodes a single production rule. Whereas in the Pittsburgh approach, each individual is represented by a variable-length string and encodes a complete set of rules, for example, GABIL [13], GIL [29], HDPDCS [45], and LS-1 [50], etc. The Pittsburgh approach is better suited for static domains and batch-mode learning, in which all training examples are available before the learning process starts, and the Michigan approach is more flexible to handle incremental-mode learning, in which training examples arrive over time and dynamically changing domains [11].

In order to alleviate the disadvantages of these two approaches, some hybrid Michigan/Pittsburgh methodologies have been proposed, for example, REGAL [22], COGIN [24], and DOGMA [25]. In the SIA system [54], a covering (sometimes called separate-and-conquer) strategy [20] was applied to learn one rule at a time that covers part of the training examples until all examples are covered and then combine all the discovered rules together to form the target concept. A system called GRaCCE uses a multistage GA-based approach to first reduce the feature set and then locate class-homogeneous regions within the data to generate classification rules [37]. Most GA-based classifier systems proposed in the literature address the task of rule extraction in the form of propositional logic, while REGAL and SIAO1 [1] can learn first-order-logic (FOL) concept descriptions. The systems described in [26] and [28] can generate an appropriate set of fuzzy rules from examples using GAs. In general, the individuals in GA-based classifiers, i.e., candidate rules, are usually encoded as binary strings and the rule quality is accessed by a fitness function. Most fitness functions proposed in the literature favor more accurate and comprehensible rules [13], [17], [29], [45]. Noda *et al.* introduced a rule "interestingness" term in the fitness function in order to discover interesting rules [43].

Koza introduced the idea of using GP to induce a decision tree classifier, which was represented by a LISP S-expression [32]. Since then, several investigations have employed GP to develop decision trees [6], [38], [42]. Such classifiers limit the generated programs to decision tree structures, which are more constrained than standard genetic programs returning real values. Since the 1990s, many GP-based frameworks have been studied for discovering explicit classification rules instead of decision trees [4], [14], [19], [30], [51], [52]. GP is receiving more attention recently because unlike most data mining algorithms, GP can discover the underlying relationships in the data and express them in any logical, mathematical combinations of input attributes. GP manipulates variable size genomes, thus allowing to adapt better the solution structure to the data compared with GA, therefore, GP is more open-ended than GAs. But this comes with a cost, i.e., GP is more difficult to navigate in much larger search spaces. In [18], Freitas presents a good survey of existing data mining approaches with evolutionary algorithms, i.e., GA and GP.

To develop a standard GP-based classifier for a given problem, one must first define the GP's terminal set and function set. The terminal set usually consists of all input attributes and a random number generator; the function set may contain some mathematical, comparative, and logical operators. Each individual, i.e., a parse tree, in the GP population, encodes a candidate rule and the objective is to minimize the classification error rate through genetic manipulations, where classification is done by comparing the output of the GP expression to a given threshold. For a two-class problem, one GP expression is sufficient to predict whether or not a given feature vector belongs to one class. The division between negative and nonnegative output values acts as a natural boundary between two classes. In this way, for an n-class ($n > 2$) problem, multiple threshold "bands" need to be determined. However, finding meaningful division points over the set of numeric values the GP expressions may return is difficult. There exist two methods to select the bands: static range selection and dynamic range selection [34]. Another simple and often-used approach for solving multiclass problems using GP is to break the n-class problem down into n binary classification problems and run the GP n times, each time solving a binary problem [14], [30]. This method is called "binary decomposition" [34] or "one-against-all learning" [21]. For each class, one GP expression is generated to predict whether a given instance belongs to that class or not.

GP-based classifiers also have some weak spots. For example, the closure property of standard GP requires that all the variables, constants, arguments for functions, and values returned from functions must be of the same data type. This property is satisfied when the standard GP is applied on classification problems with numeric data. Some systems [4], [12], [14], use only Boolean attributes or booleanize all the attributes being mined, and then apply logical operators in order to meet the closure property. *Strongly typed GP* (STGP) [41] (sometimes called constrained-syntax GP) and the grammar-based GP have

been proposed to deal with this problem when addressing classification problems with a mixture of continuous and nominal attributes GP [5], [55]. Moreover, GPs tree-based individuals typically result in bloating [56], [48], i.e., uncontrolled growth in the size of individuals over the course of genetic manipulations, such as subtree crossover and various kinds of mutation. The bloat can be controlled by proper coordination between the fitness function and the genetic learning operators. Only when there is such a proper coordination the search may progress successfully.

III. Overview of Gene Expression Programming

Like GP, five general components, the function set, terminal set, fitness function, control parameters, and stop criteria [31], must be determined when using GEP to solve a problem. Unlike the parse tree representation in canonical GP, GEP uses a fixed-length of character strings to represent computer programs, which are afterwards expressed as parse trees (called "expression tree" in GEP) of different sizes and shapes when evaluating their fitness. During reproduction it is the chromosomes of the individuals, not the expression trees (ETs), that are reproduced with modification and transmitted to the next generation. Thus, in GEP, the search space is separated from the solution space, which can result in benefits such as unconstrained search of the genome space, while still ensuring validity of the program's output as noted in [2]. The original GEP technique was proposed by Ferreira [15], in which GEP chromosomes may consist of one or more genes of equal length. In this paper, we consider one-gene chromosomes and use a slightly different version described as follows.

A. GEP Chromosomes and ETs

Each GEP chromosome is composed of a list of symbols with a fixed length, which can be any element from the function set and the terminal set. For example, from the function set $\{+, -, *, /, \mathrm{sqrt}\}$ and the terminal set $\{1, a, b, c, d\}$, a typical GEP chromosome (with size 15) can be

$$\mathrm{sqrt}.*.+.*.a.*.\mathrm{sqrt}.a.b.c./.1.-.c.d \tag{3.1}$$

where "." is used to separate elements for easy reading; sqrt is the square-root function; 1 is a constant; and a, b, c, d are variable (or attribute) names. The above is typically named as Karva notation, or K-expression [15]. A K-expression can be mapped into an ET following a width-first fashion. The conversion starts from the first position in the K-expression, which corresponds to the root of the ET and reads through the string one-by-one. For each node (from left to right) in one layer in the ET, if it is a function with $n (n >= 1)$ arguments, then the next n symbols in the K-expression are attached below it as n child branches. Otherwise, each terminal node forms a leaf of the ET. This tree expanding process continues layer-by-layer until all leaf nodes in the ET are composed of elements from the terminal set. For example, the sample chromosome (3.1) can be expressed as Fig. 1(a), which can be further expressed in a mathematical form as

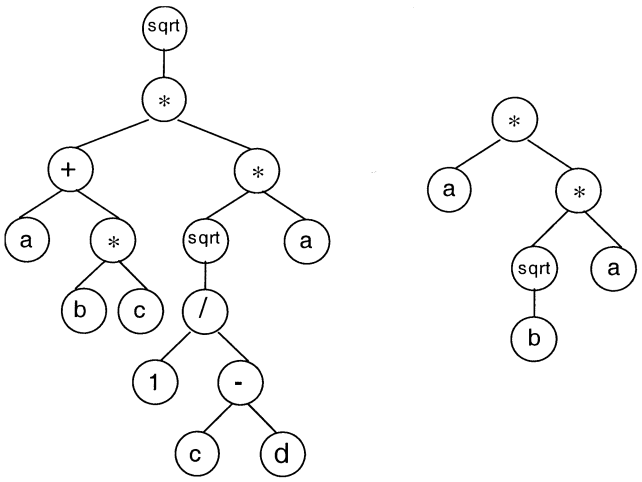

Fig. 1. Example of GEP expression trees.

$$\sqrt{(a+b*c)^{*}\, a\sqrt{\frac{1}{c-d}}}. \tag{3.2}$$

The inverse process, i.e., the conversion of an ET into a K-expression, is also very straightforward, just recording the nodes from left to right in each layer of the ET, from root layer down to the deepest one to form the string, e.g., the expression tree in Fig. 1(a) is recorded as chromosome (3.1).

