Acta Cybernetica **16** (2003) 179–205.

# Mining Dynamic databases by Weighting*

Shichao Zhang† and Li Liu‡

**Abstract**

A dynamic database is a set of transactions, in which the content and the size can change over time. There is an essential difference between dynamic database mining and traditional database mining. This is because recently added transactions can be more 'interesting' than those inserted long ago in a dynamic database. This paper presents a method for mining dynamic databases. This approach uses weighting techniques to increase efficiency, enabling us to reuse frequent itemsets mined previously. This model also considers the novelty of itemsets when assigning weights. In particular, this method can find a kind of new patterns from dynamic databases, referred to *trend patterns*. To evaluate the effectiveness and efficiency of the proposed method, we implemented our approach and compare it with existing methods.

## 1 Introduction

In real-world applications, a business database is dynamic, in which (1) its content updates over time and (2) transactions are continuously being added. For example, the content and size of the transaction database of a supermarket change time by time, and different branches of Wal-Mart receive 20 million transactions a day. This generates an urgent need for efficiently mining dynamic databases.

While traditional data mining is developed for knowledge discovery in static databases, some algorithms have recently been developed for mining dynamic databases [4, 5, 6, 13]. However, there is an essential difference between dynamic database mining and traditional database mining. This is because recently added transactions can be more 'interesting' than those inserted long ago in a dynamic database. Actually, some items such as suits, toys, and some foods are with smart in market basket data. For example, "jean" and "white shirt" were often purchased in a duration from a department store, and "black trousers" and "blue T-shirt" were often purchased in another duration. The department store made different

*This research is partially supported by the Australian Research Council Discovery Grant (DP0343109) and partially supported by a large grant from the Guangxi Natural Science Funds

†Faculty of Information Technology, University of Technology, Sydney, PO Box 123, Broadway NSW 2007, Australia, and Guangxi Teachers University, Gulin, P R China. Email: zhangsc@it.uts.edu.au

‡Faculty of Information Technology, University of Technology, Sydney, PO Box 123, Broadway NSW 2007, Australia. Email: liliu@it.uts.edu.au

decisions on buying behavior according to such different purchased models. This means, some goods are very often purchased in a duration in market basket data, and they are solely purchased in another duration. These items are called *smart goods*. Apparently, most of smart items may not be frequent itemsets in a market basket data set. But they are useful to making decisions on latest buying behavior, referred to *trend pattern*.

Consequently, mining trend patterns is an important issue in mining market basket data. Indeed, since new data may represent the changing trend of customer buying patterns, we should intuitively have more confidence on the new data than on the old data. therefore, the novelty of data should be highlighted in mining models. However, mining customer buying behavior based on support-confidence framework (see [1]) can only reflect the frequency of itemsets but the trend of data. In this paper, a new method is proposed for mining association rules in dynamic databases. The proposed approach is based on the idea of weighted methods and aims at incorporating both the size of a database and the novelty of the data in the database.

The rest of this paper is organized as follows. In the next section, we first show our motivation, and then briefly recall some related work, concepts and definitions. In Section 3, a weight model of mining association rules for incremental databases is proposed. A competition is set up for tackling the problem of infrequent itemsets in Section 4. In Section 5, we show the efficiency of the proposed approach by experiments, and finally, we summarize our contributions in the last section.

# 2 Preliminaries

This section recall some previous work and concepts needed.

## 2.1 Related Work

Data mining can be used to discover useful information from data like 'when a customer buys milk, he/she also buys Bread' and 'customers like to buy Sunshine products'.

Strictly speaking, *data mining is a process of discovering valuable information from large amounts of data stored in databases, data warehouses, or other information repositories.* This valuable information can be such as patterns, associations, changes, anomalies and significant structures [19]. That is, data mining attempts to extract potentially useful knowledge from data.

Recently, mining association rules from large databases has received much attention [1, 7, 12, 15, 17]. However, most of them [1, 10, 12] presuppose that the goal pattern to be learned is stable over time. In real world, a database is often updated. Accordingly, some algorithms have recently been developed for mining dynamic databases [4, 5, 6, 13].

One possible approach to the update problem of association rules is to re-run the association rule mining algorithms [1] on the whole updated database. This approach, though simple, has some obvious disadvantages. All the computation done initially at finding the frequent itemsets prior to the update are wasted and all frequent itemsets have to be computed again from scratch. An incremental approach for learning from databases is due to [6], which uses the maintaining ideas in machine learning [13].

The most prevailing dynamic database mining model should be the FUP model proposed by Cheung *et al* [4] (The model will be detailed in Subsection 2.3). The $FUP$ model reuses information from the old frequent itemsets. That is, old frequent itemsets and promising itemsets are required to be kept. This can significantly reduce the size of the candidate set to be searched against the original large database. Like the Aporiori algorithm [1], the $FUP$ model employs the frequencies of itemsets to mine association rules. However, the $FUP$ approach only need to scan the new data set for generating all the candidates. To deal with more dynamics, the authors also proposed an extended $FUP$ algorithm, called $FUP2$ [5], for general updating operations, such as insertion, deletion and modification on databases.

However, previous models (such as the FUP model) use *retrace technique* to handle the problem that smaller itemsets become frequent itemsets during maintenance. The retrace technique is to re-mine the whole data set. Unfortunately, because the change is unpredicatable in applications, the technique may be repeatedly applied leading to poor performance.

In particular, previous models don't work well for the novelty of data. This paper will present techniques to address the above problems. Before figuring out our approach, we now present some well-known concepts for data mining and knowledge discovery used throughout this paper.

## 2.2 Basic Concepts

Let $I = \{i_1, i_2, \cdots, i_N\}$ be a set of $N$ distinct literal called *items*. $D$ is a set of variable length transactions over $I$. Each transaction contains a set of items $i_1, i_2, \cdots, i_k \in I$. A transaction has an associated unique identifier called $TID$. An *association rule* is an implication of the form $A \Rightarrow B$ (or written as $A \rightarrow B$), where $A, B \subset I$, and $A \cap B = \emptyset$. $A$ is called the *antecedent* of the rule, and $B$ is called the *consequent* of the rule.

In general, a set of items (such as the antecedent or the consequent of a rule) is called an *itemset*. The number of items in an itemset is the *length* (or the *size*) of an itemset. Itemsets of some length $k$ are referred to as a $k$-itemsets. For an itemset $A \cdot B$, if $B$ is an $m$-itemset then $B$ is called an *m-extension* of $A$.

Each itemset has an associated measure of statistical significance called *support*, denoted as *supp*. For an itemset $A \subset I$, $supp(A) = s$, if the fraction of transactions in $D$ containing $A$ equals $s$. A rule $A \rightarrow B$ has a measure of its strength called *confidence* (denoted as $conf$) defined as the ratio $supp(A \cup B)/supp(A)$.

**Definition 1. (support-confidence model)**: *If an association rule $A \rightarrow B$ has both support and confidence greater than or equal to some user specified minimum support ($minsupp$) and minimum confidence ($minconf$) thresholds respectively, i.e. for regular associations:*

$$supp(A \cup B) \geq minsupp$$

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

*then $A \rightarrow B$ can be extracted as a valid rule.*

Mining association rules can be decomposed into the following two issues.

(1) All itemsets that have support greater than or equal to the user specified minimum support are generated. That is, generating all frequent itemsets.

(2) Generate all the rules that have minimum confidence in the following naive way: For every frequent itemset $X$ and any $B \subset X$, let $A = X - B$. If the rule $A \rightarrow B$ has the minimum confidence (or $supp(X)/supp(A) \geq minconf$), then it is a valid rule.

