Tabular Method For Frequent Itemset Generation In Large Database

Farhan Meer 1, Dr. R. R. Sedamkar 2
1Dept of Computer Engineering,
Thakur College of Engineering and Technology,
Mumbai, India
2Dean Academics
Thakur College of Engineering and Technology,
Mumbai, India

Abstract
Mining for association rules involves extracting patterns from large database and inferring useful information from these. This has been described as an important mining problem in large database of sales transactions. The Apriori algorithm has been implemented for mining association rules for quite some time. However, it has several disadvantages like I/O intensive, inefficient CPU usage while counting; number of database scans is initially unknown. Most of the early algorithms require an unpredictable large number of I/O operations, reducing I/O cost has been the primary target of the algorithm present in the literature. They also require scanning of the entire database at least or almost twice in the worst case. In existing work, partition algorithm is a well-known algorithm for generating frequent itemsets, in which the data is partitioned randomly. And then Tid list data representation has been used for generating frequent item sets. In the best case, number of database scans needed is 1 and in the worst case it is 2. Through this work, an efficient algorithm for mining frequent item sets by using Tabular method which removes the overhead maintenance Tid lists and improves the performance. This algorithm finds the frequent item sets by dividing the database transactions randomly into various partitions. Further each partition finds the frequent item sets using a tabular method which reduces the maximum number of scans to less than 2 and in most of the cases finds all association rules in about 1 scan. As result of which the complexity of proposed algorithm is O(n) as compared to that of partition algorithm which is O(n^2). Extensive experiments have been performed and the corresponding results prove that the proposed algorithm is better than previous algorithms.

Index Terms: Frequent Itemset Generation, Association Rule, Tabular Method, Database, Optimization

1 Introduction
Business organizations are increasingly turning to the automatic extraction of information from large volumes of routinely collected business data. Such high level inference process may provide a host of useful information on customer groups, buying patterns, stock trends etc. This process of automatic
information inferring is commonly known as Knowledge Discovery in Databases (KDD). Data mining is the analysis step of KDD, an interdisciplinary subfield of computer science, is the computational process of discovering patterns in large data sets involving methods at the intersection of artificial intelligence, machine learning, statistics and database systems. The overall goal of the data mining process is to extract information from a data set and transform it into an understandable structure for further use. Recently, data mining has attracted much attention in database community [1]. One of the most investigated topics in data mining is problem of discovering association rules over basket data [1,8].

Association rule problem were first introduced by Agrawal and others in 1993 [2] for supermarket data. Basket data typically contains items purchased by the customers along with the date, quantity, price etc. Such data may be collected for e.g. at supermarket checkout counters. Association rules identify the set of items that are most often purchased with another set of items. For e.g. an association rule may state that 80 % of customers who bought items P and Q also bought R and S. Association rules may be used for catalogue design, store layout, product placement, target marketing etc. Due to the usefulness of association mining, it has been applied to other databases as well, for e.g. telecommunication alarm data and university course enrolment data [3].

Several sequential and parallel algorithms have been pro-posed in the literature to find association rules [4],[3]. One of the key features of all the previous algorithms is that they require multiple passes over the database. For disk resident database, this requires reading of the entire database multiple number of times resulting in large no of disk I/Os. Apart from poor response time, this process also places huge burden on the I/O subsystem adversely affecting other users of the system. Since, the sequential algorithms developed so far have many limitations and are not suitable for massive database, there is a great need of parallel algorithms for achieving high performance. There are various challenges in exploiting parallelism like effective load balancing, efficient utilization of memory, synchronization between the processors, and minimization of communication between processors.

In this paper, an algorithm named TMFI (Tabular method for frequent itemset) has been proposed, that is fundamentally different from all the previous algorithms which reduces the maximum number of scans to less than 2 and in most of the cases find all association rules in about 1 scan. However, the savings in I/O is not achieved at the cost of increased CPU overhead. Extensive experiments has been performed and compared with one of the best contemporary algorithms. The corresponding outcome of the experiments shows that for computationally intensive cases where the number of itemsets and transactions to be considered is quite high, the proposed algorithm outperforms the other algorithms used in this comparative study both in terms of CPU and I/O overhead. The proposed algorithm is ideally suitable for parallelization.