Like GP, the function set and terminal set must have the *closure property*: each function must be able to take as its arguments any value of data type which can be returned by a function or assumed by a terminal. Furthermore, in our implementation, mathematical errors are prevented by using protected functions. For instance, if division by zero is attempted, protected division returns the value of division by a very small number $(\varepsilon = 1\mathrm{E} - 6)$.

As stated before, GEP chromosomes have fixed length, which is predetermined for a given problem. Thus, in GEP, what varies is not the length of chromosomes, but the size of the corresponding expression trees. This means that there exist a certain number of redundant elements, which are not useful for the genome-ET mapping. For example, the following chromosome:

$$\underline{*.a.*.\mathrm{sqrt}.a.b}.c./.1.-.c.d.\mathrm{sqrt}.*.+ \tag{3.3}$$

has the same length of 15 as chromosome (3.1), but its valid K-expression size is 6, i.e., only the first six elements are used to construct the solution function $a^{*}(a^{*}\sqrt{b})$, with the corresponding expression tree shown in Fig. 1(b).

So the valid length of a K-expression may be equal or less than the length of the chromosome. In order to guarantee that only legal expression trees are generated, the original GEP technique employs a head-tail method. Each chromosome is composed of a head and a tail. The head may contain symbols from

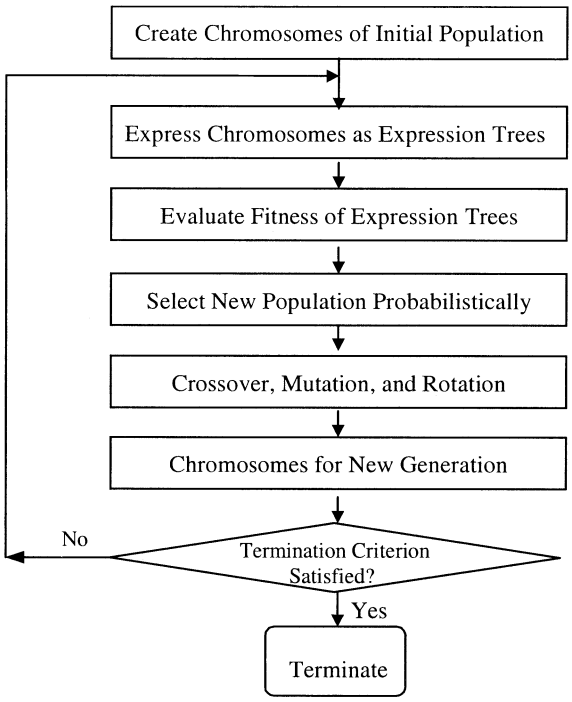

Fig. 2. Flowchart of a typical GEP algorithm.

both the function set and the terminal set, whereas the tail contains only terminals. In our GEP implementation, we applied a validity test program to dynamically check if a chromosome is able to encode a legal expression tree within the size limit, instead of using the head-tail method. The program reads through the symbol list of a candidate chromosome sequentially (from left to right) and accumulates the number of arguments required for each symbol (the number of arguments for a terminal symbols is zero) into a variable, which was set to one initially, since the shortest expression consists of one terminal. If at a certain point within the list, the value of this variable equals the total number of symbols (including the current one) that have been scanned so far, the chromosome is a valid one. Otherwise, if the two numbers cannot match till the end of the chromosome, it is invalid. All chromosomes randomly generated or reproduced by genetic operators are subject to this test to prevent illegal expressions coming up.

B. The GEP Algorithm and Operators

Fig. 2 illustrates the flowchart of a typical GEP algorithm. Like GAs and GP, the GEP algorithm begins with an initial population of chromosomes, which are randomly generated, linear strings with a fixed length. Then, the linear chromosomes are expressed as ETs and the fitness of each individual is evaluated based on a predefined fitness function. The individuals are then selected according to fitness to form a new generation, i.e., the higher the fitness value, the more chance an individual has to be selected. The selected individuals are also subject to reproduction with modification, through genetic operators like crossover, mutation, and rotation (which will be described later). The individuals of this new generation are, in their turn, subjected to the same developmental process: expression of the genomes, selection, and reproduction. The process is repeated for a certain number of generations or until a solution has been found.

In GEP, individuals are often selected and copied into the next generation according to fitness by roulette-wheel sampling [23] with elitism, which guarantees the survival and cloning of the best individual to the next generation. Variation in the population is introduced by conducting genetic operations on selected chromosomes, as illustrated in Fig. 3, which include the following.

1) Crossover, in which two parent genomes are randomly chosen and paired to exchange some elements between them. There are two kinds of crossover: one-point, and two-point crossover, working in the same fashion as that in GAs.
2) Mutation, in which the symbols at any position in a genome are subject to a random change according to a certain probability. Note that like crossover, a mutation in the coding sequence of a chromosome usually drastically reshapes the ET. For example, the case of mutation in Fig. 3 has changes at two positions, which results in the corresponding ET reshaping as shown in Fig. 4.
3) Rotation, in which two subparts of element sequence in a genome are rotated with respect to a randomly chosen point. Rotation can also drastically reshape the expression trees, as shown in the example chromosomes (3.1) and (3.3). Note that chromosome (3.3) was produced by rotating the first three elements of chromosome (3.1).

The output chromosomes from these operators must pass the validity test to ensure that they can form valid expression trees within the predefined chromosome size limit. If an individual produced by a genetic operator does not pass the test, the operator will be performed repeatedly until the offspring passes the test. For example, one-point crossover will be repeated on the same parents (not the invalid chromosome) but at a different crossover point if either of the children fails the test. An analysis of the evolutionary dynamics for each of these operators can be found in [16]. Although GEP may also evolve expressions with redundant complexity, i.e., subexpressions that could be replaced by much simpler or smaller expressions that yield the same result, the bloating problem is much less serious that traditional GP. Since the valid length of a K-expression never exceeds the limit of the predefined chromosome length, GEP has the tendency to produce shorter programs.

In summary, like GAs, the chromosomes in GEP are linear, compact, and easy to genetically manipulate; like GP, the evolved computer programs in the form of expression trees exhibit a certain amount of functional complexity. And the intertranslation of chromosomes and ETs is pretty straightforward. Moreover, GEP exhibits more simplicity, i.e., the user need not explicitly specify the genotype-to-phenotype mapping (e.g., through a BNF grammar), compared with other linear GPs such as binary GP (BGP) [2], GADS [44], or grammatical evolution (GE) [49]. On account of these characteristics, i.e., simplicity, high efficiency, and functional complexity, GEP combines the advantages of both GAs and GP, while overcoming some of their limitations, which offers great potentiality to solve complex modeling and optimization problems.

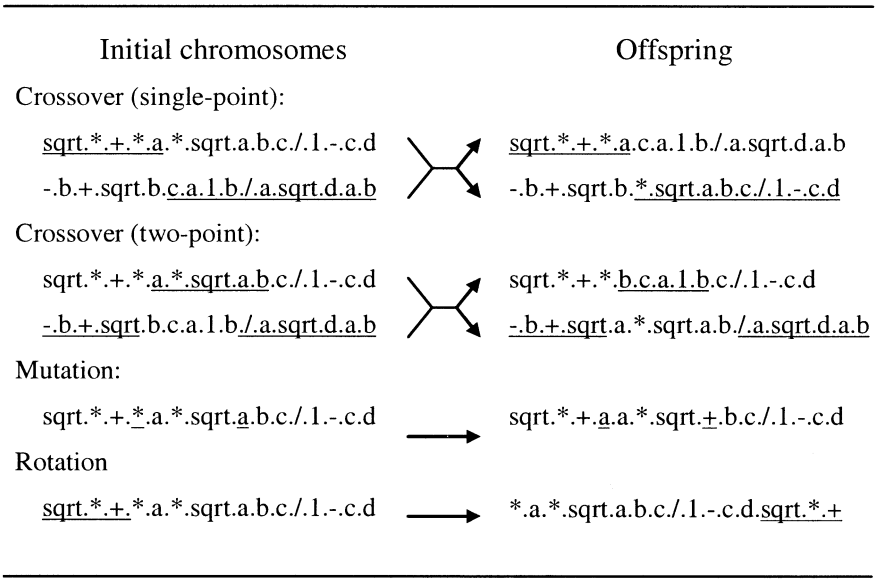

Fig. 3. Genetic operators of the GEP algorithm.

