

Anytime Mining for Multiuser Applications

Shichao Zhang and Chengqi Zhang, *Senior Member, IEEE*

Abstract—Database systems have been designed to serve multi-users in real-world applications. There are essential differences between mono- and multi-user applications when a database is very large. Therefore, this paper presents an “anytime” framework for mining very large databases which are shared by multi-users. Anytime mining has been designed to generate approximate results such that these results can be accessed at any time while the system is autonomously mining a database.

Index Terms—Association rule, anytime mining, data mining, instance selection, multiuser application..

I. INTRODUCTION

THE ABILITY to analyze and understand massive data sets lags far behind the ability to gather and store the data [2]. Mining *approximate frequent itemsets* from a sample of a large database can reduce computation costs significantly. An example is to select a sample of a large database for estimating the support of candidates using Chernoff bounds [3], [4]. This technique is effective for *mono-user applications*, which are those that can work well under a unique precision on frequent itemsets. However, *multi-user applications* require different precisions.

In real-world applications, a database is developed to be shared by multi-users. Therefore, data mining must be developed to serve multi-user applications. For a very large database, multi-users might demand different precisions on results for different applications. For example, a short-term stock investor might demand approximate frequent itemsets quickly from a shared stock database for high profits as time is money to him/her. A long-term stock investor is more likely to wait for more accurate results than the former.

Using traditional instance-selection (sampling) techniques, one must resample a database multiple times and mine the selected instance sets for different precisions when the database is very large. To demonstrate this, let us examine existing techniques and check whether some of them can serve multi-user applications.

Example 1: Consider a very large database TD shared by five users. For the time/performance tradeoff, the five users require 0.85, 0.90, 0.92, 0.95, and 0.98 precisions on estimating frequent itemsets.

Manuscript received November 2, 2001; revised February 1, 2002. This paper was recommended by Associate Editor M. Berthold.

S. Zhang is with the Faculty of Information Technology, University of Technology, Sydney, NSW 2007, Australia, and also with the School of Mathematics and Computing, Guangxi Normal University, Guilin, China (zhangsc@its.uts.edu.au).

C. Zhang is with the Faculty of Information Technology, University of Technology, Sydney NSW 2007, Australia (chengqi@it.uts.edu.au).

Digital Object Identifier 10.1109/TSMCA.2002.804793

- 1) The first solution (traditional frequent itemset mining) is to identify accurate frequent itemsets by searching the whole database. Though this solution is able to identify accurate frequent itemsets, results for the five users might be delayed due to time-consuming discovery.
- 2) The second solution (instance-selection-based mining approach) is to identify approximate frequent itemsets by searching a sample of the database.
This instance-selection-based approach is efficient for meeting the requirements of a user when identifying approximate frequent itemsets by sampling [2]–[6]. However, for the five different precisions, we need to select five instance sets and mine them.
- 3) The third solution (the anytime mining approach presented in this paper) is to search for approximate frequent itemsets by the anytime technique.

From the above observations, we can see that mining techniques for multi-users present more challenges than traditional techniques for mono-users. Both the first solution and the second solution cannot serve multi-user applications well.

In this paper, we present an anytime algorithm (the third solution) for multi-user applications. Our research problem can be formulated as follows. Given a large database shared by multi-users, we are interested in investigating techniques for building an anytime mining framework. Basically, this includes: 1) identifying frequent itemsets for which quality is gradually improved as computation time increases and 2) supporting users' inquiries made at any time for a time/performance tradeoff.

Without loss of generality, this paper focuses on identifying frequent itemsets in databases by implementing an anytime algorithm.

There are various existing instance-selection techniques [2]–[4], [6]. This paper focuses on developing anytime mining techniques. We begin by recalling some needed concepts in Section II. In Section III, we build an anytime mining model. In Section IV, we design the anytime algorithm. In Section V, we evaluate the effectiveness of the proposed approach by experiments. Finally, we summarize our contribution in Section VI.

II. PRELIMINARIES

There has been much work done on instance-selection [2]–[4]. We have already developed techniques for identifying approximate frequent itemsets based on the central limit theorem [6]. In this section, we outline our instance-selection.

