

# Improved Association Mining Algorithm for Large Dataset

Tannu Arora<sup>1</sup>, Rahul Yadav<sup>2</sup>

<sup>1</sup> Computer Science, PDM college of Engg,  
Bahadurgarh, Haryana, India  
*tannu.arora@gmail.com*

<sup>2</sup> Computer Science, PDM college of Engg,  
Bahadurgarh, Haryana, India  
*rahulyadav.project@yahoo.com*

## Abstract

Mining frequent patterns in transaction databases, time-series databases, and many other kinds of databases has been studied popularly in data mining research. Most of the previous studies adopt an Apriori-like candidate set generation-and-test approach. However, candidate set generation is still costly, especially when there exist a large number of patterns and/or long patterns. Further, dynamic itemset counting algorithm, an extension to Apriori algorithm used to reduce number of scans on the dataset. It was alternative to Apriori Itemset Generation. In this, itemsets are dynamically added and deleted as transactions are read. It relies on the fact that for an itemset to be frequent, all of its subsets must also be frequent, so we only examine those itemsets whose subsets are all frequent. Both Apriori and DIC are based on candidate generation. But frequent-pattern tree (FP-tree) structure, which is an extended prefix-tree structure for storing compressed, crucial information about frequent patterns, mine the frequent itemsets without any candidate generation. Our performance study shows that the *FP-growth* method is efficient and scalable for mining both long and short frequent patterns, and is about an order of magnitude faster than the Apriori algorithm and also faster than some recently reported new frequent-pattern mining methods. In this paper, we combine the features of both FP-tree and dynamic itemset so that we can mine the frequent itemsets dynamically without any candidate generation. We propose a new algorithm, Dynamic-FP which takes the advantages of both DIC as well as FP-tree. This algorithm will be compared with previous algorithms to give the better performance for large dataset.

**Keywords:** Association rules, Apriori, Dynamic itemset counting, Frequent pattern growth, Dynamic-FP

## 1. Introduction

Association rule mining is an important topic in data mining field. Data Mining (DM) is an important issue in the field of data and knowledge based systems, which have been prompted by an interesting new field called Knowledge Discovery in Databases (KDD). Discovered knowledge can come in many forms such as: association rules, correlations, sequences, episodes, classifiers, clusters and many more. Mining for association rules had received

great attention in recent years. The original motivation for searching association rules came from the need to analyze the so-called supermarket transaction data, that is, to examine customer behavior in terms of the purchased products.

Association rule mining finds interesting association or correlation relationships among a large set of data items. With massive amounts of data continuously being collected and stored in databases, many industries are becoming interested in mining association rules from their databases. For example, the discovery of interesting association relationships among huge amounts of business transaction records can help catalog design, cross-marketing, lossleader analysis, and other business decision making processes. The two main applications of association mining are market basket analysis and finding prediction rules.

## 2. Applications of Association Mining

The two main applications of association mining are market basket analysis and finding prediction rules.

### 2.1 Market basket analysis

A typical example of association rule mining is market basket analysis. This process analyzes customer buying habits by finding associations between the different items that customers place in their shopping baskets" as in Figure 1.

The discovery of such associations can help retailers develop marketing strategies by gaining insight into which items are frequently purchased together by customers. For instance, if customers are buying milk, how likely are they to also buy bread (and what kind of bread) on the same trip to the supermarket?

