An Efficient Data Mining Approach on Compressed Transactions

Arpana Jaiswal, Dr. Amit Khare, Rahul Moriwal
M.tech Scholar, Professor, Assistant Professor
Department of Computer Science and Engineering,
Acropolis Institute of Technology & Research, Indore, Madhya Pradesh, India.

ABSTRACT:

In an era of knowledge explosion, the growth of data increases rapidly day by day. Since data storage is a limited resource, how to reduce the data space in the process becomes a challenge issue. Data compression provides a good solution which can lower the required space. Data mining has many useful applications in recent years because it can help users discover interesting knowledge in large databases. However, existing compression algorithms are not appropriate for data mining. In [1, 2], two different approaches were proposed to compress databases and then perform the data mining process. However, they all lack the ability to decompress the data to their original state and improve the data mining performance. In this research a new approach called Mining Merged Transactions with the Quantification Table (M^2TQT) was proposed to solve these problems. M^2TQT uses the relationship of transactions to merge related transactions and builds a quantification table to prune the candidate itemsets which are impossible to become frequent in order to improve the performance of mining association rules. The experiments show that M^2TQT performs better than existing approaches.

I .INTRODUCTION :

A great amount of data is being accumulated very rapidly in the Internet era. Consequently, it takes a lot of time and effort to process these data for knowledge discovery and decision making. Data compression is one of good solutions to reduce data size that can save the time of discovering useful knowledge by using appropriate methods, for example, data mining. Data mining is used to help users discover interesting and useful knowledge more easily. It is more and more popular to apply the association rule mining in recent years because of its wide applications in many fields such as stock analysis, web log mining, medical diagnosis, customer market analysis, and bioinformatics. In this research, the main focus is on association rule mining and data pre-process with data compression.

M.C. Hung et al. proposed a knowledge discovery process from compressed databases in [1] which can be decomposed into the following two steps:

(1) Data pre-process step: Data pre-process transforms the original database into a new data representation where several transactions are merged to become a new transaction. Eventually, it generates a new transaction database at the end of the data pre-process step.

(2) Data mining step: It uses an Apriori-like algorithm [11]-[14] of association rule mining to find useful information. Details are described later. There are some problems in this approach. First, the compressed database is not reversible after the original database is transformed by the data pre-process step. It is very difficult to maintain this database in the future. Second, although some rules can be mined from the new transactions, it still needs to scan the database again to verify the result. This is because the data mining step produces potentially ambiguous results. It is a serious problem to scan the database multiple times because of the high cost of re-checking the frequent itemsets. Another solution was developed by Mafruz Zaman Ashrafi et al. [2]. However, they suffer from similar problems mentioned above. It is even a bigger challenge to maintain the compressed

database in the future. In addition, it spends too much time to check candidate itemsets in the data mining step. In this research, a more efficient approach, called Mining Merged Transactions with the Quantification Table (M^2 TQT) is proposed, which can compress the original database into a smaller one and perform the data mining process without the above problems. Our approaches have the following characteristics: (a) The compressed database can be decompressed to the original form. (b) Reduce the process time of association rule mining by using a quantification table. (c) Reduce I/O time by using only the compressed database to do data mining. (d) Allow incremental data mining. The rest of the paper is organized as follows. The background and related works are provided in Section 2. The proposed algorithm is described in Section 3. The experimental environment and results are presented in Section 4. Finally, Section 5 concludes the paper and discusses the future work.

II. RELATED WORK:

When making decisions, people would like to have enough information to avoid making wrong decisions that may cause losses. Data mining can be used to find useful information as a part of knowledge discovery in databases (KDD) for better decision-making. KDD can convert source data into useful information using the main process [3] depicted in Fig. 1.

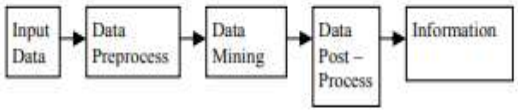

Fig. 1 The process of KDD

The goal of data preprocessing is to transform the input data into a suitable form for data mining or analysis. In general, data mining techniques can be divided into three categories: classification [4], [5], clustering [6], and association rule [11]. Classification is the process that data is divided into different classes with the known property. First, data is divided into two datasets which are training dataset and testing dataset. Second, a classification model is generated from the training dataset and then tests are made to verify the model's accuracy. Finally, the verified model is used to classify new transaction data into respective classes. Clustering is the process that data is divided into mutiple groups in which the data are similar. It is an unsupervised process. Association rule can be expressed as "if A, then B" after satisfying the measures of support and confidence. For example, assume that a customer buys milk and bread whereas another buys milk and meat. One would like to discuss "if a new customer buys milk, then he/she will buy bread too" or "if a new customer buys milk, then he/she will also buy meat". The concept of association rule mining in the next subsection. Data post–processing is to ensure the result is valid and usable. For example, visualization can be used for analysts to explore the data mining results from a variety of viewpoints. This can help users better utilize the mined rules or patterns. It is more and more popular for many users perform association rule mining in recent years. Many approaches are proposed [7-14], [17] in association rule mining Let I = {i1, i2, …, im} be a set of items. Let D be a transaction database which contains a set of transactions. Let t= (tid, t-itemset) be a transaction. Tid is a transaction number and t-itemset contains a set of items. Let X be a set of items. If a transaction t contains X if only if $X \subseteq t$. Length of a transaction which contains a K-itemset is K. There are two important measurements which are support and confidence in association rule mining. Support is the frequency of occurring patterns in D and confidence is the strength of implication. Their definitions are as follows: (1) Support (X) = |T(X)| / |D| (2) Confidence (X→Y) = Support (XY) / Support (X) T(X) is any transaction in D that contains X. |D| is the total number of transactions in D. We can define what we think an interesting relation is in a transaction database.

