A Hash based Mining Algorithm for Maximal Frequent Item Sets using Linear Probing

A.M.J. Md. Zubair Rahman, P. Balasubramanie and P. Venkata Krihsna[1]
Kongu Engineering College, Perundurai, Tamilnadu, India
[1]School of Computing Sciences, VIT University, Vellore

Abstract. Data mining is having a vital role in many of the applications like market-basket analysis, in bio-technology field etc. In data mining, frequent itemsets plays an important role which is used to identify the correlations among the fields of database. In this paper, we propose an algorithm, HBMFI-LP which hashing technology to store the database in vertical data format. To avoid hash collisions, linear probing technique is utilized. The proposed algorithm generates the exact set of maximal frequent itemsets directly by removing all non-maximal itemsets. The proposed algorithm is compared with the recently developed MAFIA algorithm and is shown that the HBMFI-LP outperforms in the order of two to three.

Key words*:* Mining-Frequent Item Sets-Hashing-Linear Probing-MAFIA etc

(Received July 26, 2008 / Accepted January 19, 2009)

1. Introduction

Frequent itemset mining has wide applications. The research in this field is started many years before but still emerging. This is a part of many data mining techniques like association rule mining, classification, clustering, web mining and correlations. The same technique is applicable to generate frequent sequences also. In general, frequent patterns like tree structures, graphs can be generated using the same principle. There are many applications where the frequent itemset mining is applicable. In short, they can be listed as market-basket analysis, bioinformatics, networks and most in many analyses. Agarwal et. al [4] is the first person to state this problem. Later many algorithms were introduced to generate frequent itemsets.

Let I = { I_1, I_2, I_3, …, I_m} be a set of items. Let D be the transactional database where each transaction T is a set of items such that $T \subseteq I$. Each transaction is associated with an identifier TID. A set of items is referred as itemset. An itemset that contains K items is a K-itemset. The number of transactions in which a particular itemset exists gives the support or frequency count or count of the itemset. If the support of an itemset I satisfies the minimum support threshold, then the itemset I is a frequent itemset.

Frequent pattern mining can be classified based on the completeness of patterns to be mined, the levels of abstraction involved in the rule set, the number of data dimensions involved in the rule, the types of values handled in the rule, the kinds of rules to be mined, the kinds of patterns to be mined. The classification of algorithms for frequent itemset mining is Apriori-like algorithms, frequent pattern growth based algorithms such as FP-growth and algorithms that use the vertical data format.

It is impractical to generate the entire set of frequent itemsets for the very large databases [1]. There is much research on methods for generating all frequent itemsets efficiently [8, 11, 14, 17, 18, 19, 20]. Most of these algorithms use a breadth-first approach, i.e. finding all k-itemsets before considering (k+1) itemsets. The performance of all these algorithms gradually degrades with dense datasets such as telecommunications and census data, where there are many, long frequent patterns.

The main drawback of frequent itemsets is they are very large in number to compute or store in computer. This leads to the introductions of closed frequent itemsets and maximal frequent itemsets. An itemset X is closed in a data set S if there exists no proper super-itemset Y such that Y has the same support count as X in S. An itemset X is closed frequent itemset in set S if X is closed and frequent in S. an itemset X is a maximal frequent itemset in set S if X is frequent and there exists no super-itemset Y such that $X \subset Y$ and Y is frequent in S. Maximal frequent itemset mining is efficient in terms of time and space when compared to frequent itemsets and closed frequent itemsets because both are subsets of maximal frequent itemset. Some of the algorithms developed for mining maximal frequent itemsets are MaxMiner [1], DepthProject [2], GenMax [3], FPMax* [5], MAFIA [6].

The organization of the paper is as follows. Section 2 discusses the related work to our approach on. In Section 3, we describe the proposed algorithm, HBMFI-LP. Section 4 discusses about the results obtained from the comparison of the proposed algorithm with MAFIA. Section 5 concludes the work proposed.

2. Related Work

Methods for finding the maximal elements include All-MFS [7], which works by iteratively attempting to extend a working pattern until failure. A randomized version of the algorithm that uses vertical bit-vectors was studied, but it does not guarantee every maximal pattern will be returned.

MaxMiner [1] is another algorithm for finding the maximal elements. It uses efficient pruning techniques to quickly narrow the search. MaxMiner employs a breadthfirst

traversal of the search space; it reduces database scanning by employing a lookahead pruning strategy DepthProject [2] finds long itemsets using a depth first search of a lexicographic tree of itemsets, and uses a counting method based on transaction projections along its branches.

