ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 8, August 2015

An Efficient Algorithm for Finding the Support Count of Frequent 1-Itemsets in Frequent Pattern Mining

P.Subhashini[1], Dr.G.Gunasekaran[2]

Research Scholar, Dept. of Information Technology, St.Peter's University, Chennai, India[1].

Professor, Meenakshi College of Engineering, Chennai, India[2]

ABSTRACT: Frequent itemset (or frequent pattern) mining is a very important issue in the data mining field. Both syntactic simplicity and descriptive potential, are the key features of the itemset-based pattern which have led to its widespread use in a growing number of real-life domains. Many efficient algorithms like Apriori & FP-Growth Algorithm are used to find frequent itemsets from large database. In almost all Frequent Pattern mining algorithms generating Frequent 1-itemsets are generated in order to find the support count(occurences) of each item in the entire transactions. This task is itself a tedious task in generating Frequent Patterns when considering the hugeness of modern databases available. No explicit strategy has been outlined in these algorithms to perform the aforesaid task. In this paper an efficient tree called Support Count tree has been proposed to perform this task. This algorithm can be easily embedded into any of the existing algorithms aimed at frequent pattern mining. With the help of this tree Frequent 1-Itemsets are found out quickly and efficiently which in-turn speeds up the generation of Frequent Patterns of the entire database.

KEYWORDS*:* Frequent Itemsets, support count, Frequent 1-itemsets.

I. INTRODUCTION

Data mining is key point in the process known as Knowledge Discovery in Databases (KDD), and consists of applying data analysis and discovery algorithms which produce a particular enumeration of structures (models or patterns) over the data. Association rule mining is one of the key tasks of data mining. It aims at discovering unknown relationships, providing results that can be the basis of forecast and decision [11]. For example, supermarkets or catalog companies collect sales data from the sale orders. These orders usually consist of the sale date,
the items in the transaction and customer-ID. Through getting and analyzing the association rules of these orders, a lot of valuable information such as the buying patterns of consumers can be inferred [8].

In Association rule mining rules are created by analyzing the data for frequent patterns and it uses the criteria support and confidence to identify the most important relationships. Support is an indication of how frequently the items appear in the database. Confidence indicates the number of times the if/then statements have been found to be true. Generally an association rule mining algorithm [1] contains the following steps:

1. The set of candidate k-item sets is generated by 1-extensions of the large (k-1) item sets generated in the previous iteration.

2. Supports for the candidate k-item sets are generated by a pass over the database.

3. Item sets that do not have the minimum support are discarded and the remaining item sets are called large k-item sets.

This process is repeated until no large item sets are found. Association Rule Mining techniques have been widely used in various applications such as marketing, modern business, medical analysis and website navigation analysis [6]. ARM algorithms aim at extracting interesting correlations, frequent patterns, associations or casual structures that satisfy a predetermined minimum support and confidence, among items present in transaction databases or other data repositories. Of all these Frequent Itemset Mining is one of the classical data mining problems in most of the data mining applications. Frequent Itemset Mining (FIM) tries to discover information from database based on frequent occurrences of an event according to the minimum frequency threshold provided by user[2].

II. PROBLEM DESCRIPTION

A set of items that appears frequently together in a transaction dataset is known as a frequent item set. Moreover, the item set containing 1 item is called a 1-itemset.Let us take an example. The Fig.1 is a transaction database D containing 5 transactions i.e. |D| = 5. Fig.2 gives the support count of the items whose occurrence is more than 3. There are many algorithms available for frequent pattern mining [12, 13, 14, 15, 16]. Apriori algorithm [6] and FP-tree algorithm [7] are the classic examples. Almost all of these algorithms generate frequent 1- item sets and their corresponding counts at some stage in their implementation. This task in itself is a time demanding task considering the enormity of modern databases. However, no explicit strategy has been suggested in the above-mentioned algorithms to perform this task.

TID	*Items bought*
200	{*f, a, c, d, g, i, m, p*}
201	{*a, b, c, f, l, m, o*}
202	{*b, f, h, j, o*}
203	{*b, c, k, s, p*}
204	{*a, f, c, e, l, p, m, n*}

Fig 1.Transaction ID and Itemsets

Item	*frequency*
f	*4*
c	*4*
a	*3*
b	*3*
m	*3*
p	*3*

Fig 2. 1-Itemsets and Support count

Almost all of the Frequent Pattern mining algorithms generate Frequent 1- itemsets and their corresponding counts at some stage in their implementation. This itself is a big task and no explicit strategy has been suggested in the above-mentioned algorithms to perform this task. To solve the above problem a quick, simple and easy-to-implement algorithm has been proposed.

