A Fast Association Rule Algorithm Based on Bitmap Computing with Multiple Minimum Supports using Maximum Constraints

Jyothi Patil, and Dr. V.D.Mytri

***Abstract*—**Mining association rules from databases is a time-consuming process. Finding the large item set fast is the crucial step in the association rule algorithm. In this paper we present a algorithm based on the Apriori approach to find the large-itemsets and association rules under the constraint multiple minimum supports using maximum constraint. Most of the previous approaches set a single minimum support threshold for all the items or itemsets. But in real applications, different items may have different criteria to judge its importance. The support requirements should then vary with different items. This algorithm also doesn't follow the generation-and-test strategy of Apriori algorithm and adopts the divide-and-conquer strategy, thus avoids the time-consuming table scan to find and prune the itemsets, all the operations of finding large itemsets from the datasets are the fast bit operations based on its corresponding granular. The experimental result of our algorithm with Apriori, AprioriTid and AprioirHybrid algorithms shows this algorithm is 2 to 3 orders of magnitudes faster. Our research indicates that multiple minimum supports using maximum constraint and granular computing. can greatly improve the performance of association rule algorithm, and are very promising for data mining applications.

***Keywords*—** Data mining, Multiple minimum supports, Association rule, Maximum constraint, Granular computing.

I. Introduction

CONSIDER a supermarket with a large collection of items. Typical business decisions that the management of the supermarket has to make include what to put on sale, how to design coupons, how to place merchandise on shelves in order to maximize the profit, etc. Analysis of past transaction data is a commonly used approach in order to improve the quality of such decisions. Until recently, however, only global data about the cumulative sales during some time period (a day, a week, a month, etc.) was available on the computer. Progress in bar-code technology has made it possible to store the so called basket data that stores items purchased on a per-transaction, basis. Basket data type transactions do not necessarily consist of items bought together at the same point of time. It may consist of items bought by a customer over a period of time. Examples include monthly purchases by members of a book club or a music club.Several organizations have collected massive amounts of such data. These data sets are usually stored on tertiary storage and are very slowly migrating to database systems. One of the main reasons for the limited success of database systems in this area is that current database systems do not provide necessary functionality for a user interested in taking advantage of this information.

In this study we considered Association Rule Mining for knowledge discovery and generate the rules by applying our developed approach on real databases. Mining Associations is one of the techniques involved in Data mining. Discovering association rules is at the heart of data mining. Mining for association rules between items in large database of sales transactions has been recognized as an important area of database research [1]. These rules can be effectively used to uncover unknown relationships, producing results that can provide a basis for forecasting and decision making. The original problem addressed by association rule mining was to find a correlation among sales of different products from the analysis of a large set of supermarket data. Today, research work on association rules is motivated by an extensive range of application areas, such as banking, manufacturing, health care, and telecommunications. It is also used for building statistical thesaurus from the text databases [2], finding web access patterns from web log files [3], and also discovering associated images from huge sized image databases [4].

A number of association rule mining algorithms have been developed in the last few years [5, 6, 7, 8, 9, 10], which can be classified into two categories: (a) candidate generation/test approach such as Apriori [6] (b) pattern growth approach [9, 10]. A milestone in the first category studies is the development of an Apriori based, level wise mining method for associations, which has sparked the development of various kinds of Apriori like association mining algorithms. Among these, the Apriori algorithm has been very influential. Since its inception, many scholars have improved and optimized the Apriori algorithm and have presented new Apriori like algorithms [11, 12, 13, 14, 15]. The Apriori like algorithms adopt an iterative method to discover frequent

Jyothi Patil is Research Scholar of CM University, Karnataka, India. Email- jyothip_pda2003@yahoo.com.

Dr V.D.Mytri is with the Electronics Engineering Department, Shetty Institute of Technology, Karnataka, India Email: mytri@gmail.com.

itemsets.

