Available online at www.sciencedirect.com

ScienceDirect

Expert Systems with Applications 36 (2009) 3066–3076

Expert Systems with Applications

www.elsevier.com/locate/eswa

Genetic algorithm-based strategy for identifying association rules without specifying actual minimum support

Xiaowei Yan[a], Chengqi Zhang[a], Shichao Zhang[b,c,*]

[a] Faculty of Information Technology, University of Technology, Sydney, P.O. Box 123, Broadway NSW 2007, Australia
[b] Institute of Logics, Zhongshan University, PR China
[c] School of Computer Science and Information Technology, Guangxi Normal University, PR China

Abstract

We design a genetic algorithm-based strategy for identifying association rules without specifying actual minimum support. In this approach, an elaborate encoding method is developed, and the relative confidence is used as the fitness function. With genetic algorithm, a global search can be performed and system automation is implemented, because our model does not require the user-specified threshold of minimum support. Furthermore, we expand this strategy to cover quantitative association rule discovery. For efficiency, we design a generalized FP-tree to implement this algorithm. We experimentally evaluate our approach, and demonstrate that our algorithms significantly reduce the computation costs and generate interesting association rules only.
© 2008 Elsevier Ltd. All rights reserved.

Keywords: Data mining; Association rule mining; Genetic algorithm; Threshold setting

1. Introduction

Association rule mining (Agrawal, Imielinski, & Swami, 1993) plays a key role in boosting the research, development and application of data mining techniques. This leads to a great many significant technologies and methodologies for identifying association rules. These techniques mainly focus on algorithm scale-up and data reduction for the efficiency issue.

However, these mining algorithms are mostly based on the assumption that users can specify the minimum support appropriate to their databases, and thus referred to as the *Apriori-like algorithms* (Zhang & Zhang, 2002; Zhang, Lu, & Zhang, 2004). Han, Wang, Lu, and Tzvetkov (2002) have pointed out that setting the minimum support is quite subtle, which can hinder the widespread applications of these algorithms. Our own experiences of mining transaction databases also tell us that the setting is by no means an easy task.

Recognizing the above limitation, a great many techniques have been developed to attack this issue. Han et al. (2002) designed a strategy to mine top-*k* frequent closed patterns for effectiveness and efficiency. Piatetsky-Shapiro and Steingold proposed a method to identify only the top 10% or 20% of the prospects with the highest score for marketing (Piatetsky-Shapiro & Steingold, 2000). Roddick and Rice (2001) presented the independent thresholds and context dependent thresholds to measure time-varying interestingness of events for temporal data. Hipp and Guntzer (2002) explored a new mining approach that postpones constraints from mining to evaluation. Wang, He, Cheung, and Chin (2001) designed a confidence-driven mining strategy without minimum support to identify new patterns. Cheung and Fu (2004) developed a technique to identify frequent itemsets without the support threshold. Zhang et al. (2004) advocated a fuzzy-logic-based method to acquire user threshold of minimum support for mining association rules. However, most of these approaches attempt to avoid specifying the minimum support. Some

* Corresponding author.
E-mail addresses: xyan@it.uts.edu.au (X. Yan), chengqi@it.uts.edu.au (C. Zhang), zhangsc@it.uts.edu.au (S. Zhang).

0957-4174/$ - see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2008.01.028

of them are actually confidence-driven methods. The last approach solves the minimum support issue by coding technique. All of these efforts provide a good insight into the difficulty of specifying a minimum-support constraint.

In this paper we use genetic algorithm to identify association rules without minimum support. Genetic algorithm is efficient for global search work, especially when the search space is too large to use a deterministic search method. It imitates the mechanics of natural species evolution with genetics principles, such as natural selection, crossover, and mutation. In particular, our approach does not require users to specify the minimum-support threshold. Instead of generating an unknown number of interesting rules in traditional mining models, only the most interesting rules are returned according to the interestingness measure defined by the fitness function. Obviously, this method is database-independent in contrast to these *Apriori*-based algorithms. This approach leads to (1) effectiveness and efficiency for global search; and (2) system automation, because our model does not require the user-specified threshold of minimum support.

The rest of this paper is organized as follows. We start with brief recalls of both concepts concerning about association rules and current work on genetic algorithm-based learning in Section 2. In Section 3, we present our genetic algorithm-based model for identifying association rules, including the encoding method, genetic operators, and the *ARMGA* algorithm. In Section 4, we expand the *ARMGA* algorithm to identify generalized association rules. In Section 5, we experimentally evaluate our approach. Finally, we conclude our work in Section 6.

2. Preliminaries

This section recalls some concepts concerning association rule mining, quantitative association rule, and genetic algorithms.

2.1. Association rules

$I = \{i_1, i_2, \dots, i_m\}$ is a set of literals, or *items*. X is an *itemset* if it is a subset of I. Itemset X is a k-itemset if X exactly has k items.

$D = \{t_1, t_2, \dots, t_n\}$ is a set of transactions, called the *transaction database*, where each transaction t_i has a transaction identifier tid_i, and a k_i-itemset X_i, that is, $t_i = (tid_i, X_i), 1 \leqslant k_i \leqslant m, i = 1, \dots, n$. A transaction, t_i, contains an itemset, X, if and only if, for any item $i \in X$, i is in t-itemset X_i.

There is a natural *lattice structure*, namely the subset/superset structure, over the set of all itemsets, 2^I. Some certain sub-lattice of it can be taken as a taxonomy.

An itemset, X, in a transaction database, D, has a *support*, denoted as $supp(X)$ or simply $p(X)$, that is the ratio of transactions in D containing X. Or

$$supp(X) = |X(t)|/|D|$$

where $X(t) = \{t \text{ in } D | t \text{ contains } X\}$, $|X(t)|$ and $|D|$ are the numbers of transactions in $X(t)$ and D, respectively.

An itemset, X, in a transaction database, D, is called a large (frequent) itemset if its support is equal to, or greater than, a threshold of minimal support (*minsupp*), which is given by user or expert.

An *association rule* is an implication $X \rightarrow Y$, where itemsets X and Y do not intersect.

Each association rule has two quality measurements, *support* and *confidence*, defined as follows.

- the support of a rule $X \rightarrow Y$ is the support of $X \cup Y$, where $X \cup Y$ is the union of X and Y;
- the confidence of a rule $X \rightarrow Y$, written as $conf(X \rightarrow Y)$, is the ratio $|(X \cup Y)(t)|/|X(t)|$ or $supp(X \cup Y)/supp(X)$.

