An Analytical Study on Frequent Itemset Mining Algorithms

K. Pazhani Kumar and S. Arumugaperumal

Dept. of Computer Science
S.T. Hindu College, Nagercoil, TamilNadu

Abstract. Data mining is the process of collecting, extracting and analyzing large data set from different perspectives. Fundamental and important task of data mining is the mining of frequent itemsets. Frequent itemsets play an important role in association rule mining. Many researchers invented ideas to generate the frequent itemsets. The execution time required for generating frequent itemsets play an important role. This study yields a detailed analysis of the FP-Growth, Eclat and SaM algorithms to illustrate the performance with standard datasets Hepatitis and Adault. The comparative study of FP-Growth, Eclat and SaM algorithms includes aspects like different support values and different datasets.

Keywords: Frequent Itemset, Mining, Hepatitis, Adult.

1 Introduction

In recent years the size of database has increased rapidly. This has led to a growing interest in the development of tools capable in the automatic extraction of knowledge from data. Data mining refers to discover knowledge in huge amounts of data. It is a scientific discipline that is concerned with analyzing observational data sets with the objective of finding unsuspected relationships and produces a summary of the data in novel ways that the owner can understand and use. Data mining as a field of study involves the merging of ideas from many domains rather than a pure discipline.

The problem of mining frequent itemsets arose first as a sub-problem of mining association rules. Frequent itemsets play an essential role in many data mining tasks that try to find interesting patterns from databases such as association rules, correlations, sequences, classifiers, clusters and many more of which the mining of association rules is one of the most popular problems. The original motivation for searching association rules came from the need to analyze so called supermarket transaction data, that is, to examine customer behavior in terms of the purchased products. Association rules describe how often items are purchased together. For example, an association rules "beer, chips (80%)" states that four out of five customers that bought beer also bought chips. Such rules can be useful for decisions concerning product pricing, promotions, store layout and many others.

2 Problem Study

2.1 Motivation

Studies of Frequent Itemset (or pattern) Mining is acknowledged in the data mining field because of its broad applications in mining association rules, correlations, and graph pattern constraint based on frequent patterns, sequential patterns, and many other data mining tasks. Efficient algorithms for mining frequent itemsets are crucial for mining association rules as well as for many other data mining tasks. The major challenge found in frequent pattern mining is a large number of result patterns. As the minimum threshold becomes lower, an exponentially large number of itemsets are generated. Therefore, pruning unimportant patterns can be done effectively in mining process and that becomes one of the main topics in frequent pattern mining. Consequently, the main aim is to optimize the process of finding patterns which should be efficient, scalable and can detect the important patterns which can be used in various ways [4].

3 Frequent Itemset Mining Algorithms

3.1 FP-Growth Algorithm

One of the currently fastest and most popular algorithms for frequent item set mining is the FP-growth algorithm [3]. It is based on a prefix tree representation of the given database of transactions (called an FP-tree), which can save considerable amounts of memory for storing the transactions. The basic idea of the FP-growth algorithm can be described as a recursive elimination scheme: in a preprocessing step delete all items from the transactions that are not frequent individually, i.e., do not appear in a user-specified minimum number of transactions. Then select all transactions that contain the least frequent item (least frequent among those that are frequent) and delete this item from them. Recourses to process the obtained reduced (also known as projected) database, remembering that the item sets found in the recursion share the deleted item as a prefix. On return, remove the processed item also from the database of all transactions and start over, i.e., process the second frequent item etc. In these processing steps the prefix tree, which is enhanced by links between the branches, is exploited to quickly find the transactions containing a given item and also to remove this item from the transactions after it has been processed.