A database D can be taken as a trial. For any itemset A , it is one if the itemset A occurs in a transaction T [written as $T(A)$], or else it is zero [written as $\neg T(A)$]. Suppose the probability of A occurring in the database is p , and the probability of A not occurring is $q = 1 - p$. Hence, the database can be taken as a

Bernoulli trial, according to the definition in [1]. In particular, we can approximate the probability p of A by the central limit theorem.

For instance-selection, first we need to estimate the sample size.

Let D be a large database, T_1, T_2, \dots, T_m be the transactions in D , X be an itemset in D , $\eta > 0$ be the degree of asymptotic to frequent itemsets, and $\xi \geq 0$ be the upper probability of $P[|Ave(X_n) - p| \leq \eta]$, where $Ave(X_n)$ is the average of X occurring in n transactions in D . Suppose records in D are matched Bernoulli trials. If the size n of D is big enough to determine the approximate frequent itemsets in D , according to the central limit theorem, n must be as follows:

$$n \geq \frac{z_{\alpha}^2}{4\eta^2} \quad (1)$$

where z_{α} is a standard normal distribution function, which can be taken from [1, Appendix].

Based on the central limit theorem, generating a random instance set from a real database by indexing is a two-step approach. The first step is to generate a set X of random numbers, where $|X| = n$. The second step is to generate the random data set RD (instance set) from D by indexing. We search frequent itemsets from a sample RD (instance set) of D by identifying approximate frequent itemsets.

Let X, Y be two itemsets, $X \cap Y = \emptyset$, $supp(X) \neq 0$, $supp(Y) \neq 0$, and the *minimum support* ($minsupp$) and *minimum confidence* ($minconf$) be given by a user. $X \rightarrow Y$ is a valid rule of interest if

- 1) $X \cap Y = \emptyset$;
- 2) $supp(X \cup Y) \geq minsupp$;
- 3) $supp(X \cup Y) - supp(X)supp(Y) \geq mininterest$;
- 4) $supp(X \cup Y)/supp(X) \geq minconf$;

where $mininterest$ is the minimum interest specified by the user, and $X \cup Y$ is a frequent itemset.

For a very large database $VLDB$, let D be an instance set generated from $VLDB$, D^t be another instance set, and A be an itemset that occurs in D . Furthermore, A^t stands for A occurring in D^t , then A is a frequent itemset in $D \cup D^t$ only if the support of A is greater than or equal to $minsupp$.

Definition 1: An association rule $A \rightarrow B$ can be extracted as a valid rule in $D \cup D^t$ only if it has both support and confidence greater than or equal to $minsupp$ and $minconf$, respectively, or

$$supp(A \cup B) = \frac{f(A \cup B) + f(A^t \cup B^t)}{c(D) + c(D^t)} \geq minsupp$$

$$confidence(A \rightarrow B) = \frac{supp(A \cup B)}{supp(A)} \geq minconf$$

where $c(D)$ and $c(D^t)$ are the cardinalities of D and D^t , respectively; and $f(A)$ and $f(A^t)$ denote the number of tuples that contain itemset A in D and D^t , respectively.

III. ANYTIME MINING MODEL

This section develops a technique for supporting multi-user applications by anytime mining.

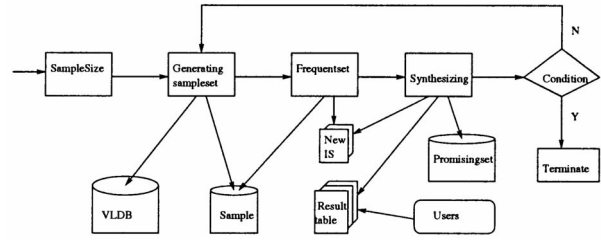


Fig. 1. Process of anytime mining.

A. A Framework of Anytime Mining

An anytime algorithm is a class of algorithms for which the quality of results improves gradually as computation time increases [7]. It is particularly useful for solving problems where the search space is very large and the quality of the results can be compromised.

