One Time Mining by Multi-Core Preprocessing on Generalized Dataset
Aviral Kumar Singh1, S. R. Tondon2, Tarun Dhar Diwan3
aviralkumarsinghlive.com

Abstract
One of the important problems in data mining is discovering association rules from databases of transactions where each transaction consists of a set of items. Many industries are interested in developing the association rules from their databases due to continuous retrieval and storage of huge amount of data. The discovery of interesting association relationship among business transaction records in many business decision making process such as catalog decision, cross-marketing, and loss-leader analysis. The enormity and high dimensionality of datasets typically available as input to problem of association rule discovery, and the time consuming operation in this discovery process is the computation of the frequency of interesting subset of items (called candidates) in the database of transactions. Hence, it is has become vital to develop a method that will make speedup the preprocessing computation. In this paper, We have proposed An Integrated approach of Parallel Computing and ARM for mining Association Rules in Generalized data set that is fundamentally different from all the previous algorithms in that multi-core preprocessing is done and by avoiding recurring scan of dataset number of passes required is reduced. The response time is calculated on space delimited text dataset.

Keywords: Data Mining, Association Rule Mining (ARM), Association rules, Apriori algorithm, Frequent pattern.
Since the diagonal elements are not affected during the transposition, that is, element aii of A equals element aii of AT, the data in the diagonal processors will stay stationary.

The advantage of using transposed array is to calculate support count for particular item. There is no need to repeatedly scan array. Only by finding the row sum of the array will give the required support count for particular item, which ultimately results in increased efficiency of the algorithm.

The remainder of this paper is organized as follows: Section 2 provides a brief review of the related work. In Section 3, we explain Frequent Itemset and Association Rule Mining through Apriori Algorithm. In Section 4, we introduce our approach of frequent itemset generation using parallel preprocessing. An illustration of the algorithm and experiment analysis is presented in section 5 and section 6 respectively. Finally, we concluded our work.

Related Work

One of the most well known and popular data mining techniques is the Association rules or frequent item sets mining algorithm. The algorithm was originally proposed by Agrawal et al. [4] [5] for market basket analysis. Because of its significant applicability, many revised algorithms have been introduced since then, and Association rule mining is still a widely researched area.

Agrawal et. al. presented an AIS algorithm in [4] which generates candidate item sets on-the-fly during each pass of the database scan. Large item sets from previous pass are checked if they are present in the current transaction. Thus new item sets are formed by extending existing item sets. This algorithm turns out to be ineffective because it generates too many candidate item sets. It requires more space and at the same time this algorithm requires too many passes over the whole database and also it generates rules with one consequent item.

Agrawal et. al. [5] developed various versions of Apriori algorithm such as Apriori, AprioriTid, and AprioriHybrid. Apriori and AprioriTid generate item sets using the large item sets found in the previous pass, without considering the transactions. AprioriTid improves Apriori by using the database at the first pass. Counting in subsequent passes is done using encodings created in the first pass, which is much smaller than the database. This leads to a dramatic performance improvement of three times faster than AIS.

Scalability is another important area of data mining because of its huge size. Hence, algorithms must be able to “scale up” to handle large amount of data. Eui-Hong et. al [16] tried to make data distribution and candidate distribution scalable by Intelligent Data Distribution (IDD) algorithm and Hybrid Distribution (HD) algorithm respectively. IDD addresses the issues of communication overhead and redundant computation by using aggregate memory to partition candidates and move data efficiently. HD improves over IDD by dynamically partitioning the candidate set to maintain good load balance. Different works are reported in the literature to modify the Apriori logic so as to improve the efficiency of generating rules. These methods even though focused on reducing time and space, in real time still needs improvement.

Frequent Item Set And Association Rule

The aim of Association rule mining is exploring relations and important rules in large datasets. A dataset is considered as a sequence of entries consisting of attribute values also known as items. A set of such item sets is called an item set. Frequent item sets are sets of pages which are visited frequently together in a single server session.

Let I={ I1, I2, … , Im } be a set of items. Let D, the task-relevant data, be a set of database transactions where each transaction T is a set of items such that T ⊆ I. Each transaction is associated with an identifier, called TID. Let A be a set of items. A transaction T is said to contain A if and only if A ⊆ T. An association rule is an implication of the form A⇒B, where A⊆I, B⊆I, and A∩B=∅. The rule A ⇒B holds in the transaction set D with support s, where s is the percentage of transactions in D that contain A∪B (i.e., the union of sets A and B, or say, both A and B). This is taken to be the probability, P(A∪B). The rule A ⇒ B has confidence c in the transaction set D, where c is the percentage of transactions in D containing A that also contain B. This is taken to be the conditional probability, P(B|A). That is,

\[ \text{support}(A \Rightarrow B) = \text{P}(A \cup B) \]  \hspace{1cm} (2.1)

\[ \text{confidence}(A \Rightarrow B) = \frac{\text{P}(B|A)}{\text{P}(A)} \]  \hspace{1cm} (2.2)