The existing mining algorithms have some drawbacks: Firstly, the existing mining algorithms are mostly designed in forms of several passes so that the whole database needs to be read from disks several times for each user's query under the constraint that the whole database is too large to be stored in memory. This is very inefficient in considering the big overhead of reading the large database even though only partial items are interested in fact. As a result, they cannot perform efficiently in terms of responding the user's query quickly. Secondly, in many cases, the algorithms generate an extremely large number of association rules, often in thousands or even millions. Further, the association rules are sometimes very large. It is nearly impossible for the end users to comprehend or validate such large number of complex association rules, thereby limiting the usefulness of the data mining results. Thirdly, no guiding information is provided for users to choose suitable settings for the constraints such as support and confidence such that an appropriate number of association rules are discovered. Consequently, the users have to use a try and -error approach to get suitable number of rules. This is very time consuming and inefficient. Therefore, one of the main challenges in mining association rules is developing fast and efficient algorithms that can handle large volumes of data. In this paper we attack the association rule mining by an apriori based approach to find the large-itemsets and association rules under the constraint multiple minimum supports using maximum constraint and uses bitmap computing to speed up the algorithm ,specifically designed for the optimization in very large transactional databases.

The rest of the paper is organized as follows. We introduce the theoretical properties and formal definition of association rule in section 2. Section 3, presents some related work. . In Section 4, the proposed method is described in detail. Section 5 presents comparisons and experiments results of our proposed approach with various Apriori algorithms. Finally, a conclusion and the future work are given in Section 6.

II. Association Rules Mining

Association rule of data mining involves picking out the unknown inter-dependence of the data and finding out the rules between those items .Let I = I1; I2; . . .; Im be a set of binary attributes, called items. Let T be a database of transactions. Each transaction t is represented as a binary vector, with t[k] = 1 if t bought the item Ik, and t[k] = 0 otherwise. There is one tuple in the database for each transaction. Let X be a set of some items in I. We say that a transaction t satisfies X if for all items Ik in X, t[k] =1. A rule is defined as an implication of the form A=>B where X⊂ I, Y⊂ I, and A∩ B≠Ø. The left-hand side of the rule is called as antecedent. The right-hand side of the rule is called as consequent. For example the rule { Onions, Potatoes}=>{beef} found in the sales data of a supermarket would indicate that if a customer buys Onions and potatoes together then the customer is likely to buy beef also. Such information is useful to make decisions about marketing activities. Association rules are also used in many applications including Web usage Mining, Intrusion Detection and Bio-informatics.

The selection of association rule is based on support and confidence. The confidence factor indicates the strength of the implication rules, i.e. the confidence for an association rule is the ratio of the number of transactions that contain X U Y to the number of transactions that contain X; whereas the support factor indicates the frequencies of the occurring patterns in the rule. i.e., the support for an association rule is the percentage of transactions in the database that contain X U Y. Given the database D, the problem of mining association rules involves the generation of all association rules among all items in the given database D that have support and confidence greater than or equal to the user specified minimum support and minimum confidence. Typically large confidence values and a smaller support are used. Rules that satisfy both minimum support and minimum confidence are called strong rules. Since the database is large and users concern about only those frequently purchased items, usually thresholds of support and confidence are predefined by users to drop those rules that are not so interesting or useful.

The discovery of association rules for a given dataset D is typically done in two steps [5, 16]: discovery of frequent itemsets and the generation of association rules. The first step is to find each set of items, called as itemsets, such that the co-occurrence rate of these items is above the minimum support, and these itemsets are called as large itemsets or frequent itemsets. In other words, find all sets of items (or itemsets) that have transaction support above minimum support. Finding large itemsets is generally very easy but very costly. The naive approach would be to count all itemsets that appear in any transaction. Suppose one of the large itemsets is Lk, Lk = {I1, I2, …, Ik}, association rules with this itemsets are generated in the following way: the first rule is {I1, I2, … , Ik1} ⇒ {Ik}, by checking the confidence this rule can be determined as interesting or not. Then other rule are generated by deleting the last items in the antecedent and inserting it to the consequent, further the confidences of the new rules are checked to determine the interestingness of them. Those processes iterated until the antecedent becomes empty.

