Global Journal of Computer Science and Technology
Volume 12 Issue 2 Version 1.0 January 2012
Type: Double Blind Peer Reviewed International Research Journal
Publisher: Global Journals Inc. (USA)
Online ISSN: 0975-4172 & Print ISSN: 0975-4350

An Enhanced Approach for Compress Transaction Databases

By I.Elizabeth shanthi, v.vidhya rani
Avinashilingam Deemed University For Women,Coimabtore

Abstract - Associative rule mining is defined as the task that deals with the extraction of hidden knowledge and frequent patterns from very large databases. Traditional associative mining processes are iterative, time consuming and storage expensive. To solve these processes, a way of representation that reduces this size and at the same time maintains all the important and relevant data needed to extract the desired knowledge from transaction databases is needed. This paper proposes a method that merges the transactions in the transaction database and uses FP-Growth algorithm for mining associative knowledge is presented. The experimental results in terms of compression ratio, both in terms of storage required and number of transactions, prove that the proposed algorithm is an improved version to the existing systems.

Keywords : *Associative Rule Mining, Compact Transactional Database, FP-Growth, FP-Tree, Frequent Pattern Generation, Merge Transactions.*

GJCST Classification: *H.2.8,H.2.4*

Strictly as per the compliance and regulations of:

© 2012 I.Elizabeth shanthi, v.vidhya rani. This is a research/review paper, distributed under the terms of the Creative Commons Attribution-Noncommercial 3.0 Unported License http://creativecommons.org/licenses/by-nc/3.0/), permitting all non-commercial use, distribution, and reproduction inany medium, provided the original work is properly cited.

An Enhanced Approach for Compress Transaction Databases

I.Elizabeth shanthi [α], v.vidhya rani [Ω]

Abstract - Associative rule mining is defined as the task that deals with the extraction of hidden knowledge and frequent patterns from very large databases. Traditional associative mining processes are iterative, time consuming and storage expensive. To solve these processes, a way of representation that reduces this size and at the same time maintains all the important and relevant data needed to extract the desired knowledge from transaction databases is needed. This paper proposes a method that merges the transactions in the transaction database and uses FP-Growth algorithm for mining associative knowledge is presented. The experimental results in terms of compression ratio, both in terms of storage required and number of transactions, prove that the proposed algorithm is an improved version to the existing systems.

Keywords : Associative Rule Mining, Compact Transactional Database, FP-Growth, FP-Tree, Frequent Pattern Generation, Merge Transactions.

I. Introduction

The beginning of the twenty first century has brought considerable advances in the field of computer-based information retrieval systems, where data with "hidden asset" called "knowledge" is quickly becoming the most valuable resource. Associative rule mining (Park *et al.*, 1995; Agrawal *et al.*, 1996) is defined as the task that deals with the extraction of hidden knowledge and frequent patterns from very large databases. Mining frequent patterns has become a focused topic in data mining research with the development of numerous interesting algorithms for mining association, partial periodicity, constraint-based frequent mining, associative classification and emerging patterns.

The popular area of application is the market basket analysis, which studies the buying habits of customers by searching for sets of items that frequently appear together. Associations among items of the same transaction lead to correlation and identification of frequent itemsets (Gionis *et al.*, 2007). Traditional associative mining processes are iterative, time consuming and storage expensive. Once the database becomes huge, it increases the number of Input/Output (I/O) scans and also reduces space complexity. In order to increase the performance of these applications, methods that can present data in a compact form is becoming crucial. The current need is to develop way of representation which reduces this size so that it can reside in the main memory and at the same time maintains all the important and relevant data needed to extract the desired knowledge from transactional database.

This paper analyzes algorithms that produce compact databases for knowledge discovery from large transaction databases like market basket database and web log databases. From these compact representations, association rule mining is applied to mine frequent patterns. In order to obtain a compact representation of the database Dai *et al.* (2008) proposed an algorithm called M^{2}TQT which uses a 'Merge Transactions Scheme (MTS)' to reduce storage requirement during analysis. The algorithm is efficient in two manners, namely (i) reduces database size and prunes irrelevant transactions, which saves time and (ii) Reduces the I/O time required. However, it has disadvantages namely (i) 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. (ii) The compressed database is reversible to its original form (iii) It has the serious problem of scanning the database multiple times, which results in high cost of re-checking the frequent itemsets and (iv) Processing time is still high when compared to uncompressed mining algorithms. To solve the above difficulties, this work enhances the M^{2}TQT in the following manner.