**Example 1.** *Let $T_1 = \{A, B, D\}$, $T_2 = \{A, B, D\}$, $T_3 = \{B, C, D\}$, $T_4 = \{B, C, D\}$, and $T_5 = \{A, B\}$ be the only transactions in a database. Let the minimum support and minimum confidence be 0.6 and 0.85 respectively. Then the frequent itemsets are the following: $\{A\}$, $\{B\}$, $\{D\}$, $\{A, B\}$ and $\{B, D\}$. The valid rules are $A \rightarrow B$ and $D \rightarrow B$.*

## 2.3 The FUP Model

For comparison, we now present the $FUP$ model [4]. Let $D$ be a given database, $D^+$ the incremental data set to $D$, $A$ be an itemset that occurs in $D$, $A^+$ stands for $A$ occurring in $D^+$, Then $A$ is a frequent itemset in $D \cup D^+$ only if the support of $A$ is great than or equal to $minsupp$. We now define the $FUP$ model as follows.

**Definition 2. ($FUP$ model)**: *An association rule $A \rightarrow B$ can be extracted as a valid rule in $D \cup D^+$ only if it has both support and confidence greater than or equal to minsupp and minconf respectively. Or*

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

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

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

According to the FUP model, the update problem of association rules can be reduced to finding the new set of frequent itemsets [4]. It can be divided into the following subproblems:

(1) Which old frequent itemsets will be become small in the updated database.

(2) Which old small itemsets will be become frequent in the updated database.

(3) Tackle these itemsets: delete the association rules $A \rightarrow B$ that $A \cup B$ became small in the updated database; apply the mining algorithms into the itemsets that became frequent in the updated database.

(4) How long would a database system be processed so as to update the factors of all itemsets.

The FUP algorithm works iteratively and its framework is Apriori-like (For details of the Apriori algorithm, please see [1]). At the $k$th iteration it performs three operations as follows:

1. Scan $D^+$ for any $k$-itemsets, $A$. If the support of $A$ in $D^+$ is greater than, or equal to, *minsupp*, $A$ is put into $L'_k$ that is the set of frequent itemsets in $D^+$.
2. For any $k$-itemsets $A$ in $L'_k$, if $A$ is not in $K_k$ that is the set of frequent itemsets in $D$, the support of $A$ in $D \cup D^+$ is computed. If the support of $A$ in $D \cup D^+$ is smaller than *minsupp*, $A$ is removed from the candidate set of $D^+$.
3. A scan is conducted on $D$ to update the support of $A$ for each itemset in the candidate set of $D^+$.

# 3 Mining Strategies for Dynamic Databases

As we argued previously, the dynamic of databases is represented in two cases: (1) the content updates over time and (2) the size changes incrementally. When some transactions of a database are deleted or modified, it says that the content of the database has been updated. And this database is referred to an *updated database.* When some new transactions are inserted or appended into a database, it says that the size of the database has been changed. And this database is referred to *incremental database.* This section designs efficient strategies for mining updated databases and incremental databases.

## 3.1 Pattern Maintenance for Updated Databases

The update operation includes deletion and modification on databases. Consider transaction database

$$TD = \{\{A, B\}; \{A, C\}; \{A, B, C\}; \{B, C\}; \{A, B, D\}\}$$

where the database has several transactions, separated by a semicolon, and each transaction contains several items, separated by a comma.

The update operation on $TD$ can basically be

**Case-1** Deleting transactions from the database $TD$. For example, after deleting transaction $\{A, C\}$ from $TD$, the updated database is $TD_1$ as

$$TD_1 = \{\{A, B\}; \{A, B, C\}; \{B, C\}; \{A, B, D\}\}$$

**Case-2** Deleting attributes from a transaction. For example, after deleting $B$ from transaction $\{A, B, C\}$ in $D$, the updated database is $TD_2$ as

$$TD_2 = \{\{A, B\}; \{A, C\}; \{A, C\}; \{B, C\}; \{A, B, D\}\}$$

**Case-3** Modifying existing attributes of a transaction in the database $TD$. For example, after modifying the attribute $C$ to $D$ for the transaction $\{A, C\}$ in $TD$, the updated database is $TD_3$ as

$$TD_3 = \{\{A, B\}; \{A, B\}; \{A, B, C\}; \{B, C\}; \{A, B, D\}\}$$

**Case-4** Modifying a transaction in the database $TD$. For example, after modifying the transaction $\{A, C\}$ to $\{A, C, D\}$ for $TD$, the updated database is $TD_4$ as

$$TD_4 = \{\{A, B\}; \{A, C, D\}; \{A, B, C\}; \{B, C\}; \{A, B, D\}\}$$

Mining updated databases generates a significant challenge: the maintenance of their patterns. To capture the changes of data, for each time of updating a database, we may re-mine the updated database. This is a time-consuming procedure. In particular, when a database mined is very large and the changed content of each updating transaction is relatively small, re-mining the database is not an intelligent strategy, where an updating transaction is a set of update operations. Our strategy for updated databases is to scan the changed contents when the *information amount* of the changed contents is relatively small; otherwise, the updated database is re-mined. The information amount is defined as follows.

Let $D$ be a database with $n$ transactions and the average number of attributes per transaction be $m$. Then the information amount of $D$ is

$$Amount(D) = mn$$

Let $UO$ be an updating transaction, consisting of the $k$ update operations, $N_1$, $N_2, \cdots, N_k$, on the database $D$. As we have seen, each update operation, $N_i$, can generates two sets: $Additems(N_i)$ and $deleteitems(N_i)$. $Additems(N_i)$ is the set of records, in which items are added to the database $D$. And $deleteitems(N_i)$ is the set of records, in which items are deleted from the database $D$. For example,

- $Additems(UO_1) = \{\}$ and $deleteitems(UO_1) = \{\{A, C\}\}$ for the example in Case-1;
- $Additems(UO_2) = \{\}$ and $deleteitems(UO_2) = \{\{B\}\}$ for the example in Case-2;

- $Additems(UO_3) = \{\{D\}\}$ and $deleteitems(UO_3) = \{\{C\}\}$ for the example in Case-3; and
- $Additems(UO_4) = \{\{D\}\}$ and $deleteitems(UO_4) = \{\}$ for the example in Case-4.

For $N_i$ $(1 \leq i \leq k)$ in $UO$, the information amount of $N_i$ is the sum of $Amount(Additems(N_i))$ and $Amount(deleteitems(N_i))$. Then the information amount of $UO$ is

$$Amount(UO) = \sum_{i=1}^{k}(Amount(Additems(N_i)) + Amount(deleteitems(N_i)))$$

For the above example, we have

(1) $Amount(UO_1) = 2$ for Case-1, where $UO_1$ is the update operation in the corresponding example;

(2) $Amount(UO_2) = 1$ for Case-2, where $UO_2$ is the update operation in the corresponding example;

(3) $Amount(UO_3) = 2$ for Case-3, where $UO_3$ is the update operation in the corresponding example; and

(4) $Amount(UO_4) = 1$ for Case-4, where $UO_4$ is the update operation in the corresponding example.

For an updating transaction $UO$ on a database $D$, if $Amount(UO)/Amount(D) > \gamma$, the updated database $D$ must be mined, where $\gamma$ is a minimal relative information amount threshold. Otherwise, we only need to mine the changed contents in the updated database.