In support-confidence framework, if it is an interesting relation for X →Y, then X and Y must be frequent. How to define a frequent relation? There are two conditions. One condition is support(X) ≧ minsupport and support(Y) ≧minsupport(Y). Another is Confidence (X→Y) ≧ minconfidence. Minsupport and minconfidence are user-defined thresholds. The problems of mining association rules are mainly divided into two sub-problems. One is to discover the frequent itemsets and another is to generate the association rules. The first problem is more difficult than the second one. Most papers are focusing on the first problem. The apriori [8] algorithm is one of the classical algorithms in the association rule mining. It uses simple steps to discover frequent itemsets. Apriori algorithm is given in Fig. 2. Lk is a set of k-itemsets. It is also called large k-itemsets. Ck is a set of candidate k-itemsets. How to discover frequent itemsets? Apriori algorithm finds out the patterns from short frequent itemsets to long frequent itemsets. It does not know how many times the process should take beforehand. It is determined by the relation of items in a transaction. The process of the algorithm is as follows: At the first step, after scanning the transaction database, it generates frequent 1-itemsets and then generates candidate 2-itemsets by means of joining frequent 1-itemsets. At the second step, it scans the transaction database to check the count of candidate 2-itemsets. It will prune some candidate 2-itemsets if the counts of candidate 2-itemsets are less than predefined minimum support. After pruning, the remaining candidate 2-itemsets become frequent 2-itemsets which are also called large 2-itemsets. It generates candidate 3-itemsets by means of joining frequent 2-itemsets. Therefore, CK is generated by joining large (K-1)-itemsets obtained in the previous step. Large K itemsets are generated after pruning. The process will not stop until no more candidate itemset is generated.

1) $L_1 = \{\text{large 1-itemsets}\}$;
2) **for** ($k = 2$; $L_{k-1} \neq \emptyset$; k++) **do begin**
3) $C_k = \text{apriori-gen}(L_{k-1})$; // New candidates
4) **forall** transactions $t \in \mathcal{D}$ **do begin**
5) $C_t = \text{subset}(C_k, t)$; // Candidates contained in t
6) **forall** candidates $c \in C_t$ **do**
7) c.count++;
8) **end**
9) $L_k = \{c \in C_k \mid c.\text{count} \geq \text{minsup}\}$
10) **end**
11) Answer = $\bigcup_k L_k$;

Fig. 2 Apriori algorithm

Since most data occupy a large amount of storage space, it is beneficial to reduce the data size which makes the data mining process more efficient with the same results. Compressing the transactions of databases is one way to solve the problem. [1] Proposed a new approach for processing the merged transaction database. It is very effective to reduce the size of a transaction database. Their algorithm is divided into data preprocess and data mining. The overview of the approach is shown in Fig. 3.

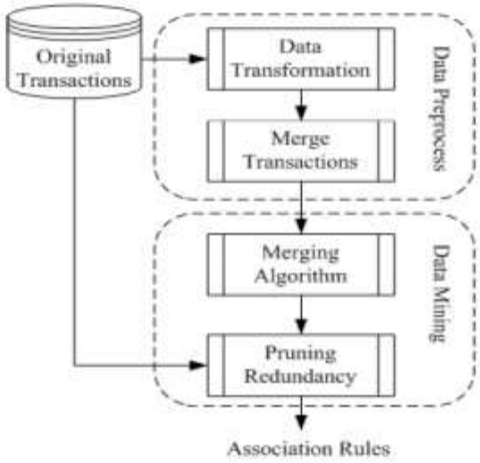

Fig. 3 An overview of the merged transaction algorithm

There are two sub-processes in the data preprocess. One sub-process transforms the original database into a new data representation. It uses lexical symbols to represent raw data. Here, it's assumed that items in a transaction are sorted in lexicographic order. Another sub-process is sorting all the transactions to various groups of transactions and then merges each group into a new transaction. For example, T1= {A, B, C, E} and T2 = {A, B, C, D} are two transactions. T1 and T2 are merged into a new transaction T3= {A2, B2, C2, D1, E1}. Fig. 4 shows an example of the sort grouping method and Fig. 5 shows an example of data mining sub-process.

