*J. Sci. Trans. Environ. Technov.* 2009, 3(1): 25 - 29

# Quick Mining Algorithm (QMA) using pattern mining techniques for roomy data base

**J. Umamaheswari* and S.Ravichandran**
Department,of Computer Science, Govt., Arts and Science College for Women, Pudukkottai - 622 001 Tamil Nadu, India
Department of Computer Science, H.H. The Rajah's College , Pudukkottai - 622 001, Tamil Nadu, India.

## Abstract

Pattern mining plays an essential role in many data mining tasks such as mining sequential patterns, multi-dimensional patterns, max-patterns, and emerging patterns. Pattern mining techniques can also be extended to solve many other problems involving computation and classification. Pattern mining is a significant research problem. This paper discusses the problem of efficient and effective pattern mining with an orientation towards a fast mining method.

**Keywords :** apriori algorithm, association rule, pattern growth, quick search

## 1. INTRODUCTION

To create the set of candidate patterns of length ($k$+1) from the set of frequent patterns of length $k$ (for $k$¸1), and check their matching occurrence frequencies in the database, the *Apriori* heuristic approach achieves fine performance gain by reducing the size of candidate sets. However, in situations with productive frequent patterns, long patterns, or quite low minimum support thresholds, an *Apriori*-like algorithm may still suffer from the nontrivial costs. It is expensive to handle a huge number of candidate sets. For example, the *Apriori* algorithm will need to produce more candidates and test their occurrence frequencies and candidates in total. This is the inherent cost of candidate generation no matter what implementation technique is applied (Han *et al.,* 2000). Furthermore, it is monotonous to repeatedly scan the database and check a large set of candidates by pattern matching, which is especially true for mining long patterns.

Thus, *Apriori algorithm* has its advantages and disadvantages. A way is to improve the effectiveness of frequent pattern mining substantially, and avoiding the costly candidate-generation-and-test and repeated database scan operations while obtaining this advantage. Frequent pattern mining often suffers not only from the lack of efficiency but also from the lack of effectiveness, i.e., there could be a huge number of frequent patterns generated from a database.

As frequent pattern mining is a vital data-mining task, developing efficient frequent mining techniques has been an important research direction in data mining. This paper tries to make good movement in that direction.

*Corresponding Author
email: *rajgokul2002@yahoo.com*

## 2. CONTRIBUTIONS

In this paper, we study the problem of frequent pattern mining (Wang *et al.,* 2002; Sriphaew and Theeramunkong, 2004), as well as some of its extensions. In particular, we make the following contributions. We systematically developed an association rule based pattern mining. A novel algorithm QMA is proposed for efficiently mining frequent patterns from voluminous dense datasets (Burdick *et al.,* 2001). Constraint-based data mining is an important approach to solve the problem of data mining. We study the problem of constraint-based fast mining using pattern-growth methods (Bayardo *et al.,* 1999). Our study shows that pattern-growth methods can push constraints deeper into the mining process.

We extend the fast pattern growth method to allow the mining of sequential patterns. Our approach shows that fast pattern mining methods are more efficient in mining voluminous sequence databases. Interesting techniques are developed to solve the sequential fast search mining problem effectively.

## 3. PROBLEM DEFINITIONS

Association rules can be derived from frequent patterns (Agrawal and Srikant, 1994). An association rule is an implication of the form $X \rightarrow Y$, Where $X$ and $Y$ are itemsets and $X \cap Y = \theta$ the rule $X \rightarrow Y$ has *support s* in a transaction database *TDB* if $sup_{TDB}(X \cup Y) = S$.

The rule $X \rightarrow Y$ holds in the transaction database *TDB* with *confidence* C where $C = sup(X \cup Y)/ sup(X)$.

The frequent pattern mining problem is mainly focused towards the mining association rules between sets of items. Based on the research findings and solution of Agrawal and Srikant, (1994) we can define

Let $I = \{ i_{1,.....,} i_m \}$ be a set of *items*.