Let

$$\begin{aligned} Additems(UO) &= Additems(N_1) \cup Additems(N_2) \cup \cdots \cup Additems(N_k) \\ deleteitems(UO) &= deleteitems(N_1) \cup deleteitems(N_2) \cup \cdots \cup deleteitems(N_k) \end{aligned}$$

When mining the changed contents over the database $D$, we need to mine both $Additems(UO)$ and $deleteitems(UO)$. Let $X_{Additems}$ be the number of itemsets $X$ occurring in $Additems(UO)$ and $X_{deleteitems}$ be the number of itemsets $X$ occurring in $deleteitems(UO)$. The the number, $f(X)$, of itemsets $X$ is

$$f(X) = X_{Additems} - X_{deleteitems}$$

There may be some *hopeful itemsets* in $Additems(UO)$. Let $A$ be an itemsets in the changed contents and $|D|$ be the number of records in $D$. The relative support of $A$, $rsupp(A)$, is defined as

$$rsupp(A) = \frac{f(A)}{|D|}$$

When $rsupp(A)$ is large enough, the itemset $A$ may be a frequent itemset in the updated database. This means that we may scan the database $D$ for checking whether or not a hopeful itemset is frequent.

On the other hand, if the total information amount of the updating transaction $UO$ and $M$ updating transactions ($allCC = \{UO_1, UO_2, \cdots, UO_M\}$) is greater than $\gamma$, the database $D$ must also be re-mined, where $UO_1 \leq \gamma$, $UO_2 \leq \gamma$, $\cdots$, $UO_M \leq \gamma$ and the $M$ updating transactions have done before the updating transaction $UO$.

Including the above idea, the updating transaction $UO$ leads to that the database $D$ must be re-mined if

$$Mut(UO, allCC) = Amount(UO)/Amount(D) > \gamma \bigvee \frac{Amount(UO) + \sum_{i=1}^{M} Amount(UO_i)}{Amount(D)} > \gamma \quad (1)$$

Below we design the algorithm for updated database mining.

**Algorithm 1.** *UpdatedDBMining;*

**Input** *$D$: original database; $UO$: set of update operations; $FS$: set of frequent itemsets in $D$; $\gamma$: minimal relative information amount; $\alpha$: relative minimal support for itemsets in the changed contents;*

**Output** *$newFS$: set of frequent itemsets in the updated database;*

1. **compute** $Amount(D)$;
2. **let** $CC \leftarrow$ the changed contents;
3. **let** $allCC \leftarrow \emptyset$
4. **compute** $Amount(UO)$;
5. **if** $Mut(UO, allCC)$ **then begin**
6. **mine** the updated database and **put** frequent itemsets into $newFS$;
7. $allCC \leftarrow \emptyset$;
8. **endif**
9. **else begin**
10. $hopeset \leftarrow \emptyset$;
11. $allCC \leftarrow$ the updating transaction $UO$;
12. **mine** the changed contents $CC$;
13. $Cset \leftarrow$ the set of items in $CC$;
14. **for** any $A$ in $Cset$ **do**
15. **if** $f(A) \geq \alpha$ **then**
16. $hopeset \leftarrow hopeset \cup \{A\}$;
17. $Candidate \leftarrow$ the set of itemsets in $FS$, in which each itemset contains at least an item in $Cset$;

18. **scan** the updated database for *hopeset* and *Candidate*;
19. **generate** $newFS$ by $FS$, *hopeset* and *Candidate*;
20. **end else;**
21. **output** $newFS$;
22. **end procedure;**

The algorithm *UpdatedDBMining* generates frequent itemsets in updated databases. The initialization is carried out in steps 1-4. Step 5 checks whether $Mut(UO, allCC)$ is true or not. When $Mut(UO, allCC)$ is true, the updated database must be mined in step 6 and the set $allCC$ is emptied in step 7. Otherwise, we only need mine the changed contents in steps 9-20. The set *hopeset* is used to save all hopeful itemsets in the changed contents. Step 11 puts the updating transaction $UO$ into $allCC$. Step 12 mines the changed contents. Steps 14-16 generate the set of hopeful itemsets. Step 17 generates the set of itemsets in $FS$, in which their supports have been changed. Step 18 takes one scan on the updated database for tackling both *hopeset* and *Candidate*, so as to generate all frequent itemsets in the updated database in step 19. Step 21 outputs all frequent itemsets in the updated database.

We now illustrate the use of this procedure by an example as follows.

Consider the above database $TD$. Let $minsupp = 0.4$. Then $Amount(TD) = 12$ and the frequent itemsets in $TD$ are

$$A, 0.8; B, 0.8; C, 0.6; AB, 0.6; AC, 0.4; BC, 0.4$$

where there are 6 frequent itemsets, separated by a semicolon, and each frequent itemset contains 2 items, its name and frequency, separated by a comma.

Let $\gamma = 0.2$, $\alpha = 0.25$ and the first updating transaction is $UO_1$, in which the transaction $\{A, C\}$ is deleted from $TD$. Then the updated database is $UD_1$ as

$$UD_1 = \{\{A, B\}; \{A, B, C\}; \{B, C\}; \{A, B, D\}\}$$

And

$$\begin{aligned} Additems(UO_1) &= \emptyset \\ deleteitems(UO_1) &= \{\{A, C\}\} \end{aligned}$$

Because $UO_1$ is the first updating transaction, $allCC = \emptyset$. For $UO_1$, $Amount(UO_1) = 2$,

$$Amount(UO_1)/Amount(TD) = 2/12$$

and $Mut(UO_1, allCC)$ is not true. Consequently, we only need to mine the changed contents $\{A, C\}$ and the itemsets are $A$, $C$ and $AC$. By the definition of the relative

support of itemsets,

$$\begin{aligned} rsupp(A) &= \frac{f(A)}{|TD|} = \frac{1}{5} = 0.2 < \alpha \\ rsupp(C) &= \frac{f(C)}{|TD|} = \frac{1}{5} = 0.2 < \alpha \\ rsupp(AC) &= \frac{f(AC)}{|TD|} = \frac{1}{5} = 0.2 < \alpha \end{aligned}$$

Therefore,

$$\begin{aligned} hopeset &= \emptyset \\ allCC &= \{UO_1\} \end{aligned}$$

For the set of frequent itemsets in $D$, we have

$$Candidate = \{A, C, AB, AC, BC\}$$

By scanning the updated database for $Candidate$, we have

$$newFS = \{A, 0.6; B, 0.8; C, 0.4; AB, 0.6; BC, 0.4\}$$

Note that the size of the updated database can approximately be equal to the size of $D$ when $D$ is relatively large. Accordingly, the above example takes 5 as the size of the updated database, aiming at showing how to deal with the frequent itemsets using the changed contents.

Now, let the second updating transaction is $UO_2$, in which the transaction $\{B, C\}$ in $UD_1$ is modified as $\{A, B, C\}$. Then the updated database is $UD_2$ as

$$UD_1 = \{\{A, B\}; \{A, B, C\}; \{A, B, C\}; \{A, B, D\}\}$$

And

$$\begin{aligned} Additems(UO_2) &= \{\{A\}\} \\ deleteitems(UO_2) &= \emptyset \end{aligned}$$

For $UO_2$, $Amount(UO_2) = 1$ and

$$Amount(UO_2)/Amount(TD) = 1/12$$

Because $allCC = \{UO_1\}$, $Mut(UO_2, allCC)$ is true. Consequently, we need to mine the updated database $UD_2$ and the itemsets are as follows

$$newFS = \{A, 1; B, 1; C, 0.4; AB, 0.75; BC, 0.5; AC, 0.5; ABC, 0.5\}$$

The above examples have shown our strategy for effectively maintaining frequent itemsets in updated databases. We will focus on identifying trend patterns from incremental databases in the following sections.

