Webology (ISSN: 1735-188X)
Volume 18, Number 4, 2021

Discovering Frequent Itemset From Long Transactions Using OMIT Algorithm In Big Data

S Thirumaran[1] **and R Nagarajan**[2]

[1]Department of Computer Applications, Alagappa Government Arts College, Karaikudi, India.

[2]Department of Computer and Information Science, Annamalai University, India.

Abstract
The discovery of the frequent itemsets from large datasets and long transaction dataset in big data is a tedious task and it requires enormous amount of power and memory to drive and unearth the underlying patterns. The idea of utilizing the map reduce will considerably reduce the complexity of huge number of candidate generation and more importantly decreases the execution time and memory footprints but in some data sets with very long transactions the computation overhead becomes a major head ache and this paper proposes a new algorithm named as OMIT (omitted items in transaction to find frequent itemset) to evade the overheads using omitted items to discover the frequent itemsets at a rapid pace with minimum memory usage and with very less candidate generation. The proposed algorithm was compared with few existing algorithms and from the experimental results the proposed algorithm outscored the existing algorithm by a good margin.

Keywords: Apriori algorithm, Big data, Frequent itemsets, Map reduce, OMIT algorithm.

1. Introduction

The method of finding the frequent itemset from large data is a humongous task as it involves lots of memory usage, run time and system capacity to unearth the desired output and as the internet grows rapidly the data collected from it also grows rapidly and this sudden surge in the data needs to be tapped with good technique to alleviate the major overheads caused during the discovery of frequent itemsets (Han et al., 2011).

A large portion of the current data mining algorithms purely depend on Apriori approach which is a primitive approach in finding the frequent itemsets (Agrawal et al., 1996). The Apriori algorithm developed by Srikanth Agarwal utilizes a bottom up, breadth first search approach that specifies each and every frequent itemset present in the raw data. The algorithm starts by examining all transactional data in the input data base and registering the 1-item frequent items which complies the minimum support criteria. Then, the frequent candidate 2-itemsets is found from the 1–itemset.

The Apriori set the establishment for plenty of calculations that rely intensely upon the Apriori property and use the Apriori generate join strategy to generate candidate set (Agrawal et al., 1996). The Apriori property expresses that "All non-empty subsets of a frequent itemset should likewise be frequent". It defines that if an item is infrequent, then all the items associated with that item will be infrequent.

Apriori-based algorithms cited in (Brin et al., 1997; Lin and Dunham, 1998; Park et al., 1995) performance exceedingly well with sparse datasets and short transactions. But when it comes to dense datasets where long transactions are present, the algorithm's performance decreased and struggled to complete the execution in most of the databases. These dwindle in the overall performance is principally due to the following reason: All these algorithms that operates like Apriori iterates repeatedly equal to the length of the longest transaction present in the database.

Most of the previous FIM research work utilizes the traditional horizontal transactional database format and uses Apriori approach. But many vertical mining algorithms instead of horizontal mining was proposed (Zaki, 2000; Zaki and Hsiao, 2000). In vertical approach each and every item in the input database is closely related with its corresponding ID. Generally, the algorithms that utilized the vertical approach showed big improvement with respect to the performance and outperformed horizontal mining methods. Instead of using complex manipulations in generating the candidates and counting, the vertical mining method uses the ID set intersection to ease the complexity arises due to the computations.

In this paper, a new algorithm named OMIT is proposed which finds the omitted items present in the transactional rows of the database, which is usually lesser than the actual items present in the transactional rows. This omitted item approach reduces the memory usage and reduces the execution time considerably.

2. Preliminaries

Let us consider "I" be a set of items present in the transactional row, and "D" is the input database with n number of transactions, where each and every transaction present in D has a unique identifier (TID) and contains a set of items. A set $M \subseteq I$ is also called an itemset, and a set $N \subseteq D$ is called a tidset. An itemset with k items is called k -itemset. For brevity an itemset {M, N, O} is written as MNO. The support of an itemset M, denoted Sup (M), is the number of transactional rows in which it 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(M) ≥ min_sup.

$$\text{Min\_sup} = \frac{\text{Frequency (M,N)}}{\text{N}} \qquad (1)$$

where n is the total number of transactional rows in the database.

