IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 16, Issue 6, Ver. VII (Nov – Dec. 2014), PP 73-78
www.iosrjournals.org

Improved Frequent Pattern Mining Algorithm with Indexing

Prof. Paresh Tanna[1], Dr. Yogesh Ghodasara[2]
[1](*Research Scholar, School of Computer Science, RK University, Rajkot, India)*
[2](*College of Information Tech., Anand Agriculture University, Anand, Gujarat, India)*

Abstract: *Efficient frequent pattern mining algorithms are decisive for mining association rule. In this paper, we examine the matter of association rule mining for items in a massive database of sales transactions. Finding large patterns from database transactions has been suggested in many algorithms like Apriori, DHP, ECLAT, FP Growth etc. But here we have introduced newer algorithm called Improved Frequent Pattern Mining Algorithm with Indexing (IFPMAI), which is efficient for mining frequent patterns. IFPMAI uses subsume indexes i.e. those itemsets that co-occurrence with representative item can be identified quickly and directly using simple and quickest method. This will become beneficial like (i) avoid redundant operations of itemsets generation and (ii) many frequent items having the same supports as representative item, so the cost of support count is reduced hence the efficiency is improved. Then an example is used to illustrate this proposed algorithm. The results of the experiment show that the new algorithm in performance is more remarkable for mining frequent patterns.*

Keywords: *Association rule, Frequent pattern mining, Subsume Indexes, IFPMAI*

I. Introduction

Association rules are if/then statements that help uncover relationships between seemingly unrelated data in a relational database or other information repository [1, 2]. An example of an association rule would be "If a customer buys a dozen eggs, he is 80% likely to also purchase milk." An association rule has two parts, an antecedent (if) and a consequent (then). An antecedent is an item found in the data. A consequent is an item that is found in combination with the antecedent. Association rules are created by analyzing data for frequent if/then patterns and using 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. The problem of association rule mining is defined as: Let $I = \{i_1, i_2,.....,i_n\}$ be a set of n binary attributes called items. Let $D = \{t_1, t_2,....., t_m\}$ be a set of transactions called the database. Each transaction in D has a unique transaction ID and contains a subset of the items in I. A rule is defined as an implication of the form X==>Y where X, Y $\subseteq$I and $X \cap Y = \theta$. The sets of items (for short itemsets) X and Y are called antecedent (left-hand-side or LHS) and consequent (right-hand-side or RHS) of the rule respectively. Example: The set of items is I = {milk, bread, butter, beer}. An example rule for the supermarket could be {butter, bread} ==> {milk} meaning that if butter and bread are bought, customers also buy milk[7]. In data mining, association rules are useful for analyzing and predicting customer behavior. They play an important part in shopping basket data analysis, product clustering, catalog design and store layout[7]. Programmers use association rules to build programs capable of machine learning. Machine learning is a type of artificial intelligence (AI) that seeks to build programs with the ability to become more efficient without being explicitly programmed. In general, association rule mining can be viewed as a two step process: (i) Find all frequent patterns and (ii) Generate strong association rules from the frequent patterns[2]. In (i), we can use some mining algorithms like Apriori, DHP, ECLAT, FP Growth etc. that we discussed later. Also we proposed newer algorithm for frequent pattern mining. In (ii), all frequent pattern rules are checked for minimum support and minimum confidence to generate association rules.

II. Existing System : Frequent Pattern Mining Algorithms

Many algorithms have been proposed for transactional database having many rows/columns. Among these we can filter out most useful methods which we can categorize them as efficient methods for mining frequent patterns. Four major frequent pattern mining approaches are: Apriori[2], Direct Hashing and Pruning (DHP)[3], Frequent pattern growth (FP-Growth)[4], Vertical data format approach (ECLAT)[5]. Apriori is the most classical and important algorithm for mining frequent itemsets. This is used to find all frequent itemsets in a given database DB. The key idea of Apriori algorithm is to make multiple passes over the database. Apriori algorithm fairly depends on the apriori property which states that "All non empty itemsets of a frequent itemset must be frequent"[2]. It also described the anti monotonic property which says if the system cannot pass the minimum support test, all its supersets will fail to pass the test [2, 3]. But Apriori having lots of challenges like huge number of database scan for generating large itemset and doing support count, large number of candidate