- Develop an algorithm to recover the original transaction from the compressed database.
- M^{2}TQT uses apriori-like algorithm which is main culprit for the multiple database scans. The present research work proposes the use of FP-Tree algorithm to avoid multiple scans of the database

The rest of the paper is organized as follows. Section 2 discusses the various existing methods available to obtain a representation of transaction databases. Section 3 presents the proposed method. The experimental results are discussed in Section 4, while Section 5 concludes the work with future research directions.

II. Existing Methods

A compact representation of transaction database can be derived in three ways. They are (i) Compressing transaction database (ii) Transforming a transaction database to a compact representation (iii) Partitioning transactions in transaction database and (iv) Merging transactions in transaction database. This section briefly explains each of these techniques.

© 2012 Global Journals Inc. (US)

a) Compressing transaction database

Data compression is a technique that has been widely used to save storage requirement of transactional database in secondary devices. A simple characterization of data compression is that it involves transforming a string of characters in some representation (such as ASCII) into a new string (of bits, for example) which contains the same information but whose length is as small as possible. There are two fundamentally different types of data compression: lossless and lossy. A lossy compression technique removes unwanted data which eventhough degrades the output quality maintains overall important information. Lossless compression technique, on the other hand, attempts to compress data while retaining all information. Examples include, GZIP, BZIP, BOA and PKZip. File systems like NTFS are also used. The main disadvantage of these techniques is that they have to be decompressed before analysis. So, even though the problem of storage requirement is solved, the high transfer rate needed still exists. Thus the problem area still remains unsolved.

b) Transforming a transaction database to a compact representation

Another method used is to use compact data structures to represent the transactions in the database. Prefix tree (Bayardo, 1998) is an ordered tree data structure that is used to store an associative array where the keys are usually strings (items in transaction database). Unlike a binary search tree, no node in the tree stores the key associated with that node; instead, its position in the tree shows what key it is associated with. All the descendants of a node have a common prefix of the string associated with that node, and the root is associated with the empty string. Values are normally not associated with every node, only with leaves and some inner nodes that correspond to keys of interest, thus reducing same required to store a database. Another compression approach using data structure is to "unravel" the data structure into a single byte array (Germann *et al.*, 2009). This approach eliminates the need for node pointers which reduces the memory requirements substantially and makes memory mapping possible which allows the virtual memory manager to load the data into memory very efficiently. Packing the trie (Liang, 1983) is another compression approach used with prefix trees. Liang describes a space-efficient implementation of a sparse packed trie applied to hyphenation, in which the descendants of each node may be interleaved in memory. The FP-growth method uses another compact data structure, FP-tree (Frequent Pattern tree), to represent the conditional databases. FP-tree is a combination of prefix tree structure and node-links (Han *et al.*, 2000).

All these algorithms efficiently reduce the both the secondary and primary storage requirements. The major disadvantage here is the number of scans required during the construction of tree and during frequent pattern mining process.

c) Partitioning transactions in transaction database

A partition is a division of a logical database or its constituting elements into distinct independent parts. Database partitioning is normally done for manageability, performance or availability reasons. Partitioning databases increases performance of regular transactional databases which are done by either building separate smaller databases (each with its own tables, indices, and transaction logs), or by splitting selected attributes of the itemsets. Two types of partitioning algorithms exist. They are horizontal and vertical partitioning. Horizontal partitioning involves putting different rows into different tables. Vertical partitioning involves creating tables with fewer columns and using additional tables to store the remaining columns. Current high end database analysis systems provide different criteria to split the database. Some common criteria are range partitioning, list partitioning, hash partitioning and composite partitioning. All these partitioning schemes, however, require additional computations and while finding frequent patterns, the inter-relationship association between partitions might be missed.

d) Merging transactions in transaction database

Merging transaction approach to create compact transaction database is used as a preprocessing step prior to frequent pattern generation. In this technique, several transactions are merged together to create a new transactional database, which has the following desirable properties.