Table 1 Sample transaction database

TID	Transactions
T1	M1,M2,M4,M5
T2	M2,M3,M5
T3	M1,M2,M4,M5
T4	M1,M2,M3,M5
T5	M1,M2,M3,M4,M5
T6	M2,M3,M4

Consider an input sample database as shown in Table 1. There are five distinct items, DItems = {M1, M2, M3, M4, M5} and six transactional rows T_{row} = {T1,T2,T3,T4,T5, T6}.

2.1 Find Distinct Items

The procedure FindDistItem() is used to find the distinct items present in the data base. This procedure first scans the input DB and then fetches each and every item present in the DB. The item fetched is compared with the array element present in the array A[] and if the item is not present in the array, then that item is stored in the array A[]. If the item fetched is present, then that item is ignored and the next element in the row is fetched. This process is the first process in the proposed OMIT algorithm. The distinct items found by this procedure are DItems = {M1, M2, M3, M4, M5}. Figure 1 shows the procedures to find distinct items from transactional DB.

2.2 Procedure to Find Omitted Items

Figure 2 shows the procedures to find omitted items from transactional DB. The working of the EstimateOmitted procedure is shown in Figure 3. From Figure 3, it is quite clear that the average length of the DB is reduced considerably.

```
Find Dist Item(Database D)
INPUT: Transactional Database D
OUTPUT: Distinct Items Di ->Array
BEGIN:
    1.  Load and Scan the input database D
    2.  Initialize an empty A[]
    3.  ∀ Transaction Row r∈D do begin
    4.  ∀ Item I ∈ r do begin
    5.  If [ I Not in A[] ] then do
    6.  Store the Item I in A[]
    7.  Close IF
    8.  Close For
    9.  Close For
    10. Return A[]
```

```
END PROCEDURE
```

Figure 1 Procedure to Find Distinct Items from Transactional DB

```
Estimate Omitted (Database B, Distinct Items Di)
INPUT: Transactional Database D, Distinct Items Di
OUTPUT: Omitted Items in each transactional row → Out
BEGIN:
    1. Load and scan the input database D
    2. ∀Row r∈D do begin
    3. Fetch one Row r
    4. Find the Set Difference Diff=Di\ r
    5. Store Diff in Out along with its ID
    6. End For
    7. Return Out
END PROCEDURE
```