It returns a superset of the **MFI** and would require post-pruning to eliminate non-maximal patterns. FPgrowth [8] uses the novel frequent pattern tree (FP-tree) structure, which is a compressed representation of all the transactions in the database.

Mafia [6] is the most recent method for mining the **MFI**. Mafia uses three pruning strategies to remove non-maximal sets. The first is the look-ahead pruning first used in MaxMiner. The second is to check if a new set is subsumed by an existing maximal set.

Apriori is the first efficient algorithm that performs on large databases which was proposed by Agrawal and Srikant [9] and Mannila et. al [10] independently at the same time. They proposed their cooperative work in [11]

MaxMiner [1] performs a breadth-first traversal of the search space as well, but also performs lookaheads to prune out branches of the tree. The lookaheads involve superset pruning, using apriori in reverse (all subsets of a frequent itemset are also frequent). In general, lookaheads work better with a depth-first approach, but MaxMiner uses a breadth-first approach to limit the number of passes over the database. DepthProject [2] performs a mixed depth-first traversal of the tree, along with variations of superset pruning. Instead of a pure depth-first traversal, DepthProject uses dynamic reordering of children nodes. With dynamic reordering, the size of the search space can be greatly reduced by trimming infrequent items out of each node's tail. Also proposed in DepthProject is an improved counting method and a projection mechanism to reduce the size of the database. The other notable maximal pattern methods are based on graph-theoretic approaches. MaxClique and MaxEclat [12] both attempt to divide the subset lattice into smaller pieces ("cliques") and proceed to mine these in a bottom-up Apriori-fashion with a vertical data representation. The VIPER algorithm has shown a method based on a vertical layout can sometimes outperform even the optimal method using a horizontal layout [13]. Other vertical mining methods for finding **FI** are presented by Holsheimer [15] and Savasere et al. [16]. The benefits of using the vertical tid-list were also explored by Ganti et al. [14].

3. Proposed Work

In general the structure of the transactional database may be in two different ways – Horizontal data format and Vertical data format. In this paper, transactions of database are stored in the vertical format. In vertical data format, the data is represented as item-tidset format, where item is the name of the item and tidset is the set of transaction identifiers containing the item. We use closed hashing technique to represent this data format. To avoid collisions, linear probing technique is used. The sample transactional database using vertical format is shown in table 1.

Table 1: The vertical data format of the transactional database D in first level

Itemset	Tidset
I1	T1, T3, T4, T5, T7, T9
I2	T1, T2, T4, T6, T8, T9
I3	T1, T4, T6, T7, T10
I4	T1, T2, T5, T8, T10
I5	T2, T3

The items in the transactions are hashed based on the hash function as h(k) = (order of item K) mod n. Here, we consider n to be 7. So, the item I1 is stored in 1st location. The transactions in which item I1 is present are connected to this location in the form of linked list. Each list contains the header node in which the count of the transactions linked to the particular item is maintained. Collisions are resolved by linear probing technique. Similarly, the process is repeated for other items in the transactional database. It can be observed from Fig.1 that I2 is located at 2, I3 at 3, I4 at 4 and I5 at location 5. From the hash table shown in the Fig. 1, it can be observed that the items I1, I2, I3, and I4 are the frequent items when support is considered to be 3. So, only these items are considered for generating 2-item frequent sets. The m-itemsets can be generated by taking all the combinations of frequent itemsets from the previous level and intersection of the tidsets of the corresponding itemsets. The 2-itemsets generated in this fashion are shown in table 2.