- Compact representation of the transactional database reduces the problem of memory requirement (both secondary and primary)
- Reduce processing time and I/O time for identifying frequent itemsets and association rules
- Frequent data mining can be performed over the compact representation without losing any mining accuracy
- Can be decompressed to get the original database at any time

As this is the method that is used by this work, the detailed description of the same is provided in the following Section.

III. Proposed Method

This section presents the M^{2}TQT along with the enhanced M^{2}TQT algorithm.

a) M^{2}TQT algorithm

The M^2 TQT algorithm uses a merge Transaction

© 2012 Global Journals Inc. (US)

Scheme (MTS) for producing a compact representation of the transaction database. The MTS is a two-step procedure. Step 1 : Preprocessing and Step 2 : Frequent Pattern Mining. The preprocessing step, uses lexical symbols, to first transform the original raw data into a new data representation. This step is based on the assumption that items in a transaction are sorted in lexicographic order. The second step of preprocessing then sorts all the transactions, groups them and then merges each group into a new transaction. The details on the algorithmic details of merging transaction algorithm, please refer to Dai *et al.* (2008).

b) Enhanced M^2tqt (E-M^2tqt) Algorithm

The general process of E-M^2TQT is shown in Figure 1. It consists of three steps, namely, preprocess, compressing database and frequent pattern mining. The proposed method focus on the problems of repeated database scans and huge number of candidate itemsets generated. The E-M^2TQT algorithm takes advantage of the FP-Tree data structure of FP-Growth algorithm to solve the first problem and the second problem is solved by using a quantification table. The usage of quantification table allows the algorithm to retrieve the original database from the compressed form and prunes irrelevant candidate itemsets, which further reduce the size of the database. Reduction of database indirectly helps to reduce the time required by the mining process. Figure 2 presents the compression algorithm.

The algorithm begins by identifying related transactions and then merging these related transactions together, for which a quantification table is constructed. A transaction T1 is said to be related to transaction T2, if T1 is a subset of T2 or if T2 is a superset of T2. The distance between T1 and T2 is calculated as the difference between the items of two transactions. For example, if T1 = {ABCD} and T2 = {ABC}, the difference (D) between T1 and T2 is 1. Similarly, if T1 = {A} and T2 = {C}, the D is given as 2.

The next step is the creation of quantification table, which is used to record details regarding the transaction relationship. Since the items in a transaction appear in a lexicographical order, the process starts from the left-most item and is termed as a prefix-item. After finding the length of the input transaction (n), for varying lengths (L = 1 .. n), the frequency of count of each itemset appearing in the transactions are recorded. The quantification table has details for each length where information regarding the prefix-item and its frequency count is recorded. Consider for example a transaction database with five items, as shown in Table I.

The quantification process begins by considering the first transaction, {ABCDE} with TID 100 and length = 5. Consider each item one by one in the transaction. Initially, For A, all the five counters L5 to L1 are incremented by 1. Second, for B, the counters L4 to L1 are incremented. Similarly, for C, counters L3 to L1 are incremented, for D, L2 to L1 is incremented and finally, for E, L1 is incremented by 1. Now considering the next transaction {CDE} with TID 200 and leng = 3.