That is, support implies frequency of occurring patterns, and confidence means the strength of implication. We now introduce the *support-confidence framework* (Agrawal et al., 1993).

Let I be the set of items in database D, $X, Y \subseteq I$ be itemsets, $X \cap Y = \emptyset$, $p(X) \neq 0$ and $p(Y) \neq 0$. The minimal support, *minsupp*, and the minimal confidence, *minconf*, are given by user or expert. Then rule $X \rightarrow Y$ is valid if

$$supp(X \cup Y) \geqslant minsupp, \text{ and}$$
$$conf(X \rightarrow Y) = \frac{supp(X \cup Y)}{supp(X)} \geqslant minconf,$$

Mining association rules can be taken into the following two subproblems.

(1) Generating all itemsets for which supports are greater than, or equal to, the user-specified minimum support, that is, generating all large itemsets; and
(2) Generating all the rules which satisfy the minimum confidence constraint in a naive way as follows. For each large itemset X, and any $B \subset X$, let $A = X - B$. If the confidence of a rule $A \rightarrow B$ is greater than, or equal to, the minimum confidence (or $supp(X)/supp(A) \geqslant minconf$), then $A \rightarrow B$ can be extracted as a valid rule.

To demonstrate the use of the support-confidence framework, we illustrate the process of mining association rules by the following example.

Example 1. Assume that we have a transaction database in a supermarket, as shown in Table 1. There are six transactions in the database with their transaction identifiers (TIDs) ranging from 100 to 600. The universal itemset $I = \{A, B, C, D, E\}$, where $A, B, C, D,$ and E can be any items in the supermarket. For instance, $A =$ "bread", $B =$ "milk", $C =$ "sugar", $D =$ "coffee", and $E =$ "biscuit".

There are totally $2^5 (= 32)$ itemsets. $\{A\}, \{B\}, \{C\}, \{D\}$, and $\{E\}$ are all 1-itemsets, $\{AC\}$ is a 2-itemset, and so on. $supp(BC) = 4/6 = 0.67$ because there are four transac-

Table 1
An example transaction database

TID	Items bought
100	*ABCD*
200	*BCE*
300	*ABCE*
400	*BE*
500	*ACD*
600	*BCE*

tions that contain both A and B. Let $minsupp = 50\%$ and $minconf = 80\%$. Then, A, B, C, E, AC, BC, BE, and BCE are all frequent itemsets. The confidence of association rule $A \rightarrow C$ is $conf(A \rightarrow C) = supp(AC)/supp(A) = 3/3 = 1.0$. Hence rule $A \rightarrow C$ is valid. Similarly we have $conf(C \rightarrow A) = 3/5 = 0.6$. Hence, rule $C \rightarrow A$ is not valid.

2.2. Quantitative association rules

Let $U = \{A_1, A_2, \ldots, A_n\}$ be the universal set of attributes in a generalized relational database. For each quantitative attribute $A_k \in U$, assume that a_k and b_k are respectively the minimum and maximum values among all the values of attribute A_k in the database. Hence the value domain of A_k, written as $dom(A_k)$, can be taken as a close interval of $[a_k, b_k]$. Given a granularity $g_k > 0$ for attribute A_k, we can partition interval $[a_k, b_k]$ into n_k intervals, $[a_k, a_k + g_k]$, $[a_k + g_k, a_k + 2g_k], \ldots,$ $[a_k + (n_k - 1)g_k, b_k]$, where $n_k = \lfloor (b_k - a_k)/g_k \rfloor + 1$. These sub-intervals are all called the base intervals of A_k. We can easily see that $b_k \in [a_k + (n_k - 1)g_k, a_k + n_k g_k]$. If A_k is a categorical attribute, we define its granularity g_k=0. We do not need to partition the value domain of a categorical attribute. Suppose that v is a value in the domain of categorical attribute A_k. The corresponding base interval is $[v, v + g_k]$, i.e. the value v itself.

For any two intervals, $I_1 = [a, b]$ and $I_2 = [c, d]$, of A_k, there are three kinds of relationship between them as follows.

(1) Adjacent, if and only if $b = c$ or $a = d$.
(2) Disjoint, if and only if $b < c$ or $a > d$.
(3) Overlapped, if and only if $c < a < d$ or $c < b < d$.

For any two base intervals of an attribute, only the first two kinds of relationship are applicable because they cannot be overlapped from the definition.

A gapped interval GI is a union of a certain finite number of base intervals, that is, $GI = [a_1, b_1] \cup [a_2, b_2] \cup \cdots \cup [a_m, b_m]$, where $[a_i, b_i]$ is a base interval, $i = 1, 2, \ldots, m$, and m is a certain finite integer. An interval I is a maximum interval of GI, if $I \subseteq GI$, and for any interval $I' \subseteq GI$, we have $I' \subseteq I$ when $I' \cap I \neq \emptyset$. It is easy to prove the following properties, which are useful for the design and implementation of our genetic algorithm.

Property 1. *Gapped intervals are closed under operations of union, intersection, and difference. In other words, the union, intersection, and difference of any two gapped intervals are all gapped intervals.*

Property 2. *A gapped intervals GI can be uniquely decomposed into the union of several maximum intervals of GI. That is,* $GI = I_1 \cup I_2 \cup \cdots \cup I_k$*, where each* I_j *is a maximum interval of GI,* $j = 1, \ldots, k$*. For any interval* $I \subseteq GI$*, there exists a* $I_i, 1 \leqslant i \leqslant k$*, such that* $I \subseteq I_i$*.*

An item then can be defined as a pair of the form (A, GI), where A is an attribute in U, and gapped interval $GI \subseteq dom(A)$. There are two extreme situations. One is that $GI = dom(A)$ and item (A, GI) is called a universal item. Another is the zero item, written as $\emptyset$, when $GI = \emptyset$. A set of items is referred to as an itemset. An itemset X is a zero itemset, denoted as $X = \emptyset$, if $\forall I \in X, I = \emptyset$. Apparently, if there are two items in an itemset with the same attribute, (A, GI_1) and (A, GI_2), we can easily merge them into one item by $(A, GI_1 \cup GI_2)$, since $GI_1 \cup GI_2$ is also a gapped interval from Property 1. Therefore, we assume that any items in an itemset have different attributes, such that we can reasonably define the length of an itemset. The length of an itemset is the number of items in the itemset. Itemset of length k is denoted as k-itemset. Then a tuple, $t = (v_1, v_2, \ldots, v_n)$, can be viewed as an itemset, i.e. $t = \{(A_1, [v_1, v_1]), (A_2, [v_2, v_2]), \ldots, (A_n, [v_n, v_n])\}$, where the gapped interval of each item has only an interval with a single value $v_i \in dom(A_i)$, $i = 1, 2, \ldots, n$. A database DB is a set of tuples over U. We shall alternatively use $[v, v]$ and v, if there is no confusion.

