Algorithmic Framework for Frequent Pattern Mining with FP-Tree

Georgina N. Obunadike[1&2*] Audu Isah[2] Arthur Umeh[3] H. C. Inyiamah[4]
1. Department of Mathematical Sciences and IT, Federal University, Dutsin-ma, Katsina State
2. Department of Mathematics and Statistics, Federal University of Technology, Minna, Niger State
3. Department of Information and Media Technology, Federal University of Technology, Minna, Niger State
4. Department of Computer Electronics, NNamdi Azikiwe University, Awka, Anambra State
*Email: nkoliobunadike@yahoo.com

Abstract
The FP-tree algorithm is currently one of the fastest approaches to frequent item set mining. Studies have also shown that pattern-growth method is one of the most efficient methods for frequent pattern mining. It is based on a prefix tree representation of the given database of transactions (FP-tree) and can save substantial amounts of memory for storing the database. The basic idea of the FP-growth algorithm can be described as a recursive elimination scheme which is usually achieved in the preprocessing step by deleting all items from the transactions that are not frequent. In this study, a simple framework for mining frequent pattern is presented with FP-tree structure which is an extended prefix-tree structure for mining frequent pattern without candidate generation, and less cost for better understanding of the concept for inexperienced data analysts and other organizations interested in association rule mining.
Keywords: Association Rule, Frequent Pattern Mining, Apriori Algorithm, FP-tree

1. Introduction

Data mining is the process of discovering interesting knowledge from large amounts of data stored in databases, data warehouses, or other information repositories. Data mining is an emerging field that has gained attention in research and industry and has recently also attracted considerable attention from database practitioners, researchers and data analysts. It has application in many fields such as marketing, financial forecasts and decision support (Jiawei, Micheline, and Jian, 2011). Data-mining algorithms and visualization tools are being used to find important patterns in data and to create useful forecasts. This technology is being applied in virtually all business sections including banking, telecommunication, manufacturing, marketing, and e-commerce (Zhao Hui and Jamie, 2005).

In performing data mining tasks such as association, clustering, classification/prediction and outlier detection, various methods and techniques are used for knowledge discovery from databases. Association rule is one of the important tasks of data mining which identifies interesting association and correlation among large data sets (Intan and Rolly, 2005). Mining frequent patterns is an important aspect in association rule mining. The FP-tree algorithm is currently one of the fastest approaches to frequent item set mining. FP-Tree was first proposed by Han, Pei, and Yin (2000). FP-Tree is a compact representation of transaction database that contains frequency information of all relevant patterns in a dataset.

Association rule mining has many important applications in life. An association rule is of the form X => Y, and each rule has two measurements: support and confidence. The association rule mining problem is to find rules that satisfy user-specified minimum support and minimum confidence. It mainly includes two steps: finding all frequent patterns; and generating association rules through the frequent patterns.
The identification of sets of items, products, symptoms, characteristics, and so forth, which often occur together in a given database, can be seen as one of the most basic tasks in Data Mining (Intan and Rolly, 2005).

Let $I = (I_1, I_2, \ldots, I_m)$ be a set of item. Let D be a set of database transactions where each transaction T is a nonempty item set such that $T \subset I$. Each transaction has an identifier, called a TID. Let A be a set of items. A transaction T is said to contain A if $A \subseteq T$. An association rule is of the form $A \rightarrow B$, where $A \subset I,\ B \subset I,\ A \neq \emptyset,\ B = \emptyset,\ and\ A \cap B = \emptyset$. The rule $A \rightarrow B$ holds in the transaction set D with support s, where s is the percentage of transactions in D that contain $A \cup B$ (i.e., the *union* of sets *A* and *B*). This is taken to be the probability, $P(A \cup B)$. The rule $A \rightarrow B$ 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, $P(B \backslash A)$. That is:

$$support\ (A \rightarrow B) = P(A \cup B). \qquad (1)$$
$$Cofidence\ (A \rightarrow B) = P(B \backslash A). \qquad (2)$$

Rules that satisfy both a minimum support threshold $(min\ sup)$ and a minimum confidence threshold $(min\ conf)$ are called strong (Jiawei *et al*, 2011). By convention, we write support and confidence values so as to occur between 0% and 100%, rather than 0 to 1.0 (Jiawei *et al*, 2011). A set of items is referred to as an item set. An itemset that contains *k* items is a *k*-itemset. The set (*computer, antivirus software*) is a 2-itemset. The occurrence frequency of an itemset is the number of transactions that contain the item set. This is also known, simply, as the frequency, support count, or count of the item set. The set of frequent *k*-item sets is commonly denoted by L_k. From equation (1) it follows that:

$$Confidence\ (A \rightarrow B) = P(B \setminus A) = \frac{support\ (A \cup B)}{support\ (A)} = \frac{support_{-count}(A \cup B)}{support - count\ (A)} \qquad (3)$$

