Volume 1, No. 3, Sept-Oct 2010
ISSN No. 0976-5697
International Journal of Advanced Research in Computer Science
RESEARCH PAPER
Available Online at www.ijarcs.info

Mining Association Rules in Large Databases using localized pattern

Kanhaiya Lal*
Assistant Professor
Department of Computer S. & Engg.
B.I.T Patna , India
klal@bitmesra.ac.in

Dr. N.C.Mahanti
Professor & Head
Department of Applied Mathematics
B.I.T Mesra Ranchi, India
ncmahanti@rediffmail.com

Abstract: Discovering association rules from very large databases is an important data mining problem. In this paper we propose an algorithm which is applicable to large databases and can be used efficiently for discovering important rules. It discovers the large itemsets without multiple passes over the database. In this paper we concentrate over the discovery of localized patterns in a sub-domain, which can be easily processed to obtain large-itemsets and valid rules, consecutively. We present an efficient algorithm for mining association rules that is faster than the previously proposed partition algorithms. The algorithm is also ideally suited for parallelization.

Keywords: Clustered-Domain; Localized-Pattern; Parallelizable Mining; Domain partitioning; Generalized Association Rule.

I. INTRODUCTION

Data mining has its applicability in many areas such as decision support, market strategy, financial forecasts[3], customer profiling, analysis of products, warranty claim analysis, inventory analysis[11], etc. Many approaches have been proposed to find out useful and invaluable information from huge databases [1] and[10]. One of the most important approaches is mining association rules, which was first introduced in [3].The mining of association rules involves the discovery of significant and valid correlations among items that belong to a particular domain [7]. The development of association-rule mining algorithms has attracted remarkable attention during the last years [8], where focus has been placed on efficiency and scalability issues with respect to the number of records[9].

The information obtained as association rules may be used to decide catalog design, store layout, product placement, target marketing, webpage layout, etc. Many algorithms have been proposed for discovering association rules[3][7][2]. Association-rule mining algorithms based on column enumeration can be severely impacted by a large domain. BFS algorithms (Apriori-like) will produce an excessive number of candidates, which drastically increase the CPU and I/O costs (when the candidates do not fit in main memory, they have to be divided into chunks and numerous passes are performed for each chunk[3]). DFS algorithms (e.g., FP-growth [5] and Éclat [4] use auxiliary data structures that summarize the database (e.g., FP-tree), which become less condensed when the domain size increases, because of the many different items' combinations. This affects the CPU and I/O costs and, more importantly, disk thrashing may occur when the size of the structures exceeds the available main memory.

We focus on mining databases with a very large number of records and with a domain that has a very large number of items. Initially, the algorithm partitions the database in small chunks, which can be accommodated easily in main memory. The partitioning method is adopted from [2], but with a slight modification that items are not in <TID, item> form. In second phase the correlations in the partitions (i.e. sub-domains) are determined. This method is based on the concept of localized-patterns in Ref [6]. In very large database the items form a localized pattern and are not uniformly distributed. Hence, with an assumption: *for a partition at random consists, at least, one localized pattern within its domain.* This method acts in second phase and detects, very quickly and with low memory consumption, the groups of items. Then, in a third phase, we propose the separate mining of association rules within groups. The separate mining of each partition is performed by focusing each time on the relevant items.

A. Problem Description

Let $I=\{i_1, i_2,\ldots, i_m\}$ be a set of distinct items and D be a set of variable length transactions, where each transaction T (a data case) is a set of items so that $T \subseteq I$. In general, a set of items is called an itemset. The number of items n an itemset is called the length of an itemset. Itemsets of some length k are referred as k-itemsets. An association rule is an implication of the form, $X \rightarrow Y$, where $X \subset I$, $Y \subset I$ and $X \cap Y = \Phi$. The rule $X \rightarrow Y$ holds in the transaction set T with confidence c, if c% of transactions in T that support X also support Y. The rule has support s in T if s% of the transactions in T contains $X \cup Y$. Here X is called the antecedent and Y is called the consequent of the rule. For a given set of transactions D (the database), the problem of mining association rules is to discover all association rules that have support and confidence greater than the user-specified minimum support (called minsup) and minimum confidence (called minconf).

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, relatively. This problem can be decomposed into the following sub problems [2]:

1. *All itemsets that have support above the user specified minimum support are generated. These itemsets are called large itemsets.*
2. *For each large itemsets, all the rule that have minimum confidence are generated as follows:*

For a large itemset X and Y $\subset$ X, if support (X) / support(X-Y) >= minimum confidence, then the rule X –Y → Y is a valid rule.

