International Journal of Scientific Research in Computer Science, Engineering and Information Technology
ISSN : 2456-3307 (www.ijsrcseit.com)
doi : https://doi.org/10.32628/CSEIT2063230

Frequent Patterns Mining

Youssef FAKIR*, Rachid ELAYACHI
Information processing and decision support laboratory, Faculty of Sciences and Techniques
Universiy Sultane Moulay Slimane, Morocco

Article Info
Volume 6, Issue 4
Page Number: 21-29
Publication Issue :
July-August-2020

Article History
Accepted : 01 July 2020
Published : 07 July 2020

ABSTRACT

Frequent pattern mining has been an important subject matter in data mining from many years. A remarkable progress in this field has been made and lots of efficient algorithms have been designed to search frequent patterns in a transactional database. One of the most important technique of datamining is the extraction rule in large database. The time required for generating frequent itemsets plays an important role. This paper provides a comparative study of algorithms Eclat, Apriori and FP-Growth. The performance of these algorithms is compared according to the efficiency of the time and memory usage. This study also focuses on each of the algorithm's strengths and weaknesses for finding patterns among large item sets in database systems.
Keywords : Apriori, Fp-growth, Eclat, itemsets, association rule.

I. INTRODUCTION

Association rule mining has its importance in fields of artificial intelligence, information science, database and many others. Data volumes are dramatically increasing by day-to-day activities. Therefore, mining the association rules from massive data is in the interest for many industries as theses rules help in decision-making processes, market basket analysis and cross marketing etc.

The association rules, being an unsupervised learning method, make it possible to discover, from a set of transactions, a set of rules that expresses a possibility of association between different items. A transaction is a succession of items expressed in a given order; similarly, the transaction set contains transactions of different lengths.

An association rule is an implication expression of the form X =>Y, where X and Y are disjoint itemsets, i.e., $X \cap Y = \emptyset$. X and Y are named respectively the ancedant and the consequent. For example if a cosumer buy coffee hence he buy probably sugar.

The strength of an association rule can be measured in terms of its support and confidence. The formal definitions of these metrics are :

$$\text{Support}(X \Rightarrow Y) = \text{support}(XUY)$$
$$\text{Confidence}(X \Rightarrow Y) = \text{sup}(XUY)/\text{Sup}(X)$$

Support: It is defined as rate of occurrence of an itemset in a transaction database.

Support (a →b) =

(Number of transactions containing both a and b)/ number of total transactions

Confidence: For all transactions, it defines the ratio of data items which contains Y in the items that contains X.

Confidence (a $\Rightarrow$ b) =
(number of transactions containing a and b)/number of transactions (containing a).

If I ={a1, a2, …an} is a set of items and a transaction database DB={T1, T2,…, Tn} of n transaction, the support for I is the number of baskets for which I is a subset. We say I is frequent if its support is equal or greater than S (minimum support). For example: Consider below table (Table 1) containing some transactions. This transaction can be represented in boolean representation as shown in Table 2.

Table 1. Example of transactions in a database

TID	List of Items
T_1	I_1, I_2, I_5
T_2	I_2, I_4
T_3	I_2, I_3
T_4	I_1, I_2, I_4
T_5	I_1, I_3
T_6	I_2, I_3
T_7	I_1, I_3
T_8	I_1, I_2, I_3, I_5
T_9	I_1, I_2, I_3
T_{10}	I_1, I_2, I_5, I_6

In this paper, a comparison between the algorithms Apriori [1,2], Fp-growth [3], and Eclat [4,5] is done. This paper is organized as follows: Section 2 deals with the Eclat algorithm while section 3 describes the Apriori algorithm. Section 4 present the FP-Growth algorithm. In section 5 we present the results of experiment carried out on the database. Section 6 conclude the paper.