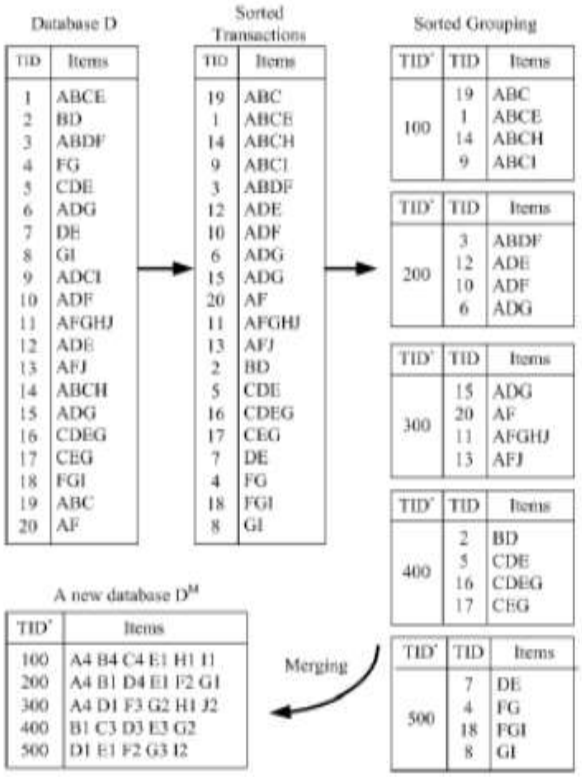

Fig. 4 An example of sort grouping

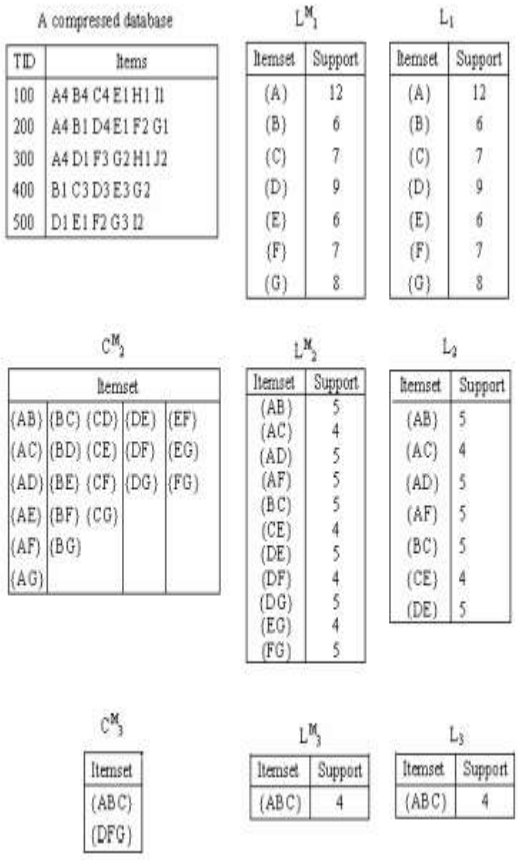

Fig. 5 An example of data mining

The process called merge-mining algorithm is used to find frequent itemsets from the new transaction DM. There are two phases in this algorithm. The first phase is finding frequent itemsets. The second phase is to prune redundancy. It is possible that frequent itemsets generated in the first phase might not exist in the DM. For this reason, it needs to verify those frequent itemsets by scanning DM again. Fig. 6 is the pseudo code of merge-mining algorithm. The notations are shown in Fig. 7.

```
// Phase one
D^M = compressed database
L_1^M = {large 1-itemsets in compressed database};
for(k = 2; L_{k-1}^M ≠ Ø; k++) do begin
  C_k^M = merging-gen(L_{k-1}^M);
  forall transactions T_M^* ∈ D^M do begin
    C_t^M = subset(C_k^M, t^*); // Candidate contained in t^*
    forall candidates c^* ∈ C_t^M do begin
      c^*.count = c^*.count + min-frequency(c^*); // the smallest item frequency in Candidate
    end
    L_k^M = {c^* ∈ C_k^M | c^*.count ≥ minsup}
  end
  Answer = ∪_k L_k^M;
// Phase two
D = original database
  forall transactions T_N ∈ D do begin
    L_t^M = subset(L_1^M, t) // Large itemset contained in t
    forall large itemsets l^* ∈ L_t^M do begin
      l^*.count++;
  end
  L_1 = {l^* ∈ L_1^M | l^*.count ≥ minsup}
  Answer = ∪_k L_k;
```

Fig. 6 The pseudo code of merge-mining algorithm

c^*	Candidate itemset in the merged database
l^*	Large itemset in the merged database
L_K	Large k-itemsets in the original transaction database
L_K^M	Large k-itemsets in the merged database
C_K^M	Candidate k-itemsets in the merged database.
D^M	The merged database after preprocess
D	The original transaction database
T	A transaction in the original database
T^*	A transaction in the merged database
T_N	All transaction in the original database
T_M^*	All transactions in the merged database
M	Number of groups after data preprocess