For a set of items | I | the number of possible itemsets is 2^m. The problem is to identify which of these large numbers of itemsets has the minimum support for the given set of transactions. For very small values of m, it is possible to set up 2^m counters, one for each distinct itemset, and count the support for every itemset by scanning the database once. However for many applications m can be more than 1,000. Clearly, this approach is impractical. To reduce the combinatorial search space, all algorithms exploits the following property: *any subset of a large itemset must also be large* [2].

II. PREVIOUS WORKS

After the initial algorithms proposed by Agawam [7], the problem has been extensively studied by other researchers and a number of fast variants have been proposed. In a subsequent paper in [3], Agawam et. al. has discussed how the algorithm for finding large item sets may be sped up substantially by introducing a pruning approach which reduces the size of the candidate Ck. This algorithm uses the pruning trick that all subsets of a large item set must also be large. If some (k-1)-subset of an itemset - 2 does not belongs to L_{k-1} then that itemset can be pruned from further consideration [17]. This process of pruning eliminates the need for finding the support of the candidate item set l. In the same paper [3], an efficient data structure known as the hash-tree was introduced for evaluating the support of an item set.

Subsequent work on the large item set method has concentrated on the following aspects:

(1) Improving the I/O costs by reducing the number of passes over the transaction database.

(2) Improving the computational efficiency of the large item set generation procedure.

(3) Finding efficient parallel algorithms for association rule generation.

(4) Introducing sampling techniques for improving the I/O and computational costs of large item set generation.

(5) Extensions of the large item set method to other problems such as quantitative association rules, generalized associations, and cyclic association rules.

(6) Finding methods for online generation of association rules by using the pre-process-once-query-many paradigm of online analytical processing.

A. Improvements

A hash-based algorithm for efficiently finding large itemsets was proposed by Park et. al. in [12]. It was observed that most of the time was spent in evaluating and finding large 2-itemsets. The algorithm in Park et. al. [12] attempts to improve this approach by providing a hash based algorithm for quickly finding large 2-itemsets.

Brin et. al. proposed a method for large itemset generation which reduces the number of passes over the transaction database by counting some (k+1)-itemsets in parallel with counting k-itemsets. Inmost previously proposed algorithms for finding large itemsets, the support for a (k+1)-itemset was counted after k-itemsets have already been generated. In this work, it was proposed that one could start counting a (k + 1)-itemset as soon as it was suspected that this itemset might be large. Thus, the algorithm could start counting for (k+1)-itemsets much earlier than completing the counting of k-itemsets. The total number of passes required by this algorithm is usually much smaller than the maximum size of a large itemset.

A partitioning algorithm was proposed by Savasere et. al. [2] for finding large itemsets by dividing the database into n partitions. The size of each partition is such that the set of transactions can be maintained in main memory. Then, large itemsets are generated separately for each partition. This method requires just two passes over the transaction database in order to find the large itemsets.

The approach described above is highly parallelizable, and has been used to generate large itemsets by assigning each partition to a processor. At the end of each iteration of the large item set method the processors need to communicate with one another in order to find the global counts of the candidate k-item sets. Often, this communication process may impose a substantial bottleneck on the running time of the algorithm. In other cases, the time taken by the individual processors in order to generate the processor-specific large itemsets may be the bottleneck.

A common feature of most of the algorithms reviewed above and proposed in the literature is that most such research is are variations on the "bottom-up theme" proposed by the *Apriori* algorithm [3,7]. For databases in which the itemsets may be long, these algorithms may require substantial computational effort. Consider for example a database in which the length of the longest itemset is 40. In this case, there are 240 subsets of this single itemset, each of which would need to be validated against the transaction database. Thus, the success of the above algorithms critically relies on the fact that the length of the frequent patterns in the database is typically short.

Since the size of the transaction database is typically very large, it may often be desirable to use random sampling in order to generate the large itemsets. The use of random sampling to generate large itemsets may save considerable expense in terms of the I/O costs. A method of random sampling was introduced by Toivonen in [13]. The weakness of using random sampling is that it may often result in inaccuracies because of the presence of *data skew*. Data which are located on the same page may often be highly correlated and may not represent the overall distribution of patterns through the entire database. As a result, it may often be the case that sampling just 5% of the transactions may be as expensive as a pass through the entire database.