Table 2 : Boolean representation of DB of Table 1

TID	I_1	I_2	I_3	I_4	I_5	I_6
T_1	1	1	0	0	0	0
T_2	0	1	0	1	0	0
T_3	0	1	1	0	0	0
T_4	1	1	0	1	0	0
T_5	1	0	1	0	0	0
T_6	0	1	1	0	0	0
T_7	1	0	1	0	0	0
T_8	1	1	1	0	1	0
T_9	1	1	1	0	1	0
T_{10}	1	1	0	0	1	1
Count	7	8	6	2	3	1

II. EQUIVALENCE CLASS TRANSFORMATION (ECLAT)

The Eclat algorithm is used to perform itemset mining. The basic idea for the Eclat algorithm is use tidset intersections to compute the support of a candidate itemset avoiding the generation of subsets that does not exist in the prefix tree. ECLAT algorithm [4,8] uses vertical database format whereas in Apriori horizontal data format (TranscationId, Items) has been used, in which transaction IDs are explicitly listed. While in Vertical Data Format (VDF) (Items, TransactionID) Items with their list of transactions are maintained. ECLAT algorithm with set intersection property uses depth-first search algorithm [6,7,8]. All frequent itemsets can be computed with intersection of TID-list. In first scan of database a TID list is maintained for each single item. k+1 itemset can be generated from k itemset using apriori property and depth first search computation. (k+1)-itemset is generated by taking intersection of TID-set of frequent k-itemset. This process continues, until no candidate itemset can be found (as shown in Table 3). The algorithm Eclat is given in algorithm 1.

Table 3. Itemsets representation

Itemset	TID
I_1	1,4,5,7,8,9,10
I_2	1,2,3,4,5,8,9,10
I_3	3,5,6,7,8,9
I_4	2,4
I_5	1,8,10
I_6	10

(a) 1-Itemset in VDF

Itemset	TID
I_1, I_2	1,4,8,9,10
I_1, I_3	5,7,8,9
I_1, I_4	4 x
I_1, I_5	1,8,10
I_2, I_3	3,6,8,9
I_2, I_4	2,4
I_2, I_5	1,8,10
I_3, I_5	8 x
I_5, I_6	10 x

(b) 2-Itemset in VDF

Itemset	TID
I_1, I_2, I_3	8,9
I_1, I_2, I_4	4 x
I_1, I_2, I_5	1,8,10
I_1, I_3, I_5	8 x

(c) 3-Itemset in VDF

Algorithm 1 : Eclat

Input: Fk = {I1, I2, ..., In} // cluster of frequent k-itemsets.

Output: Frequent l-itemsets.

```
Bottom-Up (Fk) {
   for all Ii € Fk
      Fk +1 = ∅;
  for all Ij € Fk , i < j
   N = Ii ∩ Ij;
   if N .sup1>= min_sup then
    Fk + 1 = Fk +1 U N;
   end
  end
 end
 if Fk +1 ≠ ∅ then
Bottom-Up (Fk +1);
 end
}
```

In this algorithm, numbers of items are stored in Fk as a input. Output is frequent itemsets which are frequently occurred. Searching of elements starts from bottom to top. First take Fk +1 as empty datasbase. In next step find support of individual items. Compare support of all items with minimum support. And that put all those items in Fk +1. Fk +1 contains all frequent items. Again check that Fk +1 is empty or not. If it is not empty then bottom up approach will apply on Fk + 1.

III. APRIORI ALGORITHM

Agrawal and Srikant (1994) firstly proposed Apriori algorithm [5]. This algorithm uses two steps “join” and “prune” to reduce the search space. It is an iterative approach to discover the most frequent itemsets. Join Step generates (K+1) itemset from K-itemsets by joining each item with itself. Prune step scans the count of each item in the database. If the candidate item does not meet minimum support, then it is regarded as infrequent and thus it is removed. This step is performed to reduce the size of the candidate itemsets. These processes are iteratively repeated until candidate Itemsets or large Itemsets becomes empty. Original database is scanned first time for the candidate set, consists of one sensor item and there support has counted, then these 1-Itemset candidates are pruned by simply removing those items that has an item count less than user specified threshold (minimum support). In second pass database is scanned again to generate 2-Itemset candidates consist of two items, then again pruned to produced large 2-Itemset using apriori property. According to apriori property every sub 1-Itemset of 2 frequent Itemsets must be frequent. This process ends as in fourth scan of database 4-Itemset candidate will be pruned and large itemset will be empty.