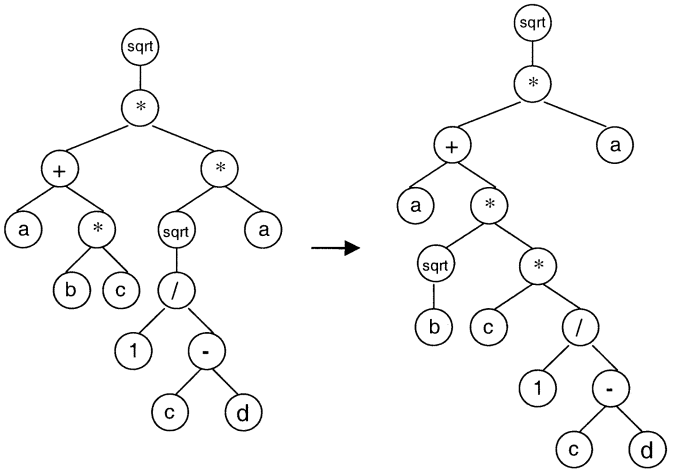

Fig. 4. Reshaping of ETs from the mutation in Fig. 3.

IV. Classification Through GEP

As a global search technique using GAs, GEP has exhibited great potential for solving complex problems [15]. The reasons to use GEP for classification include the following.

1) Flexibility: GEP is more flexible than traditional rule induction and decision tree algorithms, since it more flexibly reformulates the underlying representations.
2) Capability: Due to the employment of the powerful evolutionary search mechanisms, it can discover relationships in the form of a combination of attributes and express them mathematically.
3) Efficiency: With the help of linear representation of chromosomes with fixed-length and easy manipulation of genetic operations, GEP is more efficiently than traditional tree-based GP.

A. One-Against-All Learning

For a two-class (binary) problem, the GEP expression performs classification by returning a positive or nonpositive value indicating whether or not a given instance belongs to that class, i.e.,

$$\text{IF } \text{GEP}_j(\mathbf{X}) > 0 \quad \text{THEN } \mathbf{X} \in \text{Class } j \text{ ELSE } \mathbf{X} \notin \text{Class } j \ (j=0,1)$$

where $\mathbf{X}$ is the input feature vector. To extend this approach to n-class classification problems, where $n > 2$, we adopt the often-used *one-against-all* learning method to transform the n-class problem into n 2-class problems. These are constructed by using the examples of class i as the positive examples and the examples of classes other than i as the negative examples.

In our early work in [57], for each class, only one rule is evolved to recognize instances belonging to its own class and reject instances of other classes. We have found that many real-world problems have complex feature spaces for which a single rule may not adequately classify all instances of a given class. In this paper, we apply the covering strategy to learn multiple rules for each binary classification problem. For each class in the given training set, we learn a rule that covers as many positive examples as possible and as few negative examples as possible. Then, we remove all positive examples that have been covered from the training set and repeat the procedure to learn another rule. These steps are repeated until no positive examples remain in the training set.

There are two additional issues to be considered in developing an efficient classifier. One is the problem of over fitting noisy data. In our approach, the MDL principle [47] is applied to avoid overly specific rules that try to fit noisy examples. Another issue arises when the generated GEP rules are applied to new cases. It is possible for more than one GEP expression to return a positive value, which results in a "classification conflict." In the worst case, all rules may not be satisfied. So there is the need for a strategy to solve classification conflict and rejection. These problems are addressed later in this paper as our technique of two-phase rule pruning is explained.

B. Function Set and Terminal Set

For a given classification problem, we need to define the function set and terminal set for GEP. The function set usually consists of mathematical and logical operators, e.g., $+$, $-$, $*$, $/$, sqrt, IF. IF is a logical comparison function with three arguments (x, y, z), which take the value: *if* $x > 0$, then y else z. Using the IF operation, GEP can manipulate piecewise-continuous functions, which are very useful for real-world classification problems.

The terminal set consists of all input attributes and a list of constants, e.g., $\{1,\ 2,\ 3, \ldots,\ N\}$. In GP literature [31], there exist two ways to introduce constant values into computer programs: one is using a random number generator; the other is using a set of constants. In our implementation, we follow the second one, which enables GEP to represent any rational number using a given combination of constants and functions.

C. Numerization of Nominal Attributes

All operands are considered numeric in our GEP implementation. There are two ways to perform numerization of nominal attributes. One method is to map the values of a nominal attribute to integers, so that the attribute can be considered as numeric inside the GEP. For example, if an attribute can take three possible values, then these nominal values are mapped into a set of integers, i.e., $\{0, 1, 2\}$. The disadvantage of this approach, however, lies in the fact that it imposes an order that does not exist in the original data. Another method is to divide a nominal attribute into n binary attributes, which is called "binarization," if there are n possible values (here, $n > 2$), with 0/1 representing the absence/presence of each value. This method overcomes the shortcomings of the first-integer approach, but will generate a large set of derived attributes if n is large. In this paper, we use the binarization approach.

D. Fitness Function

The problem of defining fitness functions to measure the rule quality remains an interesting issue in data mining. Ideally, we should incorporate three criteria in a fitness function, i.e., predictive power, comprehensibility, and interestingness [18]. Here, we focus on finding accurate rules. Several formulas integrating completeness (the ratio of positive examples covered by the rule to total number of positive examples in the training set) and consistency (the ratio of positive examples covered by the rule to the total number of examples covered) have been described in the literature [9]. Rather than using a consistency metric, our approach incorporates rule consistency gain [39], because this measure takes into account the distribution of positive and negative examples in the training set. The consistency gain of rule R is defined as

$$consig(R) = \left(\frac{p}{p+n} - \frac{P}{P+N}\right) * \frac{P+N}{N}$$

where p and n are the number of positive and negative examples covered by rule R, and P and N are the total number of positive and negative examples in the training set. In the case that a rule have the identical distribution of covered positive and negative examples as the training set (just like a random guess), the consistency gain is zero; and in the case of perfect consistency, it will return one. It is also possible for a rule to return a negative consistency gain, which signifies that the rule is less accurate than a pure random guess.

In this work, the fitness function has the form

$$\text{fitness}(R) = \begin{cases} 0, & \text{if } consig(R) < 0 \\ consig(R) * \exp(compl(R) - 1), & \text{otherwise} \end{cases}$$

where $compl(R) = p/P$ is the rule completeness.

This formula is similar to the one defined by Brazdil and Torgo in [7]. The use of the *exp()* function makes the quality measure prefer the consistency gain of the rule. This fitness function has the advantages of being simple and returning a normalized value in the range $[0 \ldots 1]$.

E. Two-Phase Rule Pruning

In order to evolve an accurate, noise-tolerant and compact rule set, we apply two phases of rule pruning: prepruning and postpruning. For each binary classification task, ideally the covering strategy works by iterating the process of learning a rule that covers part of the training examples until no positive examples remain. This method, however, may not be desirable if the training data are not noise free, because it tends to generate some specific rules that fit the noisy data very well, but do not generalize the concept to be learned well, thus resulting in overfitting. One common approach to deal with this problem is early stopping [40]. In the prepruning process, we employ a stop criterion based on the MDL principle to determine when to stop learning more rules for that class. The MDL principle tries to avoid learning complicated rules that cover only a small number of examples, which may result from noisy data.

The central idea of the MDL principle originates from the information theory. Given some observed data and a collection of possible theories that might explain the data, we want to transmit the concept membership of each instance across a wire. The information being transmitted, i.e., the description length, includes the amount of bits needed to encode both the theory (e.g., a set of rules) and the exceptions, i.e., all positive examples not covered by the theory (false-negatives) and all negative examples erroneously covered by the theory (false positives). Following the Occam's razor principle [40], the MDL principle states that the best theory is the one that minimizes the total description length.