Other related, but not directly applicable work in database mining are reported in [1],[5],[6],[7],[8].

This paper is organized as follows: In Section 2, the problem has been described, in Section 3 an overview of the previous algorithms is given and Section 4 describes pro-posed algorithm. Performance results are described in Section 5. Section 6 and Section 7 contains conclusion and future work respectively.

2 Problem Description (Association Rule )

This section is largely based on the description of the probl-lem in [2],[9],[10],[11]. Formally, the problem can be stated as follows: Let $I = \{i_1, i_2, \ldots, i_m\}$ be a set of $m$ distinct literals called items. $T$ is a
set of variable length transactions over \( I \). Each transaction contains a set of items \( f_i; i_j; \ldots ; i_k \in I \). A transaction also has an associated unique identifier called Tid. An association rule is an implication of the form \( X \Rightarrow Y \), where \( X, Y \subseteq I \), and \( X \setminus Y = \emptyset \). \( X \) is called the antecedent and \( Y \) is called the consequent of the rule. In general, a set of items (such as the antecedent or the consequent of a rule) is called an itemset. The number of items in an itemset is called the length of an itemset. Itemsets of some length \( k \) are referred to as \( k \)-itemsets. For an itemset \( X \cup Y \), if \( Y \) is an \( m \)-itemset then \( Y \) is called an \( m \)-extension of \( X \). Each itemset has an associated measure of statistical significance called support. For an itemset \( X \), support(\( X \)) = \( s \), if the fraction of transactions in \( D \) containing \( X \) equals \( s \). A rule has a measure of its strength called confidence defined as the ratio support(\( X \setminus Y \)) / support(\( X \)). The problem of mining association rules is to generate all rules that have support and confidence greater than some user-specified minimum support and minimum confidence thresholds, respectively. This problem can be decomposed into the following subproblems:

I. All itemsets that have support above the user-specified minimum support are generated. These itemsets are called the large itemsets. All others are said to be small.

II. For each large itemset, all the rules that have minimum confidence are generated as follows: for a large itemset \( X \) and any \( Y \subseteq X \), if support(\( X \setminus Y \)) = \( s \), if the fraction of transactions in \( D \) containing \( X \setminus Y \) equals \( s \). A rule has a measure of its strength called confidence defined as the ratio support(\( X \setminus Y \)) / support(\( X \)). The problem of mining association rules is to generate all rules that have support and confidence greater than some user-specified minimum support and minimum confidence thresholds, respectively. This problem can be decomposed into the following subproblems:

I. All itemsets that have support above the user-specified minimum support are generated. These itemsets are called the large itemsets. All others are said to be small.

II. For each large itemset, all the rules that have minimum confidence are generated as follows: for a large itemset \( X \) and any \( Y \subseteq X \), if support(\( X \setminus Y \)) = \( s \), if the fraction of transactions in \( D \) containing \( X \setminus Y \) equals \( s \). A rule has a measure of its strength called confidence defined as the ratio support(\( X \setminus Y \)) / support(\( X \)). The problem of mining association rules is to generate all rules that have support and confidence greater than some user-specified minimum support and minimum confidence thresholds, respectively. This problem can be decomposed into the following subproblems:

I. All itemsets that have support above the user-specified minimum support are generated. These itemsets are called the large itemsets. All others are said to be small.

II. For each large itemset, all the rules that have minimum confidence are generated as follows: for a large itemset \( X \) and any \( Y \subseteq X \), if support(\( X \setminus Y \)) = \( s \), if the fraction of transactions in \( D \) containing \( X \setminus Y \) equals \( s \). A rule has a measure of its strength called confidence defined as the ratio support(\( X \setminus Y \)) / support(\( X \)). The problem of mining association rules is to generate all rules that have support and confidence greater than some user-specified minimum support and minimum confidence thresholds, respectively. This problem can be decomposed into the following subproblems:

I. All itemsets that have support above the user-specified minimum support are generated. These itemsets are called the large itemsets. All others are said to be small.