Fig. 7 The notations of merge-mining algorithm

Pincer-Search algorithm [9] is an Apriori-like algorithm. It takes advantage of Apriori algorithm to find maximal frequent itemsets. The process is mainly divided into two parts, bottom-up search and top-down search. Bottom-up search uses the process similar to Apriori algorithm. The key is the top-down search. Top-down search is to find maximal frequent itemsets by using the information generated in bottom-up search. And then bottom-up search is to find frequent itemsets by using the information generated in top-down search. They are performed in turns until all of the maximal frequent itemsets are found. The pseudo code of Pincer Search algorithm is shown in Fig. 8

Algorithm: The Pincer-Search algorithm
Input: a database and a user-defined minimum support
Output: MFS which contains all maximal frequent itemsets

1. $L_0 := \emptyset$; $k := 1$; $C_1 := \{\{i\} \mid i \in I\}$
2. MFCS := $\{\{1,2,\ldots,n\}\}$; MFS := $\emptyset$
3. **while** $C_k \neq \emptyset$
4. read database and count supports for C_k and MFCS
5. remove frequent itemsets from MFCS and add them to MFS
6. L_k := {frequent itemsets in C_k} \ {subsets of MFS}
7. S_k := {infrequent itemsets in C_k}
8. call the *MFCS-gen* algorithm if $S_k \neq \emptyset$
9. call the *join* procedure to generate C_{k+1}
10. **if** any frequent itemset in C_k is removed in line 6
11. call *recovery* procedure to recover candidates to C_{k+1}
12. call new *prune* procedure to prune candidates in C_{k+1}
13. $k := k+1$
14. **end-while**
15. **return** MFS

Fig. 8 Pincer-Search algorithm

III. PROPOSED METHOD :

The description of the proposed algorithm focuses on compressing related transactions and building a quantification table for pruning candidate itemsets that are impossible to become frequent itemsets. Finally, an example is provided to show the processes of our method. To simplify the description, it assumes the items in each transaction are presented in a lexicographical order. A. Overview Algorithms like [1], [2] compress transactions to reduce the size of a transaction database. Then, they use Apriori-like algorithms to mine the compressed database. Whereas the two phases approach of compression and data mining are used, they suffer the following problems:
(1) In the data compression phase, the original database cannot be recovered to support transaction updates.
(2) In the data mining phase, a lot of candidate itemsets could be generated in a large transaction database. Since both [1] and [2] need to scan the database more than once, they have a much higher process cost. The first problem is due to the lack of rule or constraint in the process of merging transactions in the data compression phase. Therefore, the compressed database can not be decompressed to its original form In addition, they don't use user-defined threshold to filter infrequent 1-itemsets from the original database.. Another problem is that Apriori-like algorithms generate a lot of candidate itemsets and need to check the candidate itemsets by scanning the database. It is very time-consuming. Our goal is to take the advantages of [1] and Apriori algorithm without suffering from the problem of checking candidate itemsets and recovering the database for new data. In order to provide a better performance, we limit the number of database scan to be one in the data compression phase and build a quantification table. In the data mining phase, we use the same approach of Apriori algorithm to generate candidate itemsets and reduce the number of candidate itemsets by using the quantification table. We also reduce the time of scanning the database. We present a novel approach which can (1) support local transaction variation (2) recover the transaction database to its original state (3) make the compressed database much smaller than the original one (4) reduce data mining time We called our approach the Mining Merged Transactions with the Quantification Table (M^2TQT) which has three phases: (1) merge related transactions to generate a compressed database (2) build

a quantification table (3) discover frequent itemsets B. M^{2}TQT Approach First, M^{2}TQT uses the transaction relation distance to merge the relevant transactions. The definition of the transaction relation distance is defined in Section C. Section D introduce how to build a quantification table in Section D. Then, it illustrates the process of compressing a database in Section E. Next, Section F shows how to compute support of itemsets from minimum-frequency function in Section F. Finally, it explains how to recover data from the compressed database in Section G.

C. Transaction Relation Distance Based on the relation distance between transactions one can merge transactions with closer relationship to generate a better compressed database. Here the transaction relation and transaction relation distance are defined as follows: Definition 1: (1) Transaction Relation: The relation between two different transactions T1 and T2 is that T1 is either a subset or a superset of T2. (2) Transaction Relation Distance: Distance is the number of different items between two transactions. Example 1: T1={ ABCE} and T2={ ABC} ,DT1-T2= 1 Example 2: T3={ A} and T4={ C } ,DT3-T4= 2

D. A Quantification Table To reduce the number of candidate itemsets to be generated, additional information is required to help prune non-frequent itemsets. A simple quantification table is used to record this information when each transaction is processed. Assuming the items in a transaction appear in a lexicographical order, our approach starts working from the left-most item and calls it a prefix-item. After finding the length of the input transaction as n, it records the count of the itemsets appearing in the transaction under the respective entries of length Ln, Ln-1,.. L1. A quantification table is composed of these entries where each Li contains a prefix-item and its support count. An example database in Table I is used to show the construction of a quantification table in Table II.