generation. Apriori behaves like - Generate Candidate Set and Perform Count Support from Database. While DHP behaves in a sequence – Generate candidate set, perform count support from the database and make new hash table using database for the next stage. Apriori - don't prune database but prune C_k by support counting on the original database, while DHP -Its more efficient support counting can be achieved on pruned database[3]. Eclat algorithm is a depth first search based algorithm. It uses a vertical database layout i.e. instead of explicitly listing all transactions; each item is stored together with its cover (also called tidlist) and uses the intersection based approach to compute the support of an itemset [5]. It requires less space than apriori if itemsets are small in number [5]. It is suitable for small datasets and requires less time for frequent pattern generation than apriori. FP Growth is another important frequent pattern mining method, which generates frequent itemset without candidate generation[4]. It uses tree based structure. The problem of Apriori, DHP and ECLAT algorithms were dealt with, by introducing a novel, compact data structure, called frequent pattern tree, or FP-tree then based on this structure an FP-tree-based pattern fragment growth method was developed[4]. It constructs conditional frequent pattern tree and conditional pattern base from database which satisfy the minimum support[4]. FP-growth traces the set of concurrent items[4]. It suffers from certain disadvantages: FP tree may not fit in main memory and Execution time is large due to complex compact data structure[6].

III. Proposed System: IFPMAI

In this paper we proposed new improved frequent pattern mining algorithm with indexing. This algorithm makes effective use of indexing. IFPMAI can be used for efficient large frequent pattern generation. IFPMAI uses both vertical and horizontal data format for generating large frequent pattern from the database transactions. We have found many challenges for Apriori, DHP, ECLAT and FP Growth Algorithms. With above analytics we can find some improvements that can be suggested for these algorithms like reduce passes of transaction database scans, shrink number of candidates, facilitate support counting of candidates without database scan[6]. By considering analytical study on these factors, we have experimented with subsume indexes. That is like those items having support count equal to minimum support, the item itself and possible itemsets from this item and their subsume index having the same support count as equal minimum support[9]. That is this indexing will save lots of work for support count and itemset generation.

IFPMAI Algorithm:

Steps:

1. Convert list of transactions in D into Vertical Data Format D_Ver
2. Generate TransIndex_set from D_Ver. Maintain a list for no. of iterations i.e. useful for K+1 itemsets generation.
3. Find Frequent 1-itemsets from the given TransIndex_set, is simply the length of the TransIndex_set of the itemset
4. Sort itemsets by acceding order in TransIndex_set by its minimum support.
5. Gather Items as Keyset from TransIndex_set i.e. List of Itemsets only in acceding order by its minimum support
6. Generate BitTable for each keys available in Items keyset
7. Generate Subsume for each item in Items keyset
8. For each item in Items

```
If item.subsume < > " "
    If item.Cardinality == min_sup Then
        FindItemsetsEqualsMinSup(item, item.Cardinality)
    Else
        FindItemsetsGreaterThanMinSup(item, item.Cardinality)
    End If
Else
    If item.Cardinality > min_sup
        AND  Item_Sequence<Items.Length Then
        FindItemsetsSubsumeNone(item)
    End If
End If
```

Example 1: Let us consider an example of a simple database with 10 transactions and min_sup is 2 as shown below.

Table 1 : Transactions Table

TID	Items
T1	I1,I2,I3,I5,I6,I15
T2	I1,I3,I7
T3	I5,I9
T4	I1,I3,I4,I5,I7
T5	I1,I3,I5,I7,I12
T6	I5,I10
T7	I1,I2,I3,I5,I6,I16
T8	I1,I3,I4
T9	I1,I3,I5,I7,I13
T10	I1,I3,I5,I7,I14

Table 2 : 1- Frequent Itemsets

Itemset	Sup Count
I2	2
I4	2
I6	2
I7	5
I1	8
I3	8
I5	8

Table 3 : Sorted Transactions Table

TID	Items	Ordered Items
T1	I1,I2,I3,I5,I6,I15	I2,I6,I1,I3,I5
T2	I1,I3,I7	I7,I1,I3
T3	I5,I9	I5
T4	I1,I3,I4,I5,I7	I4,I7,I1,I3,I5
T5	I1,I3,I5,I7,I12	I7,I1,I3,I5
T6	I5,I10	I5
T7	I1,I2,I3,I5,I6,I16	I2,I6,I1,I3,I5
T8	I1,I3,I4	I4,I1,I3
T9	I1,I3,I5,I7,I13	I7,I1,I3,I5
T10	I1,I3,I5,I7,I14	I7,I1,I3,I5

Table 4 : BitTable Representation of Database

TID	I2	I4	I6	I7	I1	I3	I5
T1	1	0	1	0	1	1	1
T2	0	0	0	1	1	1	0
T3	0	0	0	0	0	0	1
T4	0	1	0	1	1	1	1
T5	0	0	0	1	1	1	1
T6	0	0	0	0	0	0	1
T7	1	0	1	0	1	1	1
T8	0	1	0	0	1	1	0
T9	0	0	0	1	1	1	1
T10	0	0	0	1	1	1	1

The sorted transactions are shown in Table 3. Then the scanned database is represented by BitTable shown in Table 4. Then Calculate the intersection of transactions that contain certain frequent items one by one.