To define support and confidence more formally, let the total number of transactions be N. Support of X is the number of times it appear in the database divided by N and support of X and Y together is the number of times they appear together divided by N. Therefore, using $P(X)$ to mean probability of X in the database, we have:

$$Support\ (X) = \frac{(Number\ of\ times\ X\ appears)}{N} = P(X) \qquad (4)$$

$$Support\ (XY) = \frac{(Number\ of\ times\ X\ and\ Y\ appear\ together)}{N} = P(X \cap Y) \qquad (5)$$

Confidence of $X \rightarrow Y$ is defined as the ratio of support of X and Y together with the support of X.

$$Confidence\ of\ (X \rightarrow Y) = \frac{Support\ (XY)}{Support\ (X)} = \frac{P(X \cap Y)}{P(X)} = P(Y \setminus X) \qquad (6)$$

$P(Y \setminus X)$ is the probability of Y once X has taken place, it is therefore called conditional probability of $Y\ given\ X$. The objectives of the study is to construct an algorithm for FP-Tree that will be used to mine frequent itemsets without generating candidates, to reduce the search cost in frequent pattern-mining and apply the algorithm to a set of alphabets. The outline of the paper is as follows: section 1 introduces the FP-Tree in comparism with other methods. Section 2 is on review of related literature while section 3 discusses the FP-Tree Construction methodology. Section 4 highlights the application of FP-Tree Algorithm in a transaction database. Section 5 is on implementation with real datasets while section 6 is on conclusion.

2. Literature Review

Apriori algorithm is one of the oldest algorithms for association rule mining developed by *Agrawal, Imielinski, Swami* (1993). The algorithm has received a great deal of attention since it's introduction, many works has been done to improve the algorithm. The apriori algorithm is resource intensive for large databases that have large set of frequent items.

The FP-Tree was introduced to handle the lapses in apriori algorithm. Recent studies show that pattern-growth method is one of the most efficient methods for frequent pattern mining (Agarwal, Aggarwal, and Prasad, 2001a; Agarwal, Aggarwal, and Prasad, 2001b; Agrawal and Srikant, 1994; Bayardo, 1994; Burdick, 2001; Han and Pei, 2001; Han, Pei, and Yin, 2000; and Pei, Han, Lu, Nishio, and Shiwei Tang, 2001). As a divide-and-conquer method, this method partitions (projects) the database into partitions recursively, but does not generate candidate sets. Efficiency can be achieved in mining with FP-tree using three techniques (Han, Pei, and Yin, 2000).

A performance study shows that the FP-tree 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 such as CHARM and Tree Projection methods (Jaiwei *et al*, 2011).
This algorithm (FP-tree) mines frequent pattern without candidate generation. It uses an approach that is different from that used in Apriori algorithm.

A performance evaluation carried out by Han, et al (2000) on a number of different algorithms for association mining which include Apriori, CHARM, FP-Tree algorithm revealed that:

1) FP-Tree method was usually better than the best implementation of the Apriori algorithm
2) CHARM was usually better than Apriori and in some cases CHARM was better than FP-tree method
3) Apriori was generally better than other algorithms if the support required was high since high support leads to a smaller number of frequent items which suits the Apriori algorithm

4) At low support the number of frequent items became large and none of the algorithms were able to handle that gracefully (Jiawei et al, 2011).

3.FP-Tree Growth Methodology

FP-algorithm works by generating frequent pattern tree (FP-tree) and then mining the tree. The algorithm works as follows:

1) Scan the transaction database once, as in the Apriori algorithm to find all the frequent items and their support
2) Sort the frequent items in descending order of their support
3) Start creating the FP-tree with a "null" root
4) Get the first transaction from the transaction database. Remove all non frequent items and list the remaining item according to the order in the sorted frequent item.
5) Use the transaction to construct the first branch of the tree with each node corresponding to a frequent item and showing the item's frequency.
6) Get the next transaction from the transaction database. Remove all non frequent items and list the remaining item according to the sorted order.
7) Insert the transaction in the tree using any common prefix that may appear.
8) Increase the item count
9) continue with step 6 until all the transactions in the database are processed (Jaiwei *et al*, 2000).