The size of an itemset represents the number of items in that set. If the size of an itemset is equal to k, then this itemset is called as the k itemset. The second step is to find association rules from the frequent itemsets that are generated in the first step. The second step is rather straightforward. Once all the large itemsets are found generating association rules is straightforward. The first step dominates the processing time and for that reason, it has been one of the most popular research fields in data mining. So, focus has been made in this paper on the first step.

III. Previous Work

The problem of discovering association rules was first

introduced in [5] and an algorithm called AIS was proposed for mining association rules. For last fifteen years many algorithms for rule mining have been proposed. Most of them follow the representative approach by Agrawal et al. [16], namely Apriori algorithm. Various researches were done to improve the performance and scalability of Apriori included using parallel computing. There were also studies to improve the speed of finding large itemsets with hash table, map, and tree data structures. Here we review some of the related work that forms a basis for our algorithm.

In AIS algorithm[5] candidate itemsets are generated and counted on-the-fly as the database is scanned. After reading a transaction, it is determined which of the itemsets that were found to be large in the previous pass are contained in this transaction. New candidate itemsets are generated by extending these large itemsets with other items in the transaction. A large itemset l is extended with only those items that are large and occur later in the lexicographic ordering of items than any of the items in l. The candidates generated from a transaction are added to the set of candidate itemsets maintained for the pass, or the counts of the corresponding entries are increased if they were created by an earlier transaction.

The main drawback of the AIS algorithm is that it makes multiple passes over the database. Furthermore, it generates and counts too many candidate itemsets that turn out to be small, which requires more space and wastes much effort that turned out to be useless.

In SETM algorithm [17] was motivated by the desire to use SQL to compute large itemsets. Like AIS, the SETM algorithm also generates candidates on-the-fly based on transactions read from the database. It thus generates and counts every candidate itemset that the AIS algorithm generates. However, to use the standard SQL join operation for candidate generation, SETM separates candidate generation from counting.It saves a copy of the candidate itemset together with the TID of the generating transaction in a sequential structure. At the end of the pass, the support count of candidate itemsets is determined by sorting and aggregating this sequential structure.SETM remembers the TIDs of the generating transactions with the candidate itemsets. To avoid needing a subset operation, it uses this information to determine the large itemsets contained in the transaction read. $(L_k)^- \subseteq (C_k)^-$ and is obtained by deleting those candidates that do not have minimum support.Assuming that the database is sorted in TID order, SETM can easily find the large itemsets contained in a transaction in the next pass by sorting $(L_k)^-$ on TID. n fact, it needs to visit every member of $(L_k)^-$ only once in the TID order, and the candidate generation can be performed using the relational merge-join operation[17]. The disadvantage of this approach is mainly due to the size of candidate sets $(C_k)^-$. For each candidate itemset, the candidate set now has as many entries as the number of transactions in which the candidate itemset is present. Moreover, when we are ready to count the support for candidate itemsets at the end of the pass, $(C_k)^-$ is in the wrong order and needs to be sorted on itemsets. After counting and pruning out small candidate itemsets that do not have minimum support, the resulting set $(L_k)^-$ needs another sort on TID before it can be used for generating candidates in the next pass.

The Apriori [16] and AprioriTid algorithms generate the candidate itemsets to be counted in a pass by using only the itemsets found large in the previous pass without considering the transactions in the database. The basic intuition is that any subset of a large itemset must be large. Therefore, the candidate itemsets having k items can be generated by joining large itemsets having k-1 items, and deleting those that contain any subset that is not large. This procedure results in generation of a much smaller number of candidate itemsets. The AprioriTid algorithm has the additional property that the database is not used at all for counting the support of candidate itemsets after the first pass. Rather, an encoding of the candidate itemsets used in the previous pass is employed for this purpose. In later passes, the size of this encoding can become much smaller than the database, thus saving much reading effort. Another algorithm, called Apriori Hybrid, is introduced in [16]. The basic idea of the Apriori Hybird algorithm is to run the Apriori algorithm initially, and then switch to the AprioriTid algorithm when the generated database, i.e. large k itemset in the transaction with identifier TID, would fit in the memory.