For a very large database, we can also design an anytime search algorithm such that users can inquire the current results at anytime while the mining system is autonomously mining the database. Obviously, users should expect that frequent itemsets will become increasingly closer to real rules in the database with the lapse of time. Users can also make a tradeoff decision on time and accuracy of frequent itemsets for application purposes.

Anytime mining first generates an instance set D from a given very large database $VLDB$ using the sampling techniques in Section II (Step 1). Second, D is mined using an existing mining algorithm, and the frequent itemsets are put on a resultant table (Step 2). Third, another data set D^t is generated at time t from $VLDB$, where D^t is an *incremental data set* and all transactions in D^t do not occur in D . D^t is mined using the mining algorithm (Step 3). Fourth, the mined frequent itemsets in D^t , and old frequent itemsets in D , are synthesized into the resultant table (Step 4). Fifth, $D \leftarrow D \cup D^t$, where D is a *changing data set* (Step 5). Steps 2–5 described above are repeated until one termination condition is satisfied, as will be shown in Section IV. In the mining process, old results are updated once new results are generated. The procedure is depicted in Fig. 1.

In Fig. 1, “VLDB” is a very large database to be mined; “Sample” is used to save the selected instances; “NewIS” is used to save the new frequent itemsets searched by the procedure “Frequentset”; “Resulttable” is used to save the synthesized results, which is updated when new frequent itemsets are generated; “Promisingset” is used to save all promising candidate itemsets; “Users” is an interface for random users’ inquiries; “SampleSize” is a procedure that estimates the size of an instance set for a given accuracy; “Generatingsampleset” is a procedure that selects an instance set from “VLDB” and saves them into “Sample”; “Frequentset” is a procedure that generates all frequent itemsets; “Synthesizing” is a procedure that synthesizes the new frequent itemsets and old frequent itemsets; “Condition” is to check whether the termination conditions are satisfied (these conditions will be defined later); and “Terminate” ends the running.

Fig. 1 is a diagram of the process of our anytime mining. The first three functions are well-studied mining models. We focus on the function of the “Synthesizing” model in the following subsections.

B. Incremental Mining for Anytime Model

The ‘‘Synthesizing’’ model is used to identify association rules incrementally when a new instance set is generated. To implement the ‘‘Synthesizing’’ procedure, we build a two-phase approach for mining association rules from incremental data sets. In the first phase, a weighting model of mining association rules is presented. In the second phase, we advocate a competitive set approach to solve the new frequent itemset problem. Using the competitive set method, some infrequent itemsets can become frequent itemsets by competing in our incremental mining model.

In this subsection, we construct a weighted model (the first phase). The second phase is implemented in the next subsection.

From Definition 1, for a changing data set D , an incremental data set D^t , and an itemset X , the support of X in $D \cup D^t$ is as

$$\text{supp}(X) = \frac{f(X) + f(X^t)}{c(D) + c(D^t)}$$

or

$$\text{supp}(X) = a_1 * \text{supp}_1(X) + a_2 * \text{supp}_2(X)$$

where $a_1 = c(D)/(c(D) + c(D^t))$, $a_2 = c(D^t)/(c(D) + c(D^t))$, $\text{supp}_1(X) = f(X)/c(D)$, and $\text{supp}_2(X) = f(X^t)/c(D^t)$ are the supps of X in D and D^t , respectively; and $\text{supp}(X)$ is the support of X in $D \cup D^t$.

If we take a_1 and a_2 as the weights w_1 and w_2 of D and D^t , respectively, we define the support of X in $D \cup D^t$ as

$$\text{supp}_w(X) = w_1 * \text{supp}_1(X) + w_2 * \text{supp}_2(X)$$

where $\text{supp}_w(X)$ is the support of X in $D \cup D^t$. This is the weighted result. This method is called an *incremental mining model by weighting*. If $\text{supp}_w(X)$ is greater than or equal to minsupp , X is called a *frequent itemset*.