## 3.2 Pattern Maintenance for Incremental Databases

The incremental operation includes insertion and appending on databases. Consider transaction database

$$D = \{\{F, H, I, J\}; \{E, H, J\}; \{E, F, H\}; \{E, I\}\}$$

where the database has several transactions, separated by a semicolon, and each transaction contains several items, separated by a comma.

The incremental operation on a database $TD$ can be

(1) inserting transactions into the database $TD$. For example, after inserting two transactions $\{A, C\}$ and $\{A, B, C\}$ into $TD$ before the transaction $\{E, H, J\}$, the incremental database is $TD_1$ as

$$TD_1 = \{\{F, H, I, J\}; \{A, C\}; \{A, B, C\}; \{E, H, J\}; \{E, F, H\}; \{E, I\}\}$$

(2) Appending transactions into the database $TD$. For example, after appending transactions $\{B, C\}$ and $\{A, B, D\}$ to $TD$, the incremental database is $TD_2$ as

$$TD_2 = \{\{F, H, I, J\}; \{E, H, J\}; \{E, F, H\}; \{E, I\}; \{B, C\}; \{A, B, D\}\}$$

¿From the above observations, both the inserting and appending operations do not change the original contents in the database $TD$. Therefore, we can take an incremental operation transaction as the union of the database $TD$ and the incremental dataset $D^+$, where the dataset $D^+$ is a set of transactions that are added to $TD$ by the incremental operation transaction.

**Example 2.** *Let $TD$ be a set of a transaction database with 10 transactions in Table 1 which is obtained from a grocery store, where $A$ = bread, $B$ = coffee, $C$ = tea, $D$ = sugar, $E$ = beer, $F$ = butter and $h$ = biscuit. Assume that $D^+$ is a set of transactions in Table 2, which are new sales records in the grocery store, where $G$ = choclate.*

The databases $TD$ and $D^+$ are sets of data that represents the customer behaviors during two terms. And $D^+$ illustrates the latest customer behavior, whereas $TD$ presents the old customer behavior. A frequent itemsets in $D^+$ is referred to *trend pattern*. Trend patterns are very important in marketing because they are useful in the decision-making of merchandize buying. For example, $H$ is a trend pattern, which is frequently purchased in the new duration.

However, by using the support-confidence framework, $H$ is not a frequent itemset in $TD \cup D^+$ when $minsupp = 0.4$. Therefore, trend pattern discovery has become a key issue in incremental database mining. On the other hand, to capture the novelty of data, for each incremental operation transaction, we may also re-mine the incremental database. In particular, when an original database mined is very large and the dataset generated by an incremental operation transaction is relatively small, re-mining the incremental database is time-consuming. Like the

Table 1: Transaction databases in $TD$

| Transaction ID | Items |
|---|---|
| $T_1$ | A, B, D, H |
| $T_2$ | A, B, C, D |
| $T_3$ | B, D, H |
| $T_4$ | B, C, D |
| $T_5$ | A, C, E |
| $T_6$ | B, D, F |
| $T_7$ | A, F |
| $T_8$ | C, F |
| $T_9$ | B, C, F |
| $T_{10}$ | A, B, C, D, F |

Table 2: Transaction databases in $D^+$

| Transaction ID | Items |
|---|---|
| $N_1$ | A, B, H |
| $N_2$ | B, C, G, H |
| $N_3$ | B, H |

updated database mining, our strategy for mining incremental databases is to scan the incremental dataset when the information amount of the incremental dataset is relatively small; otherwise, the incremental database is re-mined.

For an incremental dataset $D^+$ added to a database $D$, if $Amount(D^+)/Amount(D) > \beta$, the incremental database $D \cup D^+$ must be mined, where $\beta$ is a minimal relative information amount threshold. Otherwise, we only need to mine the incremental dataset and synthesize the patterns in $D^+$ and $D$ by weighting (see Sections 4 and 5).

If the total information amount of the incremental dataset $D^+$ and $M$ incremental datasets ($allaet = \{D_1^+, D_2^+, \cdots, D_M^+\}$) is greater than $\beta$, the database $D$ must also be re-mined, where $D_1^+ \leq \beta$, $D_2^+ \leq \beta$, $\cdots$, $D_M^+ \leq \beta$ and the $M$ datasets are added to $D$ before the incremental dataset $D^+$ is.

Intuitively, the constraint $Amount(D^+)/Amount(D) > \beta$ indicates that the incremental dataset $D^+$ contains a information amount large enough to drives the mining of the incremental database. However, $D^+$ may only contain few records with much information. For example, let $D^+$ only contain a records with 200 distinct items in Example 2 and these items are also different from items in $D$. Certainly, the constraint $Amount(D^+)/Amount(D) > \beta$ holds. This leads to the mining of the incremental database. By using the support-confidence framework, there are no frequent itemsets in $D^+$ because the frequency of each item in $D^+$ is 1. Accordingly, we must take into account the size, $|D^+|$, of $D^+$ in our mining

strategy. In this paper, the constraint is constructed as

$$constraint(D^+, D) = \frac{|D^+| * Amount(D^+)}{|D| * Amount(D)}$$

Including the above idea, the incremental dataset $D^+$ leads to that the database $D$ must be re-mined if

$$\begin{aligned} Mid(D^+, allset) \quad = \quad & constraint(D^+, D) > \beta \bigvee \\ & constraint(D^+, D) + \sum_{i=1}^{M} constraint(D_i^+, D) > \beta \end{aligned} \quad (2)$$

Below we design the algorithm for incremental database mining.

**Algorithm 2.** *IncrementalDBMining;*

**Input** *D: original database; $D^+$: incremental dataset; FS: set of frequent itemsets in D; $\beta$: minimal relative information amount; minsupp: minimal support;*

**Output** *weightedFS: set of frequent itemsets by weighting;*

1. **compute** $Amount(D)$;
2. **let** $allset \leftarrow \emptyset$
3. **compute** $Amount(D^+)$;
4. **if** $Mid(D^+, allset)$ **then begin**
5. **mine** the database $D \cup D^+$ and **put** frequent itemsets into *weightedFS*;
6. $allset \leftarrow \emptyset$;
7. **endif**
8. **else begin**
9. $allset \leftarrow$ the incremental dataset $D^+$;
10. **mine** the incremental dataset $D^+$;
11. $Pset \leftarrow$ the set of frequent itemsets in $D^+$;
12. **weight** the support and confidence of $A$ in $D$ and $D^+$
13. **if** $supp(A) \geq minsupp$ **then**
14. $weightedFS \leftarrow A$;
15. **end else;**
16. **output** *weightedFS*;
17. **end procedure;**

The algorithm *IncrementalDBMining* generates frequent itemsets in incremental databases. The initialization is carried out in steps 1-3. Step 4 checks

whether $Mid(D^+, allset)$ is true or not. When $Mid(D^+, allset)$ is true, the incremental database must be mined in step 5 and the set $allset$ is emptied in step 6. Otherwise, we only need mine the incremental dataset $D^+$ in steps 9-15. Step 9 puts the incremental dataset $D^+$ into $allset$ when $Mid(D^+, allset)$ is not true. Step 10 mines the incremental dataset $D^+$. Steps 11-14 generate the set of frequent itemsets by weighting. Step 16 outputs all frequent itemsets in the incremental database.

The algorithm $IncrementalDBMining$ includes a weighting procedure which is used to identify trend patterns. We will present the weighting technique in the following Sections.