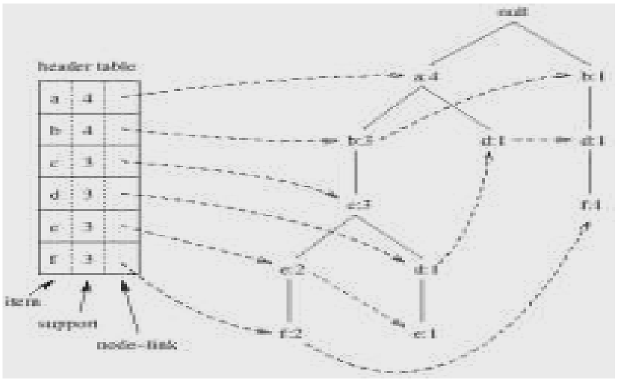


Fig. 1. An Example of FP-Tree

Table 1. An Example of Transaction Database

TID	X
1	{a,b,c,d,e,f}
2	{a,b,c,d,e}
3	{a,d}
4	{b.d.f}
5	{a,b,c,e,f}

The core operation of the FP-growth algorithm is to compute an FP-tree. A frequent pattern tree is a tree structure defined as figure 1.It consists of one root labeled as "root", a set of item prefix sub-trees as the children of the root, and a frequent item header table 1.Each node in the item prefix sub-tree consists of three fields: item-name, count and node-link, where item-name registers which item this node represents, count registers the number of transactions represented by the portion of the path reaching this node, and node-link links to the next node in the FP-tree carrying the same item-name, or null if there is none. Each entry in the frequent-item header table consists of two fields, 1. item name and 2. head of node-link, which points to the first node in the FP-tree carrying the item-name.The algorithm FP-tree[4] is as below:

Algorithm 1 (FP-tree construction):

Input: A transactional database *DB* and a minimum support threshold.

Output: Its frequent pattern tree, FP-tree

Method: The FP-tree is constructed in the following steps:

1. Scan the transaction database *DB* once. Collect the set of frequent items *F* and their supports. Sort *F* in support descending order as *L*, the *list* of frequent items.

2. Create the root of an FP-tree, *T*, and label it as "root".

After above process mining of the FP-tree will be done by Creating Conditional (sub) pattern bases:

1 Start from node constructs its conditional pattern base.

2 Then, Construct its conditional FP-tree & perform mining on such a tree.

3 Join the suffix patterns with a frequent pattern generated from a conditional GP-tree for achieving FP-growth.

4 The union of all frequent patterns found by above step gives the required frequent itemset.

In this way frequent patterns are mined from the database using FP-tree.

3.2 Eclat Algorithm

It is a set intersection, depth first search algorithm [5], unlike the Apriori. It uses vertical layout database and each item use intersection based approach for finding the support. In this way, the support of an itemset P can be easily computed by simply intersecting any two subsets Q, R ⊆ P, such that P = Q U R. In this type of algorithm, for each frequent itemset i new database is created Di. This can be done by finding j which is frequent corresponding to i together as a set then j is also added to the created database i.e. each frequent item is added to the output set. It uses the join step like the Apriori only for generating the candidate sets but as the items are arranged in ascending order of their support thus less amount of intersection is needed between the sets. It generates the larger amount of candidates then Apriori because it uses only two sets at a time for intersection [5]. There is reordering step takes place at each recursion point for reducing the candidate itemsets. In this way by using this algorithm there is no need to find the support of itemsets whose count is greater than 1because Tid-set for each item carry the complete information for the corresponding support. When the database is very large and the itemsets in the database corresponding are also very large then it is feasible to handle the Tid list thus, it produce good results but for small databases its performance is not up to mark.

The Eclat algorithm is as given below [4].

```
Input: D, σ, i ⊆ I
Output: F [I](D, σ)
1: F [I] :={}
2: for all I Є I occurring in D do
3: F [I]:= F [I] ∪ {I ∪ {i}}
4: // Create Di
5: Di: = {}
6: for all j Є I occurring in D such that j>I do
7: C: = cover ({i}) ∩ cover ({j})
8: if |C| >= σ then
9: Di: = Di ∪ {(j, C)}
10: end if
11: end for
12: //Depth-first recursion
13: Compute F [I ∪ {i}](Di, σ)
14: F [I]:= F [I] ∪ F [I ∪ {i}]
15: end for
```