Example 2: Let $c(D) = 80$, $c(D^t) = 20$, and an itemset X is with $\text{supp}_1(X) = 0.4$ and $\text{supp}_2(X) = 0.3$ in D and D^t , respectively. Then, we can get weights below by considering the sizes of the data sets

$$w_1 = a_1 = \frac{80}{80 + 20} = 0.8,$$

$$w_2 = a_2 = \frac{20}{80 + 20} = 0.2.$$

Therefore,

$$\begin{aligned} \text{supp}_w(X) &= w_1 * \text{supp}_1(X) + w_2 * \text{supp}_2(X) \\ &= 0.8 * 0.4 + 0.2 * 0.3 = 0.38. \end{aligned}$$

In fact, according to the assumption in the above example (see Example 2) we can obtain $f(X) = 32$ and $f(X^t) = 6$. By the support-confidence framework

$$\text{supp}(X) = \frac{f(X) + f(X^t)}{c(D) + c(D^t)} = \frac{32 + 6}{80 + 20} = 0.38.$$

This means that the results in the incremental mining model by weighting are the same as those in the support-confidence

framework when the weights of data sets are assigned by considering the sizes of D and D^t . Indeed, we can also consider other factors, such as the novelty of data and both the sizes of data sets and the novelty of data, when we assign weights to the data sets.

Thus, if we consider only the sizes of data sets to assign weights, the true frequent itemsets in the changing data set can be generated. In other words, the support-confidence framework is a special case of the incremental mining model by weighting.

For D, D_1, \dots, D_n with weights w_1, w_2, \dots, w_{n+1} , we define the weighted $\text{supp}_w(X)$ for an itemset X as follows:

$$\text{supp}_w(X) = w_1 * \text{supp}(X) + \dots + w_{n+1} * \text{supp}_n(X)$$

where $\text{supp}(X)$ and $\text{supp}_1(X), \dots, \text{supp}_n(X)$ are the supports of the itemset X in D , and D_1, \dots, D_n , respectively.

C. Competitive Set Method

As has been shown, our model can reflect the change of itemsets in incremental data sets. Indeed, some very low frequency itemsets, or new itemsets, may be changed into frequent itemsets. This is called the *low-frequency itemset problem*. To deal with this problem, we now advocate a competitive set approach.

To tackle the low-frequency itemset problem, a *competitive set CS* is used to store all promising itemsets, in which each itemset in CS can become a frequent itemset by competition. We now illustrate the competitive set method using an example.

An itemset A can become frequent, as follows:

$$\begin{aligned} \text{supp}(A) < 0.3 &\rightarrow 0.15 * 0.75 + 0.64 * 0.25 = 0.2725 \\ &\rightarrow A \text{ with } \text{supp}(A) = 0.2725 \Rightarrow CS' \\ &\rightarrow 0.2725 * 0.75 + 0.64 * 0.25 = 0.364375 \\ &\rightarrow A \text{ with } \text{supp}(A) = 0.2725 \Rightarrow CS \\ &\rightarrow 0.364375 * 0.75 + 0.64 * 0.25 = 0.43328 \\ &\rightarrow 0.43328 * 0.75 + 0.64 * 0.25 = 0.48496 \\ &\rightarrow 0.48496 * 0.75 + 0.64 * 0.25 = 0.52372 \\ &\rightarrow 0.52372 * 0.75 + 0.64 * 0.25 = 0.55279 \\ &\rightarrow 0.55279 * 0.75 + 0.64 * 0.25 = 0.57459 \\ &\rightarrow 0.57459 * 0.75 + 0.64 * 0.25 = 0.590945 \\ &\rightarrow 0.590945 * 0.75 + 0.64 * 0.25 = 0.60321 \end{aligned}$$

where 0.3 is minsupp , 0.15 is $\text{minsupp}/2$ that is taken as the support of an infrequent itemset, 0.75 and 0.25 are the weights of old and incremental data sets, respectively, and 0.64 is the support of A in each incremental data set.

IV. ANYTIME ALGORITHM DESIGNING

The anytime mining technique allows for very large databases to be discovered when resources are bounded. In particular, it can work for multi-users. Though the first data set is a random instance set with $P[|Ave(X_n) - p| \leq \eta]$ for each itemset X , there is still the error problem in the neighborhood of minsupp and minconf , where $Ave(X_n)$ is the average of X occurring in n transactions of a database TD . For this reason, the anytime mining algorithm is terminated only after all the data in a very large database are processed.