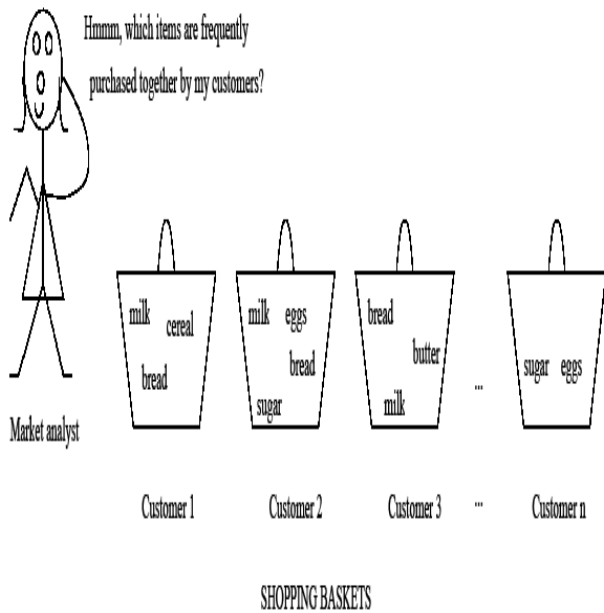


Figure 1. Market Basket Analysis

Market basket analysis identifies customers purchasing habits. It provides insight into the combination of products within a customer's 'basket'. The term 'basket' normally applies to a single order. However, the analysis can be applied to other variations. We often compare all orders associated with a single customer.

### 2.2 Interest measures

A major choice in association mining is how the interestingness of an association should be measured. The lift measure has only recently emerged on the scene. Historically, the dominant approach has been support/confidence. In this approach, we seek cells with highest prediction confidence  $p(i|j)$  subject to having support above a threshold:  $p(i,j) > t$ . This measure of association arose mainly for computational reasons, since it allows us to restrict our attention to the largest counts, which are easy to identify. (This is not possible with lift, since lift can be high even if the actual count is small, as long as the expected count is even smaller.)

For example, the information that customers who purchase computers also tend to buy financial management software at the same time is represented in association Rule below.

computer  $\rightarrow$  financial management software [support = 2%; confidence = 60%]

Rule support and confidence are two measures of rule interestingness. They respectively reflect the usefulness and certainty of discovered rules. A support of 2% for association Rule means that 2% of all the transactions

under analysis show that computer and financial management software are purchased together. A confidence of 60% means that 60% of the customers who purchased a computer also bought the software. Typically, association rules are considered interesting if they satisfy both a minimum support threshold and a minimum confidence threshold. Such thresholds can be set by users or domain experts.

## 3. Association Mining Algorithms

### 3.1 Apriori Algorithm

Apriori algorithm<sup>[7]</sup> is an influential algorithm for mining frequent itemsets for Boolean association rules. It uses a Level-wise search, where  $k$ -itemsets (An itemset that contains  $k$  items is a  $k$ -itemset) are used to explore  $(k+1)$ -itemsets, to mine frequent itemsets from transactional database for Boolean association rules.

First, the set of frequent 1-itemsets is found. This set is denoted  $L_1$ .  $L_1$  is used to find  $L_2$ , the frequent 2-itemsets, which is used to find  $L_3$ , and so on, until no more frequent  $k$ -itemsets can be found. The finding of each  $L_k$  requires one full scan of the database.

Apriori property: All non-empty subsets of a frequent itemset must also be frequent.

It performs the following tasks:

1. Reducing the search space to avoid finding of each  $L_k$  requires one full scan of the database
2. If an itemset  $I$  does not satisfy the minimum support threshold,  $min\_sup$ , the  $I$  is not frequent, that is,  $P(I) < min\_sup$
3. If an item  $A$  is added to the itemset  $I$ , then the resulting itemset (i.e.,  $I \cup A$ ) cannot occur more frequently than  $I$ . Therefore,  $I \cup A$  is not frequent either, that is,  $P(I \cup A) < min\_sup$ .

A two step process is followed, consisting of join and prune actions.

1. The join step: To find  $L_k$ , a set of candidate  $k$ -itemsets is generated by joining  $L_{k-1}$  with itself. This set of candidates is denoted  $C_k$ . The join,  $L_{k-1}$  with  $L_{k-1}$ , is performed, where members of  $L_{k-1}$  are joinable if they have  $(k-2)$  items in common.

2. The prune step:  $C_k$  is a superset of  $L_k$ , that is, its members may or may not be frequent, but all of the frequent  $k$ -itemsets are included in  $C_k$ . A scan of the database to determine the count of each candidate in  $C_k$  would result in the determination of  $L_k$  (i.e., all candidates having a count no less than the minimum support count are

frequent by definition, and therefore belong to  $L_k$ ).  $C_k$ , however, can be huge, and so this could involve heavy computation. To reduce the size of  $C_k$ , the Apriori property is used as follows. Any  $(k-1)$ -itemset that is not frequent cannot be a subset of a frequent  $k$ -itemset. Hence, if any  $(k-1)$ -subset of a candidate  $k$ -itemset is not in  $L_{k-1}$ , then the candidate cannot be frequent either and so can be removed from  $C_k$ . This subset testing can be done quickly by maintaining a hash tree of all frequent itemsets.