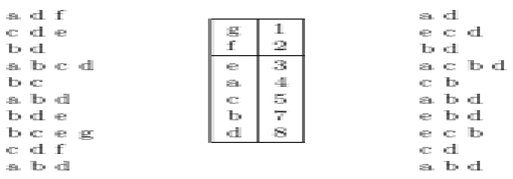
In this algorithm each frequent item is added in the output set. After that, for every such frequent item i, the projected database Di is created. This is done by first finding every item j that frequently occurs together with i. The support of this set {i, j} is computed by intersecting the covers of both items. If {i, j} is frequent, then j is inserted into Di together with its cover. The reordering is performed at every recursion step of the algorithm between line 10 and line 11. Then the algorithm is called recursively to find all frequent itemsets in the new database Di.

3.3 SaM Algorithm

The SaM (Split and Merge) algorithm established by [6] is a simplification of the already fairly simple RElim (Recursive Elimination) algorithm. While RElim represents a (conditional) database by storing one transaction list for each item (partially vertical representation), the split and merge algorithm employs only a single transaction list (purely horizontal representation), stored as an array. This array is processed with a simple split and merge scheme, which computes a conditional database, processes this conditional database recursively, and finally eliminates the split item from the original (conditional) database.

SaM preprocesses a given transaction following the steps below:

1. The transaction database is taken in its original form. 2. The frequencies of individual items are determined from this input in order to be able to discard infrequent items immediately. 3. The (frequent) items in each transaction are sorted according to their frequency in the transaction database, since it is well known that processing the items in the order of increasing frequency usually leads to the shortest execution times. 4. The transactions are sorted lexicographically into descending order, with item comparisons again being decided by the item frequencies; here the item with the higher frequency precedes the item with the lower frequency. 5. The data structure on which SaM operates is built by combining equal transactions and setting up an array, in which each element consists of two fields:An occurrence counter and a pointer to the sorted transaction (array of contained items). This data structure is then processed recursively to find the frequent item sets. The basic operations of the recursive processing are based on depth-first/divide-and-conquer scheme. In the split step the given array is split with respect to the leading item of the first transaction. All array elements referring to transactions starting with this item are transferred to a new array. The new array created in the split step and the rest of the original arrays are combined with a procedure that is almost identical to one phase of the well-known merge sort algorithm. The main reason for the merge operation in SaM is to keep the list sorted, so that: 1.All transactions with the same leading item are grouped together and 2.Equal transactions (or transaction suffixes) can be combined, thus reducing the number of objects to process.



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

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

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

Fig. 2. Transaction database (left), item frequencies (middle), and reduced transaction database with items in transactions sorted accordingly with respect to their frequency (right)

Each transaction is represented as a simple array of item identifiers (which are integer numbers). The transaction list is prepared which are stored in a simple array, each element of which contains a support counter and a pointer to the head of the list. The list elements themselves consist only of a successor pointer and a pointer to the transaction. The transactions are inserted one by one into this structure by simply using their leading item as an index.

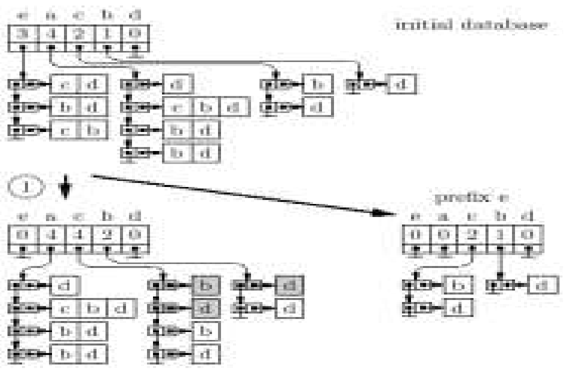


Fig. 3. Procedure of the recursive elimination with the modification of the transaction lists (left) as well as the transaction lists for the recursion (right)

4 Analytical Study

A detailed study has been conducted to assess the performance of the above-said algorithms. The metrics used in the comparison study is the total execution time taken and the number of itemsets generated for different data sets. For this comparison also same data sets were selected as for the above experiment with 30% to 60% of minimum support threshold.