A tuple t is said supporting an itemset X, if, for any item $(A, GI) \in X$, there exists a value $(A, v) \in t$, such that $v \in GI$. The support of itemset X, written as $supp(X)$, is the proportion of tuples in DB, which support X. Clearly, if there exists a zero item in X, then $supp(X)$=0.

2.3. Research into genetic algorithm-based learning

There have been many applications of genetic algorithms in the field of data mining and knowledge discovery. Most of them are addressed to the problem of classification.

Usually, genetic algorithms for rules mining are partitioned into two categories according to their encoding of rules in the population of chromosomes (Freitas, 2003). One encoding method is called Michigan approach, where each rule is encoded into an individual. Another is referred to as Pittsburgh approach, with which a set of rules are encoded into a chromosome. For example, Fidelis, Lopes, and Freitas (2000) gave a Michigan type of genetic algorithm to discover comprehensible classification rules, having an interesting chromosome encoding and introducing a specific mutation operator. But the method is impractical when the number of attribute is large. Weiss and Hirsh (1998) also followed the Michigan method to predict rare events. Pei, Goodman, and Punch (1997), on the other hand, used the Pittsburgh approach for discovery of classes and feature patterns.

Other applications can be demonstrated by *GA-Nuggets*, a system to infer the values of goal attributes given the values of predicting attributes (Freitas, 1999), and *SIAO1*, which finds the first- order-logic classification rules by generalizing a seed example. Moreover, a recent work which is also worthy to mention is the *dAR*, designed by Au and Chan (2002) for mining association rules or more exactly for discovering changing patterns in historical data. In *dAR*, the whole set of rules are encoded in a single chromosome and each rule is represented by some non-binary symbolic values. It uses a complicated fitness function and a Pittsburgh encoding.

Although it is known that genetic algorithm is good at searching for nondeterministic solution, it is still rare to see that genetic algorithm is used to mine association rules. We are going to further investigate the possibility of applying genetic algorithm to the association rules mining in the following sections.

3. Identifying association rules with genetic algorithm

Let $I = \{i_1, i_2, \ldots, i_n\}$ be the universal set of items. Then a transaction can be viewed as an itemset with variable length, and a database D can be defined as a set of transactions over I. Association rule $X \to Y$ is a k-rule if $X \cup Y$ is a k-itemset.

From Section 2.1, we know that the traditional task of mining association rules is to find all rules $X \to Y$, such that the supports and confidences of the rules are larger than, or equal to, the minimum support, *minsupp*, and the minimum confidence, *minconf*, respectively.

In our genetic algorithm, called *ARMGA*, we require that the confidence $conf(X \to Y)$ should be larger than, or equal to, $supp(Y)$, because we only deal with positive association rules of the form $X \to Y$. Hence, we define the relative confidence as the interestingness measure as follows:

$$rconf(X \to Y) = \frac{supp(X \cup Y) - supp(X)supp(Y)}{supp(X)(1 - supp(Y))}.$$

We now state our task of association rule mining as follows:

Problem 1. Given a rule length k, we search for some high-quality association k-rules, with their *rconf*s acceptably maximized, by using a genetic algorithm.

3.1. Encoding

We first present our encoding in this subsection. And then we design four basic genetic operators (see Section 3.2) and, an initializer and a fitness function based on the relative confidence (see Section 3.3). We finally integrate these modules into Algorithm *ARMGA* (see Section 3.4).

We now number and quote all items in $I = \{i_1, i_2, \ldots, i_n\}$ by their indexes. In other words, we can assume that the universal itemset $I = \{1, 2, \ldots, n\}$. Given an association k-rule $X \to Y$, where $X, Y \subset I$, and $X \cap Y = \emptyset$, we encode it into an individual as

j	A_1	...	A_j	A_{j+1}	...	A_k

where j is an indicator that separates the antecedent from the consequent of the rule. That is, $X = \{A_1, \ldots, A_j\}$ and $Y = \{A_{j+1}, \ldots, A_k\}, 0 < j < k$. Therefore, a k-rule $X \to Y$ is represented by k+1 positive integers.

3.2. Genetic operators

This subsection describe three genetic operators, select, crossover, and mutation. The fitness function is given as well.

1. Function $select(c, sp)$ acts as a filter of chromosome with considerations of their fitness and probability *sp*. It returns *TRUE* if chromosome *c* is successfully selected with probability *sp*, and otherwise *FALSE* if failed. In this selection function, *frand*() is used to return a random real number ranged from 0 to 1.
2. Function $crossover(pop, cp)$ uses two-point strategy to reproduce offspring chromosomes at a probability of *cp* from population *pop*, and returns a new population. These two crossover points are randomly generated, such that any segment of chromosome may be chosen, illustrated in Fig. 1. Here, Function $irand(k)$ is used to return a random integer ranged from 0 to k. We generate i and j through this random function as the two crossover points, and make sure $i \leqslant j$ in the loop body. After crossover, a strategy of nearest-neighbour-substitution is used to ensure that any two genes in a chromosome are different.
3. Function $mutate(c, mp)$ occasionally changes genes of chromosome *c* at a probability of *mp*, besides considering the fitness of *c* as an additional weight. The nearest-

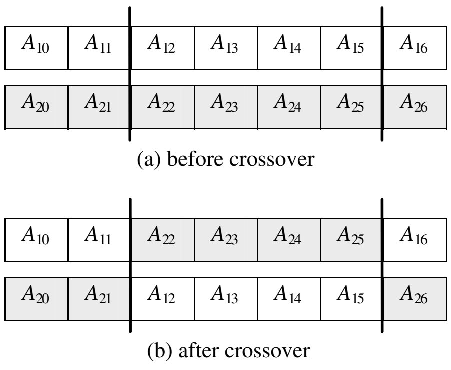
A10 A11 A12 A13 A14 A15 A16
A20 A21 A22 A23 A24 A25 A26
(a) before crossover
A10 A11 A22 A23 A24 A25 A16
A20 A21 A12 A13 A14 A15 A26
(b) after crossover