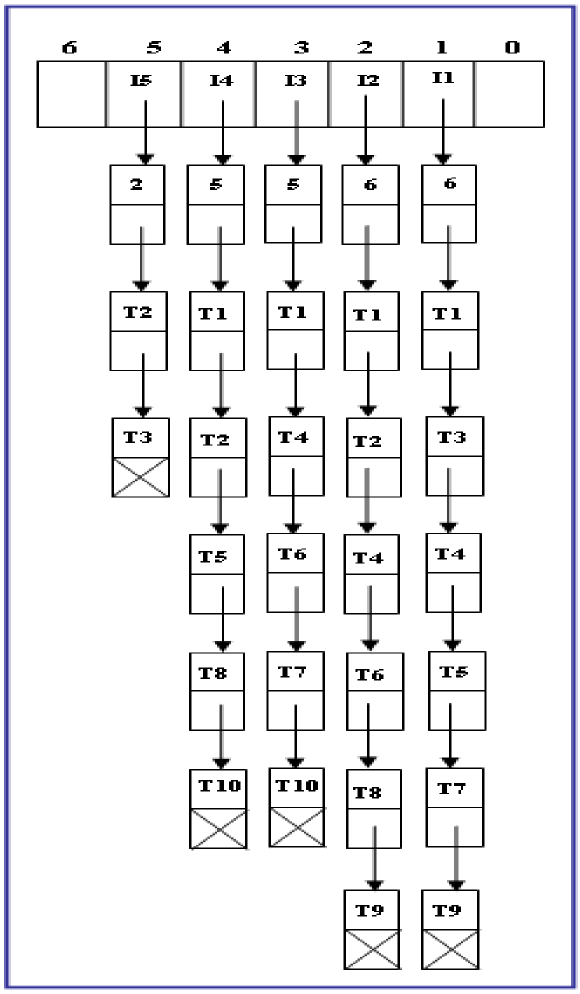

Fig. 1 Hash Table including links for the transactional database in the first level

Table 2: The vertical data format of the transactional database D in second level

Itemset	Tidset
{I1, I2}	T1, T4, T9
{I1, I3}	T1, T4, T7
{I1, I4}	T1, T5
{I2, I3}	T1, T4, T6
{I2, I4}	T1, T2, T8
{I3, I4}	T1, T10

The itemsets in the second level are hashed based on the hash function, h(k) = ((order of X)*10 + order of Y) mod n. Here n is considered to be 7. Using this hash function, itemsets {I1, I2}, {I1, I3}, {I1, I4}, {I2, I3}, {I2, I4}, {I3, I4} are mapped to locations 5, 6, 0, 2, 3, 1 respectively. Here there is a collision for {I1, I3} and {I3, I4}. Both the sets are being mapped to location 6. Then using linear probing technique, {I3, I4} is mapped to location 1. The hash table for the second level is shown in Fig. 2.

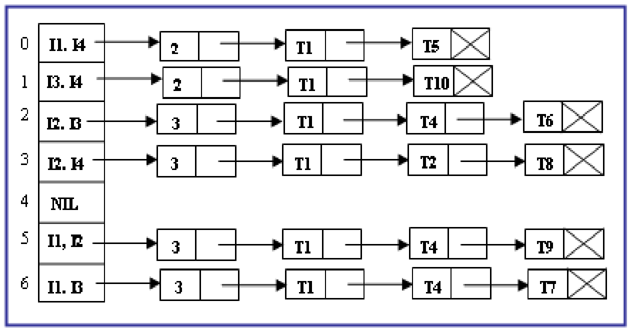

Fig. 2 Hash Table including links for the transactional database in the second level

In the second level, the itemsets {I1, I2}, {I1, I3}, {I2, I3}, {I2, I4} are frequent itemsets which can be observed from table 2 and Fig. 2. Now, total frequent itemsets are {I1, I2, I3, {I1, I2}, {I1, I3}, {I2, I3}, {I2, I4}}. From these, maximally frequent itemsets are {I1, I2}, {I1, I3}, {I2, I3}, {I2, I4}.
The 3-itemsets are generated from the maximal frequent itemsets of second level. They are shown in table 3.

Table 3: The vertical data format of the transactional database D in third level

Itemset	Tidset
{I1, I2, I3}	T1, T4
{I1, I2, I4}	T1
{I2, I3, I4}	T1

The itemsets in the second level are hashed based on the hash function, h(k) = floor(((order of X)*100 + (order of Y)*10 + order of Z) mod n). Here n is considered to be 7. Using this hash function, itemsets {I1, I2, I3}, {I1, I2, I4}, {I2, I3, I4} are mapped to locations 4, 5, 3 respectively. The hash table for the third level is shown in Fig. 3.

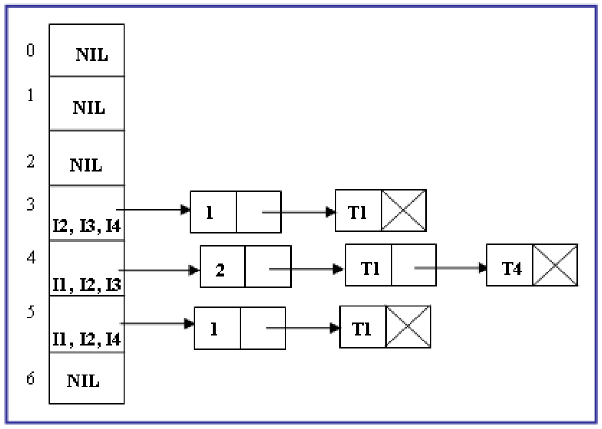

Fig. 3 Hash Table including links for the transactional database in the third level