II. For each large itemset, all the rules that have minimum confidence are generated as follows: for a large itemset \( X \) and any \( Y \subseteq X \), if support(\( X \setminus Y \)) = \( s \), if the fraction of transactions in \( D \) containing \( X \setminus Y \) equals \( s \). A rule has a measure of its strength called confidence defined as the ratio support(\( X \setminus Y \)) / support(\( X \)). The problem of mining association rules is to generate all rules that have support and confidence greater than some user-specified minimum support and minimum confidence thresholds, respectively. This problem can be decomposed into the following subproblems:
3 Related Work

The problem of generating association rules was first introduced in [2] and an algorithm called AIS was proposed for mining all association rules. After that many efficient frequent pattern mining algorithms have been proposed in the literature [4],[3]. Apriori-like algorithms, which are the most important level order mining algorithms, suffer from two main problems: huge numbers of candidate sets, and repeatedly scanning the dataset to check the frequency of each of candidates. DHP [12] is an extension of Apriori algorithm by using Hash tables. Andreas Muellers proposed four algorithms on Apriori[13]. The DIC algorithm proposed by Sergey Brin and others is a generalization of Apriori[14] in which partitioning of database is done. FP-growth and its successors [15],[16],[17],[18] and [19] are known as the second generation of pattern mining algorithms which mine frequent patterns of datasets using a depth first search method without candidate generation and without multiple scans on the dataset. High dimensional datasets, which have small number of transactions and large number of items in each transaction, cause creation of vertical algorithms which operate on vertical datasets [20]. Eclat, MaxEclat, Clique, and MaxClique [21] are the examples of these.

The algorithms vary mainly in

1) How the candidate item sets are generated?

2) How the support for the candidate item sets are counted?

1) Partition Algorithm[10]: Ashok Savasere and others proposed the two-pass Partition algorithm, which logically divides the horizontal database into non-overlapping partitions. Each partition is read, and vertical Tid lists (lists of all Tids where the item appears) are formed for each item. Partition then generates all locally frequent itemsets through Tid list intersections. Locally frequent itemsets from all partitions merge to form a global candidate set. Partition then makes a second pass through all the partitions and obtains all candidates global counts through Tid list intersections. Our results are compared with this algorithm.

4 Proposed Work

A. TMFI (Tabular method for frequent itemset)

Data base partitioning is the basis of the various parallel association rule mining algorithms and distributed association rule mining algorithms. The partitioning is based on the observation that the frequent sets are normally very few in number compared to the set of all itemsets. A large database needs to be scanned multiple number of times because the number of possible item sets to be tested for support is exponentially large if it must be done in a single scan of the database. However, suppose we are given a small set of potentially large item sets, say a few thousand item sets then the support for them can be tested in one scan of the database and the actual large item sets can be discovered. This is the main idea behind partitioning a database. We define local support for an item set as the fraction of transactions containing that item set in a partition. We define a local large item set as an item set whose local support in a partition is at least the user defined minimum support. In other words, a local large item set is large only in the context of a partition (i.e. consider the entire database as consisting of only that partition). A local candidate item set is an item set that is being tested for minimum support within a given partition. It may or may not be large in the context of the
entire database. We define the global support, global large item set, and global candidate item set as above except they are in the context of the entire database D. Clearly our goal is to find all global large itemsets.

By taking advantage of the large itemset property, i.e. a large itemset must be large in at least one of the partitions [10]. This idea can help to design algorithms more efficiently than those based on looking at the entire database. Partitioning algorithms may be able to adapt better to limited main memory. Each partition can be created such that it fits in to main memory. In addition, it would be expected that the number of itemsets to be counted per partition would be smaller than those needed for the entire database. Incremental generation of association rules may be easier to perform by treating the current state of the database as one partition and treating the new entries as a second partition. Various approaches for generating large item sets have been proposed based on partitioning the set of transactions. The partitions are formed by various methods for e.g. dividing the set of transactions based on the similarity measures between the transactions, sequential approach etc. Randomly forming the partition has an advantage over sequential partitioning due to data skew effect[10]. As the result, if the set of transactions are partitioned in to smaller segments such that each segment can be accommodated in the main memory, then the set of frequent sets of each of these partitions can be computed. Therefore, this way of finding the frequent sets by partitioning the database may improve the performance of finding large itemsets in several ways.