Fig. 1. An example of two-point crossover.

neighbour-substitution strategy is used again to guarantee that the changed genes are really new.
4. From *Problem 1*, our goal is to search the most interesting association rules. Hence, the fitness function is crucial for determining the interestingness of chromosome, and it does affect the convergence of the genetic algorithm. In Algorithm *ARMGA*, we define our fitness function as

$$fitness(c) = \frac{supp(A_1 \ldots A_k) - supp(A_1 \ldots A_j)supp(A_{j+1} \ldots A_k)}{supp(A_1 \ldots A_j)(1 - supp(A_{j+1} \ldots A_k))}$$

where $c = (j, A_1, \ldots, A_j, A_{j+1}, \ldots, A_k)$ is a given chromosome. The fitness of c is, in fact, the relative confidence of the corresponding association rule $\{A_1, \ldots, A_j\} \rightarrow \{A_{j+1}, \ldots, A_k\}$.

3.3. Initialization

Given a seed chromosome s, we use the $mutate(s, mp)$ function to produce an initial population $pop[0]$, where we have $mp = 1$. This initialization is shown in the following function.

```
population $initialise(s)$
begin
    $pop[0] \leftarrow s$;
    while $sizeof(pop[0]) < popsize/2$ do
    begin
        $pop\_temp \leftarrow \emptyset$;
        for $\forall\ c \in pop[0]$ do
        begin
            $pop\_temp \leftarrow pop\_temp \cup mutate(c, 1)$;
        end
        $pop[0] \leftarrow pop[0] \cup pop\_temp$;
    end
    return $pop[0]$;
end
```

The above function accepts a seed chromosome as its parameter and returns a population as the initial set of chromosomes. Function $sizeof(pop[0])$ returns the number of chromosomes in $pop[0]$. *popsize* is a constant given by user, representing the maximum number of chromosomes in a population.

3.4. ARMGA algorithm

Suppose that the current population is $pop[i]$. We first apply the *select* operator to $pop[i]$, and produce a new population $pop[i+1]$. Then any pair of chromosomes in $pop[i+1]$ are crossed over at a probability of *cp* to reproduce two offspring. Each new chromosome mutates at a probability of *mp*. Algorithm *ARMGA* generates a population, at last, with high-quality chromosomes.

```
population $ARMGA(s, sp, cp, mp)$
begin
    $i \leftarrow 0$;
    $pop[i] \leftarrow initialize(s)$;
    while not $terminate(pop[i])$ do
    begin
        $pop[i+1] \leftarrow \emptyset$;
        $pop\_temp \leftarrow \emptyset$;
        for $\forall\ c \in pop[i]$ do
            if $select(c, sp)$ then
                $pop[i+1] \leftarrow pop[i+1] \cup c$;
        $pop\_temp \leftarrow crossover(pop[i+1], cp)$;
        for $\forall\ c \in pop\_temp$ do
            $pop[i+1] \leftarrow (pop[i+1] - c) \cup mutate(c, mp)$;
        $i \leftarrow i + 1$;
    end
    return $pop[i]$;
end
```

Algorithm *ARMGA* stops, that is, the *terminate*() function returns a non-zero value, if and only if one of the following cases happens:

(1) The difference between the best and the worst chromosomes is less than a given value, α, which is small enough.
(2) The number of iterations, i, is larger than a given maximum number *maxloop*.

4. Expanding *ARMGA* for generalized association rules

Algorithm *ARMGA* is designed for Boolean association rule mining. This section will expand it to deal with generalized association rules.

4.1. Problem statement

As explained in Section 2.2, quantitative data ubiquitously exist in generalized relational databases. We really need to face the problem of quantitative association rule mining. In 1995, Srikant and Agrawal proposed a generalized model to handle both categorical and quantitative association rules (Srikant & Agrawal, 1995; Srikant & Agrawal, 1996). In their generalized model, the concept of itemset for Boolean attributes is expanded for both quantitative and categorical attributes, and a corresponding algorithm was given to mine generalized association rules. However, there are still some limitations, such as

(1) Partition of quantitative attribute, which is used in the model, is not easy for every attribute and every user.
(2) Users, and even experts, usually feel difficult to give those thresholds like the minimum support, the interest level, and the minimum confidence.

(3) The search space might be very large when we cope with quantitative attributes.
(4) The rules returned by the algorithm might be too many to deal with.
(5) The model does not deal with those rules like

$$\{(Age : 5\ldots 10 or 60\ldots 65), (Time : 17\ldots 18 or 20\ldots 21)\} \rightarrow \{(WatchTV : Yes)\};$$

(6) The proposed partitioning method may not be practical on some situations, for instance, when the data is highly skewed, and
(7) The logic in (Srikant & Agrawal, 1996) should be strengthened, since some concepts, such as the partial completeness level and the ancestor of an itemset, are used without correct definitions, and some results are implicatively used without proving, such as, $\overline{A} \cup \overline{B} = \overline{A \cup B}$, where $\overline{A}$, $\overline{B}$, and $\overline{A \cup B}$ are generalizations of A, B, and $A \cup B$ respectively.

Obviously, the problem of quantitative association rule mining has a much bigger search space than the problem of Boolean mining has. For the Boolean mining problem, each attribute only has one of the two values, *TRUE* and *FALSE*. On the other hand, each quantitative attribute might have an enormous amount of values appeared in the database. If we partition the value domain of a quantitative attribute into intervals, there should be still a lot of intervals we need to consider. Furthermore, we need to consider combinations of these intervals. Therefore, we might obtain much more rules than a decision-maker can process. Although we can decrease the amount of generated association rules by appropriately specifying the thresholds, such as the minimum support and interest level, the number of rules output can be unpredictable.

In summary, our evolutionary method for quantitative association rule mining is mainly motivated by (1) partition of quantitative attribute is not easy for every attribute and every user, (2) users, and even experts, usually feel difficult to specify the minimum-support, (3) the search space might be very large when we face quantitative attributes, and (4) the rules returned might be too many to deal with.