Table 5 : Frequent Pattern Generation for Each Item

ITEMS						
I2	**I4**	**I6**	**I7**	**I1**	**I3**	**I5**
I2 : 2	I4 : 2	I6 : 2	I7 : 5	I1 : 8	I3 : 8	I5 : 8
I2,I6 : 2	I4,I1 : 2	I6,I1 : 2	I7,I1 : 5	I1,I3 : 8	I3,I5 : 6	
I2,I1 : 2	I4,I3 : 2	I6,I3 : 2	I7,I3 : 5	I1,I5 : 6		
I2,I3 : 2	I4,I1,I3 : 2	I6,I5 : 2	I7,I1,I3 : 5	I1,I3,I5 : 6		
I2,I5 : 2		I6,I1,I3 : 2	I7,I5 : 4			
I2,I6,I1 : 2		I6,I1,I5 : 2	I7,I1,I5 : 4			
I2,I6,I3 : 2		I6,I3,I5 : 2	I7,I3,I5 : 4			
I2,I6,I5 : 2		I6,I1,I3,I5 : 2	I7,I1,I3,I5 : 4			
I2,I1,I3 : 2						
I2,I1,I5 : 2						
I2,I3,I5 : 2						
I2,I6,I1,I3 : 2						
I2,I6,I1,I5 : 2						
I2,I6,I3,I5 : 2						
I2,I1,I3,I5 : 2						
I2,I6,I1,I3,I5 : 2						
TOTAL = 43						
16	4	8	8	4	2	1

Let us take frequent item I2 as example, candidate
= T1 ∩ T7
= 1010111 ∩ 1010111
= 1010111
That is I2 subsume is I6 I1 I3 I5 i.e. (I2, I6 I1 I3 I5)

Continue this process accordingly for all items and finally the subsume index array is
(I2, I6 I1 I3 I5), (I4, I1 I3), (I6, I1 I3 I5), (I7, I1 I3), (I1, I3), (I3, Ø), (I5, Ø).

In above example, min_sup is 2, and items I2, I4 and I6 are having support count 2. So the support count and frequent itemsets generation for these three items is very easy. That is to find frequent itemsets, we only need to generate possible combinations for item and their subsume index, support count will be the same as min_sup. Through this way we have found 16 + 4 + 8 = 28 total out 43 for I2, I4 and I6 directly. Also for I7, which is having support count greater than min_sup, frequent itemsets will be possible combinations of item I7 and its subsume index (I1 I3) and support count will be same as I7. Item I5 which is not in subsume of I7, we calculate support count for this i.e. for I7,I5. But for rest like (I7,I1,I5), (I7,I3,I5), (I7,I1,I3,I5) having the same support count as (I7,I5). The same process is repeated for I1, I3 and I5. In this way it consumes less time for frequent patterns generation compare to other algorithms we have discussed earlier.

IV. Experimental Results and Performance Evaluation

We compared the performances of the newer algorithm with algorithms Apriori, DHP, ECLAT and FP Growth. The newer algorithm is implemented in Java and compiled with java compiler with use of Netbeans IDE 6.8. We choose the dataset from [8] for testing the performance of the new algorithm. All datasets are direct or indirect which are taken from the FIMI repository page http://fimi.cs.helsinki.fi. Table 6 below shows the characteristics of these datasets. The experiments were conducted on Windows 8 PC equipped with a core i3 processor and 2 GB of RAM memory. Execution times (in seconds) required by Apriori, DHP, ECLAT, FP Growth and newer algorithm i.e. IFPMAI are shown in below figure 1, 2, 3 and 4.