The principle of Apriori is that :
If an itemset is frequent, then all of its subsets must also be frequent.

Apriori principle holds due to the following property of the support measure :

- Support of an itemset never exceeds the support of its subsets.
- This is known as the anti-monotone property of support

The pseudo code for apriori algorithm is as follows :

Apriori_Algo(L,C,k)

```
Pass 1
1. Generate the candidate itemsets in C1
2. Save the frequent itemsets in L1
Pass k
1. Generate the candidate itemsets in Ck from the
   frequent itemsets in Lk-1
i. Join Lk-1 p with Lk-1q, as follows:
insert into Ck
select p.item1 , p.item2 , . . . , p.itemk1 , q.itemk-1 from
   Lk-1 p, Lk-1q where p.item1 = q.item1 , . . .
   p.itemk2 = q.itemk-2 , p.itemk-1 < q.itemk-1
ii. Generate all (k-1)-subsets from the candidate
   itemsets in Ck
iii. Prune all candidate itemsets from Ck where some
   (k-1)-subset of the candidate itemset is not in the
   frequent itemset Lk-1
2. Scan the transaction database to determine the
   support for each candidate itemset in Ck
3. Save the frequent itemsets in Lk
```

The basic steps to mine the frequent elements are given in the following tables (Table 4 to Table 6) by fixing the minimum support to 30%.

- **Step 1 : Items –count**

Table .4 : Items and large 1-items

Items	Count
I_1	7
I_2	8
I_3	6
I_4	2
I_5	3
I_6	1

(b) Candidate 1

Large 1-item	count
I_1	7
I_2	8
I_3	6
I_5	3

(c) Large 1-items

- **Step 2. Prune Step:**
- Table-4(b) shows that I_4 and I_6 Item does not meet min_sup=3, thus it is deleted, only I_1, I_2, I_3, I_5 meet min_sup count.

- **Step 3 Join Step:**
- Form 2-itemset. from Table-4 (c) find out the occurrences of 2-itemset.

- **Step 4. Prune Step:**
- Table-5 shows that item set {I_3, I_5} does not meet min_sup, thus it is deleted.

Step 5:
Join and Prune Step: Form 3-itemset. From the Table-1 find out occurrences of 3-itemset. We can see for itemset {I_1, I_2, I_3} subsets, {I_1, I_2}, {I_1, I_3}, {I_1, I_5},{I_2, I_3}, {I_2, I_5} are occurring in Table 5 thus {I_1, I_2, I_5} is frequent.

Table-5: Large 2-items

Item	Count
I_1,I_2	5
I_1,I_3	4
I_1,I_5	3
I_2,I_3	4
I_2,I_5	3
I_3,I_5	2

Table-6 : Large 3-items

Item	count
I_1,I_2,I_3	2
I_1,I_2,I_5	3

The last step is the generalisation of association rules by chosen a confidence threshold.

IV. FP-GROWTH ALGORITHM

In the field of data mining, the most popular algorithm used for pattern discovery is FP Growth algorithm [3,5,9]. To deal with the two main drawbacks of Apriori algorithm in [5] a novel, compressed data structure named as FP- tree is constructed, which is prefix-tree structure storing quantifiable information about frequent patterns. Based on FP tree a frequent pattern growth algorithm was developed. FP-growth has to scan the *TDB* twice to construct an FP-tree. The first scan of *TDB* retrieves a set of frequent items from the *TDB* . Then, the retrieved frequent items are ordered by descending order of their supports. The ordered list is called an F-list. In the second scan, a tree *T* whose root node *R* labeled with "null" is created. Then, the following steps are applied to every transaction in the *TDB* . Here, let a transaction represent [p | P] where p is the first item of the transaction and P is the remaining items.