The mining task in this section is now re-stated as follows:

Problem 2. Given a relational database *DB* and a rule length k, we search for some high-quality generalized association k-rules in *DB*, which are both quantitative and categorical, with their relative confidences acceptably maximized, by using a genetic algorithm.

4.2. Encoding

Similar to the encoding method introduced in Section 3, we also encode each association rule in a single chromosome. Because any item in an itemset X has different attributes, we can index all items in X by their attributes. Furthermore, suppose that the universal attribute set is $U = \{A_1, A_2, \ldots, A_n\}$. Here we index all attributes by integers $1, 2, \ldots, n$. Surely we can also index all items in X by some integers ranged from 1 to n. Thus, assume that the universal itemset $U = \{1, 2, \ldots, n\}$. Given an association k-rule $X \rightarrow Y$, where $X = \{I_1, \ldots, I_j\}$, and $Y = \{I_{j+1}, \ldots, I_k\}$ are two itemsets, and $X \cap Y = \emptyset$, we encode this rule into an individual as

j	I_1	$\ldots$	I_j	I_{j+1}	$\ldots$	I_k

where, $I_i = (A^{(i)}, GI^{(i)}), A^{(i)} \in U$, $GI^{(i)}$ is a gapped interval, $GI^{(i)} \subseteq dom(A^{(i)})$, $i = 1, 2, \ldots, k$. Therefore, a k-rule $X \rightarrow Y$ is represented by k+1 positive integers and k gapped intervals.

With this encoding approach, a rule is not uniquely represented in a chromosome, because there may be many permutations of items in either the antecedent part or the consequent part of a chromosome. For example, the following chromosomes,

2	$(1, GI_1)$	$(2, GI_2)$	$(3, GI_3)$

and

2	$(2, GI_2)$	$(1, GI_1)$	$(3, GI_3)$

stand for the same rule of $\{(1, GI_1), (2, GI_2)\} \rightarrow \{(3, GI_3)\}$.

Therefore, in order to encode a rule into a single chromosome, we further require that both antecedent attributes and consequent attributes are in an ascending order. Thus these k integers are sorted two-segmentally.

We now show that any $GI^{(i)} \subseteq dom(A^{(i)})$ can also be represented by a set of integers.

If $A^{(i)}$ is a categorical attribute, we suppose that there are n_i categories of $A^{(i)}$. All base intervals of $A^{(i)}$ then can be represented by [1, 1], [2, 2],..., and $[n_i, n_i]$, or 1, 2,..., and n_i. Note that $GI^{(i)}$ only consists of the base intervals of $A^{(i)}$. Consequently, categorical $GI^{(i)}$ can be represented by a set of integers ranged from 1 to n_i.

If $A^{(i)}$ is a quantitative attribute, as mentioned previously, it should have n_i base intervals, i.e. $[a_i, a_i + g_i]$, $[a_i + g_i, a_i + 2g_i], \ldots,$ and $[a_i + (n_i - 1)g_i, b_i]$, where $[a_i, b_i]$=$dom(A^{(i)})$, $n_i = \lfloor (b_i - a_i)/g_i \rfloor + 1$. Hence, the same proposition holds for quantitative attribute.

Suppose $A^{(i)} = A_x$. Then, gene $I_i = (A_x, GI_x)$ is represented by an integer x ranged from 1 to n, and a set of integers, written as $GI_x \subseteq \{1, 2, \ldots, n_x\}$, i=1, 2, ..., k.

We also use $I_i.attr$ to denote the attribute A_x of I_i, and $I_i.GI$ to denote the gapped interval GI_x of I_i. Similarly, for a chromosome *chr*, *chr.attr* stands for the set of all attributes $I_i.attr$, i=1, 2,..., k.

4.3. Algorithm design

Given a population of chromosomes, *pop*, and a selection probability, *sp*, function *select*2(*pop*, *sp*) filters those "bad" chromosomes according to their fitness values and the selection probabilities. It accepts a population of chromosomes and returns the new selected population, as shown below.

```
population $select2(pop, sp)$
begin
    $pop\_ret \leftarrow \emptyset$;
    for $\forall\ chr \in pop$ do
        if $(frand() *\ fitness(chr) < sp)$ then
            $pop\_ret \leftarrow pop\_ret \cup \{chr\}$;
    return $pop\_ret$;
end
```

The function, *crossover*2(*pop*, *cp*), also reproduces offspring chromosomes at a crossover probability of *cp* from population *pop*, as illustrated in Fig. 1. It is all the same as the *crossover*(*pop*, *cp*) function in Section 3.2, except the definition of items.

Given a population of chromosomes *pop* and mutation probability *mp*, function *mutate*2(*pop*, *mp*) occasionally changes chromosomes in *pop* according to the probability *mp*, and the fitness of chromosome as an additional weight.

```
population $mutate2(pop, mp)$
begin
    $pop\_ret \leftarrow \emptyset$;
    for $\forall\ chr = (I_0, \ldots, I_k) \in pop$ do
    begin
        if $(frand() *\ fitness(chr) < mp)$ then
        begin
            $attr\_set \leftarrow chr.attr$;
            $chr.I_0 \leftarrow irand(k-2)+1$;
            $i \leftarrow irand(k-1)+1$;
            $attr\_set \leftarrow attr\_set - \{I_i\}$;
            $n_{GI} \leftarrow$ number of intervals in $I_i.GI$;
            $x \leftarrow irand(n-1)+1$;
            while $x \in attr\_set$
                $x \leftarrow irand(n-1)+1$;
            $GI_x \leftarrow \emptyset$;
            for $\forall\ m \in \{1, \ldots, n_x\}$ do
            begin
        if $(frand() \leqslant n_{GI}/n_i)$ then
                $GI_x \leftarrow GI_x \cup \{m\}$;
            end
            $chr.I_i \leftarrow (x, GI_x)$;
        end
        $pop\_ret \leftarrow pop\_ret \cup \{chr\}$;
    end
    return $pop\_ret$;
end
```

Because $0 < chr.I_0 < k$ and $0 \leqslant irand(m) \leqslant m$ from the definitions, we use *irand*() with parameter $(k-2)$ to generate a random integer and assign it to $chr.I_0$ to determine the antecedent part and consequent part of a chromosome. Then the *i*th gene is chosen to change its attribute and gapped interval. The attribute is changed to *x*. The while loop guarantees that *x* is not an existing attribute of the chromosome *chr*. Note that the changed gapped interval should be a sub-domain of the new attribute. Let n_x be the number of base intervals of attribute Ax. Some of the base intervals are chosen to compose the gapped intervals GI_x. Probabilistically, the proportion of chosen intervals is the same as that of $I_i.GI$.