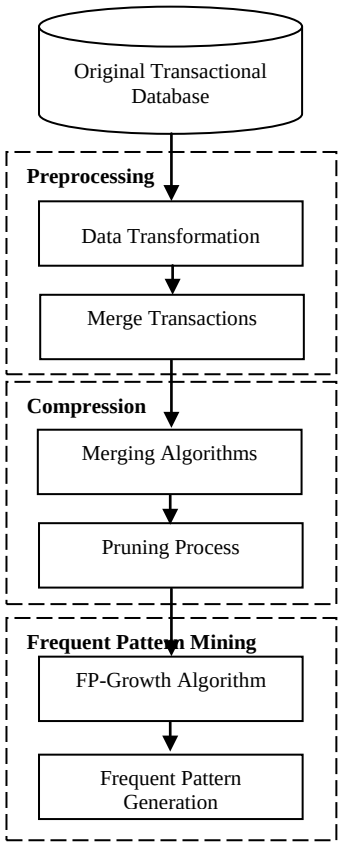


Figure 1 : Proposed Algorithm

© 2012 Global Journals Inc. (US)

```
For I = 1 to number of transactions
        Create Quantification Table
                Quantification table records the length of the transaction database (n) and records the
                count of each itemset appearing in the transaction under each length (L1 – has single
                length, L2 – has two itemset, …)
        Compute length of transaction using quantization table
        Calculate relation distance between transactions and relevant transaction groups
        Relation distance is defined as number of different items between two transactions.
        The smallest transaction of a relevant transaction group is grouped with the longest transaction of
        the same relevant transaction group.
        For d = 1 to n-1
                Consider a transaction, if the relation distance between transaction and merged group = a
                distance 'd', then merge the transactions.
        End for
End for
```

Figure 2 : Compression Algorithm

The counters L3 to L1 is incremented by 1 for C, L2 and L1 are incremented for D and for E, L1 is incremented. Thus, L3 has C2, L2 has D2 and L1 has E2. Finally, for the last transaction, {ACD} (with length 3 and TID 300), A1 is changed to A2 in L3, L2 and L1, C2 is changed to C3 in L2 and L1 and D2 is changed to D3 in L1. The final result is shown in Table II. Now, considering the minimum support, all the candidate itemsets whose counters is less than the minimum support can be pruned out. After pruning, the next step performs the merging process. The merging process is explained with an example below.

Table I : Sample Database

TID	Transactions
100	ABCDE
200	CDE
300	ACD

Let d = 1. Consider two transactions {BCG} and {BG}. After merging these two transactions, the relation transaction group will be {BCG=2.1.2}. Consider another transaction {B}. Compute relation distance for {BCG=2.1.2} and {B}. Since the relation distance is 1, {B} is merged into the relation transaction group and thus becomes {BCG=3.1.2}. The next step calculates the support count of items using a minimum-frequency function. This function takes as input the original transactions and merged transactions and returns the minimum number of itemsets in a transaction. For example, let C2 be {BC, AE} and transaction T* be {{AE=2.1}, {BCG=2.1.2}, {CDEG=2.3.3.1}, {ABCE}, {C}}. The minimum frequency function returns 2 for both BC (0+1+0+1+0=2) and AE (1+0+0+1+0=2), as the number of itemsets in the transactions. These values and the transaction set T* is used by FP-growth algorithm to generate frequent itemsets and association rules.

Table II : Quantification Table

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

c) Decompression Algorithm

The main requirement of any compression algorithm that produces a compact representation of the original transaction database is to reproduce the original database without any loss. The proposed algorithm satisfies this requirement also. The algorithm used for this purpose is shown in Figure 3.

© 2012 Global Journals Inc. (US)

1. Let merged transaction be expressed as <s1, s2… , sk .., sn-1,sn>= 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.
2. Identify the smallest count in ci (i=1..n) whose transaction is the longest transaction. Thus the count of longest transaction is ck. Thus the transaction { s1, s2… , sk , sn-1, sn } is recovered and the merged transaction becomes <s1, s2… , sn-1, sn>= c1-ck.c2-ck… cn-1-ck. cn-ck.
3. Remove items with a zero count from the merged transaction.
4. Repeat the above process to find the next longest transaction in the merged transaction until no count left.

Figure 3 : Decompression Algorithm

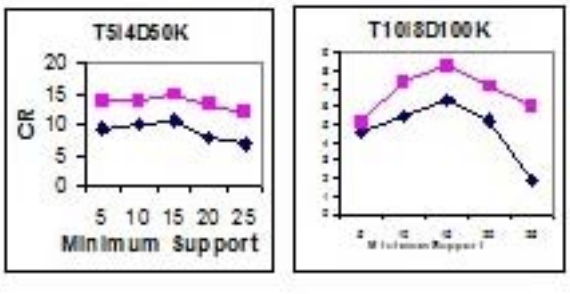
The decompression algorithm is the reverse process of compression algorithm and is explained using an example merged transaction. Consider a merged transaction <ABCD>=3.1.4.2 which has the smallest count of 1 (that is, the count of transaction <ABCD> is 1). Next, decrease the count of each item in <ABCD> by 1 to obtained <ACD>=3-1. 4-1. 2-1 = 2.3.1. Item B is removed since it has a zero count. Repeat this process, to get, <AC>==2-1.3-1=1.2. Finally, <C>=2-1=1. Combining the results we get the decompressed version of the merged transaction <ABCD>=3.1.4.2 as {ABCD}, {ACD}, {AC}, {C}.