It can be observed that there are maximal frequent itemset from the third level. So, the final maximal frequent itemsets are {I1, I2}, {I1, I3}, {I2, I3}, {I2, I4} which are obtained at second level.
It can be observed that the number of levels increase as the support is decreased. If we increase the support, then number of levels decreases and so as the time to find MFI decreases. Minimum support threshold must be properly chosen such that it is not too high where we may loose some interesting itemsets or too low where unimportant itemsets are generated
In this procedure we need not calculate the support of the itemset separately. It can be taken by the number of transactions in the tidset which is available as a count value in header node. Also the pruning can be done while finding the MFI itself, but not after finding FI completely. The proposed algorithm is given below.

HBMFI-LP Algorithm

```
Input:
D, a database of transactions
Min_sup, the minimum threshold support
Output:
M, Maximal frequent itemsets in D
Method:
    1. Generate the vertical format of the
       transactional database.
    2. N-itemsets are to be hashed
    3. Linked list of transactions for each itemset is
       created with count in its header node.
    4. generate FI
    5. if FI is NULL then go to 9.
    6. Remove any subsets that are included in
       another itemset from FI to generate MFI.
    7. Find all combinations of the MFI
    8. go to 1
    9. return MFI
```

The proposed algorithm performs better because MFI is being calculated directly before computing FI completely. At each level, after computation of FI, we are computing MFI also. So, the time taken to compute MFI is negligible. And also it shows that no separate pruning is required. Hash data structure can be maintained to store database. This makes easy in performing several tasks. As we are following vertical data format, support also need not be calculated separately. In this case, support is directly given by the number of transactions in the tidlist of each FI or can be obtained from the count value in the header node of the corresponding linked list.

4. Results and Discussions

The large datasets which have long itemsets like chess, mushroom, connect4 are used to test this algorithm. When the support values are higher, the number of items in itemsets varies from 4 to 12 items, whereas when the support values are lower, each itemset contains nearly 22 items. This makes the task of finding the MFI computationally intensive despite the small size of the databases.
Figs. 3 - 5 illustrate the results of comparing HBMFI-LP to our implementation of MAFIA method. Support is taken as X-axis and the time taken to find the MFI is taken as Y-axis.

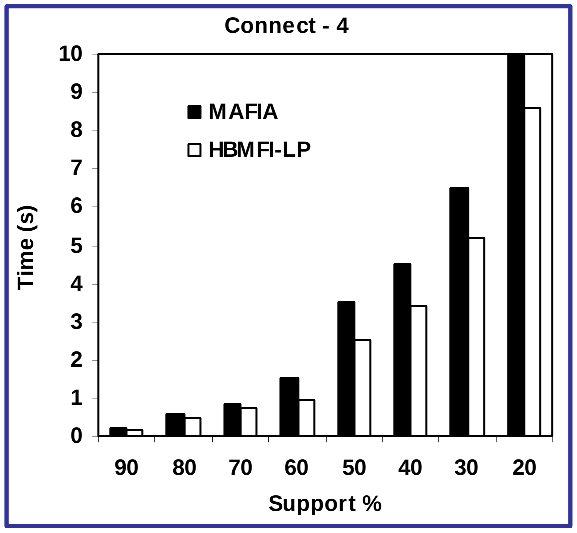

Fig. 3: Time Comparison of MAFIA and HBMFI-LP on Connect – 4 dataset

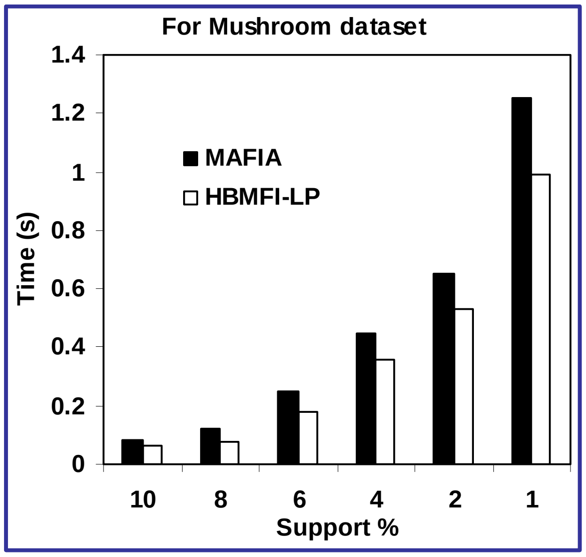

Fig. 4: Time Comparison of MAFIA and HBMFI-LP on Mushroom dataset