The DHP (Direct Hashing and Pruning) algorithm [12] is an effective hash based algorithm for the candidate set generation. It reduced the size of candidate set by filtering any k itemset out of the hash table if the hash entry does not have minimum support. The hash table structure contains the information regarding the support of each itemset. The DHP algorithm consists of three steps. The first step is to get a set of large 1-itemsets and constructs a hash table for 2itemsets. The second step generates the set of candidate itemsets Ck based on the hash table generated in previous pass, determines the set of large k-itemsets Lk, reduces the size of database for the next large itemsets and makes a hash table for candidate large (k+1)-itemsets . The third step is the same as the second step except it does not use the hash table in determining whether to include a particular itemset into the candidate itemsets. This step is used for later iterations when the number of hash buckets with a support count greater than or equal to the minimum transaction support required is less than a predefined threshold.

The partition algorithm[18] executes in two phases. In the first phase, the Partition algorithm logically divides the database into a number of non-overlapping partitions. The partitions are considered one at a time and all large itemsets for that partition are generated. At the end of phase I, these large itemsets are merged to generate a set of all potential large itemsets. In phase II, the actual support for these itemsets are generated and the large itemsets are identified. The partition sizes are chosen such that each partition can be accommodated in the main memory so that the partitions are read only once in each phase. This approach will work only if

the given set contains all actual large itemsets.

The frequent pattern growth[19] algorithm uses compact data structure, called frequent-pattern tree, or FP-tree in short, which is an extended prefix-tree structure storing crucial, quantitative information about frequent patterns. Every transaction database is mapped onto one path in the FP tree, and the frequent itemset information is completely stored in the tree .Since there are often a lot of sharing of frequent items among transactions the size of the tree is much smaller than its original database. Subsequent frequent-pattern mining will only need to work on the FP-tree instead of the whole data set.

Sampling large databases[20] for association rules uses a- A Fast Association Rule Algorithm Based On Bitmap Computing with Multiple Minimum Supports Using Maximum Constraints random sample, to determine all association rules that probably hold in the whole database, and then to verify the results with the rest of the database. The algorithms thus produce exact, association rules in one full pass over the database. In those rare cases where sampling method does not produce all association rules, the missing rules can be found in a second pass.

Mining association rules with multiple minimum [14] proposed an approach for mining association rules with nonuniform minimum support values. Their approach allowed users to specify different minimum supports to different items. The minimum support value of an itemset is defined as the lowest minimum supports among the items in the itemset. Wang et al. [22] then generalized the above idea and allowed the minimum support value of an itemset to be any function of the minimum support values of items contained in the itemset. Although their approach is flexible in assigning the minimum supports to itemsets, the mining algorithm is a little complex due to its generality.

IV. Proposed Approach

In the proposed algorithm, items may have different minimum supports and the maximum constraint is adopted in finding large itemsets. That is, the minimum support for an itemset is set as the maximum of the minimum supports of the items contained in the itemset. Under the constraint, the characteristic of level-by-level processing is kept, such that the original Apriori algorithm can be easily extended to find the large itemsets. The proposed algorithm first finds all the large 1-itemsets L1 for the given transactions by comparing the support of each item with its predefined minimum support. After that, candidate 2-itemsets C2 can be formed from L1. Note that the supports of all the large 1-itemsets comprising each candidate 2-itemset must be larger than or equal to the maximum of the minimum supports of them. This feature provides a good pruning effect before the databaseis scanned for finding large 2-itemsets.The proposed algorithm then finds all the large 2-itemsets L2 for the given transactions by comparing the support of each candidate 2-itemset with the maximum of the minimum supports of the items contained in it. The same procedure is repeated until all large itemsets have been found. The details of the proposed mining algorithm under the maximum constraint are described below.