III.RELATED WORK

In paper[1] an efficient algorithm named as BitTableFI has been proposed. In this algorithm, a special data structure BitTable is used horizontally and vertically to compress database for quick candidate itemsets generation and support count, respectively. BitTable is a set of integer whose every bit represents an Item. In BitTableFI, BitTable is used to compress the candidate itemsets and the database. For candidate itemsets compression, if candidate itemset C contains item i, bit i of the BitTable's element is marked as one; otherwise, marked as zero. For database compression, the BitTable is used vertically. If the item appears in transaction T, bit i of BitTable's T element is marked as one. Example is shown Table 1 and 2:

TID	Item
1	1,6,4
2	2,6,5
3	1,2,6,5
4	2,5

Table 1: Transactional Database

TID	Item 1	Item 2	Item 4	Item 5	Item 6
1	1	0	1	0	1
2	0	1	0	1	1
3	1	1	0	1	1
4	0	1	0	1	0
Support count	2	3	1	3	3

Table 2: Compressing the database into BitTable

In paper [2] Red black tree is used to find the support count of frequent 1-itemset. Steps involved in their algorithm are:

Step1: Instantiate a red-black tree with no element (It can be a map in C++ or a Tree Map in Java. One can also instantiate his own implementation of a red-black tree, although it may require a great deal of work).

Step 2: Scan the whole database. For each element of the database, loop through step 3.

Step 3: Insert the element in the instantiated red-black tree. If the element is not there in the tree, it will be inserted in the tree as a key with the corresponding value automatically initialized by 1. If the element already exists in the tree, its corresponding value will be incremented by 1.

Step 4: Extract each key and their corresponding values of the red-black tree. These will be the 1-itemsets and their corresponding count respectively.

IV. PROPOSED ALGORITHM

Initially number all the items which is present in the transactions from 1 and so on. It is not necessary that the items have to be sorted in the alphabetical order or in any order. If suppose the transactional database which is being considered consist of 15 items then start the numbering from 1 to 15. Next step is to form a support count tree.

Steps for generating the support count tree:

Step 1:Find the mid value of the entire item set and make it as the root node.

Step 2:Items on the left side of the mid value forms the left sub tree of the root node and items on the right side of the mid value forms the right sub tree of the root node.

Step 3:With respect to the left sub-list find the mid value and repeat step 1 and 2 until all items are included in the tree.

Step 4:With respect to the right sub-list find the mid value and repeat step 1 and 2 until all items are included in the tree.

Step 5:Now scan each and every transactions from the transactional database.

Step 6:When each transaction is scanned search each item in the transaction from the support count tree. If an item is found, increment the respective count variable of that item.

Step 7:Repeat step 6 until all the transactions have been scanned. Thus from the count variable of each item their Support counts are known.

An example has been illustrate below of how to form an initial support count tree with 15 items using the above algorithm.

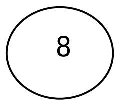
Step 1:Consider 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15 item no for 15 items. Of these 15 items 8 is the mid value and so it is made as the root node.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

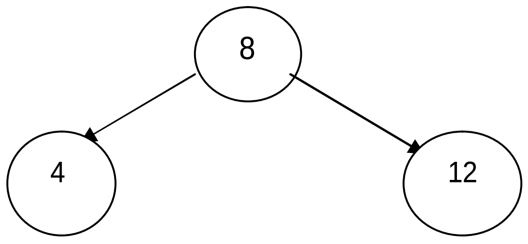
International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 8, August 2015



Step 2:Left side of item 8 forms the left sub-list and right side of item 8 forms right sub-list. Step 3:From the left sub-list find the mid value. It is item 4 and link it as the left subtree of node 8. From the right sub-list find the mid value. It is item 12 and link it as the right subtree of node 8.



The above steps are repeated until all the nodes are connected to the tree. Final tree structure is given in Figure 3.