In our approach, we follow the method used in C4.5 [46] and define the description length of a rule set H as

$$L(H) = L_{\text{exception}}(H) + W^* L_{\text{theory}}(H) \qquad (4.1)$$

where $0 < W < 1$ (here, $W = 0.5$). $L_{\text{exception}}(H)$ *is* the number of exception bits defined as:

$$L_{\text{exception}}(H) = \log_2\left(\binom{n}{N_{fp}}\right) + \log_2\left(\binom{N-n}{N_{fn}}\right) \qquad (4.2)$$

Algorithm: Covering(E^+, E^-)
Input: E^+ set of positive examples, E^- set of negative examples
Output: A set of GEP rules H

Initialize the variables:
$H := \emptyset$;
$L_{\min} := 999999$ (minimum description length obtained so far);
$L_H := 0$ (current description length);
$L_{theory} := 0$ (theory bits).
Repeat
Apply the GEP algorithm to learn a rule R covering some elements in E^+
1) $E^+ = E^+ - \{e \mid e$ is covered by $R\}$.
2) $L_{theory} = L_{theory}$ + number of bits encoding rule R,
$L_{exception}(H)$ =number of bits encoding the current exceptions,
$L_H = 0.5 * L_{theory} + L_{exception}(H)$.
3) If ($L_H \geq L_{\min}$), Then Stop; Otherwise $H = H \cup \{R\}$.
4) If $L_{\min} > L_H$, then $L_{\min} := L_H$.
Until $E^+ = \emptyset$.

Fig. 5. Covering algorithm involving a rule prepruning process.

where n is the number of examples covered by the rule set H, N is the total number of training examples, and N_{fp} and N_{fn} are the number of false positives and false negatives.

$L_{\text{theory}}(H)$ is the number of theory bits defined as

$$L_{\text{theory}}(H) = \log_2(N_c) * \sum_{i=1}^{s} L(R_i) \tag{4.3}$$

where s is the number of rules, $L(R_i)$ is the valid chromosome size for expression R_i, and N_c is the total number of different symbols used in GEP, i.e., the total number of functions, variables and constants. For each rule R_i, we need $\log_2(N_c) * L(R_i)$ bits to encode the GEP chromosome list. There is no need to specify the right-hand side of the rule, since all rules are related to the same class for each binary classification. Based on the MDL principle, a stopping criterion is applied for each binary classification task to avoid learning more rules that may overfit the data, which is called prepruning. After each rule is evolved by GEP and added to the rule set, the total description length of the rule set is computed according to formula (4.1)–(4.3). The covering algorithm (described in Fig. 5) stops adding rules when this description length is larger than the smallest description length obtained so far, or when there are no more positive examples.

The prepruning phase deals with noise during rule learning. The second phase of pruning, which is performed after the rule-learning process has been completed for all classes, tries to improve the learned theory and resolve any classification conflicts and rejections. In general, there are two methods to solve the conflict problem [9]. One is to make the unordered rules an ordered list according to their confidences or qualities. For example, Kishore *et al.* proposed a measure named "strength of association" (SA) as the criterion of ordering a set of GP rules [30]. The SA value indicates the degree to which a GP expression can recognize examples belonging to its own class, and reject examples of other classes. The higher the value of SA, the better the predictive performance of the corresponding rule will be. Then, the class of the GP expression with highest strength of association will be assigned to the input case. The other approach also considers rule quality, but goes through the entire set of rules, combines the qualities of the matched rules of the same class and assigns the new example to the class for which the combined quality is the maximum. For example, the CN2 unordered rule induction algorithm [10] follows this approach.

Since the fitness function used in our approach already takes into account the rule consistency, completeness, as well as the class distribution, it provides a natural criterion for ordering the GEP rules. We have employed the following strategy for conflict resolution and pruning redundant rules, which follows the first scheme mentioned and the integration theory in [7].

1. Order all generated rules according to their fitness values.
2. Select the rule with the highest fitness value, and add it to the ordered rule set.
3. Remove all positive and negative examples covered by the selected rule.
4. Recompute the fitness values of the remaining rules on the remaining examples.
5. Repeat steps 1–4 until there is no example left, or until none of the remaining rules yields a positive fitness value.
6. A default class is selected in case that all rules reject a new example.

The method for selecting the default class again follows the way used in C4.5 rule induction algorithm. For each training example, the ordered rules generated through steps 1–5 are applied one by one in that order until one of them is fired. We only consider the examples that cannot return a positive value by all the GEP expressions, and select the class that has the largest number of unclassified examples as the default class, resolving ties in favor of the class with the larger number of instances in the training set. After this postpruning process, redundant rules are eliminated. The final rule set is compact and ordered.

V. Empirical Evaluation

In order to test the proposed approach of GEP for classification, we have applied it to several data sets selected from the UCI repository of machine learning databases [3].

A. Monk's Problems

The Monk's problems are created as a benchmark test for comparison of different concept learning algorithms [53]. The three classification problems are derived from an artificial robot domain described by six different nominal attributes:

- a1: head_shape: {round, square, octagon};
- a2: body_shape: {round, square, octagon};
- a3: is_smiling: {yes, no};
- a4: holding: {sword, balloon, flag};
- a5: jacket_color: {red, yellow, green, blue};
- a6: has_tie: {yes, no}.

TABLE I
TEST ACCURACY ON MONK'S PROBLEMS

	M1	M2	M3
C4.5	75.70%	65%	97.20%
C4.5Rules	100%	66.20%	96.30%
GEP	**100%**	**99.07%**	**100%**

Each learning task involves finding a logical description of a class that can differentiate whether or not robots belong to the class.

- **Problem M1** [124/432]:

$$\text{IF } (a1 = a2) \text{ OR } (a5 = \text{red}) \quad \text{THEN class } 1 \tag{5.1}$$

- **Problem M2** [169/432]:

$$\text{IF Exactly two of the six attributes have their first value THEN class } 1 \tag{5.2}$$

- **Problem M3** [122/432]:

$$\text{IF } (a5 = \text{green AND } a4 = \text{sword}) \text{ OR } (a5 <> \text{blue AND } a2 <> \text{octagon}) \text{ THEN class } 1 \tag{5.3}$$

The numbers in the bracket are the numbers of instances in the training set and testing set. Problem 3 has 5% misclassifications, i.e., noise in the training set.

Since all attributes are nominal, the proposed binarization method was used to make these values numeric. For example, attribute $a5$ is divided into four binary attributes $a5\_1, a5\_2, a5\_3$, and $a5\_4$. An $a5$ value will have only one of these four attributes taking 1 and all others taking 0, e.g., if $a5 = \text{green}$, then $a5\_3$ takes 1, and the other three take 0. The GEP function set includes $\{\text{IF}, +, -, *, \text{OR}, \text{AND}\}$, and the terminal set contains constants $\{1, 2, 3\}$ and all the attribute names. Here, OR and AND are two logical functions with two arguments (x, y), which takes the following semantics: *if* $x > 0$ *or* $y > 0$, *then* $OR(x, y) = 1$, *otherwise* $OR(x, y) = 0$; *if* $x > 0$ *and* $y > 0$, *then* $AND(x, y) = 1$, *otherwise* $AND(x, y) = 0$. We compare the GEP classification results with those from the benchmark machine learning algorithms, i.e., C4.5 and C4.5Rules. Table I presents the test accuracy from these algorithms. (GEP results were obtained using the following control parameters: fixed chromosome length $= 100$, crossover probability $= 0.7$, mutation probability $= 0.02$, rotation probability $= 0.02$, population size $= 1000$, and maximum generations $= 5000$, see Table I.)

The classification rule sets found by GEP achieved the highest test accuracy on all three problems (one tie with C4.5Rules on M1). GEP also exhibits noise tolerance on problem M3. Actually, the performance of GEP on Monk's problems is outstanding compared with many other learning algorithms studied in [53].