TABLE I
AN EXAMPLE DATABASE

TID	Transactions
100	ABCDE
200	CDE
300	ACD

TABLE II
A QUANTIFICATION TABLE FOR TABLE I

L5	L4	L3	L2	L1
A1	A1	A2	A2	A2
	B1	B1	B1	B1
		C2	C3	C3
			D2	D3
				E2

For instance, after reading the transaction {ABCDE} of TID 100, it knows the transaction length n is 5. For the prefix-item A, the counters under L5 to L1 are all increased by one from the initial value of zero. That is, A1 appears in each Li, where i = 5 to 1. For the prefix-item B, the counters under L4 to L1 are all increased by one as well. That is, B1 appears in each Li, where i = 4 to 1. The same process is performed for items C, D, and E. Similarly, after reading TID 200 {CDE}, the table has C2 in L3, L2, and L1; D2 in L2 and L1; E2 in L1. Finally, with the last transaction {ACD}, it will increase the counters by one from A1 to A2 in L3, L2, and L1; C2 to C3 in L2 and L1; D2 to D3 in L1. Table 2 shows the result of building the quantification table.

With this table, we can easily prune the candidate itemsets whose counters are smaller than the minimum support. E. The Process of Database Compression Let d be a relation distance and it is initialized to 1 at the beginning. Transactions will be merged into their relevant transaction groups in the merged blocks based on the transaction relation distance. M^{2}TQT consists of the following steps: Step 1: Read a transaction at a time from the original database. Step 2: Record the information of the input transaction to build a quantification table. Step 3: Compute the length n of the transaction. Step 4: If the merged block is not empty, read the relevant transaction groups from the merged block. Step 5: Compute relation distance between the transaction and relevant transaction groups. If the

transaction is a superset of the longest transaction of a relevant transaction group, a subset of the smallest transaction of a relevant transaction group, or equal to one transaction of a relevant transaction group, it can be merged into the relevant transaction group. For example, we assume d=1. Two transactions {BCG} and {BG} are merged into a relation transaction group {BCG=2.1.2}. A "=" symbol is used to separate items and their respective support counts. We read another transaction {BC} and compute the relation distance between {BCG=2.1.2} and {BC}. Since the relation distance is 1, {BC} is merged into the relation transaction group. Finally, the relevant transaction group becomes {BCG=3.2.2}. Step 6: Compute the relation distance between the transaction and those transactions coming from (n+d) block, n block, and (n-d) block where n > d. If it finds the satisfied relevant transactions, merge the transactions to become a relevant transaction group and then classify it as (n+d) merged block, n merged block or (n–d) merged block. If no relevant transaction can be found, the transaction is classified as n merged block. Step 7: Repeat the above six steps until the last transaction is read. Step 8: Read a transaction from the merged blocks. Step 9: Compute the relation distance between the transaction and all other transactions in the relevant transaction groups. If the transaction is a sub-transaction of the maximum length transaction of a relation transaction group and its distance is satisfied, it can merge the transaction into the relation transaction group to generate a new count. The process continued until the last transaction is read. Step10: Set d to d+1. Step11: Repeat the above steps 8 - 10 until no more relation distance is found between transactions. F. Minimum-frequency Function The minimum-frequency function takes original transactions and merged transactions as input. It returns the minimum number of itemsets in the transactions. For example, let a candidate 2-itemset C2 be {BC, AE} and merged transactions of T* be {{AE=2.1}, {BCG=2.1.2}, {CDEG=2.3.3.1}, {ABCE}, {C}}. After calling minimum-frequency function with T* being the input, it returns 0+1+0+1+0=2 for BC and 1+0+0+1+0=2 for AE. This is an efficient function to count the number of itemsets in the transactions. G. Recover Data from Compressed Database to Original Database With the proposed approach it can recover data from the compressed database. Assume the relation distance is equal to 1. A merged transaction is expressed as = c1, c2…ck.. cn-1.cn, where s1, s2… , sk , sn-1, sn are items and c1, c2…, ck, cn-1, cn are their corresponding support counts separated by ".". The smallest count in ci for i = 1 to n is the support of the longest transaction, i.e., { s1, s2… , sk , sn-1, sn }. If ck is the smallest count in c1, c2…, ck, cn-1, cn , then the count of the longest transaction { s1, s2… , sk , sn-1, sn } is ck. Therefore, the transaction { s1, s2… , sk , sn-1, sn } is recovered and the merged transaction becomes = c1-ck.c2-ck… cn-1-ck. cn-ck. The items with a zero count are removed from the merged transaction. Repeat the above process to find the next longest transaction in the merged transaction until no count left. For example, the merged transaction =3.1.4.2 has the smallest count of 1 such that the count of transaction is 1. Next, decrease the count of each item in by 1 to get =3-1.4-1.2-1 = 2.3.1. Note that item B in the merged transaction is removed since it has a zero count. Next, the smallest count among A, C, and D is also 1 such that the count of transaction is 1. Then, the merged transaction becomes ==2-1.3-1=1.2 and the smallest count of 1 is the count of transaction . Finally, =2-1=1 and we have decompressed the merged transaction =3.1.4.2 to get back the original transactions {ABCD}, {ACD}, {AC}, {C}. Here, {{ABCD}, {ACD}, {AC}, {C}} also satisfy the specified relation distance. H. A Simple Example To illustrate the process of the proposed approach, a simple example is shown below. There are 9 transactions with a total number of 6 distinguished items in the original database as shown in the left-hand side of Fig. 9. Assume the minimum support is 2 meaning that an itemset is frequent if its count is greater than or equal to 2.