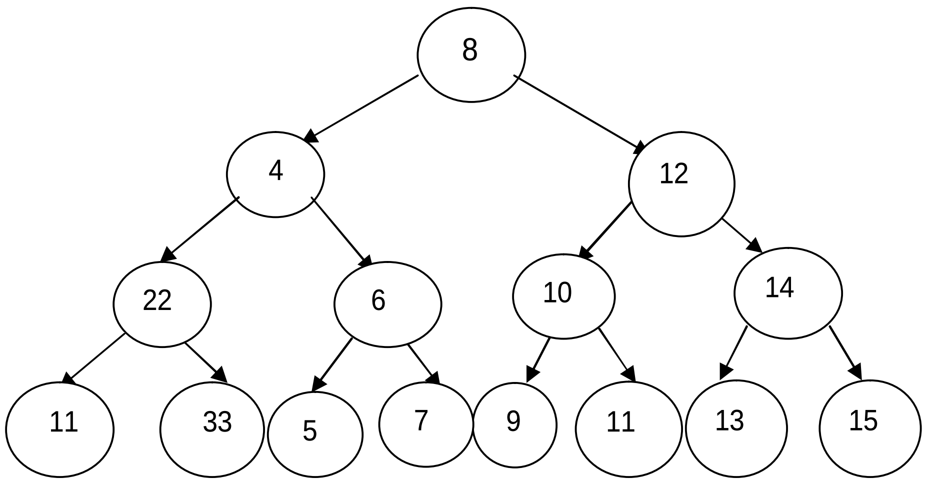


Fig 3: Initial formation of Support Count tree

Steps involved in searching an item from the support count tree:

Step 1.If the item to be searched is equal to the root node then increment its count variable and search for the next item in the transaction else go to step 2.
Step 2.check whether the value of the item to be searched is lesser than the root node. If yes go to step 3 else go step 4.
Step 3:Check whether the root node value in the left sub tree is equal to the value of the item to be searched. If yes repeat step 1 and 2.
Step 4:Check whether the root node value in the right sub tree is equal to the value of the item to be searched. If yes repeat step 1 and 2.
Repeat the above the steps until all items are searched.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 3, Issue 8, August 2015

V.EXPERIMENTAL RESULTS

This support count tree has been used in FP growth algorithm for the generating Frequent Patterns. The data set was taken from T20I7D500K. Experimental results shows that run time of FP Growth algorithm is more when compared to FP growth algorithm with support count tree which is shown in Figure 4.

