A Novel Evolutionary Data Mining Algorithm With Applications to Churn Prediction

Wai-Ho Au, Keith C. C. Chan, and Xin Yao, *Fellow, IEEE*

***Abstract*—Classification is an important topic in data mining research. Given a set of data records, each of which belongs to one of a number of predefined classes, the classification problem is concerned with the discovery of classification rules that can allow records with unknown class membership to be correctly classified. Many algorithms have been developed to mine large data sets for classification models and they have been shown to be very effective. However, when it comes to determining the likelihood of each classification made, many of them are not designed with such purpose in mind. For this, they are not readily applicable to such problem as churn prediction. For such an application, the goal is not only to predict whether or not a subscriber would switch from one carrier to another, it is also important that the likelihood of the subscriber's doing so be predicted. The reason for this is that a carrier can then choose to provide special personalized offer and services to those subscribers who are predicted with higher likelihood to churn. Given its importance, we propose a new data mining algorithm, called data mining by evolutionary learning (DMEL), to handle classification problems of which the accuracy of each predictions made has to be estimated. In performing its tasks, DMEL searches through the possible rule space using an evolutionary approach that has the following characteristics: 1) the evolutionary process begins with the generation of an initial set of first-order rules (i.e., rules with one conjunct/condition) using a probabilistic induction technique and based on these rules, rules of higher order (two or more conjuncts) are obtained iteratively; 2) when identifying interesting rules, an objective interestingness measure is used; 3) the fitness of a chromosome is defined in terms of the probability that the attribute values of a record can be correctly determined using the rules it encodes; and 4) the likelihood of predictions (or classifications) made are estimated so that subscribers can be ranked according to their likelihood to churn. Experiments with different data sets showed that DMEL is able to effectively discover interesting classification rules. In particular, it is able to predict churn accurately under different churn rates when applied to real telecom subscriber data.**

***Index Terms*—Churn prediction, customer retention, data mining, evolutionary computation, genetic algorithms.**

I. Introduction

CLASSIFICATION is an important topic in data mining research. Given a set of data records, each of which belongs to one of a number of predefined classes, the classification problem is concerned with the discovery of classification rules that can allow records with unknown class membership to be correctly classified. Many algorithms have been developed to mine large data sets for classification models and they have been shown to be very effective [3], [16], [17], [30], [36]–[38]. However, when it comes to determining the likelihood of each classification made, many of them are not designed with such purpose in mind.

Existing data mining algorithms such as decision tree based algorithms (e.g., BOAT [16], C4.5 [36], PUBLIC [37], RainForest [17], SLIQ [30], SPRINT [38]) can be used to uncover classification rules for classifying records with unknown class membership. Nevertheless, when decision tree based algorithms are extended to determine the probabilities associated with such classifications (see, e.g., [34]), it is possible that some leaves in a decision tree have similar class probabilities.

Unlike decision tree based algorithms, other classification techniques such as logit regression and neural networks [3] can determine a probability for a prediction with its likelihood. However, comparing with decision tree based algorithms, these algorithms do not explicitly express the uncovered patterns in a symbolic, easily understandable form (e.g., if-then rules).

Owing to the limitations of these existing techniques, we propose a new algorithm, called data mining by evolutionary learning (DMEL), to mine classification rules in databases. The DMEL algorithm has the following characteristics. First, instead of random generation, the initial population, which consists of a set of first-order rules,[1] is generated nonrandomly using a probabilistic induction technique. Based on these rules, rules of higher orders are then obtained iteratively with the initial population at the start of each iteration obtained based on the lower order rules discovered in the previous iteration. Second, when identifying interesting rules, DMEL uses an objective interestingness measure that does not require subjective input from the users. Third, in evaluating the fitness of a chromosome, DMEL uses a function, which is defined in terms of the probability that the attribute values of a tuple can be correctly determined using the rules it encodes. Fourth, the likelihood of predictions (or classifications) made is estimated. Fifth, DMEL is able to handle missing values in an effective manner.

Using the discovered rules, DMEL can be used to classifying records with unknown class membership. In particular, DMEL is able to predict churn, which is concerned with the loss of sub-

Manuscript received September 1, 2002; revised July 23, 2003. This work was supported in part by The Hong Kong Polytechnic University under Grant A-P209 and Grant G-V958.

W.-H. Au and K. C. C. Chan are with the Department of Computing, The Hong Kong Polytechnic University, Kowloon, Hong Kong (e-mail: cswhau@comp.polyu.edu.hk; cskcchan@comp.polyu.edu.hk).

X. Yao is with the School of Computer Science, The University of Birmingham, Birmingham B15 2TT, U.K. (e-mail: x.yao@cs.bham.ac.uk).

Digital Object Identifier 10.1109/TEVC.2003.819264

[1]In this paper, the order of the rule is related to the number of conditions in the antecedent of a rule. A one-condition rule is, therefore, a first-order rule. If a rule's antecedent contains two conditions, it is a second-order rule. If there are three conditions in the antecedent of a rule, it is a third-order rule, and so on.

scribers who switch from one carrier to another. Since competition in the telecommunications industry is very fierce, many carriers consider reducing churn as an important business venture to maintain profitability. Churn costs carriers a large amount of money annually in North America and Europe [28]. A small reduction in annual churn rate can result in a substantial increase in the valuation and the shareholder value of a carrier [28]. Consequently, analyzing and controlling churn is critical for carriers to improve their revenues.

To reduce churn rate, a carrier gives us a database of 100 000 subscribers. Among these subscribers, some of them had already switched to another carrier. The task assigned to us is to mine the database to uncover patterns that relate the demographics and behaviors of subscribers with churning so that further loss of subscribers can be prevented as much as possible. Efforts are then made to retain subscribers that are identified to have a high probability of switching to other carriers.

Since the customer services centers of the carrier only have a fixed number of staff available to contact a small fraction of all subscribers, it is important for it to distinguish subscribers with high probability of churning from those with low probability so that, given the limited resources, the high probability churners can be contacted first.

For such an application, the goal is not only to predict whether or not a subscriber would switch from one carrier to another, it is also important that the likelihood of the subscriber's doing so be predicted. Otherwise, it can be difficult for the carrier to take advantage of the discovery because the carrier does not have enough resources to contact all or a large fraction of the subscribers. Although logit regression and neural networks can determine a probability for a prediction with its likelihood, they do not explicitly express the uncovered patterns in a symbolic, easily understandable form. It is for this reason that the carrier did not consider these approaches as the best for their task concerned as they could not verify and interpret the uncovered churning patterns.

Unlike existing techniques, DMEL is able to mine rules representing the churning patterns and to predict whether a subscriber is likely to churn in the near future. Experimental results show that it is able to discover the regularities hidden in the database and to predict the probability that a subscriber churns under different churn rates. In addition, since some attributes in the subscriber database contains significant amount of missing values, the ability of DMEL to handle missing values effectively is important to the success of DMEL in churn prediction.

In the following section, we review related work in data mining and evolutionary computation literature for building predictive models. In particular, we explain how they can be used in churn prediction. In Section III, we provide the details of DMEL and explain how it can be used to discover interesting rules hidden in databases. To evaluate the performance of DMEL, we applied it to several real-world databases. The experimental results are given in Section IV. The details of the subscriber database provided by a carrier in Malaysia and the experimental results using this database to test if DMEL is effective for churn prediction are then given in Section V. Finally, in Section VI, we give a summary of the paper.

II. Related Work

Among the different approaches for building predictive models in data mining, decision-tree based algorithms are the most popular (e.g., [16], [17], [30], [36]–[38]). These algorithms usually consist of two phases: a tree-building and a tree-pruning phase (e.g., BOAT [16], C4.5 [36], RainForest [17], SLIQ [30], SPRINT [38]).

Assume that the records in a database are characterized by n attributes, $A_1, \ldots, A_i, \ldots, A_n$, and that A_i is the attribute whose values are to be predicted. In the tree-building phase, a decision tree is constructed by recursively partitioning the training set according to the values of $A_j, j = 1, \ldots, n, j \neq i$. This partitioning process continues until all, or the majority, of the records in each partition have the same attribute values, $a_{ik}, k = 1, \ldots, m_i$, where $a_{ik} \in \mathrm{dom}(A_i)$ and $|dom(A_i)| = m_i$. Since the resulting decision tree may contain branches that are created due to noises in the data, some of the branches may need to be removed. The tree-pruning phase, therefore, consists of selecting and removing the subtrees that have the largest estimated error rate. Tree pruning has been shown to increase the prediction accuracy of a decision tree on one hand and reduce the complexity of the tree on the other. Of the many decision tree based algorithms that have been used in data mining, C4.5 is by far the most popular [36].

Other than the use of decision tree based algorithms, techniques based on genetic algorithms (GAs) have also been proposed for predictive modeling. There are currently two different GA-based approaches for rule discovery: the Michigan approach and the Pittsburgh approach. The Michigan approach, exemplified by Holland's classifier system [21], represents a rule set by the entire population whereas the Pittsburgh approach, exemplified by Smith's LS-1 system [39], represents a rule set by an individual chromosome. Although the Michigan approach is able to deal with multiclass problems, one of the major difficulties in using it is the problem in credit assignment, which gives the activated classifiers a reward if the classification they produced is correct and gives them a punishment, otherwise. Specifically, it is extremely hard to come up with a good credit assignment scheme that works.

The algorithms based on the Pittsburgh approach (e.g., [10], [23], [39]) represent an entire rule set as a chromosome, maintain a population of candidate rule sets, and use selection and genetic operators to produce new generation of chromosomes and, hence, new rule sets. Each chromosome competes with one another in terms of classification accuracy on the application domain. Individuals are selected for reproduction using *roulette wheel selection* and a whole new population is generated based on *crossover* and *mutation*. The selected chromosomes produce offspring using an extended version of the standard *two-point crossover* operator such that the crossover points can occur either both on rule boundaries or within rules [10], [39]. That is, if one parent is being cut on a rule boundary, then the other parent must be cut on a rule boundary as well; similarly, if one parent is being cut at a point, say, 5 bits to the right of a rule boundary, then the other parent must be cut in a similar spot [10], [39]. The mutation operator is identical to the classical one, which performs bit-level mutations. The fitness of each individual rule set

is computed by testing the rule set on the current set of training examples [10], [39].