Figure 2 Procedure to Find Omitted Items from Transactional DB

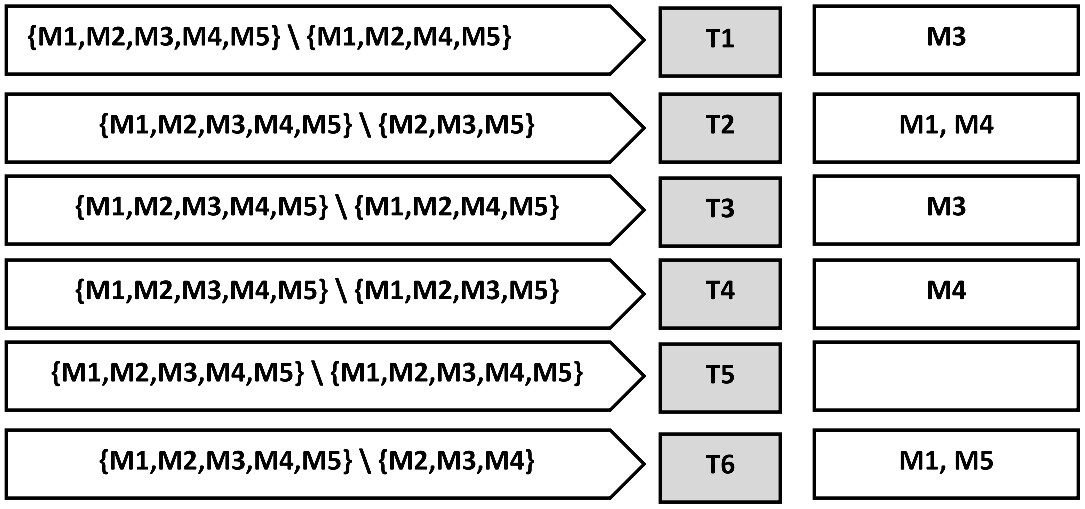

{M1,M2,M3,M4,M5} \ {M1,M2,M4,M5}
T1
M3
{M1,M2,M3,M4,M5} \ {M2,M3,M5}
T2
M1, M4
{M1,M2,M3,M4,M5} \ {M1,M2,M4,M5}
T3
M3
{M1,M2,M3,M4,M5} \ {M1,M2,M3,M5}
T4
M4
{M1,M2,M3,M4,M5} \ {M1,M2,M3,M4,M5}
T5
{M1,M2,M3,M4,M5} \ {M2,M3,M4}
T6
M1, M5

Figure 3 Working of EstimateOmitted Procedure

The database is reconverted populated with the omitted items as shown in Table 2 and here the average length is calculated to prove the effectiveness of the proposed algorithm OMIT. Figure 4 shows the procedures to convert omitted DB into binary table.

Table 2 Reconverted Sample Input Database

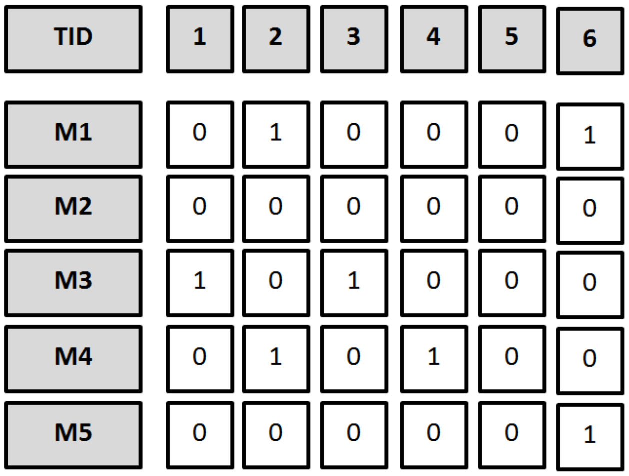
TID	Omitted Transactions
T1	M3
T2	M1,M4

T3	M3
T4	M4
T5	
T6	M1,M5

The average length of the original database is calculated as follows,
Total number of items = 23.
Average length = 23/total number of rows = 23/6 = 3.8
The average length of the reconverted database is calculated as follows,
Total number of items = 7.
Average length = 7/total number of rows = 7/6 = 1.1
The average length is reduced considerably and for very long and dense databases this method will completely minimize the overheads related to time and memory. Now the omitted item database is converted into binary table as shown in the following procedure.

```
Binary Conversion (Omited OD, Distinct Items Di)
INPUT: Omitted Database OD, Distinct Items Di
OUTPUT: Binary table →B
BEGIN:
    1. Load and Scan the Omitted database OD
    2. Fetch each Row rin OD
    3. Compare Distinct Items Di with Row r
    4. IF [ Di is present in r ] then
    5. Mark that element with "1" →B
    6. Else
    7. Mark that element with "0"→ B
    8. Close IF
    9. Return Binary table B
END PROCEDURE
```

Figure 4 Procedure to Convert Omitted DB into Binary Table



TID	1	2	3	4	5	6
M1	0	1	0	0	0	1
M2	0	0	0	0	0	0
M3	1	0	1	0	0	0
M4	0	1	0	1	0	0
M5	0	0	0	0	0	1

Figure 5 Binary Table B for Table 2

Table 2 is considered and each item is fetched and compared with the distinct items and if the item is present, ONE is marked in the binary table else ZERO is marked in the binary table as shown in Figure 5.

```
COMPUTE(Distinct Di, Binary table B, MinSup mp)
INPUT: Distinct Item Di, Binary Table B, min_sup mp
OUTPUT: Frequent Itemsets
        1.  Load the distinct Items Di
        2.  Discover candidates → C
        3.  While [ C ≠ Empty] do
        4.  Fetch the binary values of C from B
        5.  Add them and store →R
        6.  Find the number of Zeroes in R →NoZ
        7.  IF ( NoZ> = m)
        8.  STORE →RESULT
        9.  Proceed to the superset and COMPUTE
        10. Else
        11. REMOVE that ITEM
        12. Close IF
        13. Close While
    Return RESULT
END PROCEDURE
```