Table 6. CHARACTERISTICS OF DATASETS USED FOR EXPERIMENT EVALUATION
[1 – Apriori, 2 – DHP, 3 – ECLAT, 4 – FP Growth 5 – IFPMAI(New Algorithm)]

Dataset	Records	Algorithms Comparisons	Remarks
T10I4D100	100	1,2,3,4,5	Top 100 records from T10I4D100K
T10I4D1000	1000	1,2,3,4,5	Top 1000 records from T10I4D100K
T10I4D50000	50000	3,5	Top 50000 records from T10I4D100K
T10I4D100K	100000	3,5	-

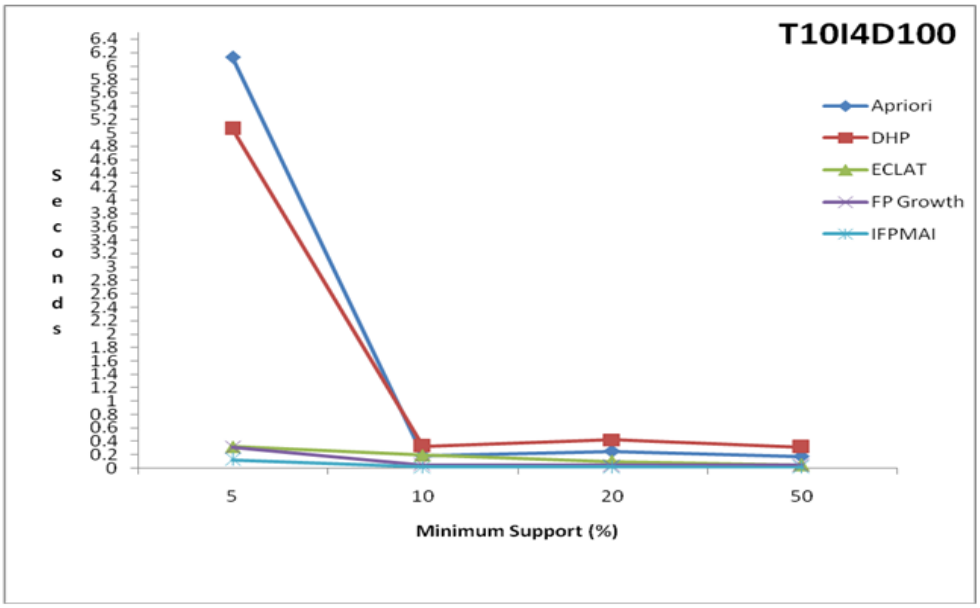

Figure 1. Execution time (in seconds) required by five different algorithms in T10I4D100 dataset with different minimum support threshold.

The dataset in Fig. 1 shows that the idea that the new algorithm runs the fastest on smaller to longer supports with small size dataset. For most supports on the datasets, the new algorithm has the best performance.

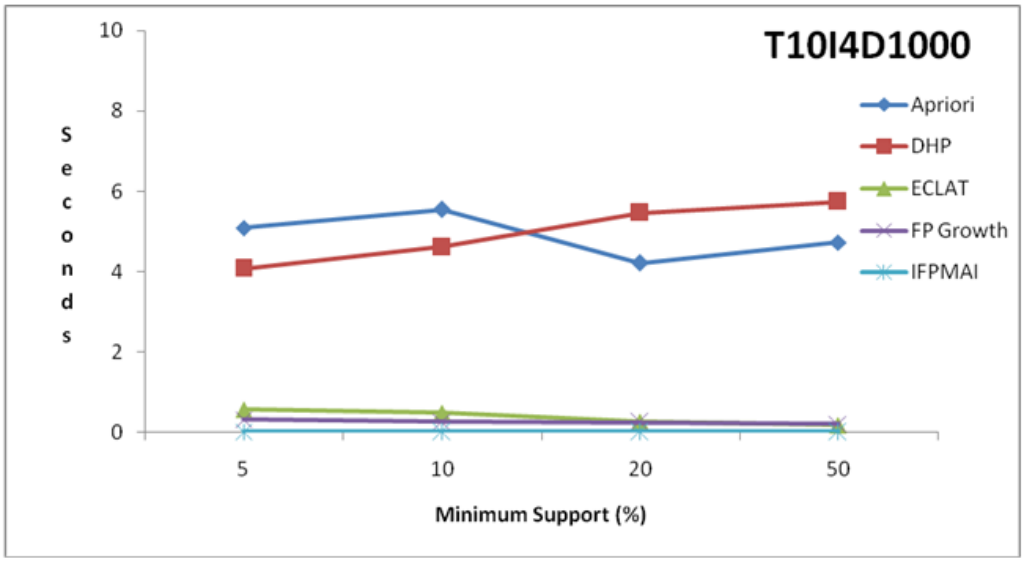

Figure 2. Execution time (in seconds) required by five different algorithms in T10I4D1000 dataset with different minimum support threshold.