An Example: In this section, an example is given to demonstrate the proposed data-mining algorithm. This is a simple example to show how the proposed algorithm can be used to generate association rules from a set of transactions with different minimum support values defined on different items. Assume the ten transactions shown in Table I are used for mining.

TABLE I
THE SET OF TEN TRANSACTION DATA

TID	Items
1	ABDG
2	BDE
3	ABCEF
4	BDEG
5	ABCEF
6	BEG
7	ACDE
8	BE
9	ABEF
10	ACDF

TABLE II
THE PREDEFINED MINIMUM SUPPORT VALUES FOR ITEMS.

Item	A	B	C	D	E	F	G
Min-Sup	0.4	0.7	0.3	0.7	0.6	0.2	0.4

TABLE III
THE SUPPORT VALUES OF ALL THE ITEMS FOR THE GIVEN TEN TRANSACTIONS.

Item	A	B	C	D	E	F	G
Support	0.6	0.8	0.4	0.5	0.9	0.3	0.3

Each transaction consists of two features, transaction identification (TID) and items purchased. Also assume that the predefined minimum support values for items are defined in Table 2.Moreover, the confidence value λ is set at 0.85 to be a threshold for the interesting association rules.In order to find the association rules from the data in Table I with the multiple predefined minimum support values, the proposed mining algorithm proceeds as follows:

STEP 1: The count and support of each item occurring in the ten transactions in Table I are to be found. Take item A as an example. The count of item A is 6, and its support value is calculated as 6/10 (=0.6). The support values of all the items for the ten transactions are shown in Table III.

STEP 2: The support value of each item is compared with its predefined minimum support value. Since the support values of items A, B, C,E and F are respectively larger than or equal to their predefined minimum supports, these five items are then put in the large 1-itemsets L1.

TABLE IV
THE SUPPORT VALUES OF ALL THE CANDIDATE 2-ITEMSETS.

2-Itemset	A,C	A,E	B,E	C,F
Support	0.4	0.5	0.7	0.2

STEP 3: r is set at 1, where r is used to keep the current number of items in an itemset.

STEP 4: The candidate set C2 is generated from L1, and the supports of the two items in each itemset in C2 must be larger than or equal to the maximum of their predefined minimum support values. Take the possible candidate 2-itemset {A,C} as an example. The supports of items A and C are 0.6 and 0.4 from STEP 1, and the maximum of their minimum support values is 0.4. Since both of the supports of these two items are larger than 0.4, the itemset {A,C} is put in the set of candidate 2-itemsets. On the contrary for another possible candidate 2-itemset {A, B}, since that the support (0.6) of item A is smaller than the maximum (0.7) of their minimum support values, the itemset {A, B} is not a member of C2. All the candidate 2-itemsets generated in this way are found as:C2 = {{A,C}, {A,E}, {B,E}, {C, F}}.

STEP 5: The count and support of each candidate itemset in C2 are found from the given transactions. Results are shown in Table IV.

STEP 6: The support value of each candidate 2-itemset is then compared with the maximum of the minimum support values of the items contained in the itemset. Since the support values of all the candidate 2-itemsets {A,C} and {B,E} satisfy the above condition, these four itemsets are then put in the set of large 2-itemsets L2.

STEP 7: Since L2 is not null, r is set at 2 and STEPS 4 to 7 are repeated. No candidate 3-itemset, C3, is generated and L3 is thus null. The next step is then executed.

STEP 8: The association rules for each large q-itemsets, q ≥ 2, are constructed by the following substeps:

(a) All possible association rules are formed as follows:

(1) "If A is bought, then C is bought";
(2) "If C is bought, then A is bought";
(3) "If B is bought, then E is bought";
(4) "If E is bought, then B is bought";

(b) The confidence factors of the above association rules are calculated.