Figure 6 Procedure COMPUTE Present in OMIT Algorithm

3. Proposed OMIT Algorithm

Figure 6 shows the procedures of COMPUTE present in OMIT algorithm. Let us consider the user defined minimum support be 3 and let us evaluate items M1 and M2 initially as shown in Figure 7.
The binary values of M1 and M2 are initially fet6ched and simple addition is performed to find whether it is frequent or not as also shown in Figure 7.

ITEM	T1	T2	T3	T4	T5	T6	
M1	0	1	0	0	0	1	
M2	0	0	0	0	0	0	**Binary addition**
M1M2	0	1	0	0	0	1	

Figure 7 First Level Addition Performed on Items M1 and M2

ITEM	T1	T2	T3	T4	T5	T6	
M1M2	0	1	0	0	0	1	
M3	1	0	1	0	0	0	**Binary Addition**
M1,M2,M3	1	1	1	0	0	1	

Figure 8 Second Level Addition Performed on Items M1, M2 and M3

The number of zeroes is found and it is equal to 4 and the count of M1 and M2 is equal to 4 which is greater than the user defined minimum support value 3. This {M1, M2} is a frequent itemset. Now the super set of the M1, M2 is computed as shown in Figure 8.
Now the number of zeroes is computed and it is found to be 2. i.e., the count value of the 3-itemset {M1, M2, M3} = 2 which is less than 3. Therefore, the itemset {M1, M2, M3} is pruned away from the calculation and no computations are carried out further. Table 3 shows the final frequent itemset result.

3.1 OMIT Algorithm

The proposed algorithm OMIT is shown in the following Figure 9 and all the sub-procedures are included in the algorithm and this algorithm is compared with some existing algorithms to check the performance regarding time and memory.

SNO	ITEMSET	SUPPORT VALUE
1	**M1**	4
2	**M2**	6
3	**M3**	4
4	**M4**	4
5	**M5**	5
6	**M1,M2**	4
7	**M1,M4**	3
8	**M1,M5**	4
9	**M2,M3**	4
10	**M2,M4**	4
11	**M2,M5**	5
12	**M3,M5**	3
13	**M4,M5**	3
14	**M1,M2,M4**	3
15	**M1,M2,M5**	3
16	**M1,M3,M5**	3
17	**M2,M3,M5**	3
18	**M2,M4,M5**	3
19	**M1,M2,M4,M5**	3

Table 3 Final Frequent Itemset Result

3.2 Advantages of Omitted Item Approach

A thorough experiment and analysis was conducted to prove the advantage of utilizing the omitted items approach and plethora of benchmarked databases are procured and executed to find the omitted items. The sample database shown in table 1 has an average length of 3.8. But the average length of the omitted item database is reduced to 1.1. The original database size is reduced more than 3 times and hence the memory usage and time taken for execution also reduces. The benchmarked databases that are used in the experiment to prove the effectiveness of the reduction in the database size is shown in Table 4.

OMIT(database D, minimum support mp)

INPUT: Transaction database D, min_sup mp
OUTPUT: Frequent Itemsets

```
1. Sort the input data D
2. Di[]=FindDistItem(D)
3. OD[] = EstimateOmitted(D, Di)
```

```
    4. B= **BinaryConversion(OD, Di)**
    5. Permute candidates→CC
    6. Freq[]=COMPUTE(Di, B, mp)

END Algorithm
```

Figure 9 Proposed OMIT Algorithm

Table 4 Tidset Size and Omitted Item Size Comparison

Database Name	Min_Sup value	Avg Tidset Size value	Avg. Omitted item size value
Chess	0.6%	1623	38
Mushroom	6%	678	77
Connect	72%	48962	136
Pumsb	30%	15819	586

From Table 4, it is quite obvious that the omitted item size database decreases considerably as the average size of the omitted database reduces to a great extent. The proposed approach is more effective in discovering the frequent itemsets for databases with long transactions and for databases with dense items.