### Limitations

1. The algorithm is of low efficiency, such as firstly it needs to repeatedly scan the database, which spends much in I/O.
2. Secondly, it create a large number of 2- candidate itemsets during outputting frequent 2- itemsets.
3. Thirdly, it doesn't cancel the useless itemsets during outputting frequent  $k$ - itemsets.

### Methods to Improve Apriori's Efficiency

- Hash-based itemset counting:** A  $k$ -itemset whose corresponding hashing bucket count is below the threshold cannot be frequent.
- Transaction reduction:** A transaction that does not contain any frequent  $k$ -itemsets is useless in subsequent scans.
- Partitioning:** Any itemset that is potentially frequent in DB must be frequent in at least one of the partitions of DB.
- Sampling:** mining on a subset of given data, lower support threshold + a method to determine the completeness.
- Dynamic itemset counting:** add new candidate itemsets only when all of their subsets are estimated to be frequent.

### 3.2 Dynamic itemset counting(DIC)

It is an extension to Apriori algorithm used to reduce number of scans on the dataset.

- Alternative to Apriori Itemset Generation
- Itemsets are dynamically added and deleted as transactions are read
- Relies on the fact that for an itemset to be frequent, all of its subsets must also be frequent, so we only examine those itemsets whose subsets are all frequent.

A dynamic itemset counting technique was proposed in which the database is partitioned into blocks marked by

start points. In this variation, new candidate itemsets can be added at any start point, unlike in Apriori, which determines new candidate itemsets only immediately prior to each complete database scan. The technique is dynamic in that it estimates the support of all of the itemsets that have been counted so far, adding new candidate itemsets if all of their subsets are estimated to be frequent. The resulting algorithm requires two database scans<sup>[1]</sup>. See in figure 2.

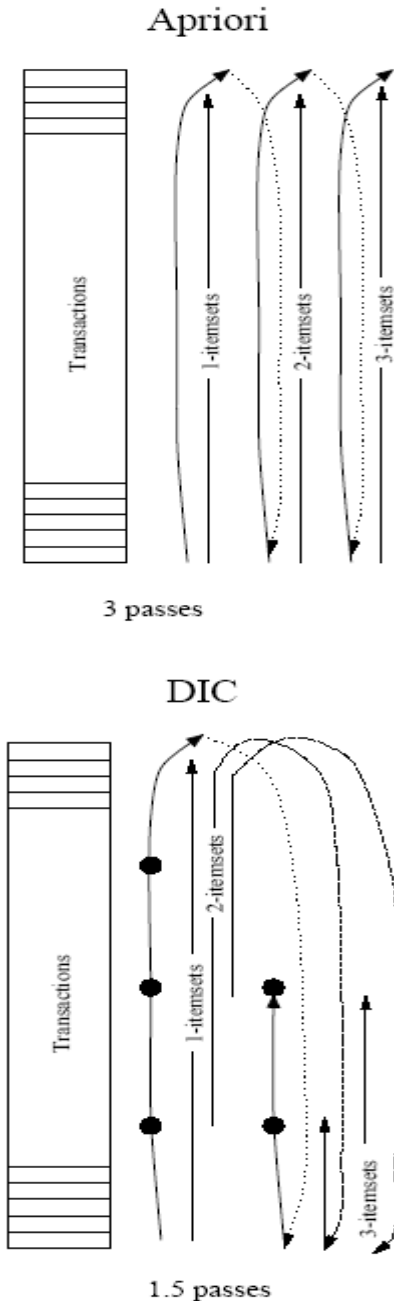






Fig 2. Apriori and DIC

Itemsets are marked in four different ways as they are counted:

- **Solid box:**  confirmed frequent itemset - an itemset we have finished counting and exceeds the support threshold *minsupp*
- **Solid circle:**  confirmed infrequent itemset - we have finished counting and it is below *minsupp*
- **Dashed box:**  suspected frequent itemset - an itemset we are still counting that exceeds *minsupp*
- **Dashed circle:**  suspected infrequent itemset - an itemset we are still counting that is below *minsupp*