Take the first possible association rule "If A is bought,then C is bought" as an example. The confidence factor for this rule is then:

$$\frac{S_{AnC}}{S_A} = \frac{0.4}{0.6} = 0.67$$

Results for all the four association rules are shown as follows:

(1) "If A is bought, then C is bought" with a confidence factor of 0.67;
(2) "If C is bought, then A is bought" with a confidence factor of 1.0;
(3) "If B is bought, then E is bought" with a confidence factor of 0.875;
(4) "If E is bought, then B is bought" with a confidence factor of 0.875;

STEP 9: The confidence factors of the above association rules are compared with the predefined confidence threshold λ. Assume the confidence λ is set at 0.85 in this example. The following four rules are thus output:

(1) "If C is bought, then A is bought" with a confidence factor of 1.0;
(2) "If B is bought, then E is bought" with a confidence factor of 0.875;
(3) "If E is bought, then B is bought" with a confidence factor of 0.875;

In this example, three large q-itemsets, q ≥ 2, and three association rules are generated. Note that if the transactions are mined using the minimum constraint proposed in [21], eighteen large q-itemsets, q ≥2, are found. The proposed mining algorithm using the maximum constraint thus finds less large itemsets and association rules than that using the minimum constraint. The proposed algorithm can, however, find the large itemsets level by level without backtracking. It is thus more time-efficient than that with the minimum constraint.

Lin's [23] approach can easily be used in our algorithm for mining from a transaction database. An item or an itemset is regarded as an equivalence class (a granule). If a transaction contains a certain item, the transaction then belongs to the equivalence class of the item and the corresponding bit in its granular representation is set at 1. The granular representation for the data in Table 1 is shown in Table V.

TABLE V
THE GRANULAR REPRESENTATION

Item	Equivalence Class	Granular Representation	Count
A	{TID1, TID3, TID5, TID7, TID9, TID10}	{1010101011}	6
B	{TID1, TID2, TID3, TID4, TID5, TID6, TID8, TID9}	{1111110110}	8
C	{TID3, TID5, TID7, TID10}	{0010101001}	4
D	{TID1, TID2, TID4, TID7, TID10}	{1101001001}	5
E	{TID2, TID3, TID4, TID5, TID6, TID7, TID8, TID9, TID10}	{0111111111}	9
F	{ TID3, TID5, TID9}	{0010100010}	3
G	{TID1, TID4, TID6}	{1001010000}	3

In this example, A, B, C, E and F are large 1-itemsets. According to our algorithm, the candidate 2-itemsets are found as: C2 = {{A,C}, {A,E}, {B,E},{C, F}}. The boolean AND operation can then be used to form the bit patterns of the 2-itemsets. The results are shown in Table VI.Since the two 2-itemsets, {A,C} and {B,E}, have their supports larger than the support constraint, they are then put in the large 2-itemsets. Itemsets with more items can be formed in the similar way.

TABLE VI
USING THE BOOLEAN AND OPERATION TO FIND THE GRANULES FOR C2

2-Item	Granular operation	Granular Representation	Count
A AND C	{1010101011} \ {0010101001}	0010101001	4
A AND E	{1111110110} \ {0111111111}	0010101011	5

B AND E	{1111110110} \ {0111111111}	0111110110	7
C AND F	{0010101001} \ {0010100010}	0010100000	2

V. Experimental Results

Three different data are used to compare the run time on the algorithms to find association rules. The data varies in the number of tuples, the attributes per relation and the minimal support. The tests were conducted using an IBM PC with 3.4GHz CPU,1 GB RAM under WindowXP. The program is coded in C++ for Apriori, AprioriTid and AprioriHybrid

Data Set	Rows	Columns	# Of Items	Table Size	Bitmap Size	Minimal Support
DS1	400k	16	185	24.6MB	10MB	20k
DS2	800k	18	238	62MB	24.5MB	40k
DS3	1MB	26	689	120MB	90MB	50k