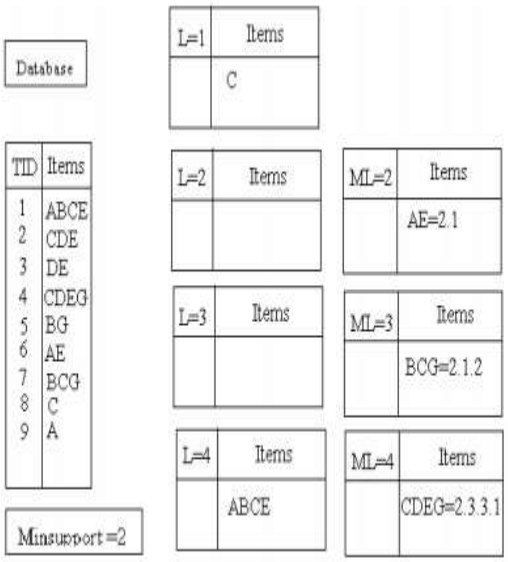

Fig. 9 The original database and its compressed database

L4	L3	L2	L1
A1	A1	A2	A3
C1	B2	B3	B3
	C2	C4	C5
	D1	D3	D3
		E1	G3
			E5

Fig. 10 A quantification Table for the database in Fig. 9

Phase 1: The first step is to compress the original database after scanning their transactions. Assuming the transaction relation distance =1, read the first transaction {ABCE} to compute its length n = 4 and put it into Length-4 block. Next, read transaction {CDE} to get the length n = 3 and then compute transaction relation distance between {ABCE} and {CDE} to get the distance of 3. They cannot be compressed because the relation distance is not equal to 1. Therefore, transaction {CDE} is put into Length-3 block. After reading the third transaction {DE} with a length of 2, it examines if the transaction appears in any merged transactions. If it exists, they are merged to generate a new merged transaction with increased counts. On the other hand, it examines whether the computed transaction relation distances with all merged transactions agree to the assumed distance. If it exists, transaction {DE} is merged with the existing merged transaction which satisfies the transaction relation distance. If transaction {DE} has no relation in the merged transactions, it will check with the items in L=1, L=2 and L=3 blocks. Since the relation distance between {DE} and {CDE} is 1, they are merged into a new transaction {CDE=1.2.2}. This new transaction is put into Length-3 merged block. Subsequent transactions are processed in the same way. The compressed database is shown in the right-hand side of Fig. 9 where the number of transactions becomes 5. Phase 2: When the compressed database is generated, it also builds a quantification table at the same time as shown in Fig. 10. Phase 3: The compressed database is used to generate frequent itemsets with any Apriori-like algorithm for association rule mining. The minimum support is set to 2. From the quantification table, it can generate frequent 1-itemsets {A, B, C, D, E, G}. Frequent 1-itemsets are used to generate candidate 2-itemsets. At

the same time, it checks the counts of itemsets in L2 of the quantification table to prune the candidate itemsets which are impossible to become frequent itemsets. The generated candidate 2-itemsets are {AB, AC, AD, AE, AG, BC, BD, BE, BG, CD, CE, CG, DE, DG, EG}. Because the item's frequency is recorded in the merged transactions, one can use the minimum-frequency function to determine the count of a candidate itemset. The minimum-frequency function returns the minimal number of item occurrences in a merged transaction and it also returns a value of 1 for an original transaction. For instance, let C2 be {{A E}, {CG}} and compressed transaction T* = {{AE=2.1}, {BCG=2.1.2}, {CDEG=2.3.3.1}, {ABCE}, {C}}. After calling the minimum-frequency function for {A E}, it returns {1, 0, 0, 1, 0}. The total frequency of {A E} is 1+0+0+1+0=2. For {CG}, it returns {0, 1, 1, 0, 0}. The total frequency of {CG} is 0+1+1+0+0=2. Using the quantification table, one can prune the candidate 2-itemset {EG} and then scan the compressed database to check if the rest of candidate 2-itemsets are frequent. Candidate 3-itemsets are generated from frequent 2-itemsets which are {AE：2, BG：2, CD：2, CE：3, CG：2, DE：2}. Finally, it outputs the frequent 3-itemset {CDE：2} after scanning the compressed database.