B. Generalizations of the association rule problem

Initially, the association rule problem was proposed in the context of supermarket data. The motivation was to find how the items bought in a consumer basket related to each other. A number of interesting extensions and applications have been proposed. The problem of mining quantitative association rules in relational tables was proposed in [14]. In such cases association rules are discovered in relational tables which have both categorical and quantitative attributes. Thus, it is possible to find rules which indicate how a given range of quantitative and categorical attributes may affect the values of other attributes in the data. The algorithm for the quantitative association rule problem discretizes the quantitative data into disjoint ranges and then constructs an item corresponding to each such range. Once these pseudo-items have been constructed, a large itemset procedure can be applied in order

to find the association rules. Often a large number of rules may be produced by such partitioning methods, many of which may not be interesting. An interest measure was defined and used in [14] in order to generate the association rules. In [15], an algorithm for clustering quantitative association rules was proposed. The aim of this algorithm was to generate rules which were more natural in terms of the quantitative clusters with which individual rules were associated. A closely related issue to finding quantitative association rules is the problem of finding profile association rules in which it is desirable to tie together rules which tie together user profiles with buying patterns. An algorithm for finding profile association rules was discussed in [16]. A method for finding optimized quantitative association rules has been discussed in [15]. This paper discusses how to choose the quantitative ranges in an optimal way so as to maximize the strength of the given association rules. An interesting issue is that of handling taxonomies of items. For example, in a store, there may be several kinds of cereal, and for each individual kind of cereal, there may be multiple brands. Rules which handle such taxonomies are called *generalized associations*. The motivation is to generate rules which are as general as possible and also as general as possible while taking such taxonomies into account. Savasere et. al.[2] also discuss how to find interesting negative association rules in the context of taxonomies of items. The focus of this work is to find rules which negatively correlate with rules which are discovered at higher levels of the taxonomy. Another useful extension of association rules which has been recently been proposed is the concept of *cyclic association rules*. It may often be the case that when association rules are computed for data which have a time component, periodic seasonal variations may be observed. For example, the monthly sales of goods correlate with each other differently on a seasonal basis.

III. ALGORITHM

The algorithm starts in its first phase by partitioning the database (*D*), which is similar to the partitioning method introduced in [2]. The major difference in this algorithm is that it does not use TID list for representing items and the mining process uses graphs to generate localized patterns within a partition ($P_i \Subset P$). This method is completely different from the frequent itemset mining algorithm used in [2], which was based on Apriori. Localized patterns are much easier to discover and without counting the support for an itemset various times in a serialized manner. Instead, it directly computes the support for 2-itemsets and represents them as adjacent vertices (i, j) of an undirected graph.

A. The algorithm can be represented as:

```
Input: D
Output: k-itemset

P=partition_domain(D)
n=number of partitions
begin
      for (i=1 to n )
            read_in_partition(P_i ∈ P)
            L_i=find_localized_patterns(P_i)
        end loop

       for(i=1 to n)
        for(j=1 to n)
            if(|L_i|=|L_j|)
            k-Itemset=merge(L_i,L_j)
            where k=|L_i|
        end loop
      end loop
end
```

B. Localized patterns

Frequent items in a partition form a localized pattern. It is illustrated in the figure as, let the nodes represent all the frequent 1-itemsets and edge between two items (i.e. vertices i and j) implies that the support is greater than the *minimum suport.*

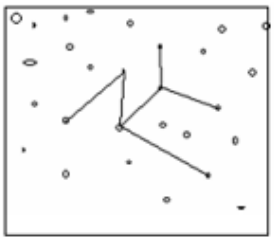

Figure 1: Localized pattern

This is the second phase of the algorithm in which it finds the localized patterns within a partition and then merges the obtained pattern with any other pattern of same size. Thus, the output is a heavy itemset which consists of all the valid rules.

The algorithm for finding localized patterns can be represented as:

```
Algorithm: find_localized_pattern(P_i)
Input: P_i
Output: L_i
Initialize graph structure;
foreach transaction T ∈ Pi
        find_support(i, j) where (i, j) ∈ T
        if support >= minsup
                if exists
                do nothing
                else
                allocate new edge(i, j)
        end loop
foreach vertices (i ∈ V)
        if there exists any (i, j) where i, j  ∈ V
        do nothing
        else
        delete i
end loop
foreach vertices (i ∈ V)
        if there exists (i, j) and (j, k) where i, j, k ∈ V
                L_i=(i, j, k) i.e group items if there exists a path between them.
end loop
```

The algorithm find_loclized_pattern has a maximum space requirement of $O(gs^2)$, where g is the size used by one graph node and s is the size for counter needed to count the support of the 2-itemset.