TABLE VII
Data Sets

Length	# Cand	# Itemset	Proposed algorithm	Apriori_Hybrid	Apriori_Tid	Apriori
DS1						
1	190	185	3.856s	4.012s	4.012s	4.105s
2	16280	98	17.426s	1498.965s	1682.165s	1397.268s
3	89	08	0.108s	1.737s	79.344s	5.876s
Total Time			21.120s	1504.714s	1765.521s	1407.241s
DS2						
1	242	224	9.262S	10.012S	10.245S	10.278S
2	2523	79	55.891S	4484.263 s	5298.439S	4489.405S
3	0	0	0S	0.01S	0.01s	0s
Total Time			65.153s	4494.285s	5308.694s	4499.683s
DS3						
1	699	700	21.7881 s	20.089s	19.278s	19.298s
2	240542	769	645.973 s	51459.742s	62252.43s	51212.081 s
3	740	39	32.164s	814.786s	952.786s	936.152s
4	0	0	0.014s	0.014s	0.014s	0.014s
Total Time			699.939 s	52294.631s	63224.508s	52167.482 s

Here are some observations and explanations on the results

(1) The total time of our comparison includes the time to write the association rules to a file; proposed algorithm is 2 to 3 orders of magnitude faster than the various Apriori algorithms (64-221 times faster). The big the test data set, the big the time difference between the proposed algorithm and the various Apriori algorithms.

(2) Proposed algorithm takes the same or litter longer time than the various Apriori algorithms in constructing the *1*-itemsets because of the extra cost of building the bitmaps for the *1*-itemsets. But after the 1-itsemtset is done, proposed algorithm is significant faster than the Apriori algorithms in constructing large frequent itemsets because it only uses the fast bit operations (AND,COUNT and SHIFT)and doesn't need to test the subsets of the newly candidate

(3) Proposed algorithm only stores the bitmaps of the frequent items, and the bitmap storage (uncompressed) is less than the original data set (1/2 to 1/4 of the original data size).

The main reasons that proposed algorithm is significant faster than Apriori and its variations are

(1) Proposed algorithm adopts the divide-and-conquer strategy,the transaction is decompose into vertical bitmap format and leads to focused search of smaller domain.There is no repeated scan of entire database in proposed algorithm.

(2) Proposed algorithm doesn't follow the traditional candidate-generate-and test approach, thus saves significant amount of time to test the candidates

(3) In proposed algorithm, the basic operations are bit Count and bit And operations, which are extremely faster than the pattern search and matching operations used in Apriori and its variations

VI. Conclusions

We present a fast association rule algorithm in which the minimum support for an itemset is set as the maximum of the minimum supports of the items contained in the itemset. The proposed mining algorithm using the maximum constraint thus finds less large itemsets and association rules than that using the minimum constraint. To speed up the algorithm granular technique is used. Proposed algorithm avoids the time-consuming table scan to find and prune the itemsets, all the operations of finding large itemsets from the datasets are the fast bit operations. The experimental result of our proposed algorithm with Apriori, AprioriTid and AprioirHybrid algorithms shows proposed algorithm is 2 to 3 orders of magnitude faster. This research indicates that bitmap and granular computing techniques can greatly enhance the performance for finding association rule, and bitmap techniques are very promising for the decision support query optimization and data mining applications. Paralleling the bitmap-based algorithms is another crucial aspect of DSS and data mining performance.

References

[1] Han and M. Kamber, "Data Mining: Concepts and Techniques", Morgan Kaufmann Publishers, 2000.

[2] J. D. Holt and S. M. Chung, "Efficient Mining of Association Rules in Text Databases" CIKM'99, Kansas City, USA, pp.234242, Nov. 1999.

[3] B. Mobasher, N. Jain, E.H. Han, and J. Srivastava, "Web Mining: Pattern Discovery from World Wide Web Transactions", Department of Compute Science, University of Minnesota, Technical Report TR96-050, (March, 1996).

[4] C. Ordonez, and E. Omiecinski, "Discovering Association Rules Based on Image Content", IEEE Advances in Digital Libraries (ADL'99), 1999.