The rules obtained by GEP learning are somewhat more difficult to understand, but they can be transformed to exactly the same disjunctive or conjunctive normal form (DNF/CNF) after logical analysis. For example, GEP found the following rule for class 1 of task M1:

$$\text{IF OR}(a5\_1, \text{IF}(a2\_2 > 0, a1\_2, (\text{IF}(a2\_3 > 0, a1\_3, a1\_1))) > 0 \text{ THEN class } 1 \tag{5.4}$$

```
Rule 1:
  IF  a4_1-((a3+(a6-(a2_1+(AND(AND((a1_1 >0 ? a5_3 :
      a5_3+a5_2), a3), (a2_3 >0 ?a1_3 : a5_2))-a2_1 >0 ?
      (AND(a5_4, a5_3) >0 ?a1_3 : 2.0) : a5_1))))-a1_1)>0
  THEN  class 0
Rule 2:
  IF  (a6-(a5_1+(a2_1+(a4_1-(a3-a1_1)))))*2.0 > 0
  THEN  class 0
Rule 3:
  IF  AND(OR(AND(a2_1, a3-((((a5_3 >0 ?a1_3 : a6-a2_1)+
      (AND (a1_1, a4_1)))+a4_3)*(1.0-OR(a4_1, a1_1)))), a6-
      (OR(a4_1, a3))), a5_2) >0
  THEN  class 1
Default class: 1
```

Fig. 6. GEP rules for Monk's problem 2.

```
Rule 1:
  IF  (a2_3 >0 ?a4_3+(a5_3 >0 ?a4_2 : 2.0) : a5_4) > 0
  THEN  class 0
Rule 2:
  IF  (a5_4 >0 ?AND(a2_1, a5_2)-1.0 : a2_1)+(((a5_3*
      a2_3)* a4_1)+a2_2) > 0
  THEN class 1
Default class: 0
```

Fig. 7. GEP rules for Monk's problem 3.

Rule (5.4) has exactly the same functionality as rule (5.1). Examples of the rules generated for problems M2 and M3 are shown in Figs. 6 and 7 individually. Some of them are even more complicated. However, the rules generated by GEP exhibit better compactness, compared with the C4.5Rules algorithm. For example, C4.5Rules produced twelve disjunctive rules (four for class 1, and eight for class 0) for task M1, and it is more difficult for human beings to derive the underlying concept as the logical description (5.1).

B. Other Problems

To further validate the proposed GEP approach for classification, we tested GEP in 12 other benchmark data sets from the UCI repository. A brief description of the data-set properties is presented in Table II, which gives the number of instances, attributes, and distinct class labels for each data set. To evaluate the classification accuracy, we used fivefold cross-validation (CV), which consists of dividing data into five subgroups. Each subgroup's examples are classified by the classification rules constructed from the remaining four subgroups and the estimated accuracy rate is the average accuracy from these five subsamples.

The function set and terminal set for GEP are often chosen according to the user's knowledge of each problem. In all the following experiments, we used the function set $\{+, -, *, /, \text{sqrt}, \text{IF}\}$. The terminal set contained all input attributes for each problem and a list of constants {1, 2, 3, 5, 7}. The choice of the GA parameters, e.g., chromosome size, population size, probability of crossover, and mutations, also

TABLE II
CHARACTERISTICS OF THE BENCHMARK DATA SETS

Database	No. Cases	No. Attr.	No. Class
Balance Scale	625	4	3
Breast Cancer W.	683	9	2
Car	1748	6	4
Glass	214	9	6
Heart Disease	270	13	2
Ionosphere	351	34	2
Iris	150	4	3
Lung Cancer	32	56	3
Pima Indian	768	8	2
Waveform	5000	21	3
Wine	178	13	3
Zoo	101	17	7

No. Case is the number of cases, **No. Attr.** is the number of attributes, and **No. Class** is the number of classes.

contribute much to the success of rule discovery. In our experiments, we used the same values of 0.8 for crossover, 0.02 for mutation, and 1000 for population size on all the databases. The chromosome size and the maximum number of GA iterations to execute may depend on the characteristics of the problems such as dimensionality. For each binary-class learning, the GEP algorithm stops when the number of generation reaches 1000 (except for the *pima Indian* data, which uses 5000 generations) or the fitness measure reaches 1.0, which means it correctly classifies all positive and negative examples. The chromosome size for the *balance scale*, *iris*, *ionosphere*, and *lung cancer* databases were set to 50, 80, 150, and 150 individually, while all other data sets using 100 as the fixed chromosome length.

1) Comparison of Classification Accuracy: The performance of GEP is compared with the canonical tree-based GP classifier as well as the C4.5 programs. A Koza-style GP system was implemented. To make the GP classifier comparable to the GEP approach, we used the same function set, terminal set, fitness function, and GA parameters as those used in GEP for each problem. We also employed the same covering algorithm and two-phase pruning strategy for rule induction in GP, as described in Section IV. To avoid GP program bloating, we specified a maximum number of nodes allowed to grow in GP trees, which equals to the size of fixed-length chromosomes used in GEP for each benchmark data. Considering the stochastic behavior of GAs, for each data set, both GEP and GP classifiers were run five times independently.

Table III presents a comparison of classification accuracy in percentage for the four algorithms, i.e., C4.5, C4.5Rules, GEP, and GP. The results were obtained from fivefold CV in terms of their average accuracy and 95% confidence interval, which is proportional to the standard deviation. For GEP and GP, the average results, as well the best results for five independent runs are reported. The best result was obtained from the run that achieved the highest accuracy of fivefold CV. Considering the best performance of GEP and GP, we found that GEP was able to achieve the highest classification accuracy on 11 out of the 12 databases, among which two have ties with GP. The only exception is the *pima Indian* data.

The result of GEP and GP was also compared with respect to their average performance, i.e., the average of five runs, thereby enabling a fair comparison with conventional classification methods. GEP achieved the highest average accuracy on 7 out of the 12 databases, among which three also had the lowest 95% confidence interval, i.e., *balance scale*, *car*, and *iris* (highlighted in bold face in Table III). For 6 of the 12 benchmarks, GEP achieved both higher accuracy and a smaller confidence interval, compared with the results for C4.5 and C4.5Rules. A comparison of GEP with GP showed that GEP achieved higher average accuracy for 11 of the 12 benchmarks and matched the performance for the remaining one (i.e., the *waveform* data).

The two-tailed t-test [40] method was also used to determine the level of significance that one algorithm outperforms another. Using this method, good outcomes should have high accuracies and low standard deviations. The t-test comparing the average results from GEP and GP indicated that GEP outperformed GP significantly on the *car* data (with $>$99% significance), and there is no significant difference for the remaining 11 benchmarks. The t-tests also showed that GEP outperforms C4.5 and C4.5Rules significantly on the *balance scale* data (with $>$99% significance). GEP also outperformed C4.5 on the *car* data (with $>$99% significance). For the other benchmarks, there was no significant difference between GEP and C4.5, or C4.5Rules.

The *balance scale* data was generated to model the results of psychology experiments. Each example is classified as having the balance scale tip to the right, tip to the left, or be balanced. The four predicting attributes are $A1$ (the left weight), $A2$ (the left distance), $A3$ (the right weight), and $A4$ (the right distance). The correct way to find the class is the greater of ($A1$*$A2$) and ($A3$*$A4$). If these two terms are equal, the scale is balanced. Rules (5.5), (5.6), and (5.7) show one example set of identified GEP rules, for which the default class is Left

$$\text{Rule 1}: \quad \text{IF } \frac{A1}{A4} * A2 - A3 > 0 \quad \text{THEN} \quad \text{class} = \text{Left}. \tag{5.5}$$

$$\text{Rule 2}: \quad \text{IF } 1 - 7.0^* \left(\frac{A4^*A3}{A1} - A2 \right) > 0 \quad \text{THEN} \quad \text{class} = \text{Balanced}. \tag{5.6}$$

$$\text{Rule 3}: \quad \text{IF } \frac{A4}{A2} * A3 - A1 > 0 \quad \text{THEN} \quad \text{class} = \text{Right}. \tag{5.7}$$

It should be noted that GEP successfully found the boundary between the three classes, i.e.,

$$(A1^*A2) - (A3^*A4) = 0. \tag{5.8}$$

As shown in Table III, GEP was always able to achieve a testing accuracy of 100% for the *balance scale* data, more than 20% more accurate than the C4.5 and C4.5Rules algorithms. It appears that the reason for this significant difference in performance is that whereas GEP generated the appropriate