The Pittsburgh approach was originally designed for single-class learning problems and, hence, only the antecedent of a rule was encoded into an allele of a chromosome [10], [23], [39]. An instance that matches one or more rules is classified as a positive example of the concept (class) and an instance that fails to match any rule is classified as a negative example [10], [23], [39]. To tackle multiclass problems, they could be extended by introducing multiple populations so that a specific population is dedicated to learn each concept. It is possible that an instance is matched by more than one rule of different concepts on one hand and it is also possible that an instance is matched by none of any rule of any concept on the other. Unfortunately, this problem has not been addressed in many of the systems based on the Pittsburgh approach (e.g., [10], [23], [39]).

Recently, the use of GAs for rule discovery in the application of data mining has been studied in [9], [12], [15], [26]. These algorithms are based on the Michigan approach in a way that each rule is encoded in a chromosome and the rule set is represented by the entire population. Unlike classifier systems (e.g., [19], [21], [29]), they 1) have modified the individual encoding method to use nonbinary representation; 2) do not encode the consequents of rules into the individuals; 3) use extended version of crossover and mutation operators suitable to their representations; 4) do not allow rules to be invoked as a result of the invocation of other rules; and 5) define fitness functions in terms of some measures of classification performance (e.g., *cover* [9], *sensitivity* and *specificity* [12], etc.).

It is important to note that these algorithms [9], [12], [15] were developed to discover rules for a single class only. When they are used to deal with multiclass problems, the GAs are run once for each class. Specifically, they would search rules predicting the first class in the first run; they would search rules predicting the second class in the second run and so on. Similar to the Pittsburgh approach, it is possible that an instance is matched by more than one rule predicting different classes on one hand and it is also possible that an instance is matched by none of any rule predicting any class on the other. This problem has not been addressed by these algorithms.

Although GA-based rule discovery approaches can produce accurate predictive models, they cannot determine the likelihood associated with their predictions. This prevents these techniques from being applicable to the task of predicting churn, which requires the ranking of subscribers according to their likelihood to churn.

A related work on churn prediction in a database of 46 744 subscribers has been presented in [32]. The performances of logit regression, C5.0 (a commercial software product based on C4.5), and nonlinear neural networks with a single hidden layer and weight decay [3] are evaluated empirically. The experimental results in [32] showed that neural networks outperformed logit regression and C5.0 for churn prediction.

An empirical comparison of DMEL with neural networks and C4.5 on the subscriber database provided by the carrier in Malaysia will be given in Section V.

$R_1 \leftarrow$ {1st-order rules obtained by probabilistic induction};
$l \leftarrow 2$;
while $R_{l-1} \neq \emptyset$ **do**
begin
  $t \leftarrow 0$;
  $population[t] \leftarrow initialize(R_{l-1})$;
  $fitness(population[t])$;
  while not $terminate(population[t])$ **do**
  begin
    $t \leftarrow t + 1$;
    $population[t] \leftarrow reproduce(population[t-1])$;
    $fitness(population[t])$;
  end
  $R_l \leftarrow decode$(the fittest individual in $population[t]$);
  $l \leftarrow l + 1$;
end
$Rules \leftarrow \bigcup_l R_l$;

Fig. 1. DMEL algorithm.

III. DMEL for Data Mining

To perform searches more effectively in a huge rule set space, we propose to use an evolutionary algorithm called DMEL. Using an evolutionary learning approach, DMEL is capable of mining rules in large databases without any need for user-defined thresholds or mapping of quantitative into binary attributes. However, DMEL requires quantitative attributes to be transformed to categorical attributes through the use of a discretization algorithm, as will be seen later.

In this paper, the order of the rule is related to the number of conditions in the antecedent of a rule. A one-condition rule is, therefore, a first-order rule. If a rule's antecedent contains two conditions, it is a second-order rule. If there are three conditions in the antecedent of a rule, it is a third-order rule, and so on. DMEL discovers rules by an iterative process. It begins with the generation of a set of first-order rules using a probabilistic induction technique. Based on these rules, it then discovers a set of second-order rules in the next iteration and based on the second-order rules, it discovers third-order rules, etc. In other words, if we refer to the initial set of first-order rules as R_1, the rules in R_1 are then used to generate a set of second-order rules, R_2. R_2 is then used to generate a set of third-order rules R_3, and so on for 4th and higher order rules. In general, at the $(l-1)$th iteration, DMEL begins an evolutionary learning process by generating an initial population of individuals (each represents a set of lth order rules) by randomly combining the rules in R_{l-1} to form a set of rules of order l. Once started, the iterative learning process goes on uninterrupted until no more interesting rules in the current population can be identified. The DMEL algorithm is given in Fig. 1.

The *decode* function in Fig. 1 is to extract all the interesting rules encoded in a chromosome and store them in R_l. If an allele in the chromosome is found interesting based on the objective measure defined in Section III-B, the *decode* function will extract the rules it encodes. The rule set returned by the *decode* function, therefore, contains interesting rules only. When none of the rules encoded in the individual is found interesting, the *decode* function will return a null set and, hence, R_l will become a null set.

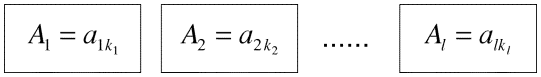


Fig. 2. An allele representing an lth order rule.

A. Encoding Rules in the Chromosomes

For the evolutionary process, DMEL encodes a complete set of rules in a single chromosome in such a way that each gene encodes a single rule. Specifically, given the following lth order rule, for example:

$$A_1 = a_{1k_1} \wedge A_2 = a_{2k_2} \wedge \cdots \wedge A_l = a_{lk_l} \Rightarrow A_p = a_{pq}[\hat{w}]$$

where $\hat{w}$, given by (14) later, is an uncertainty measure associated with it, this rule is encoded in DMEL by the allele given in Fig. 2.

It should be noted that the consequent and the uncertainty measure are not encoded. This is because the consequent is not, and in fact, should not be determined by chance. In DMEL, both the consequent and the uncertainty measure are determined when the fitness of a chromosome is computed. Given this representation scheme, the number of genes in the chromosome is, therefore, the same as the number of rules in the rule set.

B. Generating First-Order Rules

DMEL begins the evolutionary process by the generation of a set of first-order rules. When compared with randomly generated initial population, it has been shown that heuristically-generated initial populations can improve convergence speed and find better solutions [20], [22], [24], [41]. Based on these findings, DMEL first discovers a set of first-order rules and places it in the initial population. Furthermore, the initial first-order rules are generated very rapidly. The time it takes to generate the initial population that contains the first-order rules is negligible, when compared with the time it takes for the best set of rules to be evolved.

The generation of first-order rules can be accomplished by using the interestingness measure given by (5) and the weight of evidence measure given by (14) later. To do so, a probabilistic induction technique called APACS [6], [7] is used. Among all possible attribute value pairs, APACS is able to identify those that have some kind of association relationship even if a database is noisy and contains many missing values. The details of APACS are given as follows.

Let $\mathcal{A} = \{A_1, \ldots, A_n\}$ be a set of n attributes that characterize the tuples in a database and let $dom(A_i) = \{a_{i1}, \ldots, a_{im_i}\}$ denote the domain of an attribute $A_i \in \mathcal{A}$. In the case the domain is continuous, its values is mapped into m_i different categories by a discretization technique proposed in [8]. This technique is used since it has been shown to be able to minimize information lost as a result of the transformation.

Let $\mathrm{count}_{a_{ip}a_{jq}}$ be the number of tuples having both attribute values $A_i = a_{ip}$ and $A_j = a_{jq}$, where $a_{ip} \in \mathrm{dom}(A_i)$, $a_{jq} \in \mathrm{dom}(A_j)$, and $i \neq j$. If we assume that a tuple has a_{ip} is independent of whether it has a_{jq}, the number of tuples that are expected to have both a_{ip} and a_{jq} is given by

$$e_{a_{ip}a_{jq}} = \frac{\sum_{p=1}^{m_i} \mathrm{count}_{a_{ip}a_{jq}} \sum_{q=1}^{m_j} \mathrm{count}_{a_{ip}a_{jq}}}{M} \tag{1}$$

where $M = \sum_{p=1}^{m_i} \sum_{q=1}^{m_j} \mathrm{count}_{a_{ip}a_{jq}}$. The independency of A_i and A_j can be evaluated objectively by the chi-square test as follows. If the statistic

$$X^2 = \sum_{p=1}^{m_i} \sum_{q=1}^{m_j} \frac{(\mathrm{count}_{a_{ip}a_{jq}} - e_{a_{ip}a_{jq}})^2}{e_{a_{ip}a_{jq}}} \tag{2}$$

is greater than the critical chi-square $\chi^2_{d,\alpha}$, where $d = (m_i - 1)(m_j - 1)$ is the degree of freedom and α (usually taken to be 0.05 or 0.01) is the significance level, then we can conclude, with a confidence level of $1 - \alpha$, that A_i is dependent on A_j. It is important to note that the chi-square test only tells us whether an attribute A_j is helpful in determining another attribute A_i. However, it does not provide us with much information about whether a tuple having $a_{jq} \in \mathrm{dom}(A_j)$ would have $a_{ip} \in \mathrm{dom}(A_i)$ at the same time.