IV. Experimental Results

The efficiency of the proposed algorithm was tested using various test data and performance metrics. The proposed model was tested using synthetic dataset as proposed by Agarwal and Srikanth (1994). These transactions mimic the actual transactions in a retail environment. Four synthetic datasets, namely, T5I4D50K, T10I8D100K, T20I10D100K and T20I12D200K were used during experimentation. The proposed algorithm was analyzed using compression ratio in terms of speed, number of transactions and storage required. The average size of the potentially large itemset is taken as 5 and minimum support is varied from 5% to 25% in steps of five. The proposed algorithm was compared with M^2TQT. The algorithms were developed in JAVA with NetBeans 5.5 as front end. All the experiments were conducted on a Pentium IV machine with 512 MB RAM.

a) Compression Ratio with Respect to Storage Space Saved

The compression result in terms of storage size is shown in Figure 4 for the selected four datasets.



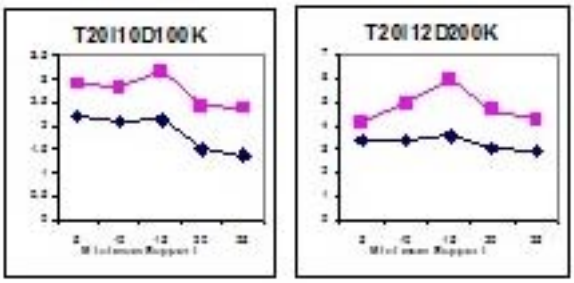
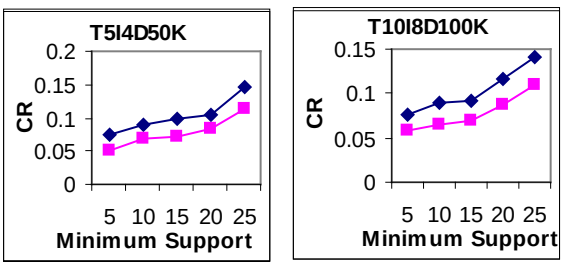


Figure 4 : Compression Ratio For Storage Space Saved

From the results, it could be seen that while both the algorithms are efficient in generating a compact version of the original database, the enhanced M^2TQT shows significant improvement in terms of storage size reduction. The results also show that the E-M^2TQT algorithm scales with different sized datasets. An interesting trend observed is that the maximum amount of compression achieved was when the minimum support is around 15% after which a slow decline in compression rate is envisaged. This means that the maximum performance of the algorithm can be achieved by setting the support value to 15%. Another pattern observed with respect to compression ratio is the relation between size of dataset and average compression achieved. As the size of the dataset increases the compression efficiency decreases. However, the efficiency gain is consistent with all databases which imply that the enhanced approach is better than the traditional algorithm. Thus, the results show that the introduction of FP-Growth algorithm to M^2TQT provides effective data compression.

b) Compression Ratio with Respect to Number of Transactions

Figure 5 shows the compactness efficiency in terms of number of transactions. These experiments were conducted to evaluate the performance of the pruning algorithm introduced in the merging transaction step of the compression algorithm.



© 2012 Global Journals Inc. (US)

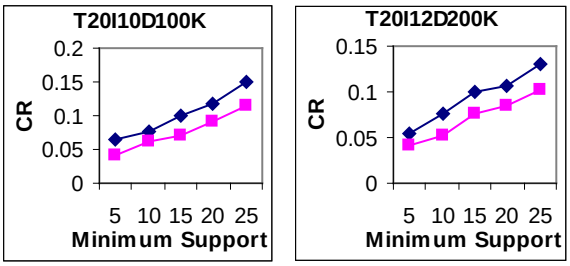


Figure 5 : Compression Ratio For No. Of Transaction

The results of compression ratio with respect to number of transactions again show that the Enhanced M²TQT algorithm is an improved version. While considering various minimum support, the results shows that the minimum support and compression ratio are directly proportional. This is evident from the increasing trend line obtained. While analyzing the scalability of the algorithm, the trend obtained shows that the efficiency is not affected by different sized datasets and remains consistent between 23% and 25%. In conclusion, it could be seen that the pruning algorithm with merging transaction scheme has improved the E-M²TQT algorithm by 25.32% on average.

c) Execution Time

Table III shows the average execution time of each dataset along with the efficiency gain obtained with respect to execution speed. The speed is calculated in seconds.