A set of items is referred to as an itemset. An itemset that contains k items is a k-itemset. The set \{bread, butter\} is a 2-itemset. The occurrence frequency of an itemset is the number of transactions that contain the itemset, it is also known, as the frequency, or support count. If the relative support of an itemset I satisfies a pre specified minimum support threshold then I is a frequent itemset. The set of frequent k-itemsets is commonly denoted by Lk. From Equation (2.2), we have

\[ \text{confidence}(A \Rightarrow B) = \frac{\text{P}(B,A)}{\text{P}(A)} = \frac{\text{support}(A \cup B)}{\text{support}(A)} \]

where a ∈ A, b ∈ B, a \nsubseteq A, b \nsubseteq B, A \subseteq I, B \subseteq I, and A \cap B = ∅.
Let \( \tau = I_1, I_2 \ldots I_m \) 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 \( I_k \), and \( t[k] = 0 \) otherwise. There is one tuple in the database for each transaction. Let \( X \) be a set of some items in \( \tau \). We say that a transaction \( t \) satisfies \( X \) if for all items \( I_k \) in \( X \), \( t[k] = 1 \).

By an association rule, we mean an implication of the form \( X \Rightarrow I_j \), where \( X \) is a set of some items in \( \tau \), and \( I_j \) is a single item in \( \tau \) that is not present in \( X \). The rule \( X \Rightarrow I_j \) is satisfied in the set of transactions \( T \) with the confidence factor \( \alpha \leq c \leq 1 \) if at least \( c \% \) of transactions in \( T \) that satisfy \( X \) also satisfy \( I_j \). We will use the notation \( X \Rightarrow I_j | c \) to specify that the rule \( X \Rightarrow I_j \) has a confidence factor of \( c \).\[3\]

### A. Apriori Algorithm

The Apriori algorithm is one of the most popular algorithms for mining frequent patterns and association rules \[4\]. It introduces a method to generate candidate itemsets \( C_k \) in the pass \( k \) of a transaction database using only frequent itemset \( L_k \)–1 in the previous pass. The idea rests on the fact that any subset of a frequent itemset must be frequent as well. Hence, \( C_k \) can be generated by joining two itemsets in \( L_{k-1} \) and pruning those that contain any subset that is not frequent as shown in Fig1.

---

**Step 1:** Scan the transaction database to get the support \( S \) of each \( I_k \) to get a set of frequent items, \( L_k \).

**Step 2:** Use \( L_{k-1} \) to generate a set of candidate itemsets, and use Apriori property to prune the unwanted elements from this set.

**Step 3:** Scan the transaction database to get the support \( S \) of each candidate \( k \) itemset in the final set, compare \( S \) with min sup, and get set of frequent itemsets, \( L_k \).

**Step 4:** The candidate set \( S \) is null.

**Step 5:** For each frequent itemset \( L \), compute all anomaly subsets of \( L \).

**Step 6:** For every anomaly subset of \( L \), update the rule \( L \Rightarrow \neg L \) with confidence \( \alpha \) of rule \( L \Rightarrow \neg L \) \( = \frac{S(L \cap \neg L)}{S(L)} \) (support of \( \neg L \) \( \cap \neg L \)).

---

: Apriori Algorithm

### Our Approach

Association rule and frequent itemset mining has become now a widely research area and hence, faster and faster algorithms have been presented. The Association Rule Mining algorithms such as Apriori, FP-Growth requires repeated scans over the entire database. All the input/output overheads that are being generated during repeated scanning the entire database decrease the performance of CPU, memory and I/O overheads.

Much work has been carried out on improving the efficiency of the apriori algorithm by reducing the I/O time and minimizing the set of candidate itemsets. However, all these works suffer from problem of scans over the database at least once. The efficiency of these algorithms can still be improved by reducing the time required for counting the supports of candidate itemsets. We aim to obtain an efficient algorithm which reduces the time needed to count the supports of candidate itemsets.

The process of finding large itemsets is divided into following parts.

- Parallel Data Preprocessing
- Generating candidate sets.