4. Experimental Results

The proposed OMIT algorithm was implemented using java programming language on a personal computer with 2.66 GHz Intel Core I5, 4GB RAM running on windows 10. All the benchmarked databases are downloaded from the UCI Database Repository. The proposed OMIT algorithm was compared with Apriori, vertical format Bi-Eclat algorithms, DBV algorithms and the results are showcased in the Figure 10 to 14.

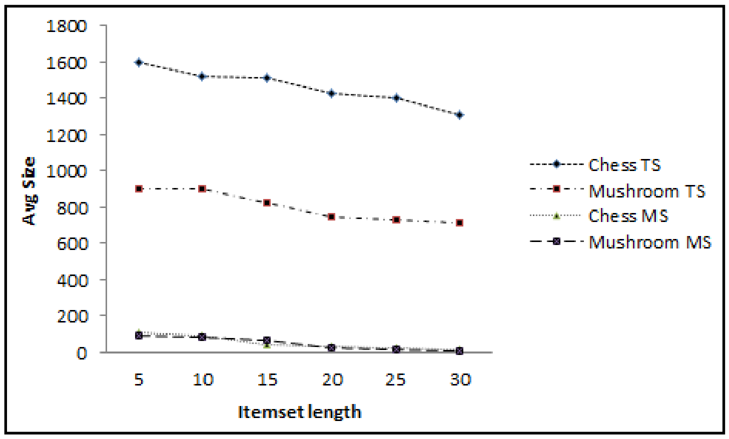


Figure 10 TIDSET Comparison with Omitted Item Size

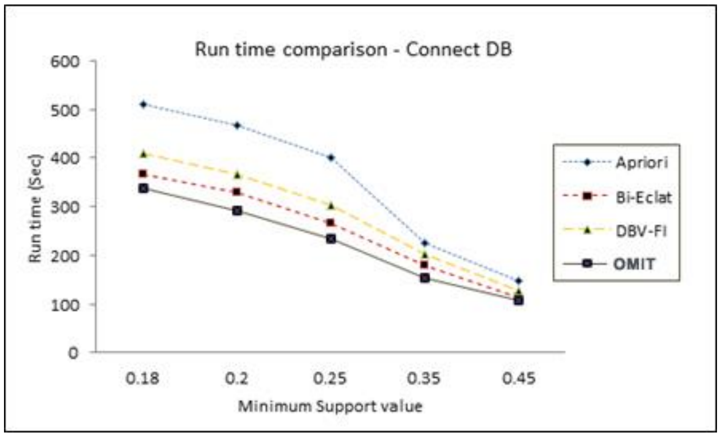


Figure 11 Runtime Comparison for Connect database

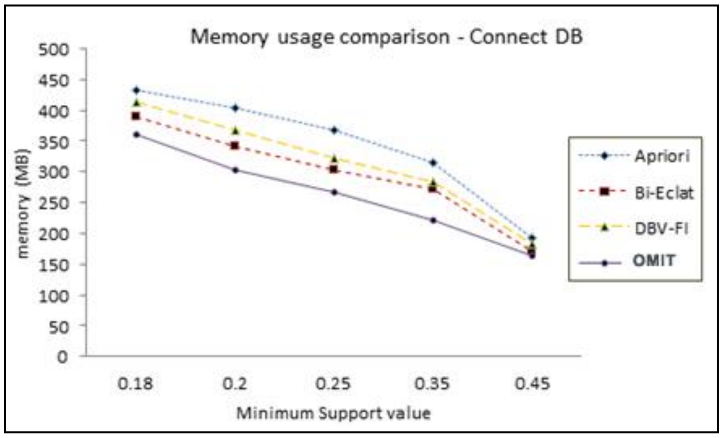


Figure 12 Memory Usage Comparison for Connect Database

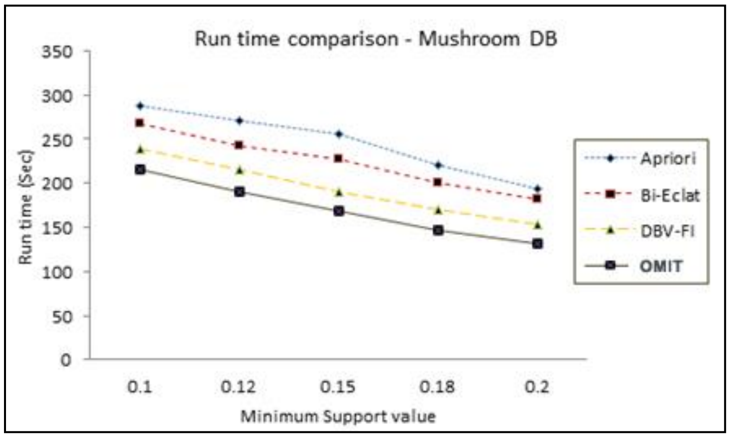


Figure 13 Run Time Comparison for Mushroom Database

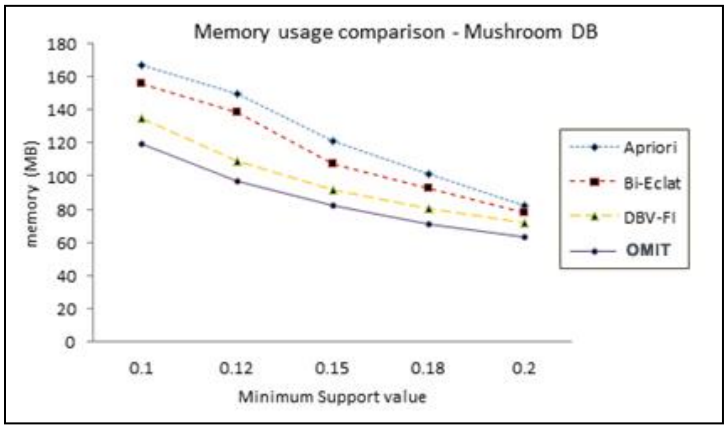


Figure 14 Memory Usage Comparison for Mushroom Database

From the Figure 10 to 14, it is quite obvious that the proposed OMIT algorithm outscored the other three algorithms by a good margin with respect to speed and memory usage and this is mainly because of reconstruction of the original database with the omitted items.

5. Conclusions

The proposed OMIT algorithm performed extremely well with the benchmarked databases and proved to be a powerful tool to deal with very dense databases with long transactions. The OMIT algorithm can be combined with map reduce to speed up the entire process