TABLE III
COMPARISON OF CLASSIFICATION ACCURACY IN PERCENTAGE

Database	C4.5	C4.5Rules	Average		Best	
			GEP	GP	GEP	GP
Balance Scale	78.7±6.8	77.3±7.1	**100.0±0.0**	99.8±0.3	100.0±0.0	100.0±0.0
Breast Cancer W.	94.7±1.5	95.6±1.6	96.2±1.8	95.3±1.2	96.5±2.0	96.5±1.6
Car	91.0±2.4	93.2±1.9	**94.6±1.7**	91.0±1.5	96.8±2.0	92.8±2.8
Glass	65.7±5.6	65.8±8.4	63.9±8.8	54.2±6.1	70.1±4.7	60.3±7.7
Heart Disease	77.7±4.4	80.6±5.1	79.9±5.2	78.8±6.8	82.2±7.6	81.1±9.1
Ionosphere	91.1±8.6	90.0±8.3	90.2±2.4	90.0±1.7	92.6±2.4	92.0±3.0
Iris	93.9±8.1	94.6±8.2	**95.3±4.6**	92.9±4.7	96.0±3.2	94.0±5.0
Lung Cancer	44.3±23.6	38.6±16.1	54.4±15.6	42.7±13.5	57.6±25.0	46.7±17.0
Pima Indian	74.8±4.7	75.4±4.3	69.7±3.8	67.7±4.2	74.9±4.5	71.0±4.7
Waveform	76.2±1.8	77.2±1.5	76.6±1.4	76.6±0.8	77.9±1.5	77.5±0.9
Wine	91.6±8.1	91.6±8.1	92.0±6.0	90.7±5.5	93.8±2.7	92.0±4.9
Zoo	92.0±7.7	91.0±7.5	93.9±6.9	92.1±5.3	95.1±7.8	95.0±5.7

Results from fivefold CV with the benchmarks are stated in terms of their average accuracy and 95% confidence interval. For GEP and GP, the best and average results out of five independent runs are listed.

TABLE IV
COMPARISON OF GEP/GP EFFICIENCY

Database	GEP/GP Size Limit	Avg. Exp. Size		Exec. Time (min)	
		GEP	GP	GEP	GP
Balance Scale	50	9	34	27.9	109.3
Breast Cancer W.	100	19	77	139.2	643.4
Car	100	13	92	1209.4	14874.2
Glass	100	20	82	214.4	854.2
Heart Disease	100	13	93	82.4	440.4
Ionosphere	150	10	131	186.9	507.8
Iris	80	30	48	42.7	91.1
Lung Cancer	150	5	137	35.9	48.3
Pima Indian	100	29	97	1849.7	5721.5
Waveform	100	11	93	1660.5	7703.1
Wine	100	15	84	70.0	214.9
Zoo	100	7	45	19.5	32.5

Avg. Exp. Size represents the average expression size, Exec. Time is the average execution time in minutes. The size limit is the fixed length of GEP chromosomes; the same size was used to control the maximum number of nodes allowed to grow in a GP tree.

curved boundaries between the classes, C4.5 relied on hypercubic boundaries and could only generate piecewise-constant approximations. On average, C4.5 generated a decision tree containing about 47 nodes, resulting in only 78.7% test accuracy, while C4.5Rules produced about 25 rules with 77.3% testing accuracy. In this case, the GEP generated rules are more accurate, more compact, and easier to understand.

2) Comparison of Efficiency Between GEP and GP: It is trivial to compare the CPU time between GEP with C4.5 algorithms, since GEP is definitely more time-consuming. But it is necessary to compare GEP with traditional GP approach. Although there is no significant difference on the accuracy rate between GEP and GP on most of the testing databases, the GEP approach is more efficient than GP with respect to the time and solution complexity. Table IV gives a comparison of the average expression size generated by GEP and GP, i.e., the number of nodes in the program tree and their average execution time over five different runs. Both GEP and GP systems were programmed using Java and tested on Pentium III 1.7-MHz machines with 512 M RAM. On average, the size of GP expressions is 7.67 times larger than that of GEP expressions; on the other hand, the GEP system runs faster than GP with an average speedup of 4.07 under the same circumstances. It can be concluded that GEP tends to generate shorter expressions and costs less time to execute compared with GP, while achieving comparable classification performance.

3) Comparison of Rule Set Compactness: Concerning the compactness of evolved rule sets, we compare the average number of rules generated by GEP with that of C4.5Rules and GP for each problem, and the results are shown in Table V. On average, the number of rules generated by C4.5Rules is 2.8 times larger than that of GEP. Thus, we claim that our GEP approach tends to produce more compact rule sets than traditional rule induction algorithm like C4.5Rules, especially

TABLE V
COMPARISON OF RULE SET COMPACTNESS

Databases \ Method	C4.5Rules	GEP	GP
Balance Scale	25	3	3
Breast Cancer	8	4	4
Car	73	14	8
Glass	13	9	4
Heart Disease	9	4	4
Ionosphere	8	4	4
Iris	4	5	5
Lung Cancer	4	3	3
Pima Indian	10	5	5
Waveform	51	7	6
Wine	5	5	5
Zoo	8	7	7

Each entry gives the average number of rules generated for each problem.

on complex problems. For example, GEP generates approximately 14 rules for the *car* data, while C4.5Rules requires about 73 rules to describe the data. The reason lies in the fact that GEP is good at representing a curved boundary by means of mathematical expressions, while traditional algorithms like C4.5 utilize piecewise constant approximation to construct hypercubic boundaries in contrast. We notice that GP generates fewer rules than GEP on three out of the twelve data sets, and equal on others. The reason for this lies in the fact that GP rules are usually more complex that GEP rules, which costs more description bits. Since we use the same MDL criterion for early stopping, it is reasonable that GP tends to generate fewer rules. But with respect to the overall complexity of the whole rule set, GEP still gets better compactness.

C. Discussion

Due to the stochastic behavior of GAs, GEP often results in different expression rules from different runs. For example, in another trial GEP found a more complex expression with the same classification power as (5.5) for class the *Left* on the *balance scale* data

$$\text{IF } A1 * \frac{A2}{\text{IF}(A1 > 0, A4, (A3 - A2)*1)} - A3 > 0 \quad \text{THEN class} = \text{Left}. \quad (5.9)$$

Since the value for attribute $A1$ is in the range of $[1.0..3.0]$, it is always larger then 0, so the expression (5.9) is actually the same as (5.5). This demonstrates the element redundancy existed in the GEP chromosomes and the fact that multiple genomes can be mapped into the same solution. GEPs nondeterminism is beneficial in the context of data mining, since it might lead to unexpected and interesting discoveries among observed data.

VI. CONCLUSION AND FUTURE WORK

In this paper, we have demonstrated the applicability of GEP to extract high-order rules for classification problems. As a new type of GP, GEP exhibits simplicity, high efficiency, and functional complexity because of its linear representation and genome-ET mapping scheme. GEP harnesses the power of evolutionary search to detect underlying but unknown relationships among data and express them as mathematical expressions, which overcomes the shortcomings of local, heuristic, and greedy search used by conventional machine learning algorithms like C4.5. The fitness measures for rule quality consider both the rule consistency gain and completeness. The covering strategy was applied to learn a group of rules for each class and the MDL principle was used to avoid overfitting. Finally, we remove redundant rules and make the final rule set ordered for conflict resolution and compact. Experiments conducted on several UCI machine learning data sets with both numeric and nominal attributes have shown that our GEP approach achieved up to 20% improvement in validation accuracy, compared with the C4.5 algorithms. The reason for this performance improvement is due to the global searching capability of GEP to produce the appropriate curved boundaries between the classes in the data space. Furthermore, the proposed GEP approach is more efficient and tends to generate shorter solutions compared with canonical tree-based GP classifiers.