### A. Parallel Data Preprocessing

The idea of our algorithm is quite simple. Since the diagonal elements are not affected during the transposition, that is, element \( a_{ii} \) of \( A \) equals element \( a_{ii} \) of \( A^T \), the data in the diagonal processors will stay stationary. An \( n \times n \) mesh of processors can be regarded as a matrix and is therefore perfectly fitted to accommodate an \( n \times n \) data matrix, one element per processor. \( A(i,j) \) is used to store \( a_{ij} \) initially and \( a_{ji} \) when transposition, that is, element \( a_{ii} \) of \( A \) equals element \( a_{ii} \) of \( A^T \). All the input/output overheads that are being generated required for counting the supports of candidate itemsets.

---


[795-802]
Matrix to be transposed, stored in mesh of processors.

Example Execution

A=x  
B=1  
C=4

A=1  
B=2  
C=

A=2  
B=3  
C=

A=3  
B=4  
C=

A=4  
B=5  
C=

A=5  
B=6  
C=

A=6  
B=7  
C=

A=7  
B=8  
C=
B. Candidate set Generation

We propose a new algorithm in which transactional database is considered as a two dimension array which works on generalized value dataset. The main difference between proposed algorithm and other algorithms is that instead of using transactional array in its natural form, our algorithm uses transpose of array i.e. rows and columns of array are interchanged and transposition is achieved using parallel matrix transpose algorithm.

Algorithm

1. Transpose(Data Set)
2. Read the database to count the support of C1 to determine L1 using sum of rows.
3. \(L_1\) = Frequent 1-itemsets and \(k:=2\)
4. While \((k-1)\neq NULL\) set do
   Begin
   \(C_k\) := Call Gen_candidate_itemsets \((L_{k-1})\)
   Call Prune \((C_k)\)
   for all itemsets \(i \in I\) do
   Calculate the support values using dot-multiplication of array;
   \(L_k\) := All candidates in \(C_k\) with a minimum support;
   \(k:=k+1\)
   End
5. End of step-4
End Procedure

An Illustration

Suppose we have a transactional database in which the user transactions from T1 to T5 and items from A1 to A5 are stored in the form of generalized values, which is shown in Table-1 (Fig 4).

Consider the transpose of transactional database as shown in Table-1 is stored in Table-2 by applying Parallel Transposition that can be used in our proposed algorithm. Assume the user specified minimum support is 40%, and then the steps for generating all frequent item sets in proposed algorithm will be repeated until NULL set is reached. In our algorithm, transactional dataset will be used in the transposed form. Therefore, candidate set and frequent itemset generation process will be changed as compared to Apriori algorithm.

Then the candidate 2-itemset will be generated by performing dot-multiplication of rows of array, as array consist of generalized values, the resultant cell will be produce in the form of 1. If the corresponding cells of the respective rows have 1, otherwise 0 will be in the resultant cell. In this approach, we will receive a new array consisting of candidate 2-itemsets to get the higher
order of itemsets. The above process between rows of array can be performed to find out the results.

![Table 1](image1)

**An Illustration of our approach**

**Experimental Evaluations**

The performance comparison of our mining algorithm with classical frequent pattern-mining algorithm Apriori is shown in Fig 7. All the experiments are performed on 1.50Ghz Pentium-iv desktop machine with 256 MB main memory, running on Windows-XP operating system. The program for Apriori and our proposed algorithm were developed in Java JDK1.5 environment.

Figure 5 compares the time taken by the apriori algorithm and our algorithm for different support values. For lower support values algorithm takes more time because it generates too many candidate itemsets. These candidate itemsets are then tested for minimum support. It is obvious from the figure that as the support value increases time taken by the algorithm decreases. In our algorithm, only by finding the row sum of the array will give the required support count for particular item, which ultimately results in increased efficiency of the algorithm.
Conclusions

ARM algorithms are important to discover frequent itemsets and patterns from large databases. In this project, we have designed An Algorithm for generation of frequent itemsets similar to Apriori algorithm. The proposed algorithm can improve the efficiency of Apriori algorithm and it is observed to be very fast. Our algorithm is not only efficient but also very fast for finding association rules in large databases. The proposed algorithm drastically reduces the I/O overhead associated with Apriori algorithm and retrieval of support of an itemset is quicker as compared to Apriori algorithm. This algorithm may be useful for many real-life database mining scenarios where the data is stored in generalized form. Our algorithm uses parallel transposition of generalized 2D data set, so the data preprocessing goes faster. This algorithm cannot be used with multimedia dataset.

References


[18] Dr (Mrs) Sujni Paul “An Optimized Distributed Association Rule Mining Algorithm In Parallel and Distributed Data Mining With Xml Data For Improved Response Time”. International Journal Of Computer Science And Information Technology, Volume 2, Number 2, April 2010

[19] Manoj Bahel, Chhaya Dule “Analysis of frequent item set generation process in Apriori & RCS (Reduced Candidate Set) Algorithm” National Conference on Information and Communication Technology, Banglore April 2010