Actually, the following two cases are possible:

- Case 1) the first N high-ranking frequent itemsets are supported by m instance sets;
- Case 2) the support and confidence of each such frequent itemset are almost identical (or contain very small differences) in the m instance sets.

If so, we would certainly terminate the algorithm at once and output the N frequent itemsets if we require only first the N frequent itemsets.

There are also other requirements which might be proposed by users. Therefore, in this section, we design an improved algorithm: the anytime algorithm for searching frequent itemsets in databases.

A. Conditions of Termination

Let TD be a very large database, D, D_1, D_2, \dots, D_n be $n + 1$ random instance sets generated from TD , $TD = D \cup D_1 \cup D_2 \cup \dots \cup D_n$, and each transaction of TD be contained only by one instance set. For any frequent itemset X mined by anytime mining

$$|Ave(X_i) - p| \rightarrow 0, \quad \text{when } i \rightarrow n$$

where X_i is the i th synthesized result of X , and p is the ratio of X that occurs in TD .

It is typically a time-consuming procedure to deal with all instance sets of TD when we mine. Sometimes we may need to terminate the algorithm for some applications. For example, when certain people need only approximate frequent itemsets, the algorithm can be terminated and support results obtained at any time. We now present three other conditions for terminating the anytime mining algorithm.

- 1) A requirement of the termination condition for approximate frequent itemsets at a certain time t is received from a user.
- 2) Existing N high-ranking frequent itemsets are supported by m instance sets, and the support and confidence of each such itemset differ very slightly in the m instance sets.
- 3) All the data in a given very large database are processed.

Condition 1 is used in response to outside inquiries about current results. The system is not to be terminated when this condition occurs. For example, a stock investor might require approximate results at time t_1 for interim decision making, and more accurate results at time t_2 for confirming the decision. The user gets rough results at time t_1 without the “stop” instruction. Therefore, the system is not stopped at that time.

Condition 2 automatically terminates the system when results are confirmed by enough instance sets. In this case, the remaining data in a given database are no longer processed, and the data in the database is in a well-distribution.

Let $N_0 > 1$ and *minratio* be *minimum number* and *minimum ratio*, respectively, as given by users or experts. For the database TD , assume each of the first N high-ranking frequent itemsets discovered are almost identical in all the instance sets $D, D_1, D_2, \dots, D_{N_0-1}$. This means that the number of instance sets is equal to the minimum number N_0 , and the ratio of support the

TABLE I
ER: THE SYNTHESIZED RESULTS

name of frequent itemset	support	rank
FI_1	$supp_1$	1
FI_2	$supp_2$	2
...
FI_m	$supp_m$	m

TABLE II
FR: THE FREQUENCIES OF FREQUENT ITEMSETS

name of frequent itemset	support	frequency
FI_1	$Supp_1$	f_1
FI_2	$Supp_2$	f_2
...
FI_m	$Supp_m$	f_m

first N high-ranking frequent itemsets is one (\geq *minratio*). Furthermore, the first N high-ranking frequent itemsets are confided. Hence, the system is stopped and the first N high-ranking frequent itemsets are output as the final results.

Condition 3 occurs only when Conditions 1 and 2 are not satisfied. In this case, the system takes a lot of time to discover patterns in a very large database.

B. Anytime Searching

As we have seen, the anytime algorithm is appropriate for serving multiple users. For outputting results at any time, we design two tables to save the mined results, where “ FI_i ” is the name of a frequent itemset, “ $supp_i$ ” is the synthesized support of the frequent itemset FI_i , and FI_i is ranked from large to small in the i th row by the synthesized supports of frequent itemsets.

Table I lists the current synthesized information concerning all frequent itemsets. Users may access information at time t_0 . Here “ FI_i ” is the name of a frequent itemset, “ $Supp_i$ ” is the synthesized support of the frequent itemset FI_i , and “ f_i ” is the number of instance sets that FI_i has extracted as a frequent itemset in the f_i instance sets. Meanwhile, FI_i is ranked from high to low in the i th row by the frequencies of frequent itemsets.