The TMFI algorithm works in 2 phases. In the first phase, it divides D into m partitions, followed by creation of bit array matrix in which rows represents Tid’s and then it generates large frequent itemsets (Lpi where i = 1 to m) from all the partitions by using tabular method. Hash map data structure has been used. For generating frequent itemsets, it uses simple bitwise addition operation which is later explained in this paper. At the end of first scan, it constructs global candidate sets (Cg) incrementally by adding Lpi to Cg whenever Lpi is available. It starts counting the number of occurrences for each candidate itemset as soon as Lpi is added to Cg. Thus, after the first scan, if TMFI adds the last candidate itemset to Cg when processing partition pij, then it only needs to scan up to partition pij for the second scan. Thus, in the best case where each partition generates same Lpi only 1 scan is needed, and in the worst case (2m - 1)=m scans are needed.

Algorithm

```
Cg = null
P = partition database(D)
No of transactions in each partition = x=m ==Phase 1
for i = 1 to m begin read in partition (Pi)
Li = gen large itemsets(Pi)
add Li to Cg with count and also increment the count of each Li during processing of each partition
if all the partitions generate the same Li end
else ==Phase 2
Scan for i = 1 to P 1 to get the final count Lg = Cg with count
end
gen large itemsets
for each transaction t Pi begin
Create a new row for t with item purchased as 1 and 0if not purchased
```
for each item Ii P begin ==for singleton itemsets find count end
If count < min sup then delete Ii from Pi;
for each item I, Ij Pi in t begin ==for 2 itemsets
convert columns of both itemset into integers perform bitwise and operations
find count end
If count < min,up then delete I, Ij from Pi; end
repeat for n itemsets until frequent itemsets occurs end

The working of proposed algorithm for generating frequent itemsets is being described here with the help of an example.

Consider a database D with 9 transactions, containing items P, Q, R, S, T consider minimum support 50%. 50% of 8 transactions is 4 but since the considered database is divided into 2 partitions, 50% of each partition is 2. Therefore, we check for 2 occurrences per partition and not 4.

Table I: (a) Transaction table (b) Partition 1 (c) Partition 2

<table>
<thead>
<tr>
<th>Tid</th>
<th>Items</th>
<th>Tid</th>
<th>Items</th>
<th>Tid</th>
<th>Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>P,Q,T</td>
<td>1</td>
<td>P,Q,T</td>
<td>5</td>
<td>P,R</td>
</tr>
<tr>
<td>2</td>
<td>Q,S</td>
<td>2</td>
<td>Q,S</td>
<td>6</td>
<td>Q,R</td>
</tr>
<tr>
<td>3</td>
<td>Q,R</td>
<td>3</td>
<td>Q,R</td>
<td>7</td>
<td>P,R</td>
</tr>
<tr>
<td>4</td>
<td>P,Q,S</td>
<td>4</td>
<td>P,Q,S</td>
<td>8</td>
<td>P,Q,R,T</td>
</tr>
<tr>
<td>5</td>
<td>P,R</td>
<td>5</td>
<td>P,R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Q,R</td>
<td>6</td>
<td>Q,R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>P,R</td>
<td>7</td>
<td>P,R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>P,Q,R,T</td>
<td>8</td>
<td>P,Q,R,T</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Step 1. Divide the database into partitions randomly. Here we get 2 partitions P1 and P2.
- Step 2. For each partition and for each transaction create a Tid lists matrix in which create a new row with the different number of items in D and mark it as 1 if the item is present in that transaction otherwise mark it as 0. Store these 1’s and 0’s in bits.
- Step 3. Calculate the support count for all items in L1 just by adding no of ones in the column. Delete the items having support count less than the minimum support from L1.
- Step 4. To calculate 2-itemsets and the support count for all items in L2 take the corresponding columns of items convert it into integer and perform bitwise addition. Delete the items having support count less than the minimum support from L2.
- Step 5. Repeat the steps for 3-itemsets and so on until we get the frequent itemsets. L gives the frequent itemset from all partitions.
here. Suppose there are 3 partitions with 4 transactions in each partition and minimum support is 50%.
Suppose in the first partition, PQ and QR has count 2 and RS has count 1, in the second partition PQ and RS has 2 counts each and in the third partition, PQ has count 1 and PS has count 2 respectively. In 3rd partition PQ with count 1 and PS with count 2. Initially Cg is empty and count of frequent itemsets is 0. Now, PQ will be added to Cg with its count and it will be incremented while processing partition 2 and so on. RS will be added in 2nd partition and count will be incremented in the 3rd partition and so on. To get cumulative count of RS, second database scan is needed as RS was missed in the first partition. Last partition will not to be scanned as large itemsets from last partition are already added in Cg and those who are not frequent, their count is been updated in the first database scan. Therefore, maximum number of database scan needed is (2m 1)=m.