The bloating problems that appear often in traditional GP are mitigated by the fixed chromosome size in GEP. However, in some cases GEP may generate very long and complex expressions that are difficult for humans to understand. GEP may generate complex expressions that "overfit" the training examples yet perform poorly over the complete input space. To avoid this problem, future research should address the identification and application of the appropriate parsimony pressure into the fitness function to restrict the expression size, increase the computational efficiency of the method, and improve the understandability of the solutions.

Future work should also address how to handle incremental learning. Currently, our system can save the learned GEP equations as "seed chromosomes" and reload them later to relearn concepts for the same data set or a modified data set. Real-world systems typically evolve over time; therefore, it is necessary to develop a computationally efficient GEP method that reuses the knowledge in a set of previously learned rules, to represent both old and new training examples.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers who provided valuable comments for the quality of this paper. They would also like to thank the Motorola Advanced Technology Center (MATC) for providing funding for this project.

REFERENCES

[1] S. Augier, G. Venturini, and Y. Kodratoff, "Learning first order logic rules with a genetic algorithm," in *Proc. 1st Int. Conf. Knowledge Discovery and Data Mining*, U. M. Fayyad and R. Uthurusamy, Eds., 1995, pp. 21–26.

[2] W. Banzhaf *et al.*, "Genotype-phenotype mapping and neutral variation—A case study in genetic programming," in *Parallel Problem Solving from Nature—PPSN III*, Y. Davidor *et al.*, Eds. News York: Springer-Verlag, 1994, vol. 866, Lecture Notes in Computer Science, pp. 322–332.

[3] C. Blake, E. Keogh, and C. J. Merz. (1998) UCI Repository of Machine Learning Databases. Univ. California at Irvine, Dept. Inform. Comput. Sci., CA. [Online]. Available: http://www.ics.uci.edu/~mlearn/ ML-Repository.html

[4] C. C. Bojarczuk, H. S. Lopes, and A. A. Freitas, "Discovering comprehensible classification rules using genetic programming: A case study in a medical domain," in *Proc. Genetic and Evolutionary Computation Conf.*, Orlando, FL, July 14–17, 1999, pp. 953–958.

[5] ——, "Data mining with constrained-syntax genetic programming: Applications to medical data sets," in *Proc. Intelligent Data Analysis in Medicine and Pharmacology (IDAMAP)*, London, U.K., 2001.

[6] M. C. J. Bot and W. B. Langdon, "Application of genetic programming to induction of linear classification trees," in *Proc. 3rd European Conf. Genetic Programming (EuroGP)*, vol. 1802, LNCS , R. Poli, W. Banzhaf, W. B. Langdon, J. F. Miller, P. Nordin, and T. C. Fogarty, Eds., Edinburgh, 2000, pp. 247–258.

[7] P. Brazdil and L. Torgo, "Knowledge acquisition via knowledge integration," in *Current Trends in Knowledge Acquisition*. Amsterdam, The Netherlands: IOS Press, 1990.

[8] L. Breiman, J. H. Friedman, R. A. Olsen, and C. J. Stone, *Classification and Regression Trees*. Belmont, CA: Wadsworth, 1984.

[9] I. Bruha, "Quality of decision rules: Definitions and classification schemes for multiple rules," in *Machine Learning and Statistics: The Interface*, G. Nakhaeizadeh and C. C. Taylor, Eds. New York: Wiley, 1997, pp. 107–131.

[10] P. Clark and T. Niblett, "The CN2 induction algorithm," *Machine Learn.*, vol. 3, no. 4, pp. 261–283, 1989.

[11] A. L. Corcoran and S. Sen, "Using real-valued genetic algorithms to evolve rule sets for classification," *Proc. 1st IEEE Conf. Evolutionary Computation*, pp. 120–124, June 1994.

[12] G. F. Davenport *et al.*, "Rule induction using a reverse polish representation," in *Proc. Genetic and Evolutionary Computation Conf. (GECCO'99)*, vol. 2, W. Banzhaf *et al.*, Eds., 1999, pp. 990–995.

[13] K. A. De Jong, W. M. Spears, and D. F. Gordon, "Using genetic algorithms for concept learning," *Machine Learn.*, vol. 13, pp. 161–188, 1993.

[14] J. Eggermont, A. E. Eiben, and J. I. van Hemert, "A comparison of genetic programming variants for data classification," in Advances in Intelligent Data Analysis, 3rd Int. Symp. (IDA-99), Amsterdam, The Netherlands, 1999.

[15] C. Ferreira, "Gene expression programming: A new adaptive algorithm for solving problems," *Complex Syst.*, vol. 13, no. 2, pp. 87–129, 2001.

[16] ——, "Mutation, transposition, and recombination: An analysis of the evolutionary dynamics," in *Proc. 6th Joint Conf. Information Sciences, 4th Int. Workshop on Frontiers in Evolutionary Algorithms*, H. J. Caulfield, S.-H. Chen, H.-D. Cheng, R. Duro, V. Honavar, E. E. Kerre, M. Lu, M. G. Romay, T. K. Shih, D. Ventura, P. P. Wang, and Y. Yang, Eds., Research Triangle Park, NC, 2002, pp. 614–617.

[17] I. W. Flockhart and N. J. Radcliffe, "A genetic algorithm-based approach to data mining," in *Proc. 2nd Int. Conf. Knowledge Discovery & Data Mining*, 1996, pp. 299–302.

[18] A. A. Freitas, "A survey of evolutionary algorithms for data mining and knowledge discovery," in *Advances in Evolutionary Computation*, A. Ghosh and S.S. Tsutsui, Eds. New York: Springer-Verlag, 2001.

[19] ——, "A genetic programming framework for two data mining tasks: Classification and generalized rule induction," in *Proc. 2nd Annu. Conf., Genetic Programming* , J. R. Koza *et al.*, Eds., 1997, pp. 96–101.

[20] J. Fürnkranz, "Separate-and-conquer rule learning," *Artif. Intell. Rev.*, vol. 13, no. 1, pp. 3–54, Jan. 1999.

[21] ——, "Round robin rule learning," in *Proc. 18th Int. Conf. Machine Learning (ICML-01)*, C. E. Brodley and A. P. Danyluk, Eds., 2001, pp. 146–153.

[22] A. Giordana and F. Neri, "Search-intensive concept induction," *Evol. Comput.*, vol. 3, no. 4, pp. 375–416, 1995.

[23] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, MA: Addison-Wesley, 1989.

[24] D. P. Greene and S. F. Smith, "Competition-based induction of decision models from examples," *Machine Learn.*, vol. 13, pp. 229–257, 1993.

[25] J. Hekanaho, "GA-based rule enhancement in concept learning," in *Proc. 3rd Int. Conf. Knowledge Discovery and Data Mining*, Newport Beach, CA, 1997, pp. 183–186.

[26] F. Herrera, M. Lozano, and J. L. Verdegay, "Generating fuzzy rules from examples using genetic algorithms," in *Fuzzy Logic Soft Computing*, B. Bouchon-Meunier, R. R. Yager, and L. A. Zadeh, Eds., 1995, pp. 11–20.

[27] J. H. Holland and J. S. Reitman, "Cognitive systems based on adaptive algorithms," in *Pattern-Directed Inference Systems*, D. A. Waterman and F. Hayes-Roth, Eds. New York: Academic, 1978.

[28] H. Ishibuchi, K. Nozaki, N. Yamamoto, and H. Tanaka, "Selecting fuzzy if-then rules for classification problems using genetic algorithms," *IEEE Trans. Fuzzy Syst.*, vol. 8, pp. 485–488, Aug. 1995.

[29] C. Z. Janikow, "A knowledge-intensive genetic algorithm for supervised learning," *Machine Learn.*, vol. 13, pp. 189–228, 1993.

[30] J. K. Kishore, L. M. Patnaik, V. Mani, and V. K. Agrawal, "Application of genetic programming for multicategory pattern classification," *IEEE Trans. Evol. Comput.*, vol. 4, pp. 242–258, Sept. 2000.

[31] J. R. Koza, *Genetic Programming*. Cambridge, MA: MIT Press, 1992.

[32] ——, "Concept formation and decision tree induction using the genetic programming paradigm," in *Parallel Problem Solving from Nature*, H. Schwefel and R. Maenner, Eds: Springer-Verlag, 1991.