Table II lists the current support information concerning all frequent itemsets in the mined instance sets. Users may also be interested in accessing information at a time t_0 .

We now design the anytime search algorithm.

Algorithm 1: AnytimeSearch

begin

Input: $VLDB$: very large database;
 $minsupp, minconf, mincruc$: threshold values;

Output: X : the synthesized frequent itemset;

- 1) **let** $ER \leftarrow \emptyset; FR \leftarrow \emptyset;$
- let** $CS \leftarrow \emptyset; CS' \leftarrow \emptyset; TD \leftarrow \emptyset; i \leftarrow 0;$
- let** $Number \leftarrow 0;$
- 2) **generate** an instance set of $VLDB$;
- let** $R \leftarrow$ all frequent itemsets in D ;
- let** $ER \leftarrow R;$

```

let  $FR \leftarrow R$ , where frequent itemsets in
 $FR$  are ranked by support and the frequency
of each frequent itemset is 1;
let  $Number \leftarrow Number + 1$ ;
3) output the first instance set is
mined;
output tables  $ER$  and  $FR$ ;
4) for any itemset  $A$  in  $D$  do
  if  $supp(A) \geq mincruc$  then
    if  $A$  don't occur in any frequent
    itemset in  $R$  then
      let  $CS \leftarrow CS \cup \{A\}$ ;
5) for all instance sets from  $VLDB$  do
  begin
    if a termination instrument is re-
    ceived then
      exit;
    let  $D^t \leftarrow$  an instance set;
    let  $Number \leftarrow Number + 1$ ;
    mine the  $Number$ th instance set  $D^t$ ;
    for all frequent itemsets in  $D^t$  do
      synthesize the new and old results;
    for all synthesized frequent itemsets
    of interest do
      begin
        update  $ER$  by new results;
        update  $FR$  by new results;
        output the  $Number$  instance sets
        are mined;
        output tables  $ER$  and  $FR$ ;
        if  $i \geq N_0$  then
          if Condition 2) is satisfied
          then
            terminate the anytime mining;
          end
        end
      end
    end.

```

The algorithm *AnytimeSearch* is designed to discover very large databases incrementally. Users can access information from the current frequent itemsets at any time while the mining system is autonomously discovering a database. The initialization is carried out in Step 1. Step 2 first generates an instance set D from $VLDB$ by sampling. Second, D is discovered to obtain all the approximate frequent itemsets in the data set. The results are saved in a set R . Finally, the frequent itemsets in R are used to form two tables: ER and FR . Step 3 is to output the tables ER and FR as the results of the first instance set so as to answer any inquiries from this made at point of time. Step 4 is to set up a competition set for the first instance set D . Step 5 consists of two parts. Of course, if a termination condition is received, the system is ended. Otherwise, an instance set D^t is first processed by synthesizing. Then, the tables ER and FR are processed. During the latter procedure, tables ER and FR are first of all updated by the new results from the data set D^t and then, to answer any inquiry at this point, the tables ER and FR are output as results immediately after the $Number$ th instance set is processed.

TABLE III
THE DATASETS IN THE FIRST SET OF EXPERIMENTS

	Record number	Attribute number
Old Database	499	10
New Database 1	50	10
New Database 2	50	10
New Database 3	50	10
New Database 4	50	10

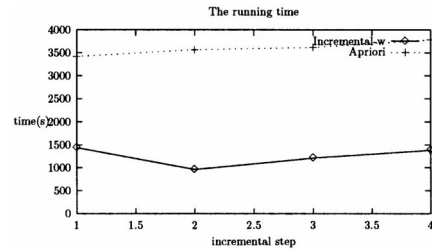


Fig. 2. Frequent itemset mining time cost comparison.

An important procedure in Step 5 is to check whether or not Condition 2 is satisfied. Obviously, the earlier Condition 2 is satisfied, the more running time is saved.