Given a seed chromosome *s*, we can use the following function to initialize the population *pop*[0].

```
population $initialise2(s)$
begin
    $pop[0] \leftarrow \{s\}$;
    while $sizeof(pop[0]) < popsize/2$ do
        $pop[0] \leftarrow pop[0] \cup mutate(pop[0], 1)$;
    return $pop[0]$;
end
```

Integration of the above modules is straightforward. We still use the relative confidence of an association rule as the fitness function.

```
population $EARMGA(s, sp, cp, mp)$
begin
    $i \leftarrow 0$;
    $pop[i] \leftarrow initialize2(s)$;
    while not $terminate(pop[i])$ do
    begin
        $pop\_temp \leftarrow \emptyset$;
        $pop[i+1] \leftarrow select2(pop[i], sp)$;
        $pop\_temp \leftarrow crossover2(pop[i+1], cp)$;
        $pop[i+1] \leftarrow pop[i+1] \cup$
            $mutate2(pop\_temp, mp)$;
        $i \leftarrow i+1$;
    end
    return $pop[i]$;
end
```

The time complexities of *select*2, *crossover*2, and *mutate*2 are O(n), O(n^2), and O(n) respectively, where *n* is the number of chromosomes in the population.

4.4. Generalized FP-tree

In this subsection, we present a generalized *FP*-tree based on (Han, Pei, & Yin, 2000), designed to implement the *EARMGA* algorithm.

As shown in (Han et al., 2000), the construction of *FP*-tree starts with finding the set of all frequent 1-itemsets from a given transaction database. When the database is

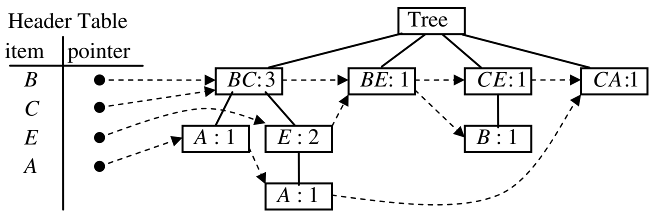

Fig. 2. The 2-*FP*-trees based on Table 1.

very dense, almost all items may be frequent and it is unnecessary to find the set of frequent 1-itemsets. On the other hand, when the database is large and sparse, *FP*-tree might be inefficient. Sometimes, we only concern those itemsets with a specific length *k*. In this section, we expand the *FP*-tree to a general *k*-*FP*-tree, where *k* is a certain non-negative integer. The construction of a *k*-*FP*-tree is demonstrated below.

Example 2. Suppose that the transaction database is given in Table 1, and the minimum support $minsupp = 0.5$. The 2-*FP*-tree is shown in Fig. 2.

After selecting an integer *k* for a given transaction database, $k \geqslant 0$, the collection of all frequent *k*-itemsets *F* and their supports are first generated by scanning the database once. Then *F* is considered as a conditional pattern base, and the patterns in *F* are sorted in support-descending order. In Example 1, we have, $k = 2$ and $F = \{BC:4, BE:4, CE:3, CA:3\}$. Let *K* be the union of all *k*-itemsets in *F*. Thus, for each item *e* in itemset *K*, there exists at least an itemset *I* in *F*, such that *e* is in *I*. From the anti-monotonic *Apriori* property, each item *e* in *K* is frequent. We create and sort the items in *K* by using the following procedure. In Example 1, we obtain, $K = \{B, C, E, A\}$.

Procedure 1. *FIMerge*(F, K)

```
//Merge all items in $F$ into $K$ according to a
certain order
begin
  let $n \leftarrow$ the number of itemsets in $F$;
  let $K \leftarrow \emptyset$;
  for $m \leftarrow 1$ to $n$ do
    for (any item e in the $m$th itemset of $F$) do
      if $e \notin K$ then append e to the end of $K$;
end.
```

We then create the root of a *k*-*FP*-tree, *Tree*, and label it as "*null*". For each itemset $I \in F$, a child of the root is created, its counter is initialized with 0, and the node is labelled BY the itemset *I*. For each transaction *T* in the database, find out the first itemset *J* in *F*, such that $J \subseteq T$. Let $S = T - J$, project *S* over *K*, sort all the resulted items by the same order as that in *K*, and obtain *R* at last. Suppose that $R = lL$, where *l* is the first item of *R* and *L* is the list of remaining items in *R*. Insert *R* to the sub-tree *J* of *k*-*FP*-tree *Tree*, by calling a procedure *FPInsert*(lL, J). We also adjust the related counters by increment and the relevant item lists by a procedure *append*(), if necessary.

In Example 1, the first transaction is $T = \{ABCD\}$. We can easily see that $J = \{BC\}$ is the first itemset in *F*, which is contained in *T*. Hence, $S = \{AD\}$. Project *S* over *K* and obtain $R = \{A\}$, where $K = \{BCEA\}$. Other transactions are similarly processed recursively. A 2-*FP*-tree is constructed finally as shown in Fig. 2.

A formal description of construction algorithm for a *k*-*FP*-tree is given below.

Algorithm 1. *k*-*FP*-tree_Constructor

```
input: $DB$: a transaction database;
  $minsupp$: the minimum support;
  $k$: a non-negative integer;
output: $Tree$: a corresponding k-FP-tree;

begin
(1)  if $k > 0$ then
     begin
       let $F \leftarrow$the collection of all frequent
       $k$-itemsets and their supports;
       sort $F$ in order of descending support;
     end

     else
       let $F \leftarrow \emptyset$;
(2)    call $FIMerge(F, K)$;
(3)    create the root, $Tree$, of a $k$-$FP$-tree;
(4)    for $\forall$ itemset $I \in F$ do
     begin
       create a child of $Tree$, labelled as $I$;
       let $I.count \leftarrow 0$;
     end
(5)    for $\forall$ transaction $T \in DB$ do
     begin
       let $J \leftarrow$ the first itemset $I \in F$ such
       that $I \subseteq T$;
       if $J \neq \emptyset$ then
       begin
         $J.count$++;
         for $\forall e \in J$ do
           if $H(e) = NULL$ then
           $append(H(e), J)$;
         let $S \leftarrow T - J$;
         sort $S$ in the same order as that of $K$;
         let $l \leftarrow$ the first item in $S$;
         let$L \leftarrow S - \{l\}$;
         call $FPInsert(lL, J)$;
         end
     end.
```

Procedure 2. *FPInsert*(lL, J)

```
//Insert itemset $lL$ into tree $J$
begin
    if $l$ is a child of $J$ then
        $l.count{+}{+}$;
    else
    begin
        create node $l$ as a child of $J$;
        let $l.count \leftarrow 1$;
        $append(H(l), l)$;
    end
    call $FPInsert(L, l)$;
end
```

In the above algorithm, $H(e)$ is the linked item list of e, which starts from the header table, and Procedure $append(H(A), B)$ adds node B to the end of the linked item list $H(A)$.