Instead of using the chi-square test, we propose to use the *residual analysis* [6], [7] to determine whether a_{ip} is dependent on a_{jq}. We consider the association between a_{ip} and a_{jq} interesting if the probability of finding a_{ip} in a tuple given that a_{jq} is in the same tuple is significantly different from the probability of finding a_{ip} in the tuple alone. In other words, therefore, there exists an interesting association between a_{ip} and a_{jq} if

$$\Pr(A_i = a_{ip} | A_j = a_{jq}) = \frac{\text{no. of tuples with } A_i = a_{ip} \text{ and } A_j = a_{jq}}{\text{no. of tuples with } A_j = a_{jq}} \tag{3}$$

is *significantly different* from

$$\Pr(A_i = a_{ip}) = \frac{\text{no. of tuples with } A_i = a_{ip}}{M}. \tag{4}$$

To decide if the difference is significant, the *adjusted residual* [6], [7] is used

$$d_{a_{ip}a_{jq}} = \frac{z_{a_{ip}a_{jq}}}{\sqrt{\gamma_{a_{ip}a_{jq}}}} \tag{5}$$

where $z_{a_{ip}a_{jq}}$ is the *standardized residual* and is defined as [6], [7]

$$z_{a_{ip}a_{jq}} = \frac{\mathrm{count}_{a_{ip}a_{jq}} - e_{a_{ip}a_{jq}}}{\sqrt{e_{a_{ip}a_{jq}}}} \tag{6}$$

and $\gamma_{a_{ip}a_{jq}}$ is the *maximum-likelihood estimate* [6], [7] of the variance of $z_{a_{ip}a_{jq}}$ and is given by

$$\gamma_{a_{ip}a_{jq}} = \left(1 - \frac{\sum_{p=1}^{m_i} \mathrm{count}_{a_{ip}a_{jq}}}{M}\right)\left(1 - \frac{\sum_{q=1}^{m_j} \mathrm{count}_{a_{ip}a_{jq}}}{M}\right). \tag{7}$$

The measure defined by (5) can be considered an objective interestingness measure as it does not depend on a user's subjective input. Since $d_{a_{ip}a_{jq}}$ has a standard normal distribution [1],

if $|d_{a_{ip}a_{jq}}| > 1.96$ (i.e., the 95th percentiles of the normal distribution), we can conclude that the difference between $\Pr(A_i = a_{ip}|A_j = a_{jq})$ and $\Pr(A_i = a_{ip})$ is significant and that the association between a_{ip} and a_{jq} is interesting.

In addition, if $d_{a_{ip}a_{jq}} > +1.96$, then a_{jq} implies a_{ip}. In other words, whenever a_{jq} is found in a tuple, the probability that a_{ip} is also found in the same tuple is expected to be significantly higher than when a_{jq} is not found. In such a case, we say that a_{jq} is *positively* associated with a_{ip}. Conversely, if $d_{a_{ip}a_{jq}} < -1.96$, whenever a_{jq} is found, the probability that a_{ip} is also found in the same tuple is expected to be significantly lower than when a_{jq} is not found. In such a case, we say that a_{jq} is *negatively* associated with a_{ip}.

Given that a_{ip} and a_{jq} are positively or negatively associated, a measure of the strength of the association can be defined. In [6], [7], such a measure is proposed and it is called the *weight of evidence* measure. This measure is defined in terms of an information theoretic concept known as mutual information. Mutual information measures the change of uncertainty about the presence of a_{ip} in a tuple given that it has a_{jq}. It is defined as follows:

$$I(A_i = a_{ip} : A_j = a_{jq}) = \log \frac{\Pr(A_i = a_{ip}|A_j = a_{jq})}{\Pr(A_i = a_{ip})}. \quad (8)$$

Based on the mutual information measure, the weight of evidence measure is defined as [6], [7]

$$\begin{aligned} w_{a_{ip}a_{jq}} &= I(A_i = a_{ip} : A_j = a_{jq}) - I(A_i \neq a_{ip} : A_j = a_{jq}) \\ &= \log \frac{\Pr(A_j = a_{jq}|A_i = a_{ip})}{\Pr(A_j = a_{jq}|A_i \neq a_{ip})}. \end{aligned} \quad (9)$$

$w_{a_{ip}a_{jq}}$ can be interpreted intuitively as a measure of the difference in the gain in information when a tuple that is characterized by the presence of a_{jq} is also characterized by a_{ip} as opposed to being characterized by other values. $w_{a_{ip}a_{jq}}$ is positive if a_{jq} is positively associated with a_{ip}, whereas it is negative if a_{jq} is negatively associated with a_{ip}.

When the number of tuples characterized by both a_{jq} and a_{ip} is sufficiently large, we can simply use the sample posterior probability of a_{ip} given a_{jq}, $\Pr(A_i = a_{ip}|A_j = a_{jq})$, as the population posterior probability. However, under skewed class distributions, the number of tuples having a_{jq} and a_{ip} can be very small and this can prohibit the use of the sample posterior probability as the population posterior probability. To obtain the population posterior probability, we propose to use an empirical Bayes method, which takes both the sample posterior and the sample prior probabilities into consideration [4]. The empirical Bayes estimation of the posterior probability of a_{ip} given a_{jq} is defined as

$$\begin{aligned} \hat{Pr}(A_i = a_{ip}|A_j = a_{jq}) = B \times \Pr(A_i = a_{ip}|A_j = a_{jq}) \\ +(1 - B) \times \Pr(A_i = a_{ip}) \end{aligned} \quad (10)$$

where B is the *shrinkage factor*, which weighs the importance of the posterior and the prior probabilities. Assuming the probability distribution of the records having a_{jq} and a_{ip} and the probability distribution of the estimated posterior probability are both Gaussian, the shrinkage factor is defined as

$$B = \frac{\sigma^2}{\sigma^2 + \sigma^2_{A_i}} \quad (11)$$

where σ^2 and $\sigma^2_{A_i}$ are the variance of the entire database and that of attribute A_i, respectively.

In order to calculate σ^2 and $\sigma^2_{A_i}$ in (11), Gini's definition of variance for categorical data [27] is used. The variance of attribute A_i, $\sigma^2_{A_i}$, is given by

$$\sigma^2_{A_i} = \frac{1}{2}\left(1 - \sum_{p=1}^{m_i} \Pr(A_i = a_{ip})\right) \quad (12)$$

and the variance of the entire database σ^2 is calculated by

$$\sigma^2 = \frac{1}{2}\left(1 - \sum_{i=1}^{n}\sum_{p=1}^{m_i} \Pr(A_i = a_{ip})\right). \quad (13)$$

The weight of evidence defined in (9) can then be modified as

$$\hat{w}_{a_{ip}a_{jq}} = \log \frac{\hat{\Pr}(A_j = a_{jq}|A_i = a_{ip})}{\hat{\Pr}(A_j = a_{jq}|A_i \neq a_{ip})}. \quad (14)$$

Given $\hat{w}_{a_{ip}a_{jq}}$ and given that a_{jq} is associated with a_{ip}, we can form the first-order rule, $A_j = a_{jq} \Rightarrow A_i = a_{ip}[\hat{w}_{a_{ip}a_{jq}}]$.

By the use of the interestingness measure given by (5) and the weight of evidence measure given by (14), a set of interesting first-order rules can be discovered. Once these rules are discovered, DMEL will begin an iterative process of initialization of population, evaluation of fitness of individuals, selection, reproduction, and termination, etc., so as to discover higher order rules.

C. Initialization of Populations

Since a good initial population may improve the speed of the evolutionary process and make it easier for an optimal solution to be found, DMEL does not generate its initial populations completely randomly. Instead, it makes use of a heuristic in which the association between $a_{jq} \wedge a_{ks}$ and a_{ip} is more likely to be interesting if the association between a_{jq} and a_{ip} and the association between a_{ks} and a_{ip} are interesting. Based on this heuristic, DMEL generates different sets of lth order rules by randomly combining the $(l-1)$th order rules discovered in the previous iteration. The details of the initialization process are given in the *initialize* function in Fig. 3.

The *initialize* function takes as argument, R_{l-1}. The $chrom_i.allele_j$ in Fig. 3 denotes the jth allele of the ith chromosome. The $rand_l(\mathcal{R})$ function returns an lth order allele constructed by randomly combining l elements in $\mathcal{R}$. For our experiments, *popsize* was set to 30 and the number of alleles in each chromosome was set to $nalleles = |R_{l-1}|$, where $|R_{l-1}|$ denotes the number of rules in R_{l-1}. We set $nalleles = |R_{l-1}|$ because each allele represents the antecedent of a rule and the chromosome is used to encode R_{l-1}.

```
population initialize(R_{l-1})
begin
    R ← {all conjuncts in the antecedent of all r ∈ R_{l-1}};
    i ← 1;
    while i ≤ popsize do
    begin
        j ← 1;
        while j ≤ nalleles do
        begin
            chrom_i.allele_j ← rand_l(R);
            j ← j + 1;
        end
        i ← i + 1;
    end
    return ⋃_i chrom_i ;
end
```

Fig. 3. The *initialize* function.

```
population reproduce(population[t − 1])
begin
    chrom_1 ← select(population[t − 1]);
    chrom_2 ← select(population[t − 1]);
    nchrom_1, nchrom_2 ← crossover(chrom_1, chrom_2);
    mutation(nchrom_1);
    mutation(nchrom_2);
    population ← steady-state(population[t − 1], nchrom_1, nchrom_2);
    return population;
end
```

Fig. 4. The *reproduce* function.

D. Genetic Operators

The genetic operators used by DMEL are implemented in the *reproduce* function shown in Fig. 4. The $select(population[t-1])$ function uses the *roulette wheel selection* scheme [13], [18], [31] to select two different chromosomes, $chrom_1$ and $chrom_2$, with respect to their fitness values from the current population, i.e., $population[t-1]$. These two chromosomes are then passed as arguments to the *crossover* function. The $crossover(chrom_1, chrom_2)$ function uses the *two-point crossover* operator because it allows the combination of schemata, which is not possible with the classical, one-point crossover [31]. DMEL uses two different strategies in choosing the crossover points, namely, *crossover*-1 and *crossover*-2. The *crossover*-1 operator allows the crossover points to occur between two rules only, whereas the *crossover*-2 operator allows the crossover points to occur within one rule only. An example of the *crossover*-1 operator and that of the *crossover*-2 operator are graphically depicted in Figs. 5 and 6, respectively.