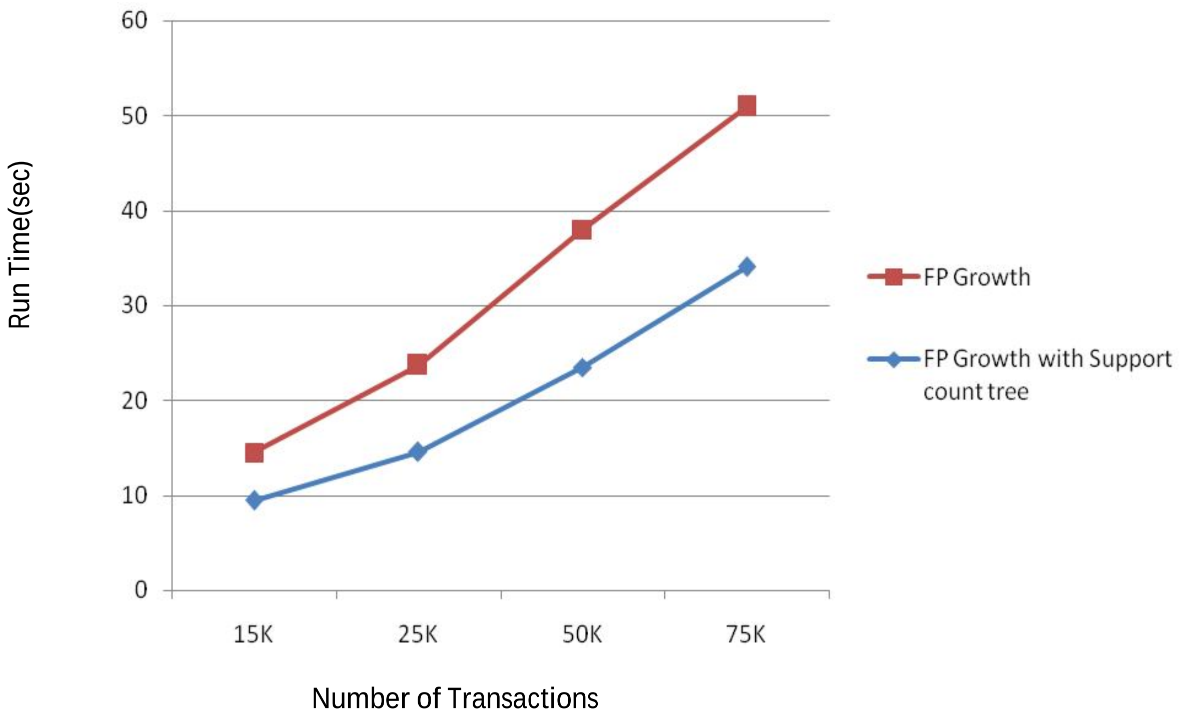


Fig 4:Performance of FP Growth and FP Growth with support count tree

VI. CONCLUSION AND FUTURE WORK

In the algorithms aimed at frequent pattern mining, extracting Frequent 1-itemsets and their corresponding counts is a computationally demanding and time consuming task in case of huge databases. In this paper a simple and easy-to-implement algorithm has been proposed to deal with this task. This algorithm can be easily embedded into any of the existing algorithms aimed at association rule mining in order to extract Frequent 1-itemsets and their corresponding counts. Thus the support count tree has been embedded in FP Growth algorithm and it has given a very good result. This work can be expanded by using the tree in other algorithms to generate Frequent Patterns efficiently.

REFERENCES

[1]Jie Dong, Min Han, "BitTableFI: An efficient mining frequent itemsets algorithm", Elsevier, Knowledge based Systems, 2006.
[2]Ming jun Song, and Sanguthevar Rajasekaran, "A Transaction Mapping Algorithm for Frequent Itemsets Mining", IEEE Transactions on Knowledge and Data Engineering, 2005.
[3]Francisco Guil, Roque Marín, "A Theory of Evidence-based method for assessing frequent patterns", Elsevier, Expert sytems with Applications, 2013.
[4]KawuuW.Lin, Sheng-hao Chung, "A fast and resource efficient mining algorithm for discovering frequent patterns in distributed computing environments" Elsevier, Future Generation computer system, 2015.
[5]S.-J. Yen, Y.-S.Lee, C.-K.Wang, J.-W.Wu, L.-Y.Ouyang, "The studies of mining frequent patterns based on frequent pattern tree, in: Advances in Knowledge Discovery and Data Mining, Springer", 2009, pp. 232–241.

[6]K.W. Lin, Y.-C. Lo, Efficient algorithms for frequent pattern mining in many task computing environments, Knowledge Based System 49 (2013) 10–21.
[7]Data Mining: Concepts and Techniques, Jiawei Han and Micheline Kamber, MORGAN KAUFMANN PUBLISHER, An Imprint of Elsevier.
[8]D. Burdick, M. Calimlim, J. Flannick, J. Gehrke, T.M. Yiu, MAFIA: a maximal frequent itemset algorithm, IEEE Transactions on Knowledge and Data Engineering 17 (11) (2005) 1490–1504.
[9]Wang, L. Tang, J. Han and J. Liu (2002), "Top-Down FP-Growth for Association Rule Mining", Lecture Notes in Computer Science Springer Berlin, Eidelberg Vol.2336, pp.334-340.
[10]E.Ozkural, B. Ucar, and C. Aykanat. Parallel frequent item set mining with selective item replication. IEEE Trans. Parallel Distrib.Syst., pages 1632–1640, 2011.
[11] M.J. Zaki, Mining non-redundant association rules, Data Mining and Knowledge Discovery 9 (3) (2004) 223–248.
[12]Agrawal, R., Imielinski, T., Swami, A. N.: Mining association rules between sets of items in large datasets, In: Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 207-216 (1993).
[13]Yuan, Y., Huang, T.: A Matrix Algorithm for Mining Association Rules. Lecture Notes in Computer Science, Volume 3644, pp. 370 - 379 Sep (2005).
[14]Tang, P., Turkia, M.: Parallelizing frequent itemset mining with FP-trees, Technical Report, Department of Computer Science, University of Arkansas at Little Rock, (2005).
[15] J. Han, J. Pei, and Y. Yin, "Mining Frequent Patterns without Candidate Generation", Proceedings of the ACM SIGMOD, Dallas, TX, May 2000, pp. 1-12.
[16]Aaron Ceglar & John F. Roddick "Association Mining" in ACM Computing Surveys, Vol. 38, No. 2, Article 5, Publication date: July 2006.