International Journal of Research in Advent Technology, Vol.7, No.4, April 2019
E-ISSN: 2321-9637
Available online at www.ijrat.org

MAX-MiBit-An Algorithm To Discover Maximal Frequent Itemsets From Large Transactional Datasets

R.Sujatha[1], Dr. S. Ravichandran[2],
Research Scholar[1] , Assistant Professor and Head[2],
PG and Research Department of Computer Science[1,2]
H.H. The Rajah's College (A)[1, 2]
Pudukkottai[1,2], Tamilnadu- 622001[1,2].
Email: rsujathaa77@gmail.com[1] , rajahsravis@gmail.com[2]

Abstract-A new algorithm for mining maximal frequent items was proposed in this paper. The proposed algorithm initially discover the missing items in the transactions and then uses simple bit vector computations to unearth maximal frequent itemsets from the large transactional datasets. The noticeable attribute of the proposed algorithm MAX-MiBit is that the maximal frequent itemsets are discovered in a straight forward method and the uncertain items are removed instantaneously to enhance the execution speed and to decrease the memory consumption. The experimental evaluation showcased that the proposed algorithm MAX-MiBit outscores the state of the art current algorithms by a huge margin with respect to execution speed and memory consumption.

Keywords-Frequent Itemset,Bitvector,Maximal,Datamining.

1.INTRODUCTION

Currently there are many algorithms which produce frequent itemsets efficiently. However, a drawback of these algorithms is that they may discover too many frequent patterns to users. A very large number of frequent patterns make it difficult for users to analyze results to gain insightful knowledge.

It may also cause the algorithms to become inefficient in terms of time and memory because the more frequent patterns the algorithms produce, the more resources they consume. The problem becomes worse when the database contains long frequent patterns.

A *frequent itemset* is nothing but that occurs in user-defined times in the transaction database. That number of time or occurrence is called minimum support value of the item. An itemset is considered to be *closed* if its immediate supersets do not contain the same minimum support. An itemset is considered as maximal frequent if its immediate supersets are not frequent.

2. PRELIMINARIES

Let **I** be a set of items,**I**={1,…,N}). Assume $X \subseteq \mathbf{I}$ an itemset, and here the *X* a *k-itemset* if the cardinality of itemset *X* is *k*. Let database **T** be a multiset of subsets of **I**,and let *support(X)* be the support of itemsets *Y* in **T** such that $X \subseteq Y$.

Here the support of an itemset describes how often *X* occurs in the transaction database. If *support(X) = minSup*, we say that *X* is a *frequent* itemset, and we denote the set of all frequent itemsets by **FI**. If *X* is frequent and none of its supersets are frequent, then *X* is called maximally frequent itesmset **MFI**. The relationship between frequent, closed frequent and maximally frequent is generally denoted as,

MFI⊆ FCI⊆ FI

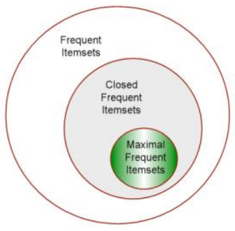

Figure 1: Relationship between FI, FCI and MFI

For brevity an itemset {A, B, C} is written as ABC. The minimum *support* of an itemset X , denoted Sup (X), is the number of transactions in which the item occurs as a subset. An itemset is *frequent* if its support is more than or equal to a user-specified *minimum support* (*min_sup*) value,
(i.e.,) if Sup(X) ≥*min sup.*

$$\text{Support} = \frac{\text{Frequency}(A,B)}{N} \quad \ldots\ldots\ldots\ldots 1$$

Where, N is the total number of transaction in the database.

Tid	Transactions
1	ABC
2	ABCD
3	BCE
4	ACDE
5	DE

Table 1: Sample transaction dataset

Consider the transaction database shown in table 1.There are five different items, I = {A; B; C; D; E } and five transactions T = {1 ; 2 ; 3 ; 4 ; 5} . The figure 2 shows all the frequent itemsets discovered for a *min sup*= 40%.