An itemset $X \subseteq$ I. Particularly, an itemset with $I$ items

is called an *I-itemset*. A transaction $T = (tid,X)$ is a tuple where *tid* is a *transaction-id* and $X$ is an *itemset*.A transaction $T = (tid,X)$ is said to contain itemset $Y$ *if* $Y \subseteq X$.A transaction database *TDB* is a set of transactions.

The support of an itemset $X$ in transaction database *TDB*, denoted as $sup_{TDB}(X)$ or $sup(X)$, is the number of transactions in *TDB* containing $X$.

### 3.1. Problem statement

Given a user-specified support threshold *min sup* $X$ is called a frequent itemset or frequent pattern if $sup(X)$ *min sup*. The problem of mining frequent itemset is to find the complete set of frequent itemset in a transaction database *TDB* with respect to a given support threshold *min- sup*. Given a transaction database *TDB*, a support threshold *min sup* and a confidence threshold *min conf*, the problem of association rule mining is to find the complete set of association rules that have support and confidence no less than the user-specified thresholds, respectively.

Association rule mining can be divided into two steps.

i. Frequent patterns with respect to support threshold *min sup* are mined.

ii. Association rules are generated with respect to confidence threshold *min conf*.

As shown in many studies, the first step, mining frequent patterns, is significantly more costly in terms of time than the rule generation step. Frequent pattern mining is not only used in association rule mining. Instead, frequent pattern mining is the basis for many data mining tasks, such as sequential pattern mining and associative classification (Li *et al.*, 2001). It also has broad applications, such as basket data analysis, cross-marketing, catalog design, sale campaign analysis, web log analysis, etc.

## 4. APRIORI ALGORITHMS

To achieve efficient mining frequent patterns, an anti-monotonic property of frequent itemsets, called the Apriori heuristic can be used. The Apriori heuristic can prune candidates based on this property, a fast frequent itemset mining algorithm, called Apriori.

**Apriori:** Any superset of an infrequent itemset cannot be frequent. In other words, every subset of a frequent itemset must be frequent.

**Proof:** To prove the theorem, we only need to show $sup(X) \cdot sup(Y)$.

Given a transaction database *TDB*, Let $X$ and $Y$ be two itemsets.

For each transaction $T$ containing itemset $X$, $T$ also contains $Y$, which is a subset of $X$.

Thus, we have $sup(X) \cdot sup(Y)$.

### 4.1. Apriori finds the complete set of frequent itemsets as follows

**1.** Scan *TDB* once to find frequent items, i.e. items appearing in transactions.

If items (*a, b, c, f, m, p*). Each of these six items forms a *length*-1 *frequent itemset*.

Let $L1$ be the complete set of *length-1 frequent itemsets.*

**2.** The set of *length*-2 *candidates*, denoted as $C2$, is generated from $L1$. Here, we use the *Apriori heuristic* to prune the candidates.

Only those candidates that consist of frequent subsets can be potentially frequent.

**3.** Scan *TDB* once more to count the support of each itemset in $C2$. The itemsets in $C2$ passing the support threshold form the *length*-2 *frequent itemsets*,

**4**. Then, we form the set of length-3 candidates. Only those length-3 itemsets for which every length-2 sub-itemset is in $L2$ are qualified as candidates. For example, *acf* is a length-3 candidate since *ac*, *af* and *cf* are all in $L2$.

One scan of *TDB* identifies the subset of length-3 candidates passing the support threshold and forms the set $L3$ of length-3 frequent itemsets. A similar process goes on until no candidate can be derived or no candidate is frequent.

One can verify that the above process eventually finds the complete set of frequent itemsets in the database *TDB* (Sriphaew and Theeramunkong, 2004).

The *Apriori* algorithm is presented as follows.