From Algorithm 1, we can see that a *k-FP*-tree contains and only contains the information of those frequent itemsets with their length equal to, or larger than, k. When k=0, we do not need to scan the database for F. Instead, we directly have $K = F = \emptyset$. When $k = 1$, *k-FP*-tree is a normal *FP*-tree. Clearly, the information contained in a *k-FP*-tree is complete in the sense that all frequent itemsets can be obtained from the *k-FP*-tree, if and only if $k < 2$.

5. Computations

We have conducted a set of experiments for evaluating the designed algorithms. For space, this section only reports on three groups of them.

5.1. Experiments on *ARMGA*

We use the *Mushroom* dataset from UCI at http://www.ics.uci.edu/~mlearn again to show the effectiveness of Algorithm *ARMGA*. Table 2 gives the parameters when we run our computations.

We run the program when *maxloop* is given from 10 to 100 stepped with 10, from 100 to 1000 stepped with 100, and from 1000 to 9000 stepped with 1000, respectively. For example, some results when *maxloop* is 10 are listed as follows:

$3485 < -86(90\%)$

$34 < -9886(100\%)$

$34 < -7886(100\%)$

$34 < -4186(100\%)$

$34 < -4286(100\%)$

$34 < -2785(100\%)$

$34 < -2685(100\%)$

$34 < -9685(100\%)$

$34 < -10185(100\%)$

$3486 < -11(100\%)$

....

Results under other *maxloop* values are similar. We also run a traditional *Apriori*-like algorithm when rule length is fixed to 3, the minimum support is set to 0%, and the minimum positive confidence is specified as 90%. The number of returned rules is 445,978. The chromosomes, returned by Algorithm *ARMGA* with their relative confidence equal to or larger than 90%, are all among those mentioned above by the *Apriori*-like algorithm. Consequently, we can see that what Algorithm *ARMGA* returns are some desired association rules with acceptable quality.

Moreover, we find that much different populations can be obtained each time we run *ARMGA*, even when the generation number grows up to 9000. The reason might be that there are too many rules with high quality according to the fitness function we defined. Therefore we need to define a more strict fitness function. One of possible approaches is to additionally consider the support measure. In fact, the support is statistically significant, and requisite in the light of traditional association rule mining area. For example, when the minimum support increases to 60%, the *Apriori*-like algorithm generates only 8 rules from the *Mushroom* dataset.

Table 2
Parameters for running Algorithm *ARMGA*

Parameter	Value	Description
sp	0.95	Selection probability
cp	0.85	Crossover probability
mp	0.01	Mutation probability
k	3	Rule length
n	118	Number of all items
popsize	100	Maximum size of population
α	0.01	Difference boundary
maxloop	10–9000	Maximum iteration number
s	(1,34,86,85)	Seed chromosome

5.2. Experiments on *EARMGA*

The *Tumour* dataset from http://lib.stat.cmu.edu/datasets/tumor is used to show the effectiveness of Algorithm *EARMGA*. The dataset contains five attributes, that is, $U = \{group, futime, number, size, recurrences\}$.

Attribute 1, *group*, is categorical. Its domain *dom*(*group*) only has two values: 1 for the category of using placebo treatment, and 2 for another category of thiotepa treatment.

Attribute 2, *futime*, is quantitative, raged from 0 to 64. We suppose that its granularity g_{futime} is 10. Then universal interval $[0, 64]$ is divided into 7 base intervals as $[0, 10]$, $[10, 20]$, $[20, 30]$, $[30, 40]$, $[40, 50]$, $[50, 60]$, and $[60, 70]$. A boundary of these intervals may belong to two base inter-

Table 3
Number of nodes at the second level for Connect-4

Support	0.1	1	10	20	30	40	50	60	70	80	90
0-*FP*-tree	129	129	129	129	129	129	129	129	129	129	129
1-*FP*-tree	125	109	73	59	46	43	38	36	31	28	21
2-*FP*-tree	5793	4294	1986	1358	894	744	633	539	420	319	108

vals. We assume that a value of boundary is in an interval only if the value is the left boundary of the interval.

Attribute 3, *number*, can be viewed as being categorical, valued as integers from 1 to 8. Granularity $g_{number} = 0$.

Attribute 4, *size*, can is taken as quantitative, ranged from 1 to 7. Granularity g_{size}=1. Again, a boundary value belongs to a base interval only if the value is the left boundary of the interval.

Attribute 5, *recurrences*, is categorical, with values of 0, 1, 2, 3, and 4.

We start with a seed chromosome $s = \{2, (1, [1, 1]), (4, [4, 7]), (5, [1, 4])\}$, and generate an initial population by the initializer *initialise*2(s). Some results when *maxloop* is 10 are listed below.

$$\{2, (1, [1, 1]), (4, [2, 2]), (3, [1, 1])\}(100.0\%)$$
$$\{2, (2, [0, 10]), (3, [1, 1]), (5, [0, 0])\}(100.0\%)$$
$$\{2, (2, [10, 20],)(5, [0, 0]), (4, [0, 1])\}(100.0\%)$$
$$\{2, (2, [40, 50]), (5, [4, 4]), (4, [0, 1])\}(100.0\%)$$

Conclusions similar to those in Section 5.1 can be drawn.

5.3. Experiments on k-FP-tree_Constructor

The Connect-4 dataset from the Irvine Machine Learning Database Repository is used to demonstrate the performance of our data structure (Blake & Merz, 1998). Connect-4 contains 67,557 records of legal positions and corresponding optimal results in the game of connect-4, with 129 items totally in the dataset and 43 items on average in transaction. Table 3 shows the number of direct children of the root for different minimum supports generated by Algorithm 1.