# 4 Weight Method

Let $D$ be a given database, $D^+$ the incremental dataset to $D$, $A$ be an itemset that occurs in $D$, $A^+$ stands for $A$ occurring in $D^+$. The support of $A$ in the incremental database $D_1 = D \cup D^+$ is as follows

$$supp(A) = \frac{t(A)}{c(D) + c(D^+)} + \frac{t(A^+)}{c(D) + c(D^+)} \tag{3}$$

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

Let $supp_1(A) = t(A)/c(D)$ and $supp_2(A) = t(A^+)/c(D^+)$ stand for the supports of $A$ in $D$ and $D^+$, respectively. Then the equation (3) can be represented as

$$supp(A) = \frac{c(D)}{c(D) + c(D^+)} supp_1(A) + \frac{c(D^+)}{c(D) + c(D^+)} supp_2(A) \tag{4}$$

Let

$$k_1 = \frac{c(D)}{c(D) + c(D^+)}$$

$$k_2 = \frac{c(D^+)}{c(D) + c(D^+)}$$

where $k_1$ and $k_2$ are the ratios of $D$ and $D^+$ in the incremental database $D_1$, respectively. And the equation (4) can be represented as

$$supp(A) = k_1 * supp_1(A) + k_2 * supp_2(A) \tag{5}$$

For the equation (5), we can take $k_1$ and $k_2$ as the weights of $D$ and $D^+$ in the incremental database $D_1$. This means, if a dataset has a larger number of transactions, the weight of the dataset is higher. And if a dataset has few transactions, the dataset is assigned a lower weight. Therefore, traditional data mining

techniques, such as the support-confidence framework, can really be regarded as trivial weighting methods.

Consider the database $D$ and the incremental dataset $D^+$ in Example 2. When $minsupp = 0.4$, the frequent itemsets in $D$ and $D^+$ are listed in Tables 3 and 4, respectively.

Table 3: Frequent itemsets in $D$

| Item | Number of Transactions | Support $p(X)$ | Item | Number of Transactions | Support $p(X)$ |
|---|---|---|---|---|---|
| A | 5 | 0.5 | B | 7 | 0.7 |
| C | 6 | 0.6 | D | 6 | 0.6 |
| F | 5 | 0.5 | BC | 4 | 0.4 |
| BD | 5 | 0.5 | | | |

Table 4: Frequent itemsets in $D^+$

| Item | Number of Transactions | Support $p(X)$ |
|---|---|---|
| B | 3 | 1 |
| H | 3 | 1 |
| BH | 3 | 1 |

After the transactions in $D^+$ are added to $D$ to form the incremental database $D_1 = D \cup D^+$, the frequent itemsets in $D_1$ are listed in Table 5.

Table 5: Frequent itemsets in $D_1$

| Item | Number of Transactions | Support $p(X)$ |
|---|---|---|
| A | 6 | 0.4615 |
| B | 10 | 0.769 |
| C | 7 | 0.5385 |
| D | 6 | 0.4615 |

From Tables 3, 4 and 5, the desirable patterns $H$ and $BH$ are not frequent itemsets in the incremental database $D_1$. To identify trend patterns such as $H$ and $BH$, the novelty of data must be emphasized. In this paper, we propose to assign the incremental dataset $D^+$ a higher weight for stressing the novelty of data. For example, for the database $D$ and the incremental dataset $D^+$, we have

$$k_1 = \frac{c(D)}{c(D)+c(D^+)} = \frac{10}{13} = 0.769$$
$$k_2 = \frac{c(D^+)}{c(D)+c(D^+)} = \frac{3}{13} = 0.231$$

Taking into account the above idea, we assign $D$ a weight $w_1 = 0.66$ and $D^+$ a weight $w_2 = 0.34$. And the support of an itemset $X$ in $D_1$ is as follows

$$supp(X) = w_1 * supp_1(X) + w_2 * supp_2(X) \quad (6)$$

Hence, for the itemsets $B$, $H$ and $BH$, we have

$$\begin{aligned} supp(B) &= 0.66 * supp_1(B) + 0.34 * supp_2(B) \\ &= 0.66 * 0.7 + 0.34 * 1 = 0.802 \\ supp(H) &= 0.66 * supp_1(H) + 0.34 * supp_2(H) \\ &= 0.66 * 0.2 + 0.34 * 1 = 0.472 \\ supp(BH) &= 0.66 * supp_1(BH) + 0.34 * supp_2(BH) \\ &= 0.66 * 0.2 + 0.34 * 1 = 0.472 \end{aligned}$$

This means that both $H$ and $BH$ are frequent itemsets in $D_1$ according to the equation (6). And all frequent itemsets in $D_1$ are listed in Table 6.

Table 6: Frequent itemsets in $D_1$

| Item | Number of Transactions | Weighted Support $supp(X)$ |
|---|---|---|
| A | 6 | 0.445 |
| B | 10 | 0.802 |
| C | 7 | 0.51 |
| H | 5 | 0.472 |
| BH | 5 | 0.472 |

Comparison with Table 6, the supports of itemsets $A$, $C$ and $D$ are decreased in Table 7 because they are not frequent itemsets in the incremental dataset $D^+$. In particular, itemset $D$ is not a frequent itemset in $D_1$ because

$$\begin{aligned} supp(D) &= 0.66 * supp_1(B) + 0.34 * supp_2(B) \\ &= 0.66 * 0.6 + 0.34 * 0 = 0.396 \end{aligned}$$

On the other hand, the supports of itemsets $B$, $B$ and $BH$ are increased in Table 7 because they are strongly supported in the incremental dataset $D^+$.

The above results have shown the following fact. In the weighting model, some infrequent itemsets (for example $H$ and $BH$) can be interested, whereas some frequent itemsets (for example $D$) can be uninterested.

We now define the weighting model for maintaining association rules in incremental databases.

**Definition 3. (Weighting model)**: *An association rule $X \rightarrow Y$ can be extracted as a valid rule in $D \cup D^+$ only if it has both support and confidence greater than or equal to minsupp and minconf respectively. Or*

$$supp_w(X \cup Y) = w_1 * supp_1(X \cup Y) + w_2 * supp_2(X \cup Y) \geq minsupp \quad (7)$$

$$conf_w(X \rightarrow Y) = \frac{supp_w(X \cup Y)}{supp_w(X)} \geq minconf \quad (8)$$

The confidence of the rule $X \rightarrow Y$ can be directly weighted as follows

$$conf_w(X \rightarrow Y) = w_1 * conf_1(X \rightarrow Y) + w_2 * conf_2(X \rightarrow Y) \geq minconf \quad (9)$$

Generally, for $D$, $D_1$, $\cdots$, $D_n$ with weights $w_1, w_2, \cdots, w_{n+1}$, we define the weighted support, $supp_w(X)$, of itemset $X$ as follows.

$$supp_w(X) = w_1 * supp(X) + w_2 * supp_1(XY) + \cdots + w_{n+1} * supp_n(X) \quad (10)$$

where, $supp(X)$, $supp_1(X)$, $\cdots$, $supp_n(X)$ are the the supports of the itemset $X$ in $D$, $D_1$, $\cdots$, $D_n$ respectively.

Let $X \rightarrow Y$ be an association rule in $D$, we define the weighted support $supp_w(X \cup Y)$ and confidence $conf_w(X \rightarrow Y)$ for $X \rightarrow Y$ as follows

$$\begin{aligned} supp_w(X \cup Y) = {} & w_1 * supp(X \cup Y) + w_2 * supp_1(X \cup Y) + \cdots \\ & \cdots + w_{n+1} * supp_n(X \cup Y) \end{aligned} \quad (11)$$

$$\begin{aligned} conf_w(X \rightarrow Y) = {} & w_1 * conf(X \rightarrow Y) + w_2 * conf_1(X \rightarrow Y) + \cdots \\ & \cdots + w_{n+1} * conf_n(X \rightarrow Y) \end{aligned} \quad (12)$$

where, $supp(X \cup Y)$, $supp_1(X \cup Y)$, $\cdots$, $supp_n(X \cup Y)$ are the the supports of the rule $X \rightarrow Y$ in $D$, $D_1$, $\cdots$, $D_n$ respectively; $conf(X \rightarrow Y)$, $conf_1(X \rightarrow Y)$, $\cdots$, $conf_n(X \rightarrow Y)$ are the confidences of the rule $X \rightarrow Y$ in $D$, $D_1$, $\cdots$, $D_n$ respectively.