### 3.3 Frequent-Pattern(FP-tree): Mining Frequent Patterns Without Candidate Generation

In many cases the Apriori candidate generate-and-test method significantly reduces the size of candidate sets, leading to good performance gain. However, it can suffer from two nontrivial costs:

- It may need to generate a huge number of candidate sets
- It may need to repeatedly scan the database and check a large set of candidates by pattern matching

FP-Growth<sup>[7]</sup> performs in following ways:

Compress a large database into a compact, Frequent-Pattern tree (FP-tree) structure  
 –highly condensed, but complete for frequent pattern mining  
 –avoid costly database scans

Develop an efficient, FP-tree-based frequent pattern mining method

A divide-and-conquer methodology: decompose mining tasks into smaller ones

Avoid candidate generation: sub-database test only!

FP-Growth Method : An Example

- First, create the root of the tree, labeled with “null”.
- Scan the database D a second time. (First time we scanned it to create 1-itemset and then L).
- The items in each transaction are processed in L order (i.e. sorted order).
- A branch is created for each transaction with items having their support count separated by colon.

- Whenever the same node is encountered in another transaction, we just increment the support count of the common node or Prefix.
- To facilitate tree traversal, an item header table is built so that each item points to its occurrences in the tree via a chain of node-links.
- Now, The problem of mining frequent patterns in database is transformed to that of mining the FP-Tree.

Table 1: Transaction database

TID	List of Items
T100	I1,I2,I5
T100	I2, I4
T100	I2, I3
T100	I1, I2, I4
T100	I1, I3
T100	I2, I3
T100	I1, I3
T100	I1, I2 ,I3, I5
T100	I1, I2, I3

Suppose min. support count required is 2 (i.e.  $\text{min\_sup} = 2/9 = 22\%$ )

- The first scan of database is same as Apriori, which derives the set of 1-itemsets & their support counts.
- The set of frequent items is sorted in the order of descending support count.
- The resulting set is denoted as  $L = \{I2:7, I1:6, I3:6, I4:2, I5:2\}$

Construction of FP-Tree

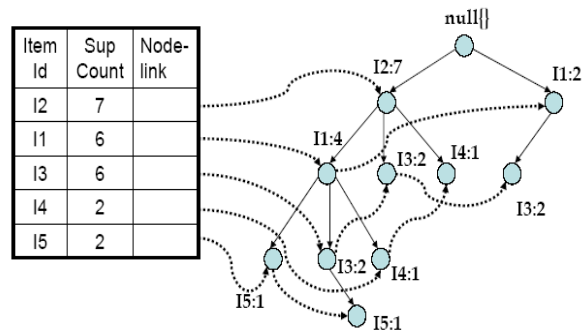


Fig 3. FP-Tree

Frequent itemset generated

I2 I5:2, I1 I5:2, I2 I1 I5: 2

I2 I4: 2

I2 I3:4, I1, I3: 2 , I2 I1 I3:2

I2 I1: 4

Efficiency of mining is achieved with three techniques:

(1) a large database is compressed into a condensed, smaller data structure, FP-tree which avoids costly, repeated database scans,

(2) our FP-tree-based mining adopts a pattern-fragment growth method to avoid the costly generation of a large number of candidate sets.

(3) a partitioning-based, divide-and-conquer method is used to decompose the mining task into a set of smaller tasks for mining confined patterns in conditional databases, which dramatically reduces the search space.

## 4. Proposed Work

### 4.1 Dynamic-FP

We propose a new algorithm, Dynamic-FP which takes the advantages of both DIC as well as FP-tree. This algorithm will be compared with previous algorithms to give the better performance for large dataset.

Algorithm

1. Mark the empty itemset (0-level) with a solid square. Mark all first-level itemsets with dashed circles. Leave all other level itemsets unmarked.

2. While leaf node is not encountered:

a) If a dashed circle's count exceeds the minimum support, turn it into a dashed square.

b) Mark all next-level itemsets to dashed circle. Make previous level dashed square itemset solid.