Fig. 2 shows the result of computing the new algorithm with the other four algorithms on dataset. On the above fig, we can find that the new algorithm demonstrates the best performance of the four algorithms

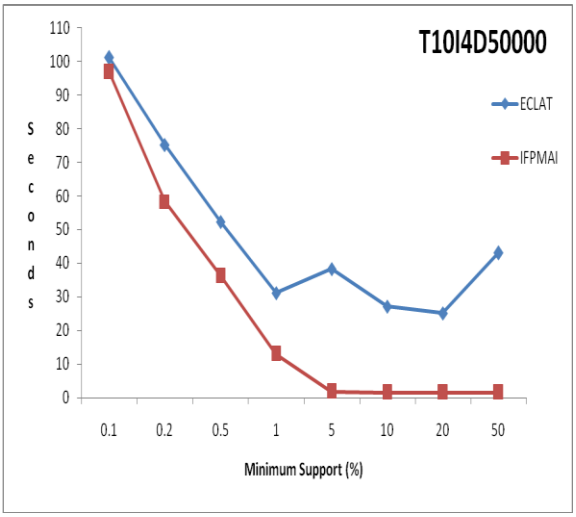

Figure 3.

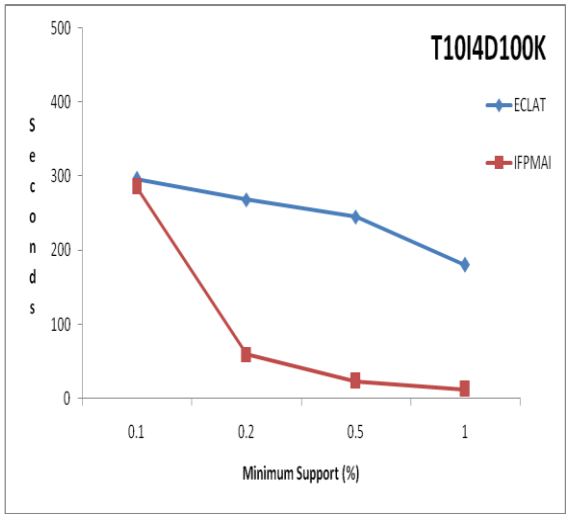

Figure 4.

Figure 3. Execution time (in seconds) required by two different algorithms in T10I4D50000 dataset with different minimum support threshold.

Figure 4. Execution time (in seconds) required by two different algorithms in T10I4D100K dataset with different minimum support threshold.

Fig. 3 and 4 show the results of computing the new algorithm with the ECLAT algorithm on different size of datasets[8]. On the above figures, we can find that the new algorithm demonstrates the best performance compare to ECLAT also.

V. Conclusion

In this paper we have used indexes which generates more frequent patterns directly. Comparing with Apriori, DHP, ECLAT and FP Growth, the new algorithm reduces time for many frequent itemsets generation and candidate frequent itemsets which support count do not need to be computed. So the efficiency of the new algorithm is better than all of the above discussed algorithms. Performance evaluation shows that new algorithm in performance is more remarkable for mining frequent patterns.

References

[1] Data Mining: Concepts and Techniques, Jiawei Han and Micheline Kamber, MORGAN KAUFMANN PUBLISHER, An Imprint of Elsevier

[2] R. Agrawal and S. Srikant, "Fast Algorithms for Mining Association Rules in Large Databases", Proceedings of the 20th International Conference on Very Large Data Bases, September 1994.

[3] J. Park, M. Chen and Philip Yu, "An Effective Hash-Based Algorithm for Mining Association Rules", Proceedings of ACM Special Interest Group of Management of Data, ACM SIGMOD'95, 1995.

[4] Han, Pei & Yin, "Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach", Data Mining and Knowledge Discovery, Volume 8, Issue 1 , pp 53-87,2004

[5] M. Zaki, S. Parthasarathy, M. Ogihara, and W. Li, " New Algorithms for Fast Discovery of Association Rules", Proc. 3rd ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD'97, Newport Beach, CA), 283-296 AAAI Press, Menlo Park, CA, USA 1997

[6] Shruti Aggarwal, Ranveer Kaur, "Comparative Study of Various Improved Versions of Apriori Algorithm", International Journal of Engineering Trends and Technology (IJETT) - Volume4Issue4- April 2013

[7] Agrawal, R., T. Imielin´ ski, and A. Swami (1993). Mining association rules between sets of items in large databases. In Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, SIGMOD '93, New York, NY, USA, pp. 207–216. ACM.

[8] Synthetic Data for Associations and Sequential Patterns. http://fimi.cs.helsinki.fi

[9] W.Song, B.R.Yang, Z.Y.Xu, "Index-BitTableFI: An improved algorithm for mining frequent itemsets," Knowledge Based Systems 21 (4) , 507–513(2008).