In DMEL, the crossover probability for the *crossover*-1 operator and that for the *crossover*-2 operator are denoted as p_1 and p_2, respectively. For our experimentation, four different setups are used and they are summarized in Table I.

The first three setups, DMEL-1, DMEL-2, and DMEL-3 use constant values of p_1 and p_2, whereas the last setup, DMEL-4, uses adaptive values of p_1 and p_2. In DMEL-4, p_1 is increased by 0.05 and p_2 is decreased by 0.05 whenever the termination criteria specified in Section III-F are satisfied. The evolutionary process ends when p_1 and p_2 reach 0.75 and 0.25, respectively, and the termination criteria are satisfied. The performance of DMEL under different setups will further be discussed in Section V.

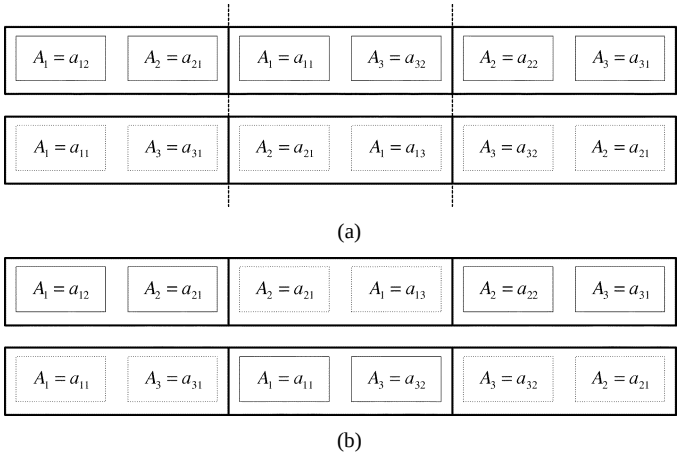


Fig. 5. An example of the *crossover*-1 operator (the thick borders indicate the rule boundaries). (a) Before crossover. (b) After crossover.

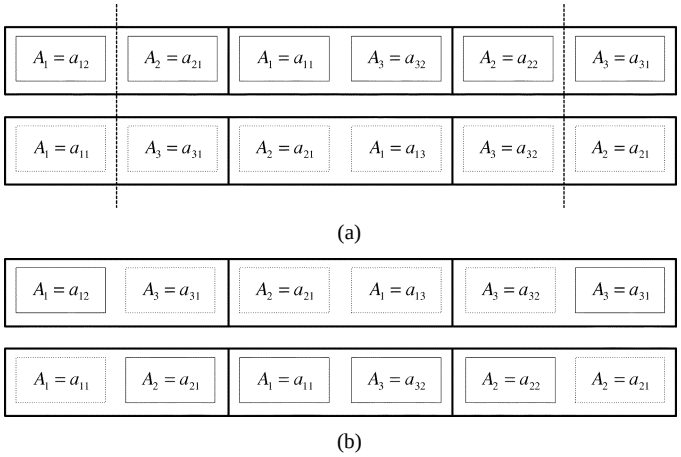


Fig. 6. An example of the *crossover*-2 operator (the thick borders indicate the rule boundaries). (a) Before crossover. (b) After crossover.

TABLE I
DIFFERENT SETUPS OF CROSSOVER PROBABILITIES p_1 AND p_2

	Beginning of evolution		End of evolution	
	p_1	p_2	p_1	p_2
DMEL-1	0.5	0.5	0.5	0.5
DMEL-2	0.75	0.25	0.75	0.25
DMEL-3	0.25	0.75	0.25	0.75
DMEL-4	0.25	0.75	0.75	0.25

The $mutation(nchrom_1)$ function, which is different from the traditional mutation operator [13], [18], [31], takes a chromosome as argument. Its details are given in Fig. 7. The *random* function returns a real number between 0 and 1 and *pmutation* contains the mutation rate and is a constant. The $random(1, l)$ function returns an integer between 1 and l. The $nchrom \cdot allele_j \cdot rule_k$ denotes the kth rule in the jth allele of chromosome *nchrom*. The $hill-climb(\mathcal{R})$ function replaces the kth rule with each element in $\mathcal{R}$ and evaluates the chromosome's fitness value. It returns the one producing the greatest fitness. Instead of replacing a rule with an element in $\mathcal{R}$ randomly, the use of the *hill-climb* function allows DMEL to search for improvements even when premature convergence occurs [13].

```
mutation(nchrom)
begin
    R ← {all conjuncts in the antecedent of all r ∈ R_{l-1}};
    j ← 1;
    while j ≤ nalleles do
    begin
        if random < pmutation then
        begin
            k = random(1, l);
            nchrom.allele_j.rule_k ← hill-climb(R);
        end
        j ← j + 1;
    end
end
```

Fig. 7. The *mutation* function.

The $steady-state(population[t-1], nchrom_1, nchrom_2)$ function in *reproduce* produces a new population, $population[t]$, by removing the two least-fit chromosomes in $population[t-1]$ and replacing them with $nchrom_1$ and $nchrom_2$, while keeping the rest of the other chromosomes intact.

E. Selection and the Fitness Function

To determine the fitness of a chromosome that encodes a set of lth order rules, DMEL uses a performance measure defined in terms of the probability that the value of an attribute of a tuple can be correctly predicted based on the rules in $R = R_1 \cup \cdots \cup R_{l-1} \cup$ (rules encoded in the chromosome being evaluated). The use of this fitness measure is to allow DMEL to maximize the number of records that DMEL can correctly predict. How exactly such fitness value can be determined is given in the following.

An attribute, say, A_i of a tuple o characterized by $A_1 = v_1, \ldots, A_i = v_i, \ldots, A_n = v_n$ is randomly selected and the value v_i deleted from o. The rules contained in R are then used to see if the value of A_i can be correctly predicted based on $v_1, \ldots, v_{i-1}, v_{i+1}, \ldots, v_n$. Assume that a rule which predicts $A_i = a_{ip} \in \mathrm{dom}(A_i)$ is matched, this rule can be considered as providing some evidence for or against A_i to have the value a_{ip} and the strength of the evidence is given by the weight of evidence associated with it. By matching $v_1, \ldots, v_{i-1}, v_{i+1}, \ldots, v_n$ against the rules in R, the value that A_i should take on can be determined based on a total weight of evidence measure [6], [7], which we describe as follows.

Suppose that, of the $n-1$ attribute values that characterize o, some combinations of them, $\mu_1, \ldots, \mu_l, \ldots, \mu_k, \ldots, \mu\eta$, where $\mu_k = \{v_j | j \in s \text{ and } s \subseteq \{1, \ldots, n\} - \{i\}\}, k = 1, \ldots, \eta, \mu_l \cap \mu_k = \emptyset, l \neq k$, match a number of rules that predict A_i to have (or not to have) a value a_{ip}, then the total weight of evidence measure for or against A_i to take on the value a_{ip} is given by [6], [7]

$$w_{a_{ip}} = \sum_{j=1}^{\eta} \hat{w}_{a_{ip}\mu_j}. \tag{15}$$

It is important to note that there may be no rule in R for predicting the value $A_i = a_{ip}$ and, hence, $w_{a_{ip}} = 0$. In this case, we do not have any evidence on hand to determine whether $A_i = a_{ip}$ or not.

Based on (15), A_i can be assigned the value a_{ip} if

$$w_{a_{ip}} \geq w_{a_{ih}}, h = 1, 2, \ldots, m_i' \text{ and } h \neq p \tag{16}$$

where m_i' ($\leq m_i$) denotes the number of different values of A_i that are implied by the matched rules.

If $a_{ip} = v_i$, the prediction is correct and we can update an accuracy count associated with the chromosome whose fitness is being evaluated. This accuracy count can be incremented by one whenever a prediction is accurately made. By putting each of the N tuples in the database to the same test above, we define the fitness measure of each chromosome to be: (value of the accuracy count) $\div N$.

F. Criteria for Termination

The $terminate(population[t])$ function in Fig. 1 implements the following termination criteria: 1) terminate when the best and worst performing chromosome in $population[t]$ differs by less than 0.1% because in this case, the whole population becomes very similar and it is not likely to achieve any improvement in the future generations; 2) terminate when the total number of generations specified by the user is reached; and 3) terminate when no more interesting rules in the current population can be identified because it is unlikely to find any interesting lth order rules if no $(l-1)$th order rule is found interesting.

IV. Experimental Results on Different Datasets

To evaluate the performance of DMEL in different data mining tasks, we applied it to several real-world databases. For each trial in each experiment, each of these databases was divided into two datasets with records in each of them randomly selected. The mining of rules was performed on one of the datasets (i.e., the training dataset). The other dataset was reserved for testing (i.e., the testing dataset). For each of these testing datasets, the values of one of the attributes were deleted. We refer to this attribute as the class attribute in the rest of this section. The rules discovered by mining the training dataset were used to predict the class attribute values in the testing dataset. The predicted values are then compared against the original values to see if they are the same. If it is the case, the accuracy count is incremented correspondingly. Based on this accuracy count, the percentage accuracy for each of DMEL, C4.5 [36], a well-known decision-tree classifier, SCS [18], a Michigan-style classifier system, and GABL [10], a Pittsburgh-style concept learner was computed. The accuracy, averaged over a total of ten trials for each experiment, were recorded and compared and they are given in Table II.