Table 2. Adault data set execution time

SUPPORT	Time in Seconds		
	FPGrowth	Eclat	SaM
30	11.2	10.35	9.25
40	8.30	7.52	6.12
50	5.55	5.20	4.10
60	3.90	3.60	2.80

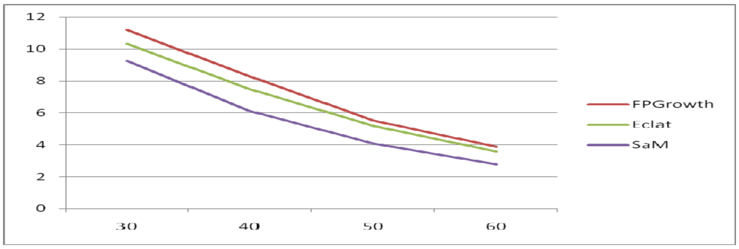


Figure 4 shows that the execution time for the FP-growth,Eclat and SaM algorithms decreases with the increase in support threshold form 30% to 60% for adult dataset FPgrowth takes more time as that compared to Eclat and SaM.

Table 3. Hepatitis data set execution time

SUPPORT	Time in Seconds		
	FPGrowth	Eclat	SaM
30	1.34	1.02	0.9
40	0.71	0.58	0.51
50	0.17	0.12	0.09
60	0.07	0.04	0.03

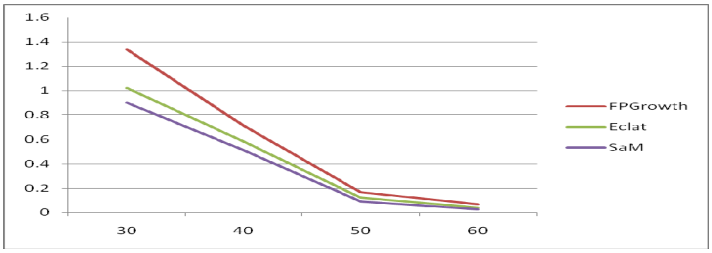


Figure 5 shows that the execution time for the FP-Growth,SaM and Eclat algorithms decreases with the increase in support threshold form 30% to 60% for adult dataset FP-Growth takes more time as that compared to Eclat and SaM.

5 Conclusion

This paper presents the comparative study of three algorithms FP-Growth,Eclat and SaM.This study shows that the SaM algorithm has high performance in various kinds of data, out forms the FP-Growth and Eclat algorithms. The performances of the algorithms strongly depend on the support levels and the feature of the data sets (the nature and the size of the data sets) is observed.

References

[1] Tan, P.N., Steinbach, M., Kumar, V.: Introduction to data mining. Addison Wesley Publishers (2006)

[2] Che, M.S., Han, Yu, P.S.: Data Mining: An Overview from a Database Perspective. Proc. of the IEEE Transactions on Knowledge and Data Engineering 8(6), 866–883 (1996)

[3] Han, J., Pei, H., Yin, Y.: Mining Frequent Patterns without Candidate Generation. In: Proc. Conf. on the Management of Data (SIGMOD 2000), Dallas, TX, ACM Press, New York (2000)

[4] Pramod, S., Vya, O.P.: Survey on Frequent Itemset Mininbg Algorithms. International Journal of Computer Applications (0975 - 8887)

[5] Borgelt, C.: Efficient Implementations of Apriori and Eclat. In: Proc. 1st IEEE ICDM Workshop on Frequent Item Set Mining Implementations. CEUR Workshop Proceedings 90, Aachen, Germany (2003)

[6] Borgelt, C.: SaM: Simple Algorithms for Frequent Item Set Mining. IFSA/EUSFLAT 2009 Conference (2009)