The motivation for the FP-growth algorithm is as follows:

1) Only the frequent items are needed to form the association rule. So, it finds frequent items and ignores the others
2) The frequent items can be stored in compact structure, thus, the original transaction database does not need to be used repeatedly.
3) If multiple transactions share a set of frequent items, it may be possible to merge shared sets with the number of occurrences registered at count.

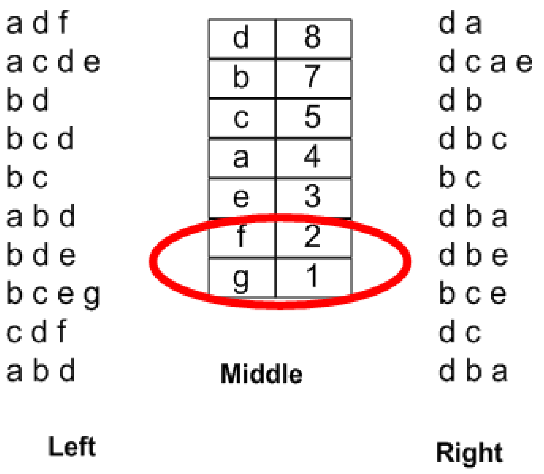
The advantages of the FP-tree algorithm are:

1) it avoids scanning the database more than twice to find the support count.
2) It completely eliminate the costly candidate generation
3) It is better than Apriori algorithm when the transaction database is huge and minimum support count is low
4) FP-Growth uses a more efficient structure to mine pattern when the database grows

4. FP-Tree Application

FP-growth preprocesses the transaction database by initially scanning the database; in scanning database the frequencies of the items (support of single element item sets) are determined. All infrequent items (that is, all items that appear in fewer transactions than a user-specified minimum number) are discarded from the transactions, since, obviously, they can never be part of a frequent item set.

In addition, the items in each transaction are sorted, so that they are in descending order according to their frequency in the database. Figure 1 and figure 2 shows an FP-tree construction process. Figure 1 shows a sample database on the left. The frequencies of the items in this database, sorted in descending order are shown in the middle. Using a user specified minimal support of 3 transactions; items f and g can be discarded. After the deletion and sorting the items in each transaction in the database in descending order according to their frequencies, a reduced database is obtained as shown on the right in figure 1.



Left
a d f
a c d e
b d
b c d
b c
a b d
b d e
b c e g
c d f
a b d

Middle	
d	8
b	7
c	5
a	4
e	3
f	2
g	1

Right
d a
d c a e
d b
d b c
b c
d b a
d b e
b c e
d c
d b a

Figure 1: Transaction database and Frequency table sorted in ascending order according to their frequency support

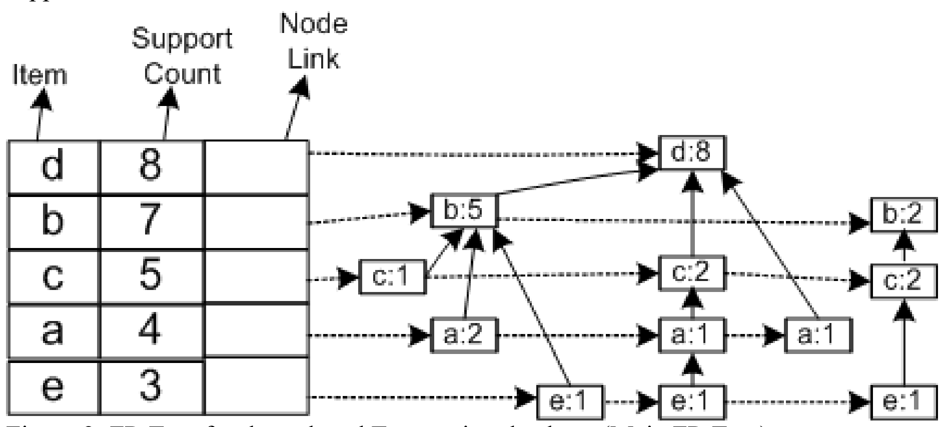


Figure 2: FP-Tree for the reduced Transaction database (Main FP-Tree)

The FP-tree in figure 2 was constructed from the (reduced) database shown in figure 1 on the right. After the deletion of the infrequent items from the transaction database, the database is turned into an FP-tree as shown in figure 2. Also in figure 2 above, each path represents a set of transactions that share the same prefix; each node corresponds to one item, and all nodes referring to the same item are linked together in a list, so that all transactions containing a specific item can easily be found and counted by traversing this list. The list can be accessed through a head element, which also states the total number of occurrences of the item in the database.