Since GABL was originally developed to solve "single-class (or concept)" problems, multiple populations had to be used in our experiments so that each of them could be dedicated to the learning of relationship between a single value in a multiple-valued attribute and other attribute values in a database. In our experiments, when a test record is matched by none of any rule of any class, we assigned the record to the most common or the majority class in the training dataset; on the other hand, when a test record is matched by more than one rule of different classes,

TABLE II
PERCENTAGE ACCURACY OF THE FOUR DIFFERENT APPROACHES

Database	No. of Records	Class Attribute	Percentage Accuracy (Standard Deviation)			
			DMEL	C4.5	SCS	GABL
zoo	101	*Type*	100.0% (0.0%)	90.9% (8.4%)	27.3% (10.3%)	28.2% (2.9%)
DNA	3,190	*Splice*	95.4% (0.6%)	92.9% (0.8%)	23.2% (15.1%)	53.7% (0.0%)
credit card	690	*Success*	95.6% (4.0%)	82.6% (4.3%)	58.9% (9.2%)	58.9% (0.0%)
diabetes	768	*Test-result*	79.8% (1.7%)	73.8% (2.6%)	61.3% (7.3%)	61.3% (0.0%)
satellite image	6,435	*Soil*	85.5% (0.8%)	85.2% (0.5%)	19.9% (5.6%)	23.1% (0.0%)
social	48,843	*Salary*	85.7% (0.2%)	85.4% (0.3%)	23.6% (17.2%)	75.8% (0.0%)
PBX	3,009	*Calling-party-identification*	99.9% (0.2%)	94.6% (5.2%)	59.6% (26.4%)	94.2% (0.2%)

we assigned the record to the majority class that matched the record.

In our experiments, the crossover rate in DMEL was set to 0.6, the mutation rate was set to 0.0001, and the population size was set to 30. Since the performances of DMEL under different setups (Table I) were more or less the same, we only report the experimental results of DMEL under the setup where both the crossover probability for the *crossover*-1 and that for the *crossover*-2 operator were set to 0.5 (i.e., DMEL-1) in this section. The performance of DMEL for churn prediction under different setups will be discussed in the next section.

For GABL, the mutation probability was set to 0.001, the crossover probability was set to 0.6, and the population size was set to 100 [10]. For SCS, the population size was set to 1,000, the bid coefficient was set to 0.1, the bid spread was set to 0.075, the bidding tax was set to 0.01, the existence tax was set to 0, the generality probability was set to 0.5, the bid specificity base was set to 1, the bid specificity multiplier was set to 0, the bid specificity base was set to 1, the bid specificity multiplier was set to 0, the reinforcement award was set to 1, the proportion to select per generation was set to 0.2, the number to select was set to 1, the mutation probability was set to 0.02, the crossover probability was set to 1, the crowding factor was set to 3, and the crowding subpopulation was set to 3 [18].

All the experiments reported in this section and Section V were performed using a personal computer with Intel Pentium III 1 GHz processor as CPU, 256 MB of main memory, and running Red Hat Linux 7.1. In the following, we describe the databases used in our experiments and present the results analyzing the performance of the different approaches.

A. Zoo Database

Each record in the *zoo* database [14] is characterized by 18 attributes. Since the unique name of each animal is irrelevant, it was ignored. All the 17 remaining attributes are categorical. The class attribute is concerned with the type of the animals are classified into. The value of the class attribute can be one of mammal, bird, reptile, fish, amphibian, insect, and coelenterate.

B. DNA Database

Each record in the *DNA* database [33] consists of a sequence of DNA, an instance name, and the class attribute. Since the unique name of each instance is irrelevant, it was ignored. A sequence of DNA contains 60 fields, each of which can be filled by one of: A, G, T, C, D (i.e., A or G or T), N (i.e., A or G or C or T), S (i.e., C or G), and R (i.e., A or G). The class attribute is concerned with the splice junctions that are points on a DNA sequence at which "superfluous" DNA is removed during the process of protein creation. It indicates the boundaries between extrons (the parts of the DNA sequence retained after splicing) and introns (the parts of the DNA sequence that are spliced out) and can be one of EI (extron-intron boundary), IE (intron-extron boundary), and N (neither extron-intron nor intron-extron boundary).

C. Credit Card Database

The *credit card* database [35] contains data about credit card applications. It consists of 15 attributes of which the class attribute is concerned with whether or not an application is successful. The meaning of these attributes are not known as the names of the attributes and their values were changed by the donor of the database to meaningless symbols to protect confidentiality of the data. Out of the 15 attributes, 6 are quantitative and 9 are categorical. The six quantitative attributes were discretized into four intervals using the discretization technique described in [8].

D. Diabetes Database

Each record in the *diabetes* database [40] is characterized by nine attributes. The value of the class attribute can be either "1" (tested positive for diabetes) or "2" (tested negative for diabetes). The other attributes are quantitative and they were discretized into four intervals using the discretization technique described in [8].

E. Satellite Image Database

Each record in the *satellite image* database corresponds to a 3×3 square neighborhood of pixels completely contained

within an area. Each record contains the pixel values in the four spectral bands of each of the 9 pixels in the 3×3 neighborhood and the class attribute is the class of the central pixel that was one of: red soil, cotton crop, grey soil, damp grey soil, soil with vegetation stubble, and very damp grey soil. All the 36 ($= 4$ spectral bands $\times$ 9 pixels in neighborhood) attributes other than the class attribute is quantitative and in the range between 0 and 255. For our experiments, these quantitative attributes were discretized into four intervals using the discretization technique described in [8].

F. Social Database

The *social* database [25] contains data collected by the U.S. Census Bureau. The records in the database are characterized by 15 attributes. Of these attributes, six of them are quantitative. These quantitative attributes were discretized into four intervals using the discretization technique described in [8]. The remaining nine attributes are all categorical. The class attribute is concerned with whether the annual salary of a person exceeds \$50 K or not.

G. PBX Database

A private branch exchange (PBX) system is a multiple-line business telephone system that resides on a company's premises. One of the significant features of a PBX system is its ability to record call activity such as keeping records of all calls and callers. In one of our experiments, we used the data from the database of a PBX system used in a telecommunication company in Indonesia. The *PBX* database contains data about the usage of the PBX system in the company. Each record in the *PBX* database is characterized by 13 attributes. Except for two attributes that are categorical, all the remaining attributes are quantitative. The quantitative attributes were discretized into four intervals using the technique described in [8]. There are many missing values in this database. In particular, 98.4% of records have missing values in one or more attributes. The class attribute is concerned with the identification of the calling party.

H. Summary

In summary, DMEL performed better than the other three approaches in all the seven databases. It achieved an average accuracy of 91.7% and correctly classified 5.2%, 52.6%, and 35.3% more test records than C4.5, SCS, and GABL, respectively.

V. Experimental Results on the Subscriber Database

A carrier in Malaysia has provided us a database of 100 000 subscribers. The subscriber database was extracted randomly from the time interval of August through October 1999. The task was to discover interesting relationships concerning with the demographics and the behaviors of the subscribers who had churned in the period between August and September 1999. By representing these relationships in the form of rules, they would then be used to predict whether a subscriber would churn in October 1999. According to the definition of the carrier, a subscriber churns when all services held by him/her are closed.

The subscriber database provided by the carrier is stored in an Oracle database. It contains three relations which are listed in Table III. It is important to note that some attributes in some relations contain significant amount of missing values, for example, 62.4% of values in attribute LOCATION in relation DEMOGRAPHICS are missing. The handling of missing values is an important problem to be tackled for mining interesting rules in this database.

TABLE III
Relations in the Subscriber Database

Relation	Description
CDR	Call detail records (each tuple, which is characterized by date, time, duration, location, etc., represents a phone call).
BILLING	Billing records (each tuple, which is characterized by fee, additional charges for roaming and other value-added services, etc., represents a monthly bill).
DEMOGRAPHICS	Demographic records (each tuple, which is characterized by service plan, handset type, etc., represents an application for services made by a subscriber).

TABLE IV
Some of the Identified Variables in the Transformed Data

Variable	Description
Location	Subscriber location.
Type	Customer type (e.g., government versus corporate).
Payment Method	Payment method (e.g., cash versus credit card).
Plan	Service plan.
Charge	Monthly charge.
Usage	Monthly usage.
Calls	Number of calls made.
Abnormal Calls	Number of abnormally terminated calls.

We, together with a domain expert from the carrier, have identified 251 variables associated with each subscriber that might affect his/her churn. Some of these variables can be extracted directly from the database whereas some of them require *data transformation*, which is one of the key steps in the *knowledge discovery process* [11], on the original data. One of the ways to perform data transformation is the use of *transformation functions* [2], [5]. Instead of discovering rules in the original data, we applied DMEL to the *transformed data*. Table IV lists some of these variables in the transformed data.

To manage the data mining process effectively, the transformed data are stored in a relation in the Oracle database. We refer to this relation as *transformed relation* in the rest of this paper. Each attribute in the transformed relation corresponds to an identified variable. The interested readers are referred to [2] and [5] for the details of the use of transformation functions.

Instead of mining the subscriber database, we used DMEL to mine the transformed relation. The transformed relation was divided into two partitions: the data concerning with whether subscribers have churned or have not churned in the time interval from August to September 1999 and the data concerning with whether subscribers would churn or would not churn in October 1999. The former was used as the training dataset for DMEL to discover rules and the latter was used as the testing dataset for DMEL to make the "churn" and "no churn" predictions based on the discovered rules.