We now present the algorithm for weighting the support and confidence of association rules.

Let $D$ be the given database, $D^+$ the incremental data set, $supp$ and $conf$ the support and confidence functions of rules in $D$, $supp^+$ and $conf^+$ the support and confidence functions of rules in $D^+$, $minsupp$, $minconf$, $mincruc$: threshold values given by user, where $mincruc$ ($< Min\{minsupp, minconf\}$) is the crucial value that an infrequent itemset can become frequent itemset in a system.

**Procedure 1.** *Weighting*

**Input:** $D^+$: *database; minsupp, minconf, mincruc: threshold values; R: rule set;* $CS$, $CS'$: *sets of itemsets;*

**Output:** $X \rightarrow Y$: *rule;* $CS$, $CS'$: *sets of itemsets;*

(1) **input** $w_1 \leftarrow$ the weight of $D$;
**input** $w_2 \leftarrow$ the weight of $D^+$;
**let** $RR \leftarrow R$; $R \leftarrow \emptyset$; $temp \leftarrow \emptyset$;
**let** $Itemset \leftarrow$ all itemsets in $D^+$;
**let** $CS_{D^+} \leftarrow$ all frequent itemsets in $D^+$;
**let** $i \leftarrow i + 1$;

(2) **for** any $X \rightarrow Y \in RR$ **do**
**begin**
**let** $supp(X \cup Y) \leftarrow w_1 * supp(X \cup Y) + w_2 * supp(X^+ \cup Y^+)$;
**let** $conf(X \rightarrow Y) \leftarrow w_1 * conf(X \rightarrow Y) + w_2 * conf^+(X \rightarrow Y)$;
**if** $supp \geq minsupp$ and $conf \geq minconf$ **then**
**begin**
**let** $R \leftarrow$ rule $X \rightarrow Y$;
**output** $X \rightarrow Y$ as a valid rule of $i$th mining;
**end**;
**else let** $temp \leftarrow temp \cup \{X, X \cup Y\}$;
**end**;

(3) **for** any $B \in CS$ **do**
**begin**
**let** $supp(B) \leftarrow w_1 * supp(B) + w_2 * supp(B^+)$;
**if** $supp(B) \geq minsupp$ **then**
**for** any $A \subset B$ **do**
**begin**
**let** $supp(A) \leftarrow w_1 * supp(A) + w_2 * supp(A^+)$;
**let** $conf(A \rightarrow (B - A)) \leftarrow supp(B)/supp(A)$;
**if** $conf(A \rightarrow (B - A)) \geq minconf$ **then**
**begin**
**let** $R \Leftarrow$ rule $A \rightarrow (B - A)$;
**output** $A \rightarrow (B - A)$ as a valid rule of $i$th mining;
**end**;
**else let** $temp \leftarrow temp \cup \{B, A\}$;
**end**
**end**;

(4) **call** competing;

(5) **return**;

The procedure *Weighting* generates association rules that are weighted. Here the initialization is done in Step (1). Step (2) performs the weighting operations on

rules in $RR$, where $RR$ is the set of valid rules in the last maintenance. In this Step, all valid rules are appended into $R$ and, the itemsets of all invalid rules weighted is temporarily stored in *temp*. Step (3) extracts all rules from competitive set $CS$ and all invalid itemsets weighted in $CS$ is temporarily stored in *temp*. (Note that any itemset in $CS'$ can generally become as a hopeful itemset and may be appended into $CS$ by competition. However, it cannot become a frequent itemset. In other words, $CS'$ can be ignored when rules are mined.) Step (4) calls procedure *competing* to tackle the competing itemsets for $CS$ and $CS'$, which will be described in next section.

# 5 Competitive Set Method

As has been shown, the proposed weighted model is efficient to mine trend patterns in incremental databases. To capture the novelty, some infrequent itemsets or new itemsets may be changed into frequent itemsets. We refer to this as *the problem of infrequent itemsets*.

To deal with this problem, we use a competitive model to deal with this problem so as to avoid retracing the whole data set. A *competitive set* $CS$ is used to store all hopeful itemsets, which each itemset in $CS$ can become frequent itemset by *competition*. We now define some operations on $CS$.

Let $D$ be given database, $D^+$ the incremental data set to $D$, $A$ be an itemset, $supp(A)$ the support of $A$ in $D$, $supp(A^+)$ the relative support of $A$ in $D^+$. Firstly, all hopeful itemsets in $D$ is appended into $CS$, which are defined in Theorem 2.

Secondly, an itemset may become invalid after each mining is done. Such an itemset is appended into $CS$ if its weighted support $\geq mincruc$.

Thirdly, some frequent itemsets in $D^+$ are appended into $CS$ after each mining if their weighted supports $\geq mincruc$. These itemsets are neither in the set of frequent itemsets, nor in $CS$. But their supports are pretty high in $D^+$. This means that their supports in $D$ are unknown. For unknown itemsets, a compromise proposal is reasonable. So we can regard their supports in $D$ as $mincruc/2$. For any such itemset $X$, $supp_w(X) = w_1 * mincruc/2 + w_2 * supp(X^+)$ according to Weight model. And if $supp_w(X) \geq mincruc$, itemset $X$ is appended into $CS$. In other words, if

$$supp(A^+) \geq \frac{mincruc(2 - w_1)}{2w_2}$$

in $D^+$, itemset $X$ is appended into $CS$; else itemset $X$ is appended into $CS'$ if its weighted support $mincruc/2$, which $CS'$ is an extra competitive set. $CS'$ is used to record another kind of hopeful itemsets. The operations on $CS'$ are similar to those on $CS$. The main use of $CS'$ is to generate a kind of itemsets with middle supports in $D^+$. For example, let $mincruc = 0.3$ and $minsupp = 0.6$. Assume the support of an itemset $A$ be less than 0.3 in a given database $D$, and the supports of $A$ in incremental data sets: $D_1$, $D_2$, $\cdots$ $D_9$ be all 0.64. Because the support of $A$ is less than 0.3 in $D$, $A$ is not kept in system. Let $w_1 = 0.75$ be the weight of the old database and $w_2 = 0.25$ the weight of the new incremental data set.

According to the operations on $CS$, $supp_w(A) = w_1 * mincruc/2 + w_2 * supp(A^+) = 0.75 * 0.15 + 0.25 * 0.64 = 0.2725$. This means that itemset $A$ cannot be appended into $CS$. But the support is greater than $mincruc/2 = 0.15$. From the novelty of data, it can be generated as a frequent itemsets if there are enough incremental data sets. Accordingly, we use $CS'$ to capture this feature of new data. This kind of itemsets such as $A$ can become frequent as follows.

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

Fourthly, some itemsets in $CS'$ are appended into $CS$ after each mining if their weighted supports $\geq mincruc$

Finally, some itemsets are deleted from $CS$ after each mining of association rules is done. By the weighted model, for any $A \in CS$, $supp_w(A) = w_1 * supp(A) + w_2 * supp(A^+)$. If $supp_w(A) < mincruc$, $A$ is deleted from $CS$; else $A$ is kept in $CS$ with new support $supp_w(A)$.