4.1 Mining Frequent Items

To mine the frequent pattern from FP-tree start from the lowest level of the FP-tree (as an initial suffix pattern), construct its conditional pattern base (a “sub-database,” which consists of the set of prefix paths in the FP-tree co-occurring with the suffix pattern), then construct its (conditional) FP-tree, and perform mining recursively on the tree. The pattern growth is achieved by the concatenation of the suffix pattern with the frequent patterns generated from a conditional FP-tree.

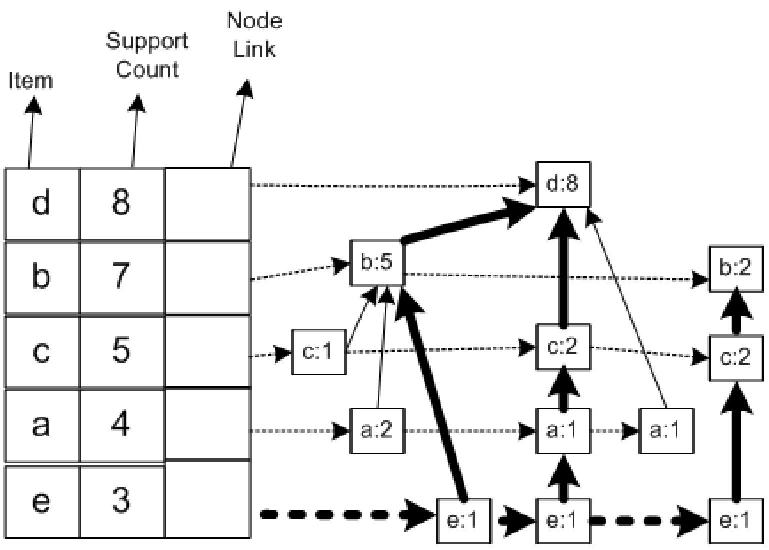


Figure 3: The identified paths to node e

Mining of the FP-tree is summarized in figure 3 and figure 4; the sub-trees were generated by considering item e, which is the last item in the bottom of the tree (in figure 2). *e* occurs in three FP-tree branches of Figure 2. (The occurrences of e can easily be found by following its chain of node-links.) as shown in figure 3. The paths formed by these branches are *dbe, bce, dcae.* Considering e as a suffix, its corresponding three prefix paths are *dbe, bce and dcae* which form its conditional pattern base. Using this conditional pattern base as a transaction database, we build an e-conditional FP-tree, which contains three single paths as shown in figure 5. However, in this FP-tree all items are infrequent (and thus all item sets containing item e and one other item are infrequent). Hence in this example, no recursive processing would take place. This is of course, due to the chosen example database and the support threshold.