We applied DMEL to the training dataset to discover rules and predict the "churn" or "no churn" of the subscribers in the testing dataset. In the telecommunications industry, the "churn"

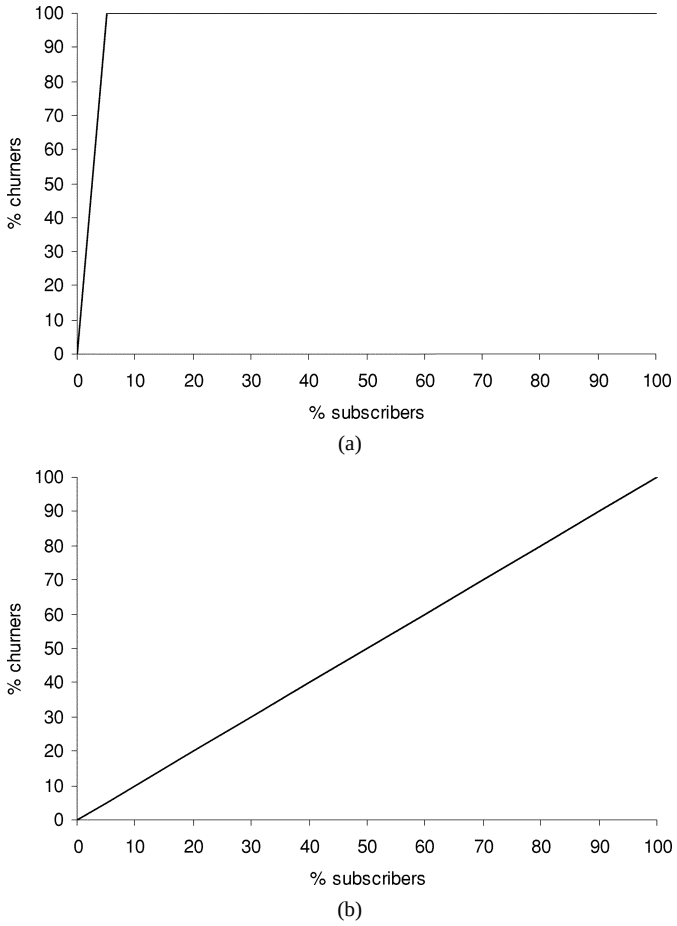


Fig. 8. Reference lift curves. (a) Lift curve representing perfect discrimination of churners from nonchurners. (b) Lift curve representing no discrimination of churners from nonchurners.

and "no churn" prediction is usually expressed as a *lift curve*. The lift curve plots the fraction of all churners having churn probability above the threshold against the fraction of all subscribers having churn probability above the threshold. The lift curve indicates the fraction of all churners can be caught if a certain fraction of all subscribers were contacted. Since the customer services centers of a carrier only have a fixed number of staff that is able to contact a fixed fraction of all subscribers, the lift curve, which can estimate the fraction of churners can be caught given the limited resources, is very useful in the telecommunications industry.

The lift curve representing perfect discrimination of churners from nonchurners and that representing no discrimination of churners from nonchurners under a churn rate of 5% are shown in Fig. 8(a) and (b), respectively. We refer to the former and the latter as *perfect churn predictor* and *random churn predictor*, respectively.

In order to evaluate the performance of DMEL using lift curve, we rank the tuples in the testing dataset according to the total weight of evidence. Given the prediction and the total weight of evidence produced by DMEL over the testing dataset, the tuples predicted to churn are sorted in the descending order of the total weight of evidence, whereas those tuples predicted to not churn are sorted in the ascending order of the total weight of evidence. The tuples predicted to churn come before the tuples predicted to not churn. Using the above method, we have an ordering of the tuples in the testing dataset such that the ones with higher probability to churn come before the ones with lower probability.

Since the churn rates of different carriers are different and the churn rate of a specific carrier varies from time to time, we have created several datasets with different monthly churn rates by randomly deleting tuples in the training and the testing datasets until appropriate fractions of churners and nonchurners are obtained. We can then plot the performance of DMEL in the form of lift curves under different monthly churn rates (Fig. 9). The performance of DMEL under different setups (Table I) is also shown in Fig. 9.

In order to facilitate comparisons, we also applied C4.5 and nonlinear neural networks to these datasets. The neural networks used in our experiments are multilayer perceptrons with a single hidden layer which contains 20 nodes and they were trained by the back propagation algorithm with the learning rate was set to 0.3 and the momentum term was set to 0.7. The lift curves for C4.5 and neural networks are also shown in Fig. 9.

As shown in Fig. 9, the performances of DMEL were more or less the same under different setups of the crossover probability for the *crossover*-1 and the *crossover*-2 operator. This is a nice feature because it is usually difficult for human users to determine the appropriate values of an algorithm's parameters for it may perform well under a specific setup in a certain environment and may perform poorly under the same setup in another environment.

Regardless of the values of p_1 and p_2, the performance of DMEL was always better than that of the random churn predictor when different fraction of subscribers were contacted under different monthly churn rates. When compared with C4.5, DMEL identified more churners than C4.5 under different monthly churn rates. It is important to note that neural networks also identified more churners than C4.5, which is consistent with the study in [32]. When compared with neural networks, DMEL identified more churners than neural networks did when a small fraction ($\leq$10%) of subscribers were contacted under different monthly churn rates. When the fraction of subscribers contacted were relatively large ($>$10%), the performance of DMEL was better than that of neural networks under a monthly churn rate of $\leq$4%, whereas its performance was comparable to neural networks' under a monthly churn rate of 6% and 8%. It is interesting to note that DMEL outperformed neural networks when $\leq$80% of subscribers were contacted under a monthly churn rate of 10%.

To better compare the performance of DMEL, C4.5, and neural networks, let us consider the *lift factor*, which is defined as the ratio of the fraction of churners identified and the fraction of subscribers contacted. For example, if $x\%$ of churners were identified when $y\%$ of subscribers were contacted, the lift factor were x/y. It is important to note that the lift factor for the random churn predictor is 1. Owing to the limited number of staff in the carrier's customer services center, it can only contact 5% of all subscribers. The lift factors for DMEL, C4.5, and neural networks when 5% of subscribers were contacted under different monthly churn rates are shown in Fig. 10.

Again, regardless of the values of p_1 and p_2, DMEL obtained higher lift factors than neural networks, which in turn obtained

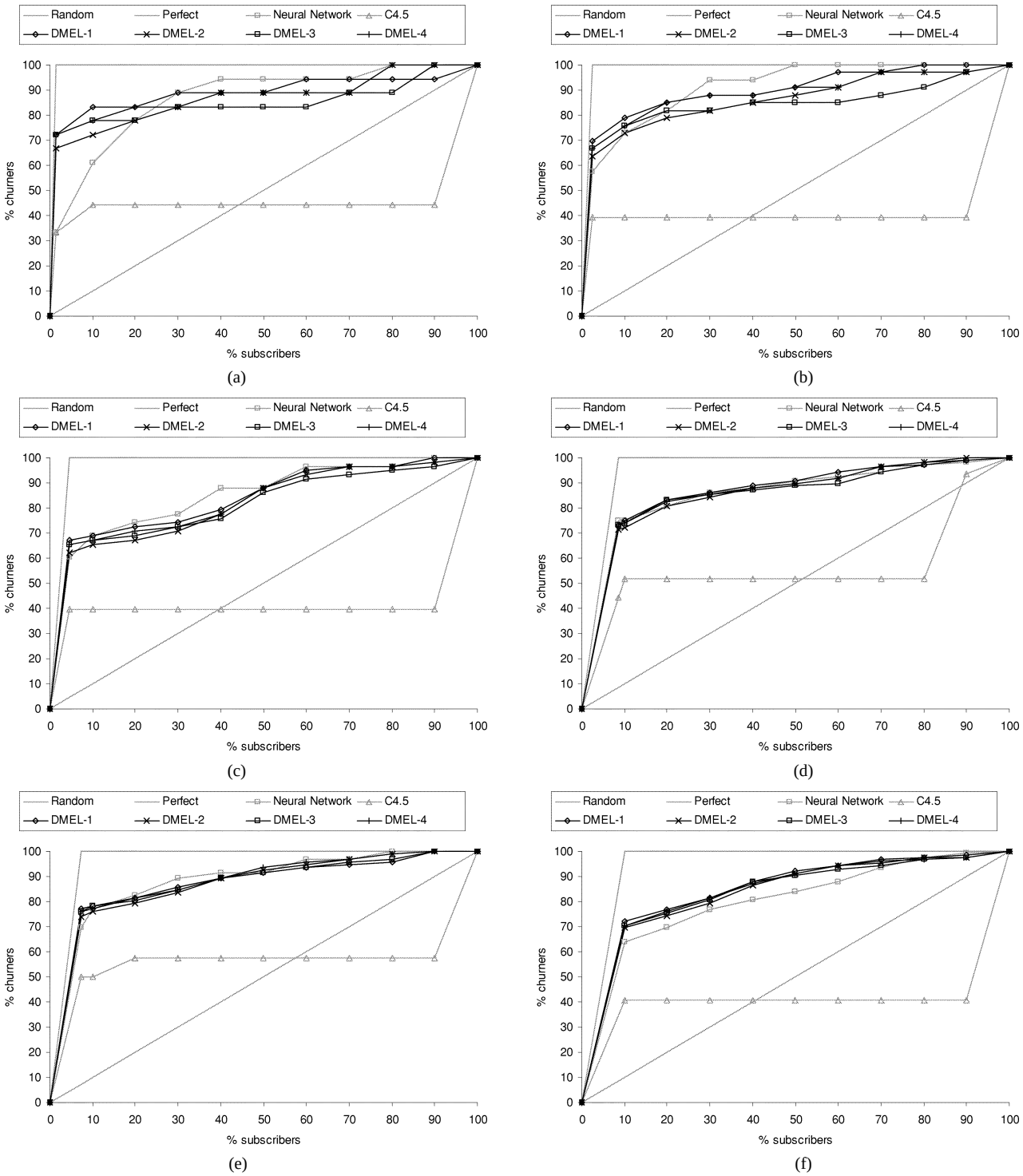


Fig. 9. Lift curves for DMEL, C4.5, and neural network under different monthly churn rates averaged over ten runs. (a) Monthly churn rate = 1%. (b) Monthly churn rate = 2%. (c) Monthly churn rate = 4%. (d) Monthly churn rate = 6%. (e) Monthly churn rate = 8%. (f) Monthly churn rate = 10%.

higher lift factors than C4.5, when 5% of subscribers were contacted under different monthly churn rates. The experimental results showed that DMEL is able to make accurate churn prediction under different churn rates. Furthermore, the relationships discovered by neural networks are encoded in the weights of the connections. It is difficult, if not impossible, to decode the discovered relationships and present them to the domain expert in an interpretable form. Unlike neural networks, DMEL is able to present the discovered relationships in the form of rules, which are easy for the domain expert to comprehend. Although the relationships discovered by C4.5 can also be represented in the form of rules, the experimental results showed that DMEL outperformed C4.5.