We now design the algorithm for pattern competition.

**Procedure 2.** *Competing*

**Input:** *mincruc: threshold values; temp, Itemset, $CS_{D+}$, $CS'$: sets of itemsets; $w_1$, $w_2$: weights;*

**Output:** *$CS$, $CS'$: competitive sets;*

(1) **let** *temp*1 $\leftarrow \emptyset$; *temp*2 $\leftarrow \emptyset$;

(2) **for** $A \in$ *temp* **do**
  **if** $suupp(A) \geq mincruc$ **then**
    **let** *temp*1 $\leftarrow A$;
  **else if** $suupp(A) \geq mincruc/2$ **then**
    **let** *temp*2 $\leftarrow A$;

(3) **for** $A \in CS'$ **do**
  **begin**
    **let** $supp(A) \leftarrow w_1 * supp(A) + w_2 * supp(A^+)$;

**if** $suupp(A) \geq mincruc$ **then**
**let** $temp1 \leftarrow A$;
**else if** $suupp(A) \geq mincruc/2$ **then**
**let** $temp2 \leftarrow A$;
**end**

(4) **for** $A \in CS_{D^+}$ **do**
**begin**
**let** $supp(A) \leftarrow w_1 * mincruc/2 + w_2 * supp(A^+)$;
**if** $suupp(A) \geq mincruc$ **then**
**let** $temp1 \leftarrow A$;
**else if** $suupp(A) \geq mincruc/2$ **then**
**let** $temp2 \leftarrow A$;
**end**

(5) **let** $CS \leftarrow temp1$; **let** $CS' \leftarrow temp2$;

(6) **return**;

The procedure *Competing* generates a competitive set for infrequent itemsets. Here the initialization is done in Step (1). Step (2) handles all itemsets in *temp*. And all itemsets with supports in interval $[mincruc, minsupp)$ is appended into $CS$ and the itemsets with supports in interval $[mincruc/2, mincruc)$ is appended into $CS'$. Step (3) and (4) are as similar as Step (2) to deal in the itemsets in $CS'$ and $CS_{D^+}$, respectively.

# 6 Experiments

To evaluate the proposed approach, we have done some experiments using synthetic databases in the Internet. Our experiments shown, this model is efficient and promising. For simplicity, we choose the UCI database BreastCancer to illustrate the effectiveness and efficiency, which contains 699 records. For maintenance, the set of the first 499 records is taken as the initial data set. And the set of each next 50 records is viewed as an incremental new data set. There are four incremental new data sets. And they will be appended into the database one by one. It needs to maintain the association rules once a new data set is appended into. The parameters of experiment databases is summarized as follows.

Table 7: The Experiment Databases

| | 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 |

### Comparing Running Time of Algorithms

We compare the large item set mining time with the Apriori and $FUP$. Undoubtedly the Apriori will spent the most time cost because it need scan for the candidate items in the old plus new database. The FUP model gets a good improvement. It scan the candidate items in the new database, it need scan in the old database only when the item in old large item set but not in new item set, or in new item set but not in old item set. But our algorithm only need to scan in the new database, so it spent the least time cost. Same conclusion shown in our experiment as in Figure 1.

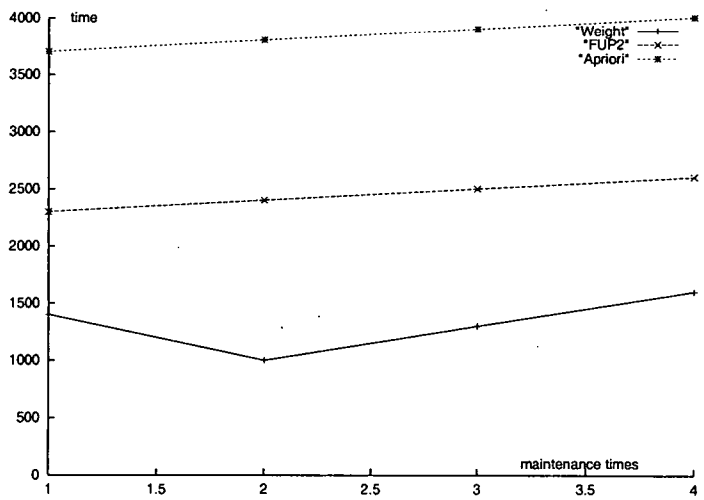


Figure 1: The large item set mining time cost comparison

Our algorithm gets a significant improvement by only scanning the new database. But there are some different rules in our rule set to the rule set made by Apriori which scan all old and new data. But on earth what is the difference, how about its influence? Because Cheung's $FUP$ algorithm also scans the old database, which saves some cost by reducing the candidate number in old database, so it generate the same large itemsets as Apriori. Then they will generate same rules according certain confidence. So we only need to compare the result rule set with one of them after generate Large itemset. But our algorithm don't scan the old data but only the new database, then generates the Competitive Rule Set by weighting and choose the winners as result (There is some similarity like genetic algorithm). so it can consist with the new data better than their algorithms. According the result rule sets from large itemset 1 to 4, we compared the difference between our algorithm and Cheung's $FUP$ model.

**Comparison with $FUP$**

Let $minsupp = 0.2$, $minconf = 0.7$, the confidence threshold $mincruc = 0.25$, the weight of old and new rule confidence is 0.7 and 0.3 respectively, results shown as follows.

Table 8: The rules in the maintenance

| | After 1st maintenance | After 2nd maintenance | After 3rd maintenance | After 4th maintenance |
|---|---|---|---|---|
| $FUP_2$ | 1062 | 1096 | 1130 | 1189 |
| Weight algorithm | 1095 | 1185 | 1204 | 1397 |
| Both in $A\&W$ | 1044 | 1089 | 1114 | 1160 |
| Only in Apriori | 18 | 7 | 16 | 29 |
| Only in our Weight algorithm | 51 | 96 | 90 | 237 |

All the rules have two supports and two confidences generated relatively by the two algorithms, which $F_{confidence}$ and $F_{support}$ are for $FUP$ algorithm, $W_{confidence}$ and $W_{support}$ are for weight algorithm. If the rule not generated by an algorithm, we let its confidence and support equal zero. We define $error = |W_{support} - F_{support}|$ to measure the difference between two confidence. For those results generated by both $FUP$ and weight algorithm, the comparison shown as follows.

Table 9: The comparison with generated results

| | After 1st maintenance | After 2nd maintenance | After 3rd maintenance | After 4 th maintenance |
|---|---|---|---|---|
| Average error | 0.015 | 0.031 | 0.032 | 0.02 |
| Max error in a rule | 0.296 | 0.2 | 0.085 | 0.252 |
| Rule number with error over 0.1 | 1 | 1 | 0 | 28 |

We can see that they are almost equal at the first 3 maintenance, at most one rule with error over 0.1. But at the 4th maintenance, there are 28 rules with error over 0.1, because some rules from new database with significant different confidence begin influence the old rule set by the Competitive Set.