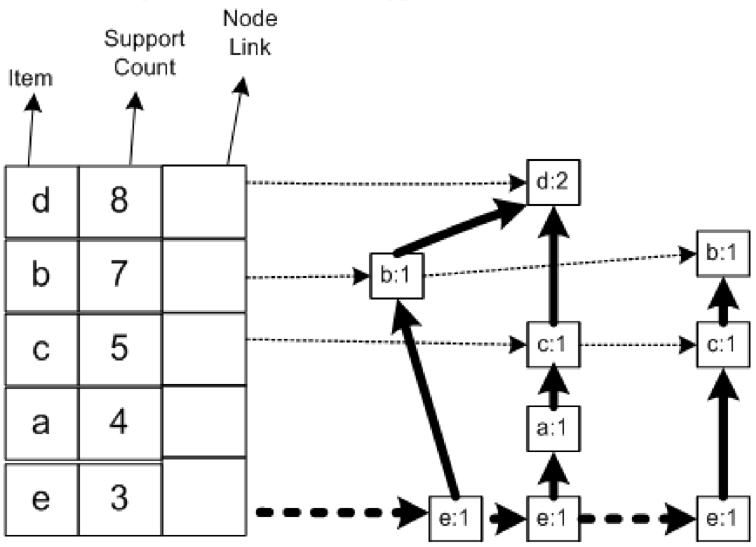


Figure 4: A conditional FP-Tree Extracted from the main FP-Tree in figure 2

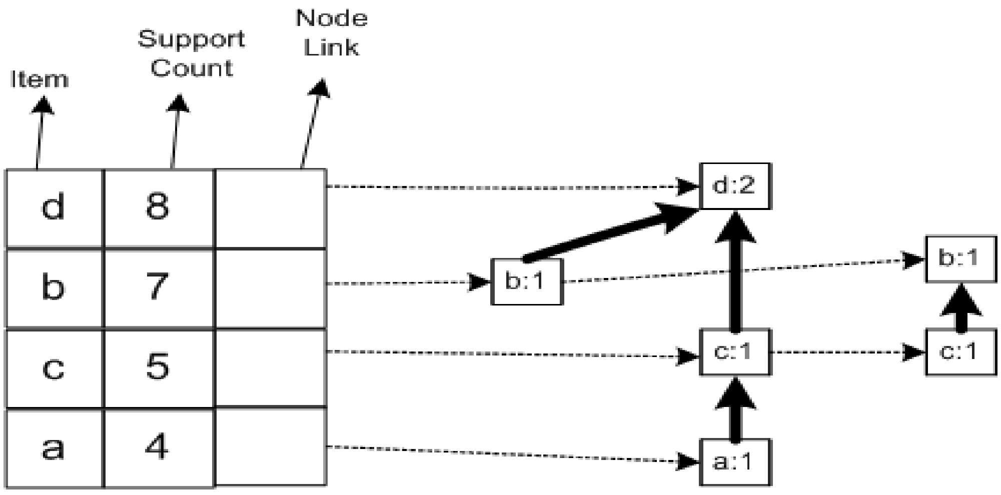
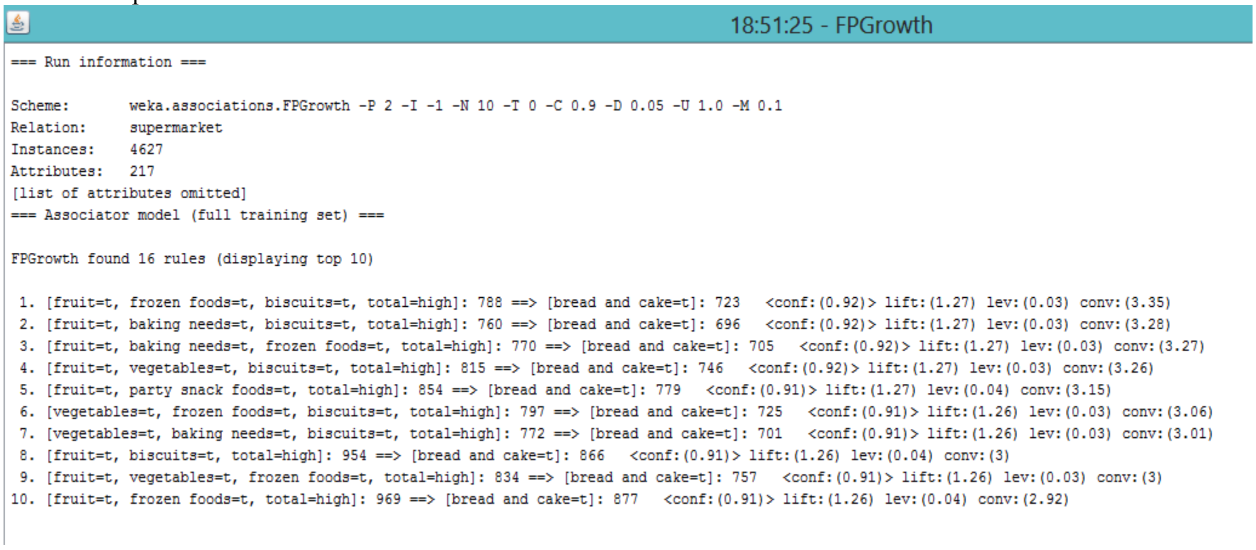


Figure 5: Generated FP-Tree after deletion of node e from the conditional FP-Tree in figure 4

The FP-growth method transforms the problem of finding long frequent patterns into searching for shorter ones in much smaller conditional databases recursively and then concatenating the suffix. It uses the least frequent items as a suffix, offering good selectivity. The method substantially reduces the search costs. When the database is large, it is sometimes unrealistic to construct a main memory based FP-tree. An interesting alternative is to first partition the database into a set of projected databases, and then construct an FP-tree and mine it in each projected database. This process can be recursively applied to any projected database if its FP-tree still cannot fit in main memory.