V. EXPERIMENTS

To evaluate our model, we have performed two sets of experiments. The algorithm is implemented on Dell using Java. We first check the effectiveness of the incremental mining model and then demonstrate the efficiency of the anytime mining algorithm.

A. Effectiveness of the Incremental Mining Model

To illustrate the effectiveness of the incremental mining model in our anytime framework, we choose the UCI database Breast-Cancer which contains 699 records. For mining databases incrementally, we randomly select 499 records as the changing data set and then randomly select 50 records as an incremental data set. There are four incremental data sets. They will be appended into the changing data set one by one. The parameters of the experiment databases are summarized in Table III.

We compare the running time for identifying frequent itemsets with the *apriori* [3]. We expect the *apriori* algorithm to be of low efficiency because it needs to scan for the candidate items at the union of old and new data sets. Our algorithm only needs to scan the new data set, which is highly efficient. The experimental results are shown in Fig. 2.

Let $minsupp = 0.2$, $minconf = 0.7$, $mincruc = 0.1$, and the weights of old and new rule confidence be $w_1 = 0.7$ and $w_2 = 0.3$. The general results are presented in Table IV, where “A” indicates *apriori* algorithm and “W” stands for our weight model.

B. Efficiency of Anytime Algorithm

To our knowledge, no work on anytime mining algorithms for multiple users has been reported in current literature. An incremental mining model has been tested in in the above Section V.A. We now check, in two sets of experiments, the main feature

TABLE IV
RULES IN INCREMENTAL MINING

	1st-incr	2nd-incr	3rd-incr	4th-incr
A	1062	1096	1130	1189
W	1095	1185	1204	1397
A& W	1044	1089	1114	1160
Only in A	18	7	16	29
Only in W	51	96	90	237

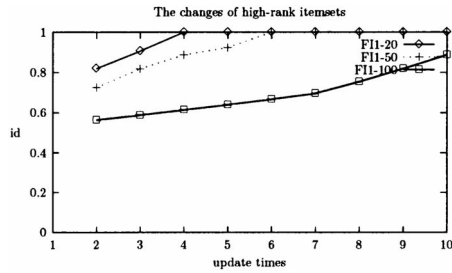


Fig. 3. Changes of a set of high-ranking itemsets in *VLDB1*.

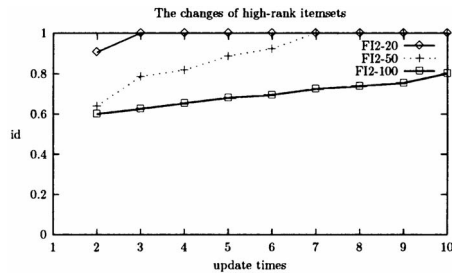


Fig. 4. Changes of a set of high-ranking itemsets in *VLDB2*.

of the anytime mining algorithm *AnytimeSearch*: the changes of sets of 20, 50, and 100 high-ranking frequent itemsets.

In our experiments, two databases, *VLDB1* and *VLDB2*, are also generated by databases from the synthetic classification data sets on the Internet. The main properties of the two data sets are as follows. There are $|R| = 1000$ attributes, and the average number of attributes per row is six and eight. The number $|r|$ of rows is approximately 10^6 . The average size I of maximal frequent sets is five. The size of each instance set is 20 000, which is randomly generated from the large databases.

Let $minsupp = 0.01$, and FI_i^t ($i = 20, 50, 100$) be the set of the first i frequent itemsets at time t . To test the changes of the sets of high-ranking frequent itemsets, we use the size of the intersection of a pair of sets of high-rank frequent itemsets to measure the identical degree of the two sets. That is, a large intersection corresponds to a high degree of sameness, whereas two sets with a small intersection are considered to be significantly different.