Proof:

Step 1: For a given set $A = \{x_1, x_2, \ldots\ldots\ldots\ldots, x_n\}$ the total number of pairs generated is n(n-1)/2 [7]. Formally, this can be represented as:

$$|A| = n(n-1)/2 = 0.5 * (n^2 - n);$$

which can be expressed as $O(n^2)$. As the total number of the elements in one partition (i.e. the domain size in the partition) is s, the space requirement is $O(s^2)$.

Step 2: Assuming that the support of all the pairs generated is greater than the minimum support threshold. Then the total number of nodes needed to represent the graph is equal to the size of domain. In our case the domain size is s. Hence, the space required to represent this graph structure in memory is $s^2 * g$, where g is the space needed to represent one node. This can be represented as $O(gs^2)$.

From above we can see the space requirement of this complete algorithm is $\max \{O(s^2), O(gs^2)\} = O(gs^2)$ for g >= 1

C. Final Phase

This phase proceeds further with the output generated by the second phase. The localized patterns obtained from different partitions are merged to form a potential large itemset of size-k. This itemset is the first appropriate domain which can be mined efficiently for generation of valid association rules. The approach proposed is primarily focused on condensing the size of domain.

The largest item set derived from the entire domain can be processed for obtaining rules according to some traditional approach.

IV. SPACE REQUIREMENT OF PARTITIONING ALGORITHM AND COMPARISON BETWEEN PARTITIONING AND FIND_LOCALIZED_PATTERN (PROPOSED ALGORITHM)

Let s be the size of domain in a partition. i.e. total number of items representing the Universe of Discourse All possible number of subsets than can be generated from these s elements is $2^s - 1$. The number of counters is exponential and can be estimated as $O(2^s)$.

Comparing the requirement of space for partitioning algorithm and the proposed algorithm:

Space requirement for the proposed method: $O(gs^2)$

Space requirement o partitioning approach: $O(2^s)$

The proposed method shows a space requirement of polynomial order whereas the older approach has a space complexity of exponential order.

V. CONCLUSION

This algorithm is primarily focused on reducing the size of a large domain. It seems unmanageable to deal with a very large domain since, the number of frequent item sets generated increases exponentially with every element added to the domain. This approach is much similar to the partitioning algorithm but differs primarily in the approach used for finding the item set inside the partition [2] as it adopts the approach of localized pattern. In pure partitioning algorithm the item sets generation was purely apriori in nature as a result for large domain size it is simply not feasible. This fact is supported from the above section which compares the traditional approach of partitioning the database and the proposed method which partitions the domain inside a partitioned database.

VI. APPENDIX

The Apriori algorithm

In mining association rules the two important measures are the *support* and the *confidence*. A *large item set* is an item set with support larger than the support threshold. The common algorithm to compute large item set is the Apriori algorithm. The Apriori algorithm [3] has become a data mining classic and most data mining algorithms are based upon it. The algorithm is depicted below. The most important step of this algorithm is step 3 in the prune step in apriori-gen function, which makes sure that all subsets of a candidate item set are frequent. The basic idea is that any subset of a large item set must be large. Therefore, the candidate item sets having *k* items can be generated by joining large item sets having $k - 1$ items, and deleting those that contain any subset that is not large. The algorithm works as follows:

L_1 = {large 1-itemsets}

for ($k = 2$; $L_{k-1} \neq \Phi$; k++) do begin

C_k = apriori-gen (L_{k-1}) //New candidates

for all transactions t in database do begin
C_t = subset (C_k, t) //Candidates contained in t
for all candidates $c \in C_t$ do begin
c.count++;
end
$L_k = \{c \in C_k | c.count \geq minsup\}$
end
Answer = $\bigcup_k L_k$

The apriori-gen function takes as argument L_{k-1}, the set of all large $(k-1)$-item sets. It returns a superset of the set of all large k-item sets. The function works as follows. First, the join step joins L_{k-1} with L_{k-1}:
1. insert into C_k
2. select $p.item_1$, $p.item_2$, $p.item_{k-1}$, $q.item_{k-1}$
3. from $L_{k-1}p$, $L_{k-1}q$
4. where $p.item_1 = q.item_1$, $p.item_{k-2} = q.item_{k-2}$, $p.item_{k-1} < q.item_{k-1}$;
Next, the prune step deletes all item sets $c \in C_k$ such that some $(k-1)$-subset of c is not in L_{k-1}:
1. forall item sets $c \in C_k$ do
2. forall $(k-1)$-subsets s of c do
3. if ($s \notin L_{k-1}$) then
4. delete c from C_k
5. end
6. end
7. end
This can be easily illustrated from the example given below:

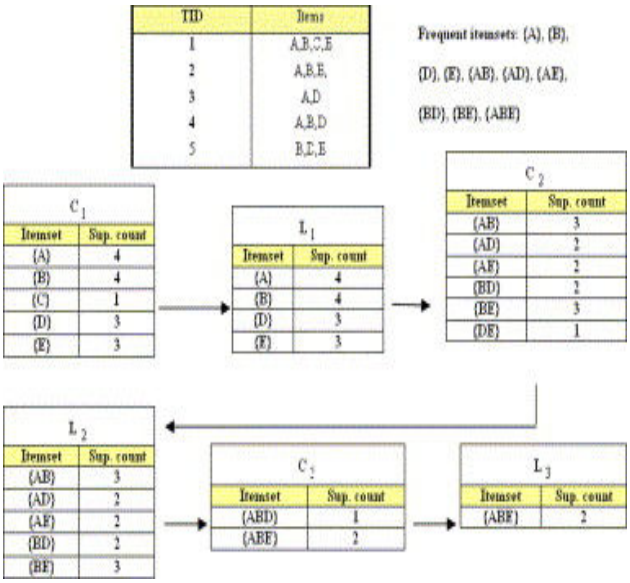

Figure 2: Example

VII. REFERENCES

[1] M.S. Chen, J. Han and P.S. Yu, Data mining: an overview from a database perspective, IEEE Transactions on Knowledge and Data Engineering 8 (1996), pp. 866–883

[2] A. Savasere, E. Omiecinski and S. Navathe, An efficient algorithm for mining association rules in large databases, Proceedings of the VLDB Conference (1995), pp. 432–444.

[3] R. Agrawal and R. Srikant, Fast algorithms for mining association rules, Proceedings of the 20th Very Large DataBases Conference (VLDB'94), Santiago de Chile, Chile (1994), pp. 487–499.

[4] M.J. Zaki, Scalable algorithms for association mining, IEEE Trans. Knowl. Data Eng. 12 (2000) (3), pp. 372–390.

[5]J. Han, J. Pei, Y. Yin and R. Mao, Mining frequent patterns without candidate generation: a frequent-pattern tree approach, Data Min. Knowl. Discovery 8 (2004), pp. 53–87.

[6] C. Aggarwal, C. Procopiuc and P. Yu, Finding localized associations in market basket data, *IEEE Trans. Knowl. Data Eng.* 14 (2002) (1), pp. 51–62.

[7] R. Aggrawal, T. Imielinski and A. Swami, Mining association rules between sets of items in very large databases, Proceedings of the ACM SIGMOD Conference (1993), pp. 207–216.

[8] J. Hipp, U. Guntzer and G. Nakhaeizadeh, Algorithms for association rules mining—a general survey and comparison, SIGKDD Explore. 2 (2000) (1), pp. 58–64.

[9] P. Bradley, J. Gehrke, R. Ramakrishnan and R. Srikant, Scaling mining algorithms to large databases, Comm. ACM 45 (2002) (8), pp. 38–43.

[10] J. Han and M. Kamber, Data Mining: Concepts and Techniques, Morgan Kaufmann Publisher, San Francisco, CA, USA (2001).

[11] Girish K. Palshikar, Manan S. Kale & Manoj M. Apte, Association rules mining using heavy itemsets, *IEEE Trans. Knowl. Data Eng.*

[12] Park J. S., Chen M. S., and Yu P. S. "An Effective Hash-based Algorithm for Mining Association Rules." Proceedings of the ACM-SIGMOD Conference on Management of Data, 1995. Extended version appears as: "Using a Hash-based Method with Transaction Trimming for Mining Association Rules." IEEE Transactions of Knowledge and Data Engineering, Volume 9, no 5, September 1997, pages 813-825.

[13] Toivonen H. "Sampling Large Databases for Association Rules". Proceedings of the 22nd International Conference on Very Large Databases, Bombay, India, September 1996.

[14] Srikant R., and Agrawal R. "Mining quantitative association rules in large relational tables". Proceedings of the ACM SIGMOD Conference on Management of Data, 1996. pages 1-12.

[15] Lent B., Swami A., and Widom J. "Clustering Association Rules." Proceedings of the Thirteenth International Conference on Data Engineering. pages 220-231, Birmingham, UK, April 1997.

[16] Aggarwal C. C., Sun Z., and Yu P. S. "Generating Profile Association Rules." IBM Research Report, RC-21037.

[17] Agrawal Rakesh and John C.Shafer "Parallel Mining of Association Rules: Design, Implementation and Experience" IBM Research Report, RJ-10004.