- In each transaction, infrequent items are discarded.
- Then, only the frequent items are sorted by the same order of F-list.
- Call insert_tree (p | P, R) to construct an FP-tree.

The function insert_tree (p | P, R) appends a transaction [p |P] to the root node R of the tree T.

An example of an FP-tree is shown in Figure 1 and Figure 2. This FP-tree is constructed from the *TDB* shown in Table 7 with *min_sup* = 3. In Figure 1, every node is represented by (*item name* : *count*). Links to next same item name node are represented by dotted arrows. FP-Growth allows frequent itemset discovery without candidate itemset generation. Two steps approach:

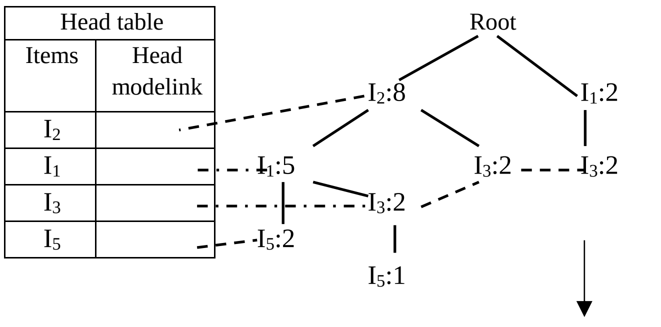
Step 1: Build a compact data structure called the FP-tree built using 2 passes over the data-set.

Step 2: Extracts frequent item-sets directly from the FP-tree.

FP-tree: is a compact data structure that stores important, crucial and quantitative information about frequent patterns. The main components of FP tree are: 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.

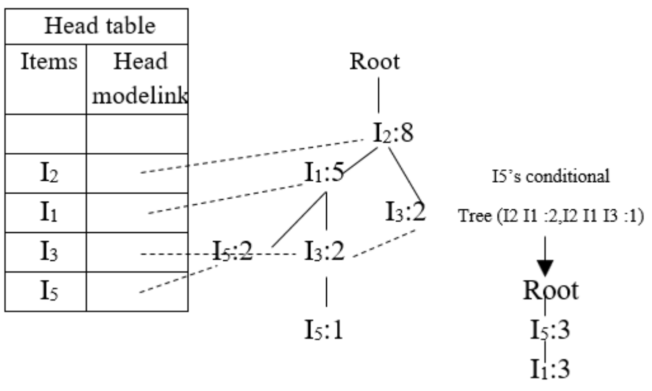
Each entry in the frequent-item header table consists of two fields, (1) item-name and (2) head of node-link, which points to the first node in the FP-tree carrying the item-name.

The tree construction algorithm is listed in Algorithm 2, the insert_tree(.) procedure is defined in Algorithm 3 and FP-Growth is given in algorithm 4.



Item	**Conditional Pattern base**
I_5	I_2 I_1:2, I_2 I_1 I_3 :1
I_3	I_2 I_1:2, I_2:2,I_1:2
I_1	I_2:5
I_2	{}

Figure 1. Conditional database



(a) I_5's conditional Tree

Item	**Conditional Tree**	**Frequent Pattern**
I_5	{(I_2 I_1)}| I_5	I_2 I_5, I_1 I_5 , I_2 I_1 I_5
I_3	{(I_2 I_1)}| I_3	I_2 I_3 , I_1 I_3 , I_2 I_3 I_5

(b) Frequent conditional items

Figure 2. (a) Conditional tree and (b) frequent conditional items

Algorithm 2: FP-tree_construction

Input: A transaction database DB and a minsupcount ξ.