5. Implementation with Real Datasets

In order to implement the FP-Tree algorithm the standard benchmark datasets from the UCI Machine Learning Repository termed Supermarket was used in Weka Environment. The analysis was performed in comparism with apriori algorithm with the real supermarket dataset, it was observed that the FP-tree algorithm is faster and more effective than the apriori algorithm in mining association rule. All reports of the iterations and number of rules generated are as shown in Figure 7 and figure 8. The graph in figure 9 shows that the FP-Tree outperforms the apriori algorithm as the number of iteration performed in finding frequent pattern is less and less time was also used to complete the iterations.



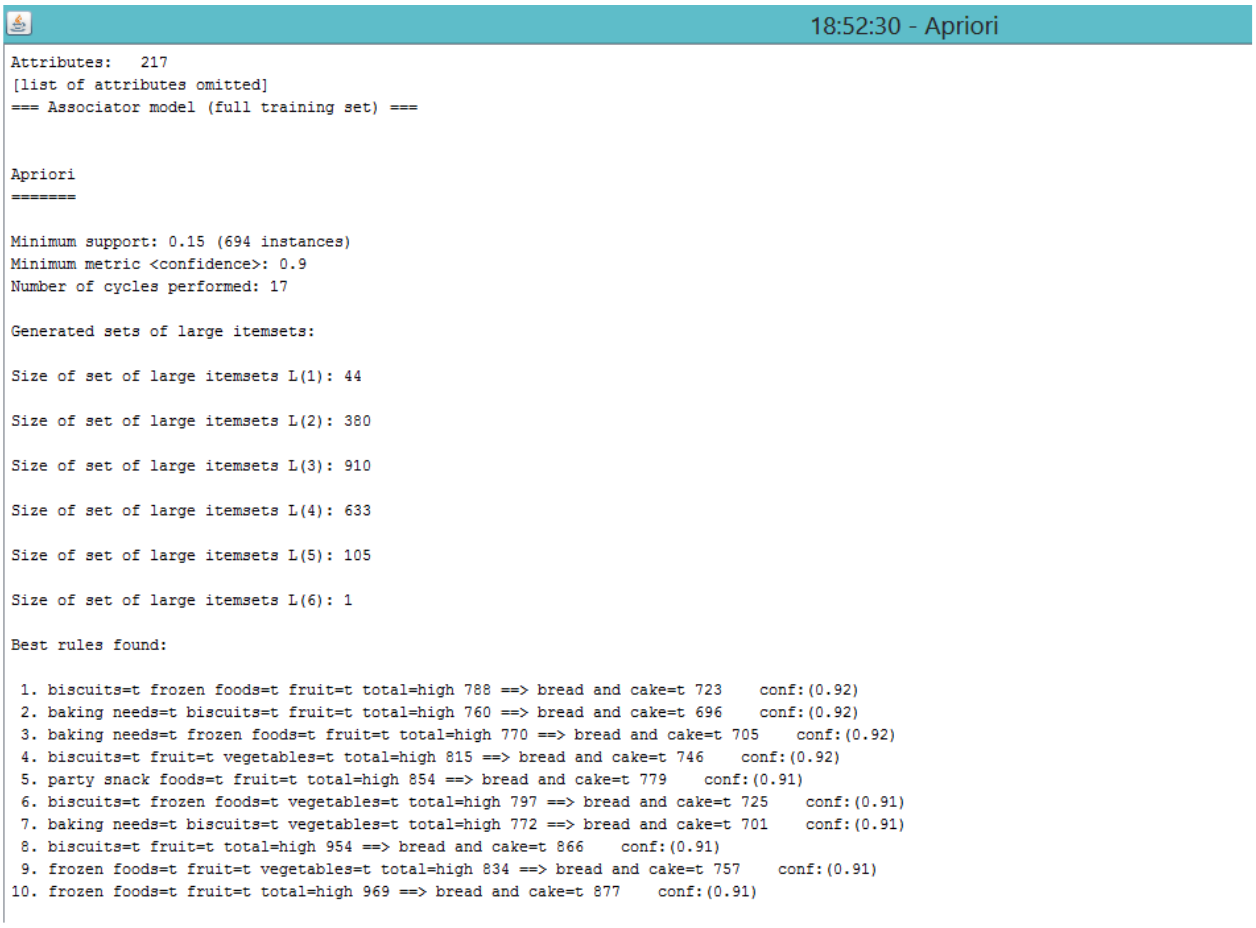
```
=== Run information ===

Scheme:       weka.associations.FPGrowth -P 2 -I -1 -N 10 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.1
Relation:     supermarket
Instances:    4627
Attributes:   217
[list of attributes omitted]
=== Associator model (full training set) ===

FPGrowth found 16 rules (displaying top 10)

 1. [fruit=t, frozen foods=t, biscuits=t, total=high]: 788 ==> [bread and cake=t]: 723   <conf:(0.92)> lift:(1.27) lev:(0.03) conv:(3.35)
 2. [fruit=t, baking needs=t, biscuits=t, total=high]: 760 ==> [bread and cake=t]: 696   <conf:(0.92)> lift:(1.27) lev:(0.03) conv:(3.28)
 3. [fruit=t, baking needs=t, frozen foods=t, total=high]: 770 ==> [bread and cake=t]: 705   <conf:(0.92)> lift:(1.27) lev:(0.03) conv:(3.27)
 4. [fruit=t, vegetables=t, biscuits=t, total=high]: 815 ==> [bread and cake=t]: 746   <conf:(0.92)> lift:(1.27) lev:(0.03) conv:(3.26)
 5. [fruit=t, party snack foods=t, total=high]: 854 ==> [bread and cake=t]: 779   <conf:(0.91)> lift:(1.27) lev:(0.04) conv:(3.15)
 6. [vegetables=t, frozen foods=t, biscuits=t, total=high]: 797 ==> [bread and cake=t]: 725   <conf:(0.91)> lift:(1.26) lev:(0.03) conv:(3.06)
 7. [vegetables=t, baking needs=t, biscuits=t, total=high]: 772 ==> [bread and cake=t]: 701   <conf:(0.91)> lift:(1.26) lev:(0.03) conv:(3.01)
 8. [fruit=t, biscuits=t, total=high]: 954 ==> [bread and cake=t]: 866   <conf:(0.91)> lift:(1.26) lev:(0.04) conv:(3)
 9. [fruit=t, vegetables=t, frozen foods=t, total=high]: 834 ==> [bread and cake=t]: 757   <conf:(0.91)> lift:(1.26) lev:(0.03) conv:(3)
10. [fruit=t, frozen foods=t, total=high]: 969 ==> [bread and cake=t]: 877   <conf:(0.91)> lift:(1.26) lev:(0.04) conv:(2.92)
```