Table III : Average Execution Speed (Seconds)

DATASET	M²TQT	E-M²TQT	EFFICIENCY (%)
T5I4D50K	1.692	0.970	42.67
T10I8D100K	1.872	1.114	40.49
T20I10D200K	2.222	1.560	29.79
T20I12D300K	2.312	1.840	20.42

As with compression ratio, the E-M²TQT algorithm shows an average 33.34% speed efficiency with respect to M²TQT algorithm. This shows that the speed efficiency has improved in the enhanced version. As expected, the algorithm takes more time to execute large sized datasets than small sized datasets. The small sized datasets (T5I4D50K and T10I8D100K) shows maximum speed efficiency of 42.67% and 40.49%, while it decreased to 29.79% and 20.42% with large sized datasets (T20I10D200K and T20I12D300K).

These results indicate that the performance of E-M²TQT algorithm in terms of compactness achieved with respect to storage size, number of transactions and execution speed with different datasets and different minimum support is efficient when compared with to its existing version.

V. Conclusion

This research work proposed an enhanced transaction database compacting algorithm for frequent mining. Experiments were conducted to analyze the performance of the algorithm. The results showed high performance and there can be reliably used in various applications where pattern mining is needed. Moreover, the speed of the algorithms further makes it suitable for online applications. In future, plans for including frequent pattern-based clustering algorithms and partitioning algorithm to further improve the efficiency of the pattern-mining algorithm are considered.

References Références Referencias

1. Park, J.S., Chen, M.S. and Yu, P.S. (1995) An effective hash-based algorithm for mining association rules. Proceeding of the 1995 ACM-SIGMOD International Conference on Management of Data (SIGMOD'95), San Jose, CA, Pp 175–186
2. Agrawal R. and Shafer, J.C. (1996) Parallel mining of association rules: design, implementation and experience, IEEE Transactions Knowledge Data Engineering, Vol. 8, Pp.962–969.
3. Gionis, A., Mannila, H., Mielikainen, T. and Tsaparas, P. (2007) Assessing Data Mining Results via Swap Randomization, ACM Transactions on Knowledge Discovery from Data (TKDD), Vol. 1 , Issue 3, Article No. 14.
4. Dai, J.Y., Yang, D.L., Wu, J. and Hung, M.C. (2008) An Efficient Data Mining Approach on Compressed Transactions, International Journal of Electrical and Computer Engineering, Vol. 3, No. 2, Pp. 76-83
5. Bayardo, R.J. (1998) Efficiently mining long patterns from databases, Proceeding of the 1998 ACM-SIGMOD International Conference on Management of Data (SIGMOD'98), Seattle, WA, Pp 85–93.
6. Germann, U., Joanis, E. and Larkin, S. (2009) Tightly packed tries: how to fit large models into memory and make them load fast, too, ACL Workshops: Proceedings of the Workshop on Software Engineering, Testing and Quality Assurance for Natural Language Processing, Association for Computational Linguistics, Pp. 31–39.
7. Liang, F.M. (1983). Word Hy-phen-a-tion By Com-put-er, Doctor of Philosophy Thesis, Stanford University.
8. Han, J., Pei, J. and Yin, Y. (2000) Mining frequent patterns without candidate generation, Proceeding of the 2000 ACM-SIGMOD International Conference on Management of Data (SIGMOD'00), Dallas, TX, Pp. 1–12.
9. Agrawal, R. and Srikant, R. (1994) Fast Algorithms for Mining Association Rules, Proceedings of the 20th International Conference on Very Large Data Bases, Pp. 487-499.

© 2012 Global Journals Inc. (US)