IV. EXPERIMENTAL RESULT:

M^2TQT and Merged Transactions Approach [1] were implemented in java programming language and all experiments run on a PC of Intel Pentium 4 3.0GHz processor with DDR 400MHz 4GB main memory. Synthetic datasets are generated by using the IBM dataset generator [15] for our experiments. Table III lists the parameters used in the IBM dataset generator.

TABLE III
PARAMETERS USED IN THE IBM DATASET GENERATOR

D	Number of transactions
T	Average size of transactions
I	Average size of maximal potentially–large itemsets
L	Number of potentially-large itemsets
N	Number of items

The parameter settings used to generate experimental datasets are shown in Table IV. The average length of the transaction T is set as 4 and 10 and the average size of maximal potentially-large itemsets I as 4 and 5. To compare with the Merged Transactions Approach, the generated datasets have the number of items N=8 and 50, and the number of potentially-large itemset L=1000.

TABLE IV
DATABASE PARAMETER SETTINGS

Dataset	T	I	D
T4I5D10K	4	5	228K
T10I4D10K	10	4	407K
T4I5D1K	4	5	21K

The dataset T4I5D10k is used to run, our algorithm, merged transactions approach and Apriori algorithm. Let the average size of the potentially large itemset be 5 for the minimum supports 5%, 10%, 15%, and 20%, and compare our algorithm with Apriori algorithm and merged transactions approach. The performance of our algorithm is much better than the other two approaches as shown in Fig. 11. When the minimum support is 5%, the execution time of merged transactions approach is about 1.72 times of our approach. Our method outperforms the other two more than 40% as an average.

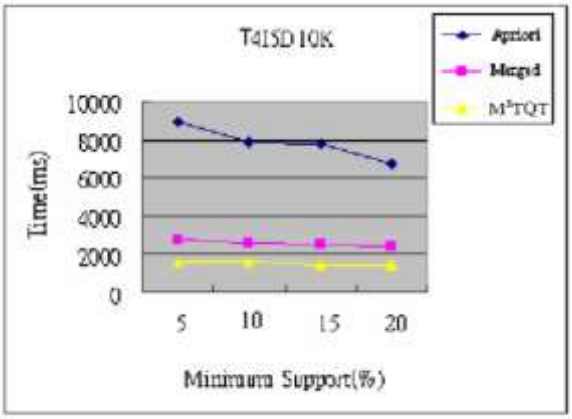

Fig. 11 The experiment result of T4I5D10K

The performance of using a quantification table is also analyzed in our experiment. From Fig. 12, it shows that the effect of using the table is low when the minimum support is lower. But the effect gets higher when the minimum support is high because it is possible to reduce more I/O time. In general, the performance of using a quantification table is better than without using it.

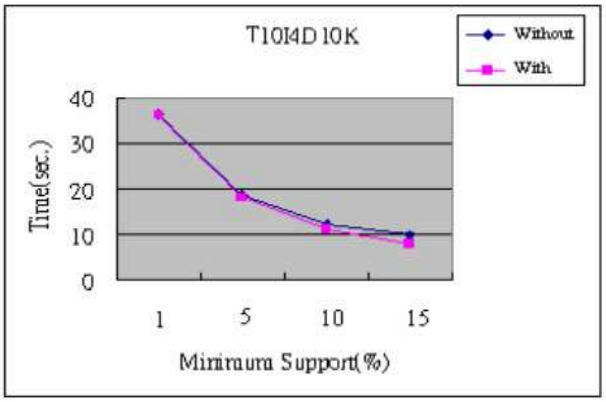

Fig. 12 With and without using a quantification table

For incremental mining, the dataset T4I5D1K is used to run the experiments of merged transactions approach and our algorithm. Let the average size of the potentially large itemset be 5 for the minimum supports 15%, 20%, 25%, and 30%. Here, 20% of the dataset T4I5D1K are used for the updates and 80% as the original data. Fig. 13 shows that four cases need to be considered in the incremental data mining. As in [16], our approach considers Case2 and Case3. The performance of our algorithm is much better than merged transactions approach as shown in Fig. 14. The average improvement of the execution time is about 8 times in favor of our approach.

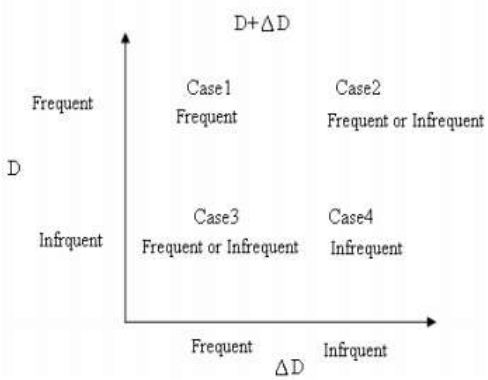

Fig. 13 Four cases regarding an itemset being frequent or not after transaction additions