Now we analysis those different rules: (1) To those rules generated by our algorithm but not $FUP$, they are the new rules can not be found by $FUP$. They keep more consistency with new database. For example, this is a rule with confidence only 0.4 before maintenance, but in the new database it is a significance rule, always with confidence over 0.9. In $FUP$, because the new databases are relatively

small than the old database, so the significant association in new data still can not be shown in the all data. During maintenance, it got a confidence serials: 0.4, 0.44, 0.46,0.49, 0.55. But according our algorithm, the confidence change in the maintenance should be:

$$\begin{aligned} 0.4 &\rightarrow 0.4 * 0.7 + 0.9 * 0.3 = 0.45 \\ &\rightarrow 0.45 * 0.7 + 0.9 * 0.3 = 0.585 \\ &\rightarrow 0.585 * 0.7 + 0.9 * 0.3 = 0.66 \\ &\rightarrow 0.66 * 0.9 + 0.3 = 0.732 \end{aligned}$$

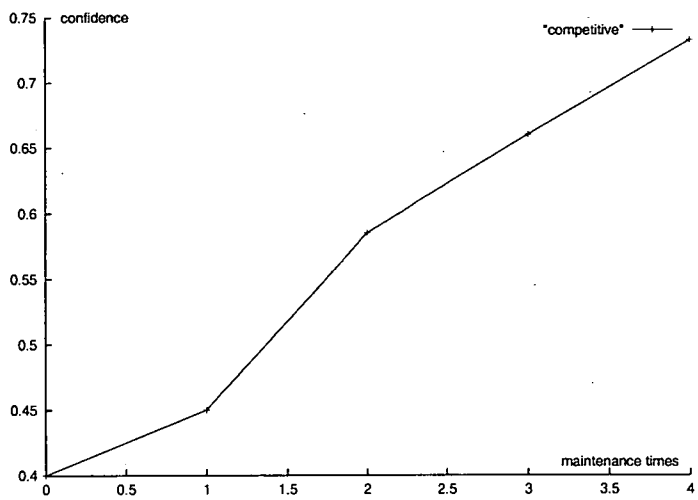


Figure 2: The competitive procedure of a rule

(2) To those rules generated by $FUP$ algorithm but not our algorithm, they are the "lost" rules from $FUP$. It needs to know why they lost, how about the influence when lost those rules?

So we analysis all those "lost" rules, and found that, for every rule, at least one of it's $F_{support}$ and $F_{confidence}$ near the relative thresholds. We define the neighbor of $minsupp$ and the neighbor of $minconf$ as $S_{interval}$ and $C_{interval}$ respectively. There are four classes of lost rules as follows.

Class1: Rules with $F_{confidence}$ in $C_{interval}$ and $F_{support}$ in $S_{interval}$;

Class2: Rules with $F_{confidence}$ in $C_{interval}$ and $F_{support}$ not in $S_{interval}$;

Class3: Rules with $F_{confidence}$ not in $C_{interval}$ and $F_{support}$ in $S_{interval}$;

Class4: Rules with $F_{confidence}$ not in $C_{interval}$ and $F_{support}$ not in $S_{interval}$.

The following figure shown the rule distribution:

Table 10: The rule distribution

| | The 1st maintenance | After 2nd maintenance | The 3rd maintenance | The 4th maintenance |
|---|---|---|---|---|
| Class1 | 5.5% | 0 | 0 | 0 |
| Class2 | 66.5% | 0 | 19% | 3% |
| Class3 | 27.8% | 100% | 81% | 97% |
| Class4 | 0 | 0 | 0 | 0 |

As we know, those rules with support in $S_{interval}$ or confidence in $C_{intervals}$ usually mean the uncertain and debated knowledge. We often discard those rules in practical decision, so we can say that those "lost" rule only generate some neglectable influence. In order to capture the novelty, this error is certainly reasonable and necessary.

We have seen, our algorithm can save much time cost in association rule maintenance. It keeps more association with the new data than old data, especially in a time serial of continually maintenance. It can generate many new rules to describe the new association in new data. At same time, it discards some old rules unfitting the new data. Their confidence or support near the threshold, so deleting them will only generate a slight influence to the final decision. At a word, our algorithm is efficient in association maintenance, especially suit the practice company decisions which pay more attention to the new market trend, need to a time serial support analysis.

# 7 Conclusions

Database mining generally presupposes that the goal pattern to be learned is stable over time. This means that its pattern description does not change while learning proceeds. In real-world applications, however, pattern drift is a natural phenomenon which must be accounted for by the mining model. To capture more properties of new data, we advocated a new model of mining association rules in incremental databases in this paper. Our concept of mining association rules in this paper is different from previously proposed ones. It is based entirely on the idea of weighted methods; the main feature of our model is that it reflects the novelty of dynamic data and the size of the given database. Actually, previous frequency-based models are the special cases of our method working without respect to the novelty. Our experiments shown, the proposed model is efficient and promising.

# References

[1] R. Agrawal, T. Imielinski, and A. Swami, Mining association rules between sets of items in large databases. In: *Proceedings of the ACM SIGMOD Conference on Management of Data*, 1993: 207-216.

[2] S. Brin, R. Motwani and C. Silverstein, Beyond Market Baskets: Generalizing Association Rules to Correlations. In: *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1997: 265-276.

[3] M. Chen, J. Han and P. Yu, Data Mining: An Overview from a Database Perspective, *IEEE Trans. Knowledge and Data Eng.*, vol. 8, 6(1996): 866-881.

[4] D. Cheung, J. Han, V. Ng and C. Wong, Maintenance of discovered association rules in large databases: An incremental updating technique, *Proceedings of 12nd International Conference on Data Engineering*, New Orleans, Louisiana, 1996: 106-114.

[5] D. Cheung, S. Lee and B. Kao, A gerneral incremental technique for maintaining discovered association rules, *Proceedings of the Fifth International Conference on Database Systems for Advanced Applications*, Melbourne, Australia, 1997.4: 185-194.

[6] R. Godin and R. Missaoui, An incremental concept formation approach for learning from databases. *Theoretical Computer Science*, 133(1994): 387-419.

[7] J. Han, J. Pei, and Y. Yin, Mining frequent patterns without candidate generation. In: *Proceedings of ACM SIGMOD*, 2000: 1-12.

[8] C. Hidber, Online Association rule mining, In: *Proceedings of the ACM SIGMOD Conference on Management of Data*, 1999.

[9] H. Liu and H. Motoda, *Instance Selection and Construction for Data Mining*. Kluwer Academic Publishers, Feburary 2001.

[10] G. Piatetsky-Shapiro, Discovery, analysis, and presentation of strong rules. In: *Knowledge discovery in Databases*, G. Piatetsky-Shapiro and W. Frawley (Eds.), AAAI Press/MIT Press, 1991: 229-248.

[11] T. Shintani and M. Kitsuregawa, Parallel mining algorithms for generalized association rules with classification hierarchy. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1998: 25-36.

[12] R. Srikant and R. Agrawal, Mining generalized association rules. *Future Generation Computer Systems*, Vol. 13, 1997: 161-180.

[13] P. Utgoff, Incremental induction of desion trees. *Machine Learning*, 4(1989): 161-186.

[14] D. Tsur, J. Ullman, S. Abiteboul, C. Clifton, R. Motwani, S. Nestorov and A. Rosenthal, Query flocks: A generalization of association-rule mining. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1998: 1-12.

[15] G. Webb, Efficient search for association rules. In: *Proceedings of ACM SIGKDD*, 2000: 99-107.

[16] X. Wu, Building Intelligent Learning Database Systems, *AI Magazine*, 21(2000), 3: 59-65.

[17] S. Zhang and C. Zhang, Estimating Itemsets of Interest by Sampling. In: *Proceedings of the 10th IEEE International Conference on Fuzzy Systems*, Dec. 2001.

[18] S. Zhang and C. Zhang, Pattern discovery in probabilistic databases. In: *Proceedings of AI'01*, Dec. 2001.

[19] Chengqi Zhang and Shichao Zhang, *Association Rules Mining: Models and Algorithms*. Springer-Verlag Publishers in Lecture Notes on Computer Science, Volume 2307, p. 243, 2002.

*Received December, 2001*