B. Minimum Support Optimization

<table>
<thead>
<tr>
<th></th>
<th>P</th>
<th>Q</th>
<th>R</th>
<th>S</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>PQ</th>
<th>PS</th>
<th>QS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 1: Time complexity for 100000 transactions

(a)  
(b)
The bitwise addition operation can be performed faster by utilizing minimum-support value. For example, assume that minimum-support is 100 and bitwise addition of two item-sets has been performed, PQ with support 119 and PR with support 200. This process will be stopped as once 20 zeroes (mismatches) are found, since the support of PQR is bounded above by 119. As the entire column is converted into integer and bitwise addition operation is performed, this technique cannot be used directly in future. But with little

<table>
<thead>
<tr>
<th></th>
<th>PQS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

TABLE II: (a) L1 Large ItemSets-1 (b) L2 Large ItemSets-2 (c) L3 Large ItemSets-3.

The frequent itemsets are PQ and QS.

How the proposed algorithm requires only \((2^m - 1)^m\) database scans to generate global itemsets is being discussed modification this can prove out to be a very good optimization technique.

4 Experimental Results

The experimental results has been implemented in Python on Intel Core i3 processor, 4 GB RAM and 256 GB Hard drive to check the results and clearly found that the efficiency of algorithm is much better than native partition algorithm. The test was conducted for 1 lakh, 5 lakh and 1 million transactions and 5000 products with varying minimum supports 1, 0.75, 0.5, 0.30, 0.25, 0.20, 0.15. Experimental outcomes are demonstrated in Fig 1, Fig 2, Fig 3, Fig 4, Fig 5 and Fig 6 respectively.

![Fig. 2: Space complexity for 100000 transactions](image)

![Fig. 5: Time complexity for 1000000 transactions](image)
6 Conclusion

The developed algorithm is not only efficient but also fast for discovering association rules. It reduces the database scan to 2 which is \((2m - 1)/m\). Instead of Tid list, tabular method has been used for storing transactions. The proposed approach calculates frequent itemsets by simple bitwise addition operation. The results have been compared with the partition algorithm which clearly indicates that proposed algorithm is faster than partition algorithm \(m\) times, where \(m\) is the number of partitions. Also, Partition algorithm requires 16 times more space than our algorithm for frequent itemsets generation. The time complexity of the proposed algorithm is linear i.e. \(O(n)\), whereas that of the partition algorithm is polynomial i.e. \(O(n^2)\). Thus, the presented approach saves a lot of time as well as space.

7 Future Work

In this work, the database has been divided into partitions randomly instead of sequentially due to effect of data skew [10]. However, this conflicts with the goal of exploiting sequential I/O to speed up the reading process. Better technology is needed to partition the database. The problem of accurately estimating the number of partitions given the available memory so that each partition fits into the memory, however needs further work. With little modification, the algorithm is best suited for parallelization. Currently, this problem is being addressed and an effort is being made to combine our algorithm with the previous algorithms to develop a hybrid algorithm which performs best for all the cases.
8 References