Figure 7: Report of the FP-Tree algorithm



18:52:30 - Apriori

```
Attributes:   217
[list of attributes omitted]
=== Associator model (full training set) ===


Apriori
=======

Minimum support: 0.15 (694 instances)
Minimum metric <confidence>: 0.9
Number of cycles performed: 17

Generated sets of large itemsets:

Size of set of large itemsets L(1): 44

Size of set of large itemsets L(2): 380

Size of set of large itemsets L(3): 910

Size of set of large itemsets L(4): 633

Size of set of large itemsets L(5): 105

Size of set of large itemsets L(6): 1

Best rules found:

 1. biscuits=t frozen foods=t fruit=t total=high 788 ==> bread and cake=t 723    conf:(0.92)
 2. baking needs=t biscuits=t fruit=t total=high 760 ==> bread and cake=t 696    conf:(0.92)
 3. baking needs=t frozen foods=t fruit=t total=high 770 ==> bread and cake=t 705    conf:(0.92)
 4. biscuits=t fruit=t vegetables=t total=high 815 ==> bread and cake=t 746    conf:(0.92)
 5. party snack foods=t fruit=t total=high 854 ==> bread and cake=t 779    conf:(0.91)
 6. biscuits=t frozen foods=t vegetables=t total=high 797 ==> bread and cake=t 725    conf:(0.91)
 7. baking needs=t biscuits=t vegetables=t total=high 772 ==> bread and cake=t 701    conf:(0.91)
 8. biscuits=t fruit=t total=high 954 ==> bread and cake=t 866    conf:(0.91)
 9. frozen foods=t fruit=t vegetables=t total=high 834 ==> bread and cake=t 757    conf:(0.91)
10. frozen foods=t fruit=t total=high 969 ==> bread and cake=t 877    conf:(0.91)
```