### 4.2. Apriori algorithm flow

**Input :** Transaction database *TDB* and support threshold *min sup*

**Output :** The complete sets of frequent patterns in TDB with respect to support threshold *min sup*

**Method**

**1.** Scan transaction database *TDB* once to find $L1$, the set of frequent *1-itemsets*;

**2.** for ($k = 2$; $L_{k-1} \neq \theta$;; $k++$) do

(a) Generate $Ck \subseteq$ length-$k$ candidates.

A *k-itemset* $X$ is in $Ck$ if and only if

every length-($L_{k-1}$) $\subseteq X$ is in $L_{k-1}$;

(b) if $Ck = \theta$; then go to Step 3;

(c) Scan transaction database *TDB* once to count the support for every itemset in $Ck$;

(d) $Lk = \{X \mid (X \in Ck) \wedge (sup(X) \geq min\_sup)\}$;

**3.** Return $U^{k}_{i=1} Li$

The Apriori heuristic helps reducing the number of candidates significantly. Since there are in number of items appearing in the database, there could be possible length of itemsets with the Apriori heuristic, we only need to check the support counts for length of candidates. Apriori cuts maximum at the length of itemset level. As the length of candidates becomes longer, the number of possible combinations becomes larger, thus the cutting effect of the Apriori heuristic is sharper.

Even though Apriori can cut a lot of candidates, it could still be costly to handle a huge number of candidate itemsets in large transaction databases. For example, if there are 1 million items and only 1% is frequent length itemsets, Apriori has to generate more length candidates, test each of their support and save them for length candidates' generation. It is tedious to repeatedly scan the database and check a large set of candidates by pattern matching, which is particularly true if a long pattern exists. Apriori is a level by level candidate-generation-and-test algorithm. To find a frequent itemset X, Apriori has to scan the database 100 times.

Apriori encounters difficulty in mining long patterns. For example, to find a frequent itemset *X,* it has to generate and test candidates.

## 5. FAST SEARCH APPLY THROUGH PATTERN GROWTH METHOD

The major costs in Apriori-like methods are the generation of a huge number of candidates and the repeated scanning of large transaction databases to test those candidates. In short, the candidate-generation and test operation is the bottleneck for Apriori-like methods. So we need to avoid candidate-generation and test in frequent pattern mining. To attack this problem, we develop *Quick Mining Algorithm (QMA),* a pattern growth method for frequent pattern mining. We propose an efficient algorithm for mining frequent patterns from an FP and we discuss how to scale the method to mine large databases.

Information from transaction databases is essential for mining frequent patterns. Therefore, if we can extract the concise information for frequent pattern mining and store it into a compact structure (Pramudiono and Kitsuregawa, 2004), then it may facilitate frequent pattern mining. Motivated by this thinking, in this paper, we develop a compact data structure, called *QMA,* to store complete but no redundant information for frequent pattern mining.

To design a compact data structure for efficient frequent-pattern mining, let's first examine an example.

A compact data structure can be designed based on the following observations:

1. Since only the frequent items will play a role in the frequent-pattern mining, it is necessary to perform one scan of transaction database *TDB* to identify the set of frequent items.
2. If the *set* of frequent items of each transaction can be stored in some compact structure,it may be possible to avoid repeatedly scanning the original transaction database.
3. If multiple transactions share a set of frequent items, it may be possible to merge the shared sets with the number of occurrences registered as *count*. It is easy to check whether two sets are identical if the frequent items in all of the transactions are listed according to a fixed order.
4. If two transactions share a common prefix, according to some sorted order of frequent items, the shared parts can be merged using one prefix structure as long as the *count* is registered properly.

### 5.1 Algorithm - Quick Mining Algorithm (QMA) construction

**Input:** A transaction database *TDB* and a minimum support threshold min sup.

**Output:** FP, the frequent-pattern tree of *TDB*.

**Method:** The FP is constructed as follows.

**1.** Scan the transaction database *TDB* once. Collect *F*, the set of frequent items, and the support of each frequent item.