For the two databases, we use t_j to represent the time that the synthesized resultant table *ER* is updated in the j th times.¹

The degree of sameness between the sets $FI_i^{t_j}$ and $FI_i^{t_{j+1}}$ is defined as follows:

$$ID(FI_i^{t_j}, FI_i^{t_{j+1}}) = \frac{|FI_i^{t_j} \cap FI_i^{t_{j+1}}|}{|FI_i^{t_j} \cup FI_i^{t_{j+1}}|}$$

where “ \cap ” denotes set intersection, “ \cup ” denotes set union, and “ $|FI_i^{t_j} \cap FI_i^{t_{j+1}}|$ ” is the number of elements in set $FI_i^{t_j} \cap FI_i^{t_{j+1}}$. In ten updates of *ER*, changing of the degree of sameness is illustrated in Figs. 3 and 4.

Figs. 3 and 4 demonstrate that

- 1) the data in *VLDB1* and *VLDB2* follow a Bernoulli distribution;
- 2) the elements of the set of the first i high-rank itemsets are stably kept. In other words, the proposed approach can support inquiries from multiple users at any time.

VI. CONCLUSION

This paper has developed an anytime mining framework which can support inquiries from multiple users at any time. In this way, users can make tradeoff decisions when they choose, depending upon the required accuracy of results. Our approach is different from traditional mining techniques because it aims at attacking multi-user application problem.

As there are many large databases shared by multi-users, the mining of large databases for serving multi-user applications is a new and pressing topic in data mining. Because of the essential differences between mining tasks for multi- and mono-user applications, research into the multi-user application problem will impact on both industry and academia.

ACKNOWLEDGMENT

The authors would like to thank the three anonymous reviewers for their constructive comments on the first version of this paper.

REFERENCES

- [1] R. Durrett, *Probability: Theory and Examples*: Duxbury Press, 1996.
- [2] H. Liu and H. Motoda, *Instance Selection and Construction for Data Mining*. Norwell, MA: Kluwer, 2001.
- [3] R. Srikant and R. Agrawal, “Mining generalized association rules,” *Future Gener. Comput. Syst.*, vol. 13, pp. 161–180, 1997.
- [4] H. Toivonen, “Sampling large databases for association rules,” in *Proc. VLDB*, 1996, pp. 134–145.
- [5] S. Zhang and X. Wu, “Large scale data mining based on data partitioning,” *Appl. Artif. Intel.*, vol. 15, no. 2, pp. 129–139, 2001.
- [6] S. Zhang and C. Zhang, “Estimating itemsets of interest by sampling,” in *Proc. of IEEE International Conference on Fuzzy Systems*, Dec. 2001.
- [7] S. Zilberstein, “Using anytime algorithms in intelligent systems,” *AI Mag.*, vol. 17, no. 3, pp. 73–83, 1996.

¹The real time is too large. Furthermore, it is not easy to identify the update of *ER*. Therefore, we do not directly use it to indicate the time dimension of FI_i^t .



Shichao Zhang received the B.Sc. degree from the Guangxi Normal University in China, the M.Sc. degree from the Computing Division at the China Atomic Energy Institute, and the Ph.D. degree from the Deakin University, Australia.

He is now an Assistant President of the Guangxi Normal University. He has been a Professor in the School of Mathematical and Computing Sciences at the Guangxi Normal University since 1994. He was a Research Fellow at the National University of Singapore and at the University of New England and Deakin University, Australia. He is currently a Senior Research Fellow, Faculty of Information Technology, University of Technology, Sydney, Australia. His research interests include database classification, data analysis, and data mining. He has authored three monographs (two of them by Springer), over 20 refereed international journal articles, and over 30 refereed international conference papers. He is a member of the editorial board of *Asian Journal of Information Technology*.

Dr. Zhang has served as a PC member for four international conferences, including the 2003 IEEE International Conference on Data Mining (ICDM).



Chengqi Zhang (M'91–SM'95) received the Ph.D. degree in computer science from the University of Queensland, Australia, and the Dr.Sci. degree from Deakin University, Australia.

He is currently a Research Professor, Faculty of Information Technology, University of Technology, Sydney, Australia. His areas of interest are data mining and multi-agent systems. He has published more than 150 refereed papers, edited seven books, and published two monographs. He is a senior member of the IEEE Computer Society and Associate Editor of the editorial board for *Knowledge and Information Systems*; an *International Journal of Web Intelligence and agent systems*. He has also served as a member of the Program Committees in many international or national conferences.