[33] P. L. Lanzi, W. Stolzmann, and S. W. Wilson, Eds., *Learning Classifier Systems: From Foundations to Applications*. Berlin, Germany: Springer-Verlag, 2000, vol. 1813, Lecture Notes on Artificial Intelligence.

[34] T. Loveard and V. Ciesielski, "Representing classification problems in genetic programming," in *Proc. Congress Evolutionary Computation*, vol. 2, 2001, pp. 1070–1077.

[35] S. Luke, "Code growth is not caused by introns," in *Late Breaking Papers at the 2000 Genetic and Evolutionary Computation Conf. (GECCO-2000)*, 2000, pp. 228–235.

[36] R. E. Marmelstein and G. B. Lamont, "A method for mining simplified decision rule sets," in Int. ICSC Congress Computational Intelligence: Methods and Applications (CIMA'99), Rochester, NY, 1999.

[37] ——, "GraCCE: A genetic environment for data mining," in *Late Breaking Papers at the Genetic Programming 1998 Conf.*, J. R. Koza, Ed., 1998, pp. 22–25.

[38] ——, "Pattern classification using a hybrid genetic program decision tree approach," in *Proc. 3rd Annu. Conf. Genetic Programming*, J. R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, M. H. Garzon, D. E. Goldberg, H. Iba, and R. Riolo, Eds., Madison, WI, July 1998, pp. 223–231.

[39] R. S. Michalski *et al.*, "Learning patterns in noisy data: The AQ approach," in *Machine Learning and Its Applications*, G. Paliouras *et al.*, Eds: Springer-Verlag, 2001, vol. 2049, Lecture Notes in Artificial Intelligence (LNAI), pp. 22–38.

[40] T. Mitchell, *Machine Learning*. New York: McGraw-Hill, 1997.

[41] D. J. Montana, "Strongly typed genetic programming," *Evol. Comput.*, vol. 3, no. 2, pp. 199–230, 1995.

[42] N. Nikolaev and V. Slavov, "Inductive genetic programming with decision trees," *Intell. Data Anal.: An Int. J.*, vol. 2, no. 1, pp. 31–44, 1998.

[43] E. Noda, A. A. Freitas, and H. S. Lopes, "Discovering interesting prediction rules with a genetic algorithm," presented at the 1999 Conf. Evolutionary Computation (CEC-99), Washington, DC, July 1999.

[44] M. R. Paterson and M. Livesey, "Distinguishing genotype and phenotype in genetic programming," in *Late Breaking Papers at the Genetic Programming 1996 Conf.*, J. R. Koza, Ed., 1996, pp. 141–150.

[45] M. Pei, E. D. Goodman, and W. F. Punch, "Pattern discovery from data using genetic algorithms," presented at the 1st Pacific-Asia Conf. Knowledge Discovery and Data Mining (PAKDD-97), Singapore, Feb. 1997.

[46] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann, 1993.

[47] J. Rissanen, "Modeling by shortest data description," *Automatica*, vol. 14, pp. 465–471, 1978.

[48] J. P. Rosca *et al.*, "Generality versus size in genetic programming," in *Proc. 1st Annu. Conf. Genetic Programming 1996*, J. R. Koza *et al.*, Eds., Cambridge, MA, 1996, pp. 381–387.

[49] C. Ryan, J. J. Collins, and M. O'Neill, "Grammatical evolution: Evolving programs for an arbitrary language," in *1st European Workshop on Genetic Programming 1998*, 1998, vol. 1391, Lecture Notes in Computer Science, pp. 83–95.

[50] S. F. Smith, "Flexible learning of problem solving heuristics through adaptive search," in *Proc. 8th Int. Joint Conf. Artificial Intelligence*, 1983, pp. 422–425.

[51] W. A. Tackett, "Genetic programming for feature discovery and image discrimination," in *Proc. 5th Int. Conf. Genetic Algorithms*, 1993, pp. 303–309.

[52] K. C. Tan, A. Tay, T. H. Lee, and C. M. Heng, "Mining multiple comprehensible classification rules using genetic programming," in *Proc. Congr. Evolutionary Computation (CEC'02)*, vol. 2, 2002, pp. 1302–1307.

[53] S. B. Thrun, T. Mitchell, and J. Cheng, "The Monk's problems—A performance comparison of different learning algorithms," Carnegie Mellon Univ., Comput. Sci. Dept., CS-CMU-91–197, 1991.

[54] G. Venturini, "A supervised inductive algorithm with genetic search for learning attributes based concepts," in *Proc. European Conf. Machine Learning*, 1993, pp. 280–296.

[55] M. L. Wong and K. S. Leung, *Data Mining Using Grammar-Based Genetic Programming and Applications* Norwell, MA, 2000.
[56] B.-T. Zhang and H. Muhlenbein, "Balancing accuracy and parsimony in genetic programming," *Evol. Comput.*, vol. 3, no. 1, pp. 17–38, 1995.
[57] C. Zhou, P. C. Nelson, W. Xiao, and T. M. Tirpak, "Discovery of classification rules by using gene expression programming," presented at the Int. Conf. Artificial Intelligence (IC-AI'02), Las Vegas, NV, June 24–27, 2002.

Chi Zhou received the B.S. and M.S. degrees in computer science from Nanjing University, Nanjing, China, in 1996 and 1999, respectively, and the Ph.D. degree in computer science from the University of Illinois at Chicago, in 2003.

He is currently a Senior Software Engineer with the Motorola Advanced Technology Center, Schaumburg, IL. His applied research has focused on developing advanced knowledge discovery and management techniques for manufacturing. His interests include machine learning, neural networks, evolutionary algorithms, and data mining.

Weimin Xiao received the B.S. degree in structural engineering from Zhejiang University, Zhejiang, China, the M.S. degree in computational structural mechanics from Chongqing University, Chongqing, China, and Ph.D. degree in computational structural mechanics from University of Kentucky, Lexington.

He was a Lecturer at Zhejiang Industrial University, Hangzhou, China. Currently, he is a Principal Software Engineer with the Motorola Advanced Technology Center, Schaumburg, IL. His research interests include distributed computing, automated FEA system, mixed integer linear and nonlinear optimization, machine learning, automated mathematical model discovery, and knowledge discovery from database.

Thomas M. Tirpak (M'91) received the B.S. and M.S. degrees in general engineering (robotics), the Ph.D. degree in electrical and computer engineering from the University of Illinois at Urbana-Champaign, and the Master of Engineering Management degree from Northwestern University, Evanston, IL.

He is a Principal Staff Engineer with the Motorola Advanced Technology Center, Schaumburg, IL, where he has led efforts to develop new methods for improving the cycle time, quality, and cost of electronics manufacturing and product design operations. In cooperation with Motorola University, he developed and taught classes on "SMT manufacturing optimization" and "factory physics." He has mentored research programs with universities in the U.S., Europe, Asia, and South America, and was a Visiting Lecturer at the University of Mining and Metallurgy, Krakow, Poland. His current research interests include process modeling and optimization, multidisciplinary design optimization, and decision support systems.

Dr. Tirpak is a Motorola Science Advisory Board Associate and a Member of the Institute for Operations Research and Management Science, and Tau Beta Pi.

Peter C. Nelson received the B.A. degree in computer science and mathematics from North Park College, Chicago, IL, in 1984, and the M.S. and Ph.D. degrees in computer science from Northwestern University, Evanston, IL, in 1986 and 1988, respectively.

Currently, he is a Professor and Head of the Department of Computer Science, University of Illinois at Chicago. His interests include developing useful AI techniques for intelligent transportation systems, manufacturing optimization, molecular biology, and intelligent tools for managing high availability, high-performance computer clusters. He has published the results of these projects by coauthoring over 50 technical articles with his collaborators. His research has been funded by the U.S. Department of Transportation, Illinois Department of Transportation, National Institutes of Health, National Research Council, National Science Foundation, Illinois State Toll Highway Authority, National Institute of Statistical Sciences, Manufacturing Research Center, Motorola, and Sun Microsystems.