Output: The frequent pattern tree F

```
1. Scan the DB to get the list L of frequent items,
   and sort it in support descending order.
2. Create a FP-tree F by:
3. Create the header table, and set all the headof-
   node-links to null.
4. Create the root node T of the tree having the
   item-name of null.
5. Set the parent-link and node-link of T to null.
6.  Scan the DB again
7. For each transaction Tran in DB do
8. Get the list of frequent items.
9. Sort it according to the order L.
10. Let this list be [p|P], where p is the first item and
    P is the  remaining items.
11. Call insert_tree([p|P], T).
```

Table 7. Ordered frequent items

TID	List of Items	Items ordered by decending order
T_1	I_1, I_2,I_5	I_2,I_1, ,I_5
T_2	I_2,I_4	I_2
T_3	I_2, I_3	I_2, I_3
T_4	I_1, I_2,I_4	I_2, I_1
T_5	I_1,I_3	I_1,I_3
T_6	I_2, I_3	I_2, I_3
T_7	I_1, I_3	I_1, I_3
T_8	I_1, I_2, I_3 , I_5	I_2, I_1, I_3 , I_5
T_9	I_1, I_2, I_3	I_2, I_1, I_3
T_{10}	I_1, I_2, I_5, I_6	I_2, I_1, I_5

Algorithm 3: insert_tree

Input: the ordered list [p|P] of frequent items, and a node T of a FP-tree.

Output: the updated FP-tree.

```
1.    If T has a child node N such that the item name
      of N and p is the same then
2.    Increase the count of N by 1
3.    Else
4.    Create a new node N.
5.    Set the item_name of N to p.item_name.
6.    Set the count of N to 1.
7.    Link the parent-link of N to T.
8.   Set the node-link of N to null.
```

```
9.  If the head-of-node-links of the item h in the
      header table having the same name as p is null
      then
10. Set head-of-node-links of h to p;
11. Else
12. Traverse through the head-of-node-links of h to
      the end of the list, and link the node-link of the
      end-node to p.
13. If P is not empty then
14. Let P=[p1|P1]
15. Call insert_tree([p1|P1], N)
```

Algorithm 4: FP_growth

Input: FP-tree constructed based on Algorithm 2, using DB and a minsupcount ξ, and a pattern prefix α

Output: The complete set of frequent patterns.

```
Procedure FP-growth (Tree, α)
{
1.  If Tree contains a single path P then
2.  For each combination (denoted as β) of the nodes
    in the path P do generate pattern β ∪α with support
    =minimum support of nodes in β;
3.  Else for each ai in the header of Tree do{
4.  Generate pattern β = ai∪α with support = ai.
    Support;
5.  Construct β 's conditional pattern base and then β
    's conditional FP-tree Tree β with respect to
    minsupcount ξ;
6.  If Treeβ ≠∅ then call FP-growth (Tree β, β)
}
```

V. EXPIRIMENTAL RESULTS

For the experimental study, the same dataset is used for the tree algorithms as input in order to compare the efficiency and rapidity of the algorithms. Results obained by the above mentioned algorithms are as follows :

APRIORI

```
Candidates count: 15
The algorithm stopped at size : 5
Frequent itemsets count: 15
Maximum memory usage: 1.282684326171875 mb
Total time : ~ 17 ms
```

FP-GROWTH

```
Transactions count from database : 7
Frequent itemsets count : 15
Max memory usage: 1.2811813354492188 mb
Total time : ~ 12 ms
```

ECLAT

```
Transactions count from database: 5
Frequent itemsets count: 15
Maximum memory usage: 0.6226730346679688 mb
Total time : ~ 5 ms
```

The statistics above contain the number of generated patterns, the pattern size at which the algorithm stopped wihch in this case is five, then the memory used for the algorithm's variables and finally, the runtime.