Based on Table 3, we can see that we would rather use 0-*FP*-tree than use 1-*FP*-tree if the minimum support is set to 1 or 0.1. For dataset Connect-4 and most other support thresholds, 1-*FP*-tree is better than 0-*FP*-tree. Because Connect-4 is a rather dense dataset, *k*-*FP*-tree is not considered here for $k > 1$.

Suppose that the total number of items is m and the total number of transactions is n for a given transaction database *DB*. Let the expected probability that an item appears in a transaction of *DB* be p. Then, the expected number of items that appear in a transaction is $m \cdot p$. The expected number of item-pairs that appear in a transaction is

$$\binom{m}{2} \cdot p^2 = \frac{m(m-1)}{2} \cdot p^2.$$

If

$$p < \frac{2}{m-1} \text{ or } m - 1 < \frac{2}{p}, \quad (1)$$

Then,

$$\frac{m-1}{2} \cdot p \cdot mp < mp, \text{ that is, } \binom{m}{2} \cdot p^2 < mp.$$

This implies that when the database *DB* is so sparse that inequality 1 holds, the number of frequent 1-itemsets is probabilistically more than the number of frequent 2-itemsets. Consequently, 2-*FP*-tree is smaller than 1-*FP*-tree.

Generally, when

$$p < \frac{k}{k-m+1}, \quad (2)$$

we have,

$$\binom{m}{k} \cdot p^k < \binom{m}{k-1} \cdot p^{k-1}.$$

Therefore, $(k-1)$-*FP*-tree is larger than *k*-*FP*-tree if condition (2) holds.

6. Summary

We have designed a genetic algorithm-based strategy and its corresponding *ARMGA*/*EARMGA* algorithm. Our approach has delivered two benefits: (1) high-performance association rule mining; (2) system automation. Computation results show that our model can be taken as an alternative for effective association rule mining.

The most important difference between our algorithm and existing mining strategies is that our approach does not require the minimum-support threshold. The experimental results of our approach encourage us to improve and apply this strategy in real-world applications.

Acknowledgements

This work was supported in part by an Australian large ARC grant (DP0667060), a China NSF major research Program (60496327), China NSF grants (90718020, 60625204), a China 973 Program (2008CB317108), an Overseas Outstanding Talent Research Program of Chinese Academy of Sciences (06S3011S01), an Overseas-Returning High-level Talent Research Program of China Hunan-Resource Ministry, the MOE Project of Key Research Institute of Humanities and Social Sciences at Universities (07JJD720044), and Guangxi NSF grants.

References

Agrawal, R., Imielinski, T., & Swami, T. (1993). Mining association rules between sets of items in large databases. In *Proceedings of ACM SIGMOD international conference on management of data* (SIGMOD'93) (pp. 207–216).

Au, W., & Chan, K. (2002). An evolutionary approach for discovering changing patterns in historical data. In *Proceedings of 2002 SPIE* (Vol. 4730, pp. 398–409).

Blake, C., & Merz, C. (1998). UCI repository of machine learning databases (<http://www.ics.uci.edu/~mlearn/MLRepository.html>). Irvine, CA: University of California, Department of Information and Computer Science.

Cheung, Y., & Fu, A. (2004). Mining frequent itemsets without support threshold: With and without item constraints. *IEEE Transactions on Knowledge and Data Engineering* (available online).

Fidelis, M., Lopes, H., & Freitas, A. (2000). Discovering comprehensible classification rules with a genetic algorithm. In *Proceedings of the 2000 congress on evolutionary computation* (pp. 805–810).

Freitas, A. (1999). A genetic algorithm for generalized rule induction. In P. K. Chawdhry, T. Furuhashi, & R. Roy (Eds.), *Advances in soft computing – Engineering design and manufacturing* (pp. 340–353). Berlin: Springer-Verlag.

Freitas, A. (2003). A survey of evolutionary algorithms for data mining and knowledge discovery. In A. Ghosh & S. Tsutsui (Eds.), *Advances in evolutionary computing* (pp. 819–846). Berlin/Heidelberg/New York: Springer-Verlag.

Han, J., Pei, J., & Yin, Y. (2000). Mining frequent patterns without candidate generation. In *Proceedings of the ACM SIGMOD international conference on management of data (SIGMOD)* (pp. 1–12).

Han, J., Wang, J., Lu, Y., & Tzvetkov, P. (2002). Mining top-*k* frequent closed patterns without minimum support. In *Proceedings of the 2002 IEEE international conference on data mining (ICDM 2002)* (pp. 211–218).

Hipp, J., & Guntzer, U. (2002). Is pushing constraints deeply into the mining algorithms really what we want? *SIGKDD Explorations, 41*, 50–55.

Pei, M., Goodman, E., & Punch, W. (1997). Pattern discovery from data using genetic algorithm. In *Proceedings of the first Pacific–Asia conference knowledge discovery and data mining* (pp. 264–276).

Piatetsky-Shapiro, G., & Steingold, S. (2000). Measuring lift quality in database marketing. *SIGKDD Explorations, 22*, 76–80.

Roddick, J. F., & Rice, S. (2001). What's interesting about cricket? – On thresholds and anticipation in discovered rules. *SIGKDD Explorations, 31*, 1–5.

Srikant, R., & Agrawal, R. (1995). Mining generalized association rules. In *Proceedings of the 21st international conference on very large databases (VLDB'95)* (pp. 407–419).

Srikant, R., & Agrawal, R. (1996). Mining quantitative association rules in large relational tables. In *Proceedings of the ACM SIGMOD international conference on management of data (SIGMOD'96)* (pp. 1–12).

Wang, K., He, Y., Cheung, D., & Chin, F. (2001). Mining confident rules without support requirement. In *Proceedings of the 10th ACM international conference on information and knowledge management (CIKM 2001)*, (pp 89–96). Atlanta.

Weiss, G., & Hirsh, H. (1998). Learning to predict rare events in event sequences. In *Proceedings of the 4th international conference on knowledge discovery and data mining* (pp. 359–363). AAAI Press.

Zhang, S., Lu, J., & Zhang, C. (2004). A fuzzy-logic-based method to acquire user threshold of minimum-support for mining association rules. *Information Sciences* (available online).

Zhang, C., & Zhang, S. (2002). *Association rules mining: Models and algorithms. Lecture notes in computer science* (Vol. 2307, p. 243). Springer-Verlag.