Pseudo code

```
SS=NULL // solid square (frequent)
SC = NULL// solid circle (infrequent)
DS = NULL // dashed square (suspected frequent)
DC = {Ist level itemset} // dashed circle (suspected infrequent)
ptr=root
while (ptr != NULL) do begin
```

```
DC=ptr
for each itemset c in DC do
if ( c.counter >= threshold ) then
move c from DC to DS;
add ptr->child to DC & ptr to SS ;
end
ptr++;
end
Answer = { c < SS } ;
```

Frequent itemset generated for above example(figure)

will be:

{I2I!},{I2I3},{I2I1I3},{I1I3},{I2I1},{I1}

### 4.2 Relative Performance

We have compared previous algorithms with the proposed algorithm, that is, dynamic itemset counting, frequent pattern and dynamic-FP are compared. As dynamic-FP takes the advantages of both, it performs faster than DIC and FP.

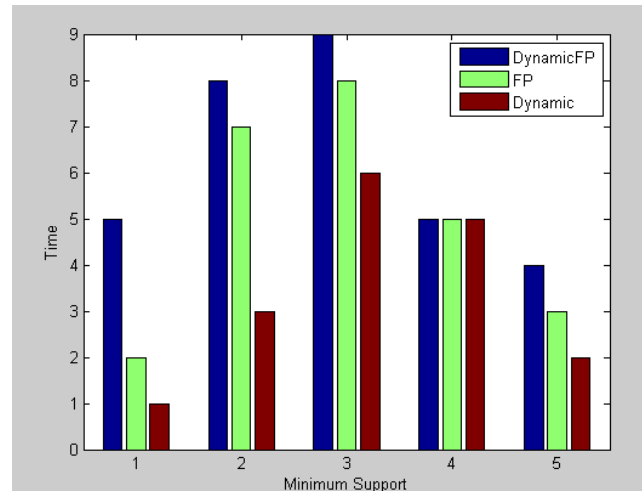


Fig 4. Minimum support vs Execution Time

As the minimum support decreases, the execution times of all the algorithms increase because of increases in the total number of candidate and large itemsets. Clearly, Dynamic-FP beats FP by more than an order of magnitude for large datasets which further beats dynamic for all problem sizes, by factors ranging from 2 for high minimum support to more than an order of magnitude for low levels of support. FP always did considerably better than DIC.

## 5. Conclusions

This paper is mainly based on the study of existing algorithms for mining association rules. We propose a new algorithm, Dynamic-FP which takes the advantages of both DIC as well as FP-tree. This algorithm is to be compared with previous algorithms to give the better performance for large dataset. Further, Dynamic-FP can be implemented using various data mining tools like Weka, XLMiner. The objective is to develop algorithms that exploits these redundancies to alter the original algorithms in such a way to achieve better results.

## References

- [1] R. Srikant, R. (1994). Fast algorithms for mining association rules. In J. B. Bocca, M. Jarke, & C. Zaniolo (Eds.), Proc 20th Int Conf Very Large Data Bases VLDB (Vol. 1215, pp. 487-499).
- [2] R. Agarwal, T. Imielinski, and A. Swami. Mining association rules in large databases. In Proc. Of ACM SIGMOD Conference on Management of Data, Washington.
- [3] R. Agrawal, T. Imielinski, and A. Swami. Database mining: A performance perspective. IEEE Transactions on Knowledge and Data Engineering, 5(6):914-925, December 1993. Special Issue on Learning and Discovery in Knowledge-Based Databases.
- [4] J. Han, Y. Cai, and N. Cercone. Knowledge discovery in databases: An attribute oriented approach. In Proc. of the VLDB Conference, pages 547-559, Vancouver, British Columbia, Canada, 1992.
- [5] Cheung, D., Han, J. & Wong, C. (1996). Maintenance of discovered association rules in large databases: An incremental updating technique. New Orleans: IEEE computer society press
- [6] Chen, M. S., Han, J., & Yu, P.S. (1996). Data Mining: An overview from a database perspective. IEEE Transactions on Knowledge and Data Engineering, 8(6), 866-833.
- [7] J Han, M Kamber, ' Data mining: Concepts and techniques', Morgan Kaufman Publishes, 1992.