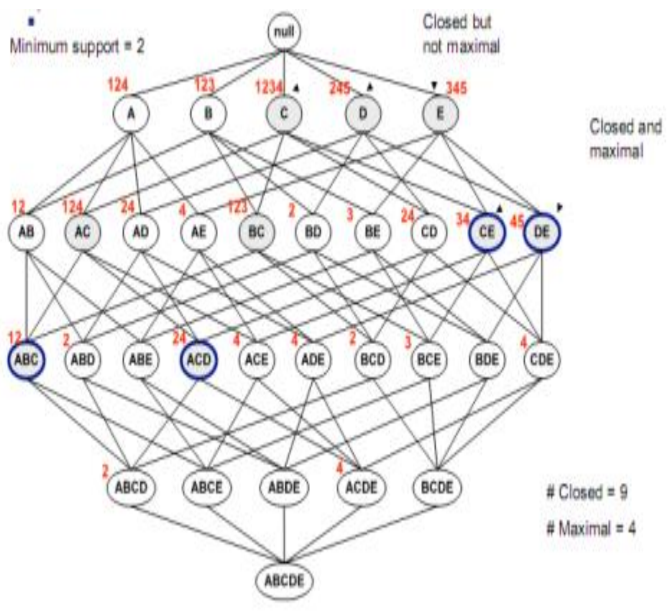

Figure 2: All frequent itemsets discovered from the sample dataset

3.PROPOSED APPROACH

The proposed approach employs missing item technique and then finds the maximal frequent itemsets after converting the transactional database into bit vector values. Initially the distinct items present in the transaction database are found and the procedure to discover distinct items is shown in the figure 3.

```
PROCEDURE DistinctITEM
( Database dT)
INPUT: Transaction Database dT
OUTPUT: DistinctItems
BEGIN:
    1.  Load the input database dT
    2.  Set Distinct[] = ϕ
    3.  ∀ TransactionRow tR present in
        dT do
    4.  ∀ RowItem rI present in tR do
    5.  If [ rI present in Distinct[] ] then
    6.  Fetch Next Item
    7.  else
    8.  Store RowItem rI → Distinct[]
    9.  End IF
    10. End For
    11. End For
    12. Return Distinct[]
END PROCEDURE
```

Figure 3: Pseudo code to find distinct items in the transaction database

The procedure to find distinct items first scans the database and then fetches each and every row and then fetches each and every items present in the transactional database to compare, if the item compared is present in the Distinct[] array, the item is ignored else, the item will be stored as distinct item. The process of discovering the distinct items present in the database is the first task during the discovery of maximal frequent itemset using the proposed MAX-MiBit algorithm.

4. FIND MISSING ITEMS IN THE TRANSACTIONS

The distinct items found by the procedure DistinctITEM is {A, B, C, D, E} and the sample dataset is again scanned to find the missing items present in every transactions and the pseudo code is shown in the figure 4.

```
PROCEDURE
GetMissingItems(Database dT,
DistinctItems d)
INPUT:Transaction Database dT,
      DistinctItems d
OUTPUT: Missing Items Array[]
BEGIN:
    1.  Load the database dT
    2.  ∀ TransactionRow $R_t$∈dT do
    3.  Find The Set Difference diff=d \ $R_t$
    4.  Store diff in Output array with
        Transaction ID
    5.  End For
    6.  Return Output array
END PROCEDURE
```

Figure 4: Pseudo code to discover the missing items in transactions.

The transactional rows present in the database dT is fetched and the set difference between the distinct item d and the transaction row fetched is discovered, stored along the TID in the missing item output array as shown in the figure 5.

{A,B,C,D,E} \ {A,B,C}	1	D,E
{A,B,C,D,E}\{A,B,C,D}	2	E
{A,B,C,D,E} \ {B,C,E}	3	A,D
{A,B,C,D,E}\{A,C,D,E}	4	B
{A,B,C,D,E} \ {D,E}	5	A,B,C

Figure 5: Working of missing item procedure

Tid	**Transactions**
1	D,E
2	E
3	A,D
4	B
5	A,B,C

Table 2: Missing items discovered

The discovered missing items are now converted into bit vectors and then used to perform simple manipulations to unearth the maximal frequent itemsets. The item present in the missing item database shown in table 2 is marked with "1", else marked with "0". The procedure to convert the missing items into bit vectors is shown in figure 5 and the converted missing values are shown in figure 7.

```
PROCEDURE ConvertToBit(Database mT, DistinctItems d)
INPUT:Database mT, DistinctItems d
OUTPUT: Bit Vectors
BEGIN:
    1. Load the database mT
    2. ∀ transaction Row $R_t$ in mT
    3. Compare Distinct Items d with Row $R_t$
    4. IF [ d present in $R_t$ ] then
    5. Mark 1
    6. Else
    7. Mark 0
    8. End IF
    9. Store Bit vectors
END PROCEDURE
```

Figure 6: Procedure to convert missing items to bit vectors

TID	**1**	**2**	**3**	**4**	**5**
A	0	0	1	0	1
B	0	0	0	1	1
C	0	0	0	0	1
D	1	0	1	0	0
E	1	1	0	0	0

Figure 7: Converted Missing Bit vectors

5. MAX-MiBit ALGORITHM

```
ALGORITHM MAX-MiBit(Database dT, Minimum Support M)
INPUT:Database dT, min_sup M
OUTPUT: Maximal Frequent Itemsets
    1. Array1[ ]= DistinctItems( Database dT)
    2. mT = GetMissingItems(Database dT, Array1[ ])
    3. BitRes= ConvertToBit(mT, Array1[ ])
    4. Fetch Row in BitRes and ADD with the next Row
    5. Apply simple addition to sum and store in RES
    6. G:Find The support Sup[RES]
    7. IF [Sup[RES]≥ M and RES not in any Superset] then
    8. Store RES in MaxR
    9. ELSE
    10. Combine the Union and Next Row in BitRes, Sum it and GOTO G
    11. End If
Return MaxR
```

Figure 8: MAX-MiBit Algorithm to find maximal frequent itemset

Let us consider the missing bit vector shown in the figure 7, the minimum support threshold provided by the user is 2(40%) and enumerate the MAX-MiBit algorithm's working. The Step 4 in the algorithm loads the missing bit vector table and the first item in the figure 7 "A" is fetched with the next item "B" and it forms the 2-itemset {AB}. The values of A and B are 00101 and 00011 respectively. The addition of {AB} is 00101 + 00011=00111. The step 6 finds the support of the itemset{AB}= 2, that is equal to the minimum support threshold value provided by the user2. This {AB} is frequent itemset but the algorithm checks if any super set equal to the minimum support value. Now {AB} value is added with the next distinct item C. The values of {AB} and {C} are added.