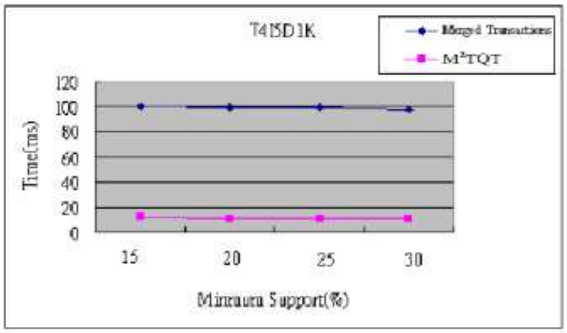

Fig. 14 The experiment of T4I5D1K

V. CONCLUSIONS AND FUTURE WORK :

In this paper, a new approach called Ming Merged Transactions with the Quantification Table is proposed to compress related transactions into a new transaction by scanning the transaction database only once. The M^2TQT approach utilizes the compressed transactions to mining association rule efficiently with a quantification table. There are several advantages of M^2TQT over the other approaches: (1) No multiple database scans, because M^2TQT reads the database only once if the compressed database fits into main memory. (2) Reduce the process time of association rule mining because M^2TQT prunes candidate itemsets which are impossible to become frequent. (3) A compressed database can be decompressed to the original database to support transaction updates. The M^2TQT algorithm was implemented to compare with the Apriori and Merged Transactions Approach for large datasets. The experiment results show that our approach performs the other two approaches. In the future, more improvements on the compression rate are under investigation. Some interesting research issues related to compression-based mining include the study of the best rate of compression for discovering frequent patterns. The extension of M^2TQT method for FP-tree [12], [14] is another interesting topic for future work.

REFERENCES :

s[1] M. C. Hung, S. Q. Weng, J. Wu, and D. L. Yang, "Efficient Mining of Association Rules Using Merged Transactions," in WSEAS Transactions on Computers, Issue 5, Vol. 5, pp. 916-923, 2006.

[2] M. Z. Ashrafi, D. Taniar, and K. Smith, "A Compress-Based Association Mining Algorithm for Large Dataset," in Proceedings of International Conference on Computational Science, pp. 978-987, 2003.

[3] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, "The KDD process for extracting useful knowledge from volumes of data," Communications of the ACM, Vol. 39, pp. 27-34, 1996.

[4] E. Hullermeier, "Possibilistic Induction in Decision-Tree Learning," in Proceedings of the 13th European Conference on Machine Learning, pp. 173-184, 2002.

[5] J. R. Quinlan, "C4.5: programs for machine learning," Morgan Kaufmann Publishers Inc, 1993.

[6] A. K. Jain and R. C. Dubes, Algorithm for clustering data: Prentice-Hall, Inc., 1988.

[7] R. Agrawal, T. Imielinski, and A. Swami, "Mining Association Rules Between Sets of Items in Large Databases," in Proceedings of the International Conference on Management of Data, pp. 207-216, 1993.

[8] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules," in Proceedings of the 20th International Conference on Very Large Data Bases, pp. 487-499, 1994.

[9] D. I. Lin and Z. M. Kedem, "Pincer-search: an efficient algorithm for discovering the maximum frequent set," IEEE Transactions on Knowledge and Data Engineering, Vol. 14, pp. 553-566, 2002.

[10] S. Brin, R. Motwani, J. D. Ullman, and S. Tsur, "Dynamic Itemset Counting and Implication Rules for Market Basket Data," in Proceedings of the International Conference on Management of Data, pp. 255-264, 1997.

[11] A. Savasere, E. Omiecinski, and S. Navathe, "An Efficient Algorithm for Mining Association Rules in Large Databases," in Proceedings of the 21st International Conference on Very Large Data Bases, pp. 432-444, 1995.

[12] J. Han, J. Pei, and Y. Yin, "Mining Frequent Patterns without Candidate Generation," in Proceedings of the International Conference on Management of Data, pp. 1-12, 2000.

[13] D. Burdick, M. Calimlim, J. Flannick, J. Gehrke, and T. Yiu, "MAFIA: A maximal frequent itemset algorithm," IEEE Transactions on Knowledge and Data Engineering, Vol. 17, pp. 1490-1504, 2005.

[14] G. Grahne and J. Zhu, "Fast algorithms for frequent itemset mining using FP-trees," IEEE Transactions on Knowledge and Data Engineering, Vol. 17, pp. 1347-1362, 2005.

[15] IBM Almaden Research Center, "Synthetic Data Generation Code for Associations and Sequential Patterns," URL:http://www.almaden.ibm. com/software/quest/, 2006.

[16] D. W. L. Cheung, S. D. Lee, and B. Kao, "A general incremental technique for maintaining discovered association rules," in Proceedings of the 15th International Conference on Database Systems for Advanced Applications, pp. 185-194, 1997.

[17] D. Xin, J. Han, X. Yan, and H. Cheng, "Mining Compressed Frequent-Pattern Sets," in Proceedings of the 31st international conference on Very Large Data Bases, pp. 709-720, 200