Figure 8: Report of the Apriori Algorithm

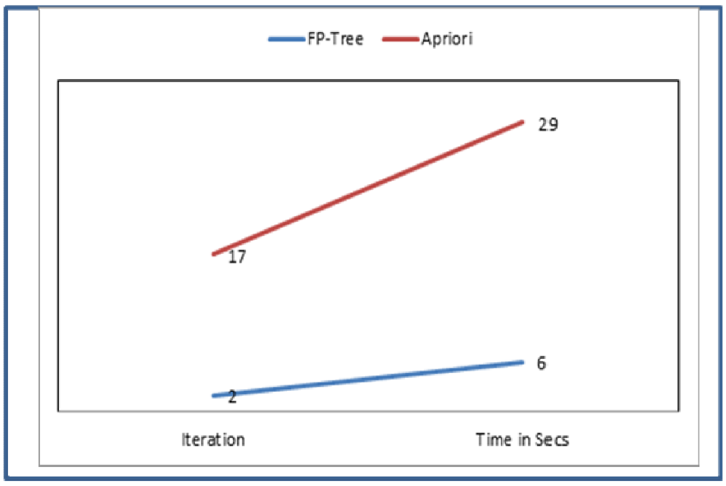


Figure 9: No of Iteration versus time taken to complete the iterations

6. Conclusion

FP-tree algorithm directly mines frequent item sets without generating candidates. By gathering sufficient statistics into a suitable data structure (called an FP tree), all the frequent patterns are generated without going back to the database. Only two passes through the database that are required to generate the FP Tree, and from the FP Tree, all frequent patterns are generated. The FP-tree encourages a divide and conquer approach to data mining. The FP-Tree is a compressed representation of the database.

The FP-growth method transforms the problem of finding long frequent patterns into searching for shorter ones in much smaller conditional databases recursively and then concatenating the suffix. It uses the least frequent items as a suffix, offering good selectivity. The method substantially reduces the search costs and thus is regarded as one of the most efficient methods for frequent pattern mining. The algorithm of the FP-Tree and its application is hereby simplified for better understanding by anyone interested in data mining that will want to understand the concept of this method.

References

Agrawal R., Imielinski, T. Swami A., (1993): "Mining Associations rules between Sets of Items in Large Databases", Proceedings of the ACM SIGMOD Int'l Conference on Management of Data, Washington D.C., USA

Agrawal R. and Srikant R. (1994) "Fast algorithms for mining association rules", In J. B. Bocca, M. Jarke, and C. Zaniolo, editors, *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, pages 487–499. Morgan Kaufmann.

Agarwal R. C. , Aggarwal C. C., and Prasad V. V. V. (2001a) "Depth first generation of long patterns". In *Proc. 6th ACM SIGKDD Int. Conf. on Knowledge discovery and data mining*, pages 108–118. ACM Press.

Agarwal R. C., Aggarwal C. C., and Prasad V. V. V. (2001b) "A tree projection algorithm for generation of frequent item sets", *Journal of Parallel and Distributed Computing*, 61:350–371,

Bayardo R. J. (1998), "Efficiently mining long patterns from databases",. In *1998 ACM SIGMOD Intl. Conference on Management of Data*, pages 85–93. ACM Press.

Burdick D., Calimlim M., and Gehrke J. (2001), "MAFIA: A maximal frequent itemset algorithm for transactional databases", In *2001 Intl. Conference on Data Engineering,ICDE*, pages 443–452.

Han J. and Pei J. (2001) "Mining frequent patterns by pattern-growth: Methodology and implications". In *ACM SIGKDD Explorations*. ACM Press.

Han J., Pei J., and Yin Y.(2000) "Mining frequent patterns without candidate generation. In W. Chen, J. Naughton, and P. A. Bernstein, editors, *2000 ACM SIGMOD Intl. Conference On Management of Data*, pages 1–12.ACM Press.

Intan and Rolly (2005). "A Proposal of an Algorithm for Generating Fuzzy Association Rule Mining in Market Based Analysis", Proceedings of the 3rd International Conference on Computational Intelligence,Robotics and Autonomous System. Issues and an Application, Geographical Analysis 27:4, (2005) pp 286-305.

Jiawei H., Jian P., and Yiwen Y. (2000) "Mining Frequent Patterns without Candidate Generation", In Proc. Conf. on the Management of Data (SIGMOD'00, Dallas, TX), ACM Press, New York, NY, USA

Jiawei H., Micheline K., and Jian P. (2011)"Data mining: Concept and Techniques" 3rd edition, Elsevier,

Pei J., Han J., Lu H., Nishio S., and Shiwei Tang D. Y. (2001) "H-mine:hyper-structure mining of frequent patterns in large databases", In *2001 IEEE Conference on Data Mining*. IEEE.

ZhaoHui T. and Jamie M. (2005), "Data Mining with SQL Server 2005",Wiley Publishing Inc, Indianapolis, Indiana, 2005.