VI. CONCLUSION AND DISCUSSION

The comparative analysis provides a framework that clearly shows the technique, database scan, and execution time of various frequent pattern mining algorithms. The above mentioned algorithms also compared on the basis of used data format and storage structure.

The performance of any algorithm can be estimated by the number of required database scan to extract patterns. The storage consumption of the above mentioned algorithms can be assessed for their utilization of memory to generate less candidate Itemset or avoid candidate Itemset generation process. Apriori algorithm charactersitics is as follows :

- Database is scanned for each time a candidate itemset is generated.

- Uses large itemset property and easy to implement
- Uses breadth first search

The drawback for apriori is that requires large memory space ,too many candidate itemset and too many passes over database

FP-Growth is good in achieving three important objectives :

- the database is scanned only two times and computational cost is decreased dramatically.
- no candidates itemset are generated.
- uses divide and conquer approach which consequently reduces the search space.

On the other hands FP-Growth has one drawback. It is difficult to use in incremental mining, FP tree needs to be updated and the whole process needs to repeat.

ECLAT algorithm is better than Apriori and near equivalent to FP-Growth performs better than Apriori and FP-Growth in terms of its execution time. Eclat has some advantages:

- The database is scannd few times (best case=2),
- Execution time less than apriori
- No need to scan database each time a candidate is generated as support count information will be obtained from previous itemset
- uses depth first search
- Eclat has drawback, requires the virtual memory to perform the transformation.

VII. REFERENCES

[1]. Hahsler, M .; Gruen, B. & Hornik, K. arules - Un environnement informatique pour les règles de l'Association minière et les ensembles d'articles fréquents . Journal of Statistical Software, 2005, 14, 1-25

[2]. Garima Jain, Diksha Maurya, Extraction of Association Rule Mining using Apriori algorithm with Wolf Search Optimisation in R Programming International Journal of Recent Technology and Engineering (IJRTE) ISSN: 2277-3878, Volume-8, Issue-2S7, July 2019

[3]. Yong Qiu, Yongjie Lan, Qing-Song Xie, An Improved Algorithm Of Mining From Fp-Tree, Proceedings of the Third International Conference on Machine Learning and Cybernedcs, Shanghai, 26-29 Aygust 2004

[4]. M. Kaur, U. Garg, et S. Kaur, « Advanced Eclat Algorithm for Frequent Itemsets Generation », p. 19.

[5]. X. Wu, V. Kumar, J. Ross Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, P. S. Yu, Z.-H. Zhou, M. Steinbach, D. J. Hand, and D.Steinberg, "Top 10 algorithms in data mining," Knowledge and Information Systems, vol. 14, no. 1, pp. 1–37, Dec. 2007.

[6]. Bo Wu, Defu Zhang, Qihua Lan, Jiemin Zheng, An Efficient Frequent Patterns Mining Algorithm based on Apriori Algorithm and the FP-tree Structure, Third 2008 International Conference on Convergence and Hybrid Information Technology

[7]. Manjit kaur , Urvashi Grag , ECLAT Algorithm for Frequent Itemsets Generation , International Journal of Computer Systems , Volume 01– Issue 03, December, 2014

[8]. Manjit kaur , Urvashi Grag, ECLAT Algorithm for Frequent Itemsets Generation, International Journal of Computer Systems (ISSN: 2394-1065), Volume 01– Issue 03, December, 2014

[9]. Xiaomei Yu, Hong Wang, Improvement of Eclat Algorithm Based on Support in Frequent Itemset Mining, journal of computers, vol. 9, no. 9, september 2014

Cite this article as :

Youssef Fakir, Rachid Elayachi, "Frequent Patterns Mining ", International Journal of Scientific Research in Computer Science, Engineering and Information Technology (IJSRCSEIT), ISSN : 2456-3307, Volume 6 Issue 4, pp. 21-29, July-August 2020. Available at doi : https://doi.org/10.32628/CSEIT2063230
Journal URL : http://ijsrcseit.com/CSEIT2063230