**2.** Create the root of an FP-tree, *T*, and label it as "null". For each transaction *t* in *TDB* do the following.

Select the frequent items in transaction t and sort them according to the order of *FList*.

Let the sorted frequent-item list in t be *[p | P],*
*where p* is the first element and P is the remaining list.
Call *insert tree ([p | P], T).*
The function *insert tree ([p | P], T)* is performed as follows.
*If T* has a *child N* such that
*N.item-name = p.item-name*
*then* increment *N's* count by 1;
*else* create a new *node N*, with count initialized to 1,
parent link linked to *T*, and node-link linked to the nodes with the same *item-name* via the node-link structure.
*If P* is nonempty, call *insert tree* (*P,N*) recursively.

**Analysis:** The *FP-tree* construction takes exactly two

scans of the transaction database[10]:

**1.** The first scan collects the set of frequent items.

**2.**The second scan constructs the Quick Mining

### 5.2 Algorithm - Quick Mining Algorithm (QMA)

The cost of inserting a *transaction t* into the *FP-tree* is the set of frequent items in *t*.

**Input :** A database DB, represented by QMA constructed

**Output :** The complete set of frequent patterns.

**Method :** call QMA-growth (FP-tree, null).

```
Procedure QMA-growth (Tree, A)
 {
  if
    Tree contains a single_ prefix- path
     // Mining single prefix-path QMA-tree
  then
  {
   let P be the single_ prefix-path part of Tree;
   let Q be the multiple-path part with the top
      branching node replaced by a null root;
   for each combination of the nodes in the
        path P
  do   generate_ pattern B U A with
    support = minimum support of nodes in B;
   let   pattern_mining_set(P)   Í
                              patterns_generated;
     }
      else
       let Q be Tree;
       for  each item ai in Q
       do
{
 // Mining multiple-path QMA-tree
 Generate_ pattern B= ai U A
              [ support = ai:support ];
 construct B's  conditional pattern-base and
              then  B conditional QMA-tree ;
  if  Tree_B '" è;
  then call QMA-growth(Tree_B,B);
   let pattern_ set(Q) Í patterns_ generated;
          }
  return
   (pattern_set(P)Upattern_set(Q)U
          pattern_set(P) X pattern_set(Q)))
    }
```

With the properties, we show that the algorithm correctly finds the complete set of frequent itemsets in transaction database *DB*.

## 6. SOLUTIONS

In this paper, we review some important drawbacks in *Apriori* algorithm and suggest strategies for improvement.

The *Apriori* algorithm needs to scan the database multiple times. When mining a huge database, multiple database scans are costly. One feasible strategy to improve the efficiency of *Apriori* algorithm is to reduce the number of database scans.

Further more, the *Apriori* algorithm has to generate a huge number of candidates. Storing and counting these candidates are tedious. To hit this problem, some studies focus on reducing the number of candidates. One dominant operation in the *Apriori* algorithm is support counting.

Our new approach *quickly* reads *M transactions* at a time and updates the appropriate support counts. For e.g., when the "first search" reaches the end of the transaction database, it has made one scan over the data and it starts over at the beginning for the next scan. The "*search object*" are on the *Roomy Data Base(RDB)* is *candidate itemset*. If an *itemset* is on the *RDB*, its support is updated each time a transaction containing the *itemset* is scanned.

At the start of the first scan, the "*search object*" on the *RDB*, is the set of *length-1* candidates.

At each stop it checks the "*search object*" on the *RDB*, according to the following rules.

When the support count of a candidate *itemset X* passes the support threshold, it checks whether *X* can be joined with some other *frequent itemsets* [5] with the same length to generate new candidates. If so, it adds the new candidate on the bus.

It generates a *length-k, itemset X, subsets of X* which have accumulated support greater than or equal to the support threshold. When a candidate *itemset X* has travelled for one complete scan, it is removed from the bus. If at that time, the support for *X* is greater than or equal to the support threshold, output *X* and its support as a frequent pattern.