{AB} + C = ABC.

00101 + 00001 =>00111.

The step 6 again calculates the support for {ABC} = 2, which is equal to the minimum support value provided by the user and this 3-itemset is frequent but the next item D has to be added to check whether the superset has the same minimum support.
{ABC} + D
00011 + 10100 => 10111

The step again calculates the support of the 4-itemset {ABCD} which is found to be 1 and since it is less than the minimum support provided by the user, all the itemset which contains {ABCD} will also be non-maximal and other items are pruned. The itemset {ABC} is found to be maximal. Similarly all the maximal itemsets are discovered and shown in the table 3.

Table 3: maximal frequent itemset discovered

SNO	Maximal Itemset
1	ABC
2	ACD
3	CE
4	DE

6.EXPERIMENTAL RESULTS

The proposed MAX-MiBit algorithm was implemented using java programming language on a personal computer with 2.66GHz Intel Pentium dual core processor, 1GB RAM running on windows 7 ultimate. Benchmarked dataset like chess, Mushroom, Connect4 and Pumsb are procured from the UC Irvine Machine Learning Database Repository. The proposed algorithm MAX-MiBit is compared with state of the art algorithms like MAFIA [1] and genMAX [5]. After executing the algorithms along with the proposed algorithm the execution time is noted and compared as shown in the figures 9 and figure 10

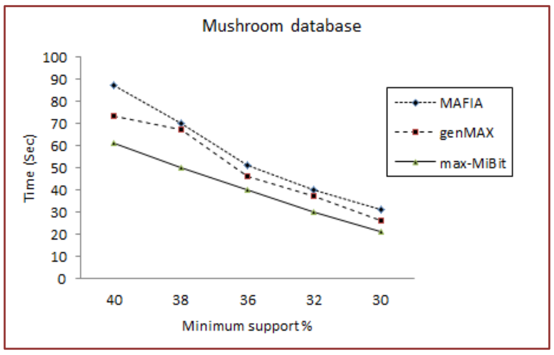

Figure 9: Comparison with respect to execution time on Mushroom dataset

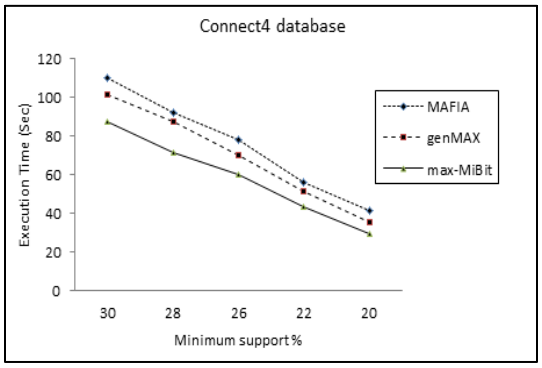

Figure 10: Comparison with respect to execution time on Connect4 dataset

7. CONCLUSION

A new vertical data representation blended with the missing items and bit vector is employed with simple arithmetic manipulation presented in this paper to discover maximal frequent itemsets performed extremely well. When dealing with denser datasets like Connect4 with long transactions, the data compression can be employed to reduce the size of the dataset which in turn will reduce the memory usage considerably. The proposed algorithm quite clearly showcased that the missing item approach with simple calculation employed, drastically reduced the execution time. From the experimental results shown in the figure 9 and figure 10 it is quite evident that the proposed MAX-MiBit algorithm outscored the existing algorithms MAFIA and genMax and took 20 to 40 % less time to produce results.

REFERENCES

[1].Doug Burdick, Manuel Calimlim, Johannes Gehrke, “MAFIA: A maximal frequent itemset algorithm for transactional databases”, Proceedings of the 17th International Conference on Data Engineering, Heidelberg, Germany, 2001.

[2] Hong-Zhen Zheng, Dian-Hui Chu, De-Chen Zhan, “Association Rule Algorithm Based on Bitmap and Granular Computing”, AIML Journal, Volume (5), Issue (3), September, 2005.

[3] Jinlin Chen, Keli Xiao, “BISC: A bit map itemset support counting approach for efficient frequent itemset mining”, ACM Transactions on Knowledge Discovery from Data (TKDD), October 2010.

[4] J. Han and M. Kamber. Data Mining: Concepts and Techniques. Morgan Kaufmann Publishers, San Francisco, CA, 2001.

[5] K. Gouda and M. J. Zaki. Efficiently mining maximal frequent itemsets. In Proceedings of the 1st IEEE International Conference on Data Mining, pages 163–170, 2001.