[5] R.Agrawal, T. Imielinski, and A. Swami, "Mining association rules between sets of items in large databases", In Proceedings of the ACM

SIGMOD International Conference on Management of Data (ACM SIGMOD '93), pages 207216, Washington, USA, May 1993.

[6] R.Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo, "Fast discovery of association rules", In Advances in Knowledge Discovery and Data Mining, pages 307328. AAAI Press, 1996.

[7] R. Bayardo and R. Agrawal, "Mining the most interesting rules", In Proceedings of the 5th International Conference on Knowledge Discovery and Data Mining (KDD '99), pages 145154, San Diego, California, USA, August 1999.

[8] J. Hipp, U. Güntzer, and U. Grimmer, "Integrating association rule mining algorithms with relational database systems", In Proceedings of the 3rd International Conference on Enterprise Information Systems (ICEIS 2001), pages 130137, Setúbal, Portugal, July 710 2001.

[9] R. Ng, L. S. Lakshmanan, J. Han, and T. Mah, "Exploratory mining via constrained frequent set queries", In Proceedings of the 1999 ACM-SIGMOD International Conference on Management of Data (SIGMOD '99), pages 556558, Philadelphia, PA, USA, June 1999.

[10] Y. Guizhen, "The complexity of mining maximal frequent itemsets and maximal frequent patterns", Proceedings of the 2004 ACM SIGKDD international conference on Knowledge discovery and data mining, pages 343353, August 2004, Seattle, WA, USA.

[11] L. Klemetinen, H. Mannila, P. Ronkainen, et al. (1994) "Finding interesting rules fromlarge sets of discovered association rules", Third International Conference on Information and Knowledge Management pp. 401407, Gaithersburg, USA.

[12] J. S. Park, M.S. Chen, and P.S. Yu, "An Effective HashBased Algorithm for Mining Association Rules", Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose, CA, USA, 1995, 175186.

[13] H. Toivonen, "Sampling large databases for association rules", 22nd International Conference on Very Large Data Bases, pp. 134–145, 1996.

[14] P. Kotásek and J. Zendulka, "Comparison of Three Mining Algorithms for Association Rules", Proc. of 34th Spring Int. Conf. on Modelling and Simulation of Systems (MOSIS'2000), Workshop Proceedings Information Systems Modelling (ISM'2000), pp. 8590. Rožnov pod Radhoštěm, CZ, MARQ. 2000.

[15] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns Candidate generation", In Proc. 2000 ACMSIGMOD Int. Management of Data (SIGMOD'00), Dallas, TX. 2000.

[16] R. Agrawal , and R. Srikant, "Fast Algorithms for Mining Association Rules in Large Databases," Prof. 20th Int'l Conf. Very Large Data Bases, pp. 478499, 1994.

[17] M. Houtsma and A. Swami, Set-oriented mining of association rules, Research Report RJ 9567, IBM Almaden Research Center, San Jose, California, October 1993.

[18] A. Savasere, E. Omiecinski, and S. Navathe, "An Efficient Algorithm for Mining Association Rules in Large Databases", Proceedings of 21th International Conference on Very Large Data Bases (VLDB'95 September 1115, 1995, Zurich, Switzerland, Morgan Kaufmann, 1995, 432444.

[19] J. Han and J Pei, "Mining frequent patterns by pattern growth: methodology and implications", ACM SIGKDD Explorations Newsletter 2, 2, 1420. 2000.

[20] H. Toivonen, "Sampling large databases for association rules," In 22nd VLDB Conference, 1996.

[21] B. Liu, W. Hsu, Y. Ma, "Mining association rules with multiple minimum supports," The 1999 International Conference on Knowledge Discovery and Data Mining, 1999, pp. 337-341

[22] K. Wang, Y. H, J. Han, "Mining frequent itemsets using support constraints," In Proceedings of the 26th International Conference on Very Large Data Bases, 2000, pp.43-52

[23] T.Y. Lin, "Mining: granular computing approach," The Third Pacific-Asia Conference on Knowledge Discovery and Data Mining, Beijing, 1999, pp.24-33.