For example, let us consider mining a Voluminous Transactional database (*VTDB)* with 20,000 transactions and support threshold 50. Let the interval between stops be 5,000. By overlapping the counting of different lengths of *itemsets*, it can save some database scans. On the other hand, it can also explore efficient support counting. It optimizes the structure used in the *Apriori* algorithm for counting candidates. Frequent items in each candidate are sorted in support ascending order according to their popularity in the first *M transactions*. Such an order can reduce the number of inner loops in counting. Reordering items incurs some overhead, but for some data, it may be beneficial overall.

## 7. CONCLUSIONS

*Quick Mining Algorithm (QMA)* is testified in roomy database. Database is divided into multiple partitions where each partition can be held in main memory. The whole voluminous database is rapidly scanned only twice.

**First scan :** Partitions are read into main memory one by one. Local frequent patterns are mined with respect to relative support threshold using the apriori method.

**Second scan:** Consolidates global frequent patterns. Each global frequent pattern must be frequent in at least one partition. Therefore, only those local frequent patterns should be counted and tested in the second scan.

The major conclusions for this fast mining method are in two aspects.

i. Partitioning the database is non-trivial when the database is biased.

ii. On the other hand, a low global support threshold may lead to a much lower local threshold and thus produce a huge number of local frequent patterns. By fast mining, we can observe the patterns hidden behind the data more accurately, more steadily and more proficiently.

## ACKNOWLEDGEMENT

The authors are grateful to their Ph.D. guide Dr.R.Balasubramnian, Principal, Government Arts and Science College, Sivagangai, Tamilnadu for his valuable guidance in the area of Knowledge Management.

## REFERENCES

Agrawal, R. and Srikant, R. 1994. Fast algorithms for mining association rules. *In* : Proc. 1994 Int Conf. Very Large Data Bases (VLDB'94), P. 487–499, Santiago, Chile.

Bayardo, R. J., Agrawal, R. and Gunopulos, D. 1999. Constraint-based rule mining on large dense data sets. *In:* Proc. 1999 Int. Conf. Data Engineering (ICDE'99), Sydney, Australia.

Kephant, J. and Arnold, W. 2000. Frequent pattern-projected sequential pattern mining. *In:* Proc. 2000 Int. Conf. Knowledge Discovery and Data Mining (KDD'00), P. 355–359, Boston, MA, August 2000.

Han, J., Pei, J. and Yin, Y. 2000. Mining frequent patterns without candidate generation. *In:* Proc. 2000 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD' 00), P. 1–12, Dallas, TX.

Pei, J., Han, J. and Mao, R. 2000. An efficient algorithm for mining frequent closed itemsets. *In:* Proc.ACM/ SIGMOD International Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD), Dallas, TX, P.21-30.

Li, W., Han, J. and Pei, J. 2001. Accurate and efficient classification based on multiple class association rules. *In:* Proc. IEEE 2001 Int. Conf. Data Mining (ICDM'01), San Jose, A..

Burdick, D., Calimlim, M. and Gehrke, J. 2001. MAFIA: A maximal frequent itemset algorithm for transactional databases. *In:* Proc. International Conference on Data Engineering (ICDE), Heidelberg, Germany, P.443-452.

Wang, K., Tang, L., Han, J. and Liu, J. 2002. Top down FP-growth for association rule mining Techniques. *In:* Proc. Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD), Taipei, Taiwan, P.334-340.

Sriphaew, K. and Theeramunkong, T. 2004. Fast algorithms for mining generalized frequent patterns of generalized association rules. *In:* IEICE Transactions on Information and Systems.

Pramudiono, I. and Kitsuregawa, M. 2004. FP-tax: Tree structure based generalized association rule mining. *In:* Proc. ACM/SIGMOD International Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD), Paris, France, P.60-63.