To evaluate their computation efficiencies, Table V shows the execution times for DMEL, C4.5, and neural networks under different monthly churn rates. At a specific monthly churn rate, the execution times for DMEL-1, DMEL-2, DMEL-3, and DMEL-4 are more or less the same because they differ

TABLE V
EXECUTION TIMES FOR DMEL, C4.5, AND NEURAL NETWORK UNDER DIFFERENT MONTHLY CHURN RATES AVERAGED OVER TEN RUNS

Monthly Churn Rate	Execution Time (sec.)					
	Neural Network	C4.5	DMEL-1	DMEL-2	DMEL-3	DMEL-4
1%	54,852.0	1,305.6	21,852.1	18,134.5	19,055.7	17,743.7
2%	55,117.2	1,765.2	24,234.0	21,498.5	26,534.3	21,885.5
4%	55,691.4	2,071.4	33,028.0	28,054.1	28,103.0	28,806.9
6%	58,021.6	1,646.2	31,805.6	31,074.7	30,115.4	28,883.6
8%	55,446.7	1,280.0	34,333.4	35,365.7	34,186.1	34,210.5
10%	55,567.5	1,046.2	38,902.7	38,168.7	39,128.3	42,980.9

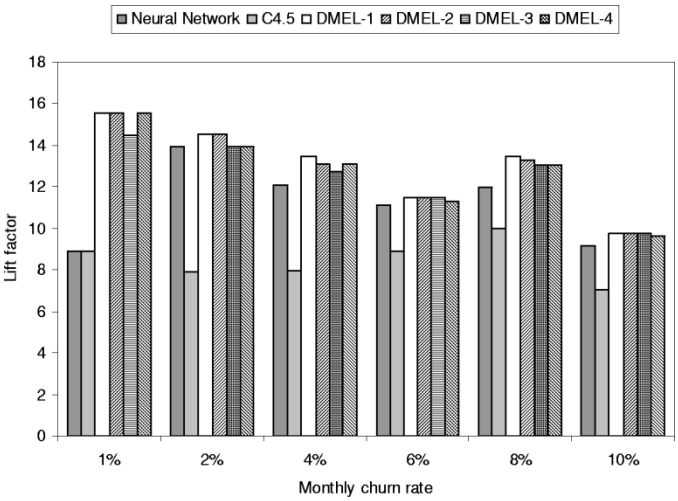


Fig. 10. Lift factors for DMEL, C4.5, and neural network under different monthly churn rates averaged over ten runs.

from each other by using different values of p_1 and p_2 only. Since $p_1 + p_2 = 1$ in different setups, their time complexities should be more or less the same. When the monthly churn rate increases, the execution time for DMEL increases because more and more relationships are found interesting and, hence, the number of alleles in a chromosome increases.

The experimental results showed that DMEL accomplished the data mining task faster than neural networks. Of the three approaches, C4.5 required the least execution time to complete since C4.5 used less number of iterations than neural networks and DMEL. However, C4.5 is unable to produce churn prediction as accurate as neural networks and DMEL (Figs. 9 and 10).

In the rest of this section, we present the rules discovered by DMEL and found to be interesting and useful by the domain expert from the carrier in Malaysia. The domain expert has found the following rule very useful:

$$\text{Type} = \text{Personal} \wedge \text{Bonus} = \text{No} \Rightarrow \text{Churn} = \text{True}[\hat{w} = 1.75].$$

This rule states that a subscriber churns if he/she subscribes the service plan personally and he/she is not admitted to any bonus scheme with weight of evidence of 1.75. According to this rule, the domain expert suggested that the carrier could admit those subscribers who subscribe the service plan personally and have not already admitted to any bonus scheme to a bonus scheme so as to retain them.

Another rule the domain expert found to be interesting is listed in the following:

$$\text{Sex} = \text{Male} \wedge \text{Tenure} \in [378, 419] \Rightarrow \text{Churn} = \text{True}[\hat{w} = 0.78].$$

The above rule states that a male subscriber who has used the service plan for a period between 378 and 419 days churns with weight of evidence of 0.78. Although the domain expert cannot explain why this rule is applicable to male subscribers only, he found this rule meaningful because a new subscriber is usually entitled a rebate after using the service plan for a period of one year and one can still keep the money even though he churns after receiving the rebate. In order to retain these subscribers, the domain expert suggested that the carrier could offer them incentives or rebates after using the service plan for another year when they have used the service plan for a period of one year.

In addition to the above rules, DMEL has discovered the following rule:

$$\begin{aligned} \text{District} =& \text{Kuala Lumpur} \wedge \text{Payment Method} \\ &= \text{Cash} \wedge \text{Age } \epsilon [36, 44] \\ \Rightarrow \text{Churn} &= \text{True}[\hat{w} = 1.20]. \end{aligned}$$

This rule states that a subscriber churns if he/she lives in Kuala Lumpur, is of age between 36 and 44, and paid bills using cash with weight of evidence of 1.20. Although the domain expert can hardly explain why this rule applies to those subscribers in this age group living in Kuala Lumpur only, he found it meaningful because it is easier for a subscriber to churn if he/she pays bills using cash when compared with one who pays bills using autopay. The domain expert found this rule useful because it identifies a niche for the carrier to retain its subscribers.

Furthermore, the domain expert also found the following rule interesting:

$$\begin{aligned} \text{Sex} =& \text{Male} \wedge \text{District} \\ &= \text{Penang} \wedge \text{Subscription Channel} \\ &= \text{Dealer} \wedge \text{Dealer Group} \\ &= \text{A} \Rightarrow \text{Churn} = \text{True}[\hat{w} = 1.84]. \end{aligned}$$

This rule states that a male subscriber who lives in Penang and subscribed the service through a dealer, which is under Dealer

Group A,[2] churns with weight of evidence of 1.84. The domain expert suggested that the churn of the subscribers might be due to the poor customer services provided by the dealers, which are under Dealer Group A, in Penang. He recommended the carrier to investigate into the service level of these dealers so as to introduce corrective actions.

VI. Conclusion

In this paper, we proposed a new data mining algorithm, called DMEL, to mine rules in databases. DMEL searches through huge rule spaces effectively using an evolutionary approach. Specifically, DMEL encodes a complete set of rules in one single chromosome. It performs its tasks by generating a set of initial first-order rules using a probabilistic induction technique so that, based on these rules, rules of higher orders are obtained iteratively. DMEL evaluates the fitness of a chromosome using a function defined in terms of the probability that the attribute values of a record can be correctly determined using the rules it encodes. With these characteristics, DMEL is capable of finding both positive and negative relationships among attributes for predictive modeling without any subjective input required of the users. To evaluate the performance of DMEL, it is applied to several real-world databases and the experimental results showed that DMEL is able to provide accurate classification.

In particular, we have applied DMEL to a database of 100 000 subscribers provided by a carrier in Malaysia. Using the discovered rules, DMEL is able to predict whether a subscriber will churn in the near future. The carrier can then offer incentives to the potential churners in order to retain them. The "churn" or "no churn" prediction is expressed as a lift curve, which indicates the fraction of all churners can be caught if a certain fraction of all subscribers were contacted. In our experiments, we also applied C4.5 and neural networks for churn prediction. The experimental results showed that DMEL outperformed neural networks, which in turn outperformed C4.5. Specifically, DMEL identified more churners than neural networks when a small fraction ($\leq$10%) of subscribers were contacted whereas the performance of DMEL is comparable to that of neural networks when the fraction of subscribers contacted was relatively large ($>$10%). The ability to identify more churners when only a small fraction of subscribers were contacted is important because the customer services center of a carrier has fixed number of staff and they can contact a small fraction of subscribers only. The experimental results on the subscriber database also showed that DMEL is robust in a way that it is able to discover rules hidden in the database and to predict the churns of subscribers under different churn rates. Since the churn rates of different subscribers are different and the churn rate of a specific carrier varies from time to time, robustness is necessary to an effective churn predictor.

References

[1] A. Agresti, *Categorical Data Analysis*. New York: Wiley, 1990.

[2] W.-H. Au and K. C. C. Chan, "Mining fuzzy association rules in a bank-account database," *IEEE Trans. Fuzzy Syst.*, vol. 11, pp. 238–248, Apr. 2003.

[3] C. Bishop, *Neural Networks for Pattern Recognition*. New York: Oxford Univ. Press, 1995.

[2]In order to maintain the anonymity of the carrier, we cannot disclose the name of the dealer group and we simply call it Dealer Group A in this paper.

[4] B. P. Carlin and T. A. Louis, *Bayes and Empirical Bayes Methods for Data Analysis*, 2nd ed. London , U.K.: Chapman & Hall, 2000.

[5] K. C. C. Chan and W.-H. Au, "Mining fuzzy association rules in a database containing relational and transactional data," in *Data Mining and Computational Intelligence*, A. Kandel, M. Last, and H. Bunke, Eds. New York: Physica-Verlag, 2001, pp. 95–114.

[6] K. C. C. Chan and A. K. C. Wong, "APACS: A system for the automatic analysis and classification of conceptual patterns," *Comput. Intell.*, vol. 6, pp. 119–131, 1990.

[7] ——, "A statistical technique for extracting classificatory knowledge from databases," in *Knowledge Discovery in Databases*, G. Piatetsky-Shapiro and W. J. Frawley, Eds. Menlo Park, CA:/Cambridge, MA: AAAI/MIT Press, 1991, pp. 107–123.