further and the database size can be further reduced by compressing the database to increase the speed of the execution by a large volume. The proposed OMIT algorithm quite clearly showcased that the omitted item approach employed cuts down the size of memory required to store the candidates since the size of the database is reduced the execution time or the run time required also reduces considerably.

References

[1] Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., Verkamo, A.I., others, 1996. Fast discovery of association rules. Adv. Knowl. Discov. Data Min. 12, 307–328.
[2] Brin, S., Motwani, R., Ullman, J.D., Tsur, S., 1997. Dynamic itemset counting and implication rules for market basket data, in: Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data. pp. 255–264.
[3] Han, J., Pei, J., Kamber, M., 2011. Data mining: concepts and techniques. Elsevier.
[4] Lin, J.-L., Dunham, M.H., 1998. Mining association rules: Anti-skew algorithms, in: Proceedings 14th International Conference on Data Engineering. IEEE, pp. 486–493.
[5] Park, J.S., Chen, M.-S., Yu, P.S., 1995. An effective hash-based algorithm for mining association rules. Acm Sigmod Rec. 24, 175–186.
[6] Zaki, M.J., 2000. Scalable algorithms for association mining. IEEE Trans. Knowl. Data Eng. 12, 372–390.

[7] Zaki, M.J., Hsiao, C.-J., 2000. An efficient algorithm for closed association rule mining, in: 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 34–43.