[8] J. Y. Ching, A. K. C. Wong, and K. C. C. Chan, "Class-Dependent discretization for inductive learning from continuous and mixed-mode data," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 17, pp. 1–11, July 1995.

[9] A. Choenni, "Design and implementation of a genetic-based algorithm for data mining," in *Proc. 26th Int. Conf. Very Large Data Bases*, Cairo, Egypt, 2000, pp. 33–42.

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

[11] U. M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, "From data mining to knowledge discovery: An overview," in *Advances in Knowledge Discovery and Data Mining*, U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, Eds. Menlo Park, CA:/Cambridge: MA: AAAI/MIT Press, 1996, pp. 1–34.

[12] M. V. Fidelis, H. S. Lopes, and A. A. Freitas, "Discovering comprehensible classification rules with a genetic algorithm," in *Proc. 2000 Congress Evolutionary Computation*, San Diego, CA, 2000, pp. 805–810.

[13] D. B. Fogel, *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. Piscataway, NJ: IEEE Press, 1995.

[14] R. Forsyth, *PC/BEAGLE User's Guide*. Nottingham, U.K.: Pathway Research Ltd., 1990.

[15] A. A. Freitas, "Understanding the critical role of attribute interaction in data mining," *Artif. Intell. Rev.*, vol. 16, pp. 177–199, 2002.

[16] J. Gehrke, V. Ganti, R. Ramakrishnan, and W.-Y. Loh, "BOAT – Optimistic decision tree construction," in *Proc. ACM SIGMOD Int. Conf. Management of Data*, Philadelphia, PA, 1999, pp. 169–180.

[17] J. Gehrke, R. Ramakrishnan, and V. Ganti, "RainForest – A framework for fast decision tree construction of large datasets," in *Proc. 24th Int. Conf. Very Large Data Bases*, New York, 1998, pp. 416–427.

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

[19] D. P. Greene and S. F. Smith, "Using coverage as a model building constraint in learning classifier systems," *Evol. Comput.*, vol. 2, no. 1, pp. 67–91, 1994.

[20] R. R. Hill, "A Monte Carlo study of genetic algorithm initial population generation methods," in *Proc. 31st Conf. Winter Simulation–A Bridge to the Future*, Phoenix, AZ, 1999, pp. 543–547.

[21] J. Holland, "Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems," in *Machine Learning: An Artificial Intelligence Approach*, R. Michalski, J. Carbonell, and T. Mitchell, Eds. San Mateo, CA: Morgan Kaufmann, 1986.

[22] H. Ishibuchi and T. Nakashima, "Improving the performance of fuzzy classifier systems for pattern classification problems with continuous attributes," *IEEE Trans. Ind. Electron.*, vol. 46, pp. 1057–1068, Dec. 1999.

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

[24] B. A. Julstrom, "Seeding the population: Improved performance in a genetic algorithm for the rectilinear steiner problem," in *Proc. ACM Symp. Applied Computing*, Phoenix, AZ, 1994, pp. 222–226.

[25] R. Kohavi, "Scaling up the accuracy of naive-bayes classifiers: A decision tree hybrid," in *Proc. 2nd Int. Conf. Knowledge Discovery and Data Mining*, Portland, OR, 1996.

[26] W. Kwedlo and M. Kretowski, "Discovery of decision rules from databases: An evolutionary approach," in *Proc. 2nd European Symp. Principles of Data Mining and Knowledge Discovery*, Nantes, France, 1998, pp. 370–378.

[27] R. J. Light and B. H. Margolin, "An analysis of variance for categorical data," *J. Amer. Statist. Assoc.*, vol. 66, pp. 534–544, 1971.

[28] J. Lockwood, "Study predicts 'Epidemic' churn," in *Wireless Week*, Aug. 25, 1997.

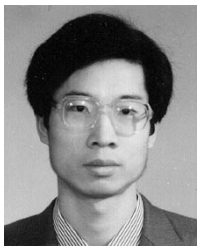
[29] A. D. McAulay and J. C. Oh, "Improving learning of genetic rule-based classifier systems," *IEEE Trans. Syst. Man, Cybern.*, vol. 24, pp. 152–159, Jan. 1994.
[30] M. Mehta, R. Agrawal, and J. Rissanen, "SLIQ: A fast scalable classifier for data mining," in *Proc. 5th Int. Conf. Extending Database Technology*, Avignon, France, 1996, pp. 18–32.
[31] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, 3rd Revised and Extended ed. New York: Springer-Verlag, 1996.
[32] M. C. Mozer, R. Wolniewicz, D. B. Grimes, E. Johnson, and H. Kaushansky, "Predicting subscriber dissatisfaction and improving retention in the wireless telecommunications industry," *IEEE Trans. Neural Networks*, vol. 11, pp. 690–696, May 2000.
[33] M. O. Noordewier, G. G. Towell, and J. W. Shavlik, "Training knowledge-based neural networks to recognize genes in DNA sequences," in *Advances in Neural Information Processing Systems*, R.P. Lippmann, J.E. Moody, and D.S. Touretzky, Eds. San Mateo, CA: Morgan Kaufmann, 1991, vol. 3.
[34] J. R. Quinlan, "Decision trees as probabilistic classifiers," in *Proc. 4th Int. Workshop Machine Learning*, Irvine, CA, 1987, pp. 31–37.
[35] ——, "Simplifying decision trees," *Int. J. Man-Mach. Stud.*, vol. 27, pp. 221–234, 1987.
[36] ——, *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann, 1993.
[37] R. Rastogi and K. Shim, "PUBLIC: A decision tree classifier that integrates building and pruning," in *Proc. 24th Int. Conf. Very Large Data Bases*, New York, 1998, pp. 404–415.
[38] J. Shafer, R. Agrawal, and M. Mehta, "SPRINT: A scalable parallel classifier for data mining," in *Proc. 22nd Int. Conf. Very Large Data Bases*, Mumbai (Bombay), India, 1996, pp. 544–555.
[39] S. Smith, "Flexible learning of problem solving heuristics through adaptive search," in *Proc. 8th Int. Joint Conf. Artificial Intelligence*, Karlsruhe, Germany, 1983, pp. 422–425.
[40] J. W. Smith, J. E. Everhart, W. C. Dickson, W. C. Knowler, and R. S. Johannes, "Using the ADAP learning algorithm to forecast the onset of diabetes mellitus," *Proc. Symp. Computer Applications and Medical Cares*, pp. 261–265, 1988.
[41] C. H. Yang and K. E. Nygard, "The effects of initial population in genetic search for time constrained traveling salesman problems," in *Proc. ACM Conf. Computer Science*, Indianapolis, IN, 1993, pp. 378–383.

Wai-Ho Au received the B.A. degree (first class honors) in computing studies and the M.Phil. degree in computing from The Hong Kong Polytechnic University (HKPU), Kowloon, in 1995 and 1998, respectively. He is currently working toward the Ph.D. degree in computing at HKPU.

He has been in charge of several large-scale software development projects, including a system integration project for an international airport, a data warehouse project for a utility company, and an intelligent home system for a high-tech startup. He is now a Manager of software development in the Department of Computing, The Hong Kong Polytechnic University. His research interests include data mining, data warehousing, fuzzy computing, and evolutionary computation.

Keith C. C. Chan received the B.Math. (honors) degree in computer science and statistics, and the M.A.Sc. and Ph.D. degrees in systems design engineering from the University of Waterloo, Waterloo, ON, Canada, in 1983, 1985, and 1989, respectively.

He has a number of years of academic and industrial experience in software development and management. In 1989, he joined the IBM Canada Laboratory, Toronto, ON, where he was involved in the development of image and multimedia software, as well as software development tools. In 1993, he joined the Department of Electrical and Computer Engineering, Ryerson Polytechnic University, Toronto, as an Associate Professor. He joined The Hong Kong Polytechnic University, Kowloon, in 1994, and is currently the Head of the Department of Computing. He is an Adjunct Professor with the Institute of Software, The Chinese Academy of Sciences, Beijing, China. He is active in consultancy and has served as consultant to government agencies, as well as large and small-to-medium sized enterprises in Hong Kong, China, Singapore, Malaysia, Italy, and Canada. His research interests are in data mining and machine learning, computational intelligence, and software engineering.



Xin Yao (M'91–SM'96–F'03) received the B.Sc. degree from the University of Science and Technology of China (USTC), Hefei, the M.Sc. degree from the North China Institute of Computing Technologies (NCI), Beijing, and the Ph.D. degree in computer science from the USTC, in 1982, 1985, and 1990, respectively, all in computer science.

He is currently a Professor of Computer Science and the Director of the Centre of Excellence for Research in Computational Intelligence and Applications (CERCIA), University of Birmingham, U.K., and a Visiting Professor at four other universities in China and Australia. He was a Lecturer, Senior Lecturer, and an Associate Professor at University College, University of New South Wales, the Australian Defence Force Academy (ADFA), Canberra, Australia, between 1992–1999. He held Postdoctoral Fellowships from the Australian National University (ANU), Canberra, and the Commonwealth Scientific and Industrial Research Organization (CSIRO), Melbourne, between 1990 and 1992. His major research interests include evolutionary computation, neural network ensembles, global optimization, computational time complexity, and data mining.

Dr. Yao is the Editor-in-Chief of the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, an Associate Editor and an Editorial Board Member of five other international journals, and the Chair of the IEEE Neural Networks Society Technical Committee on Evolutionary Computation. He is the recipient of the 2001 IEEE Donald G. Fink Prize Paper Award and has given more than 20 invited keynote and plenary speeches at various conferences. He has chaired/co-chaired more than 25 international conferences in evolutionary computation and computational intelligence, including CEC 1999, PPSN VI 2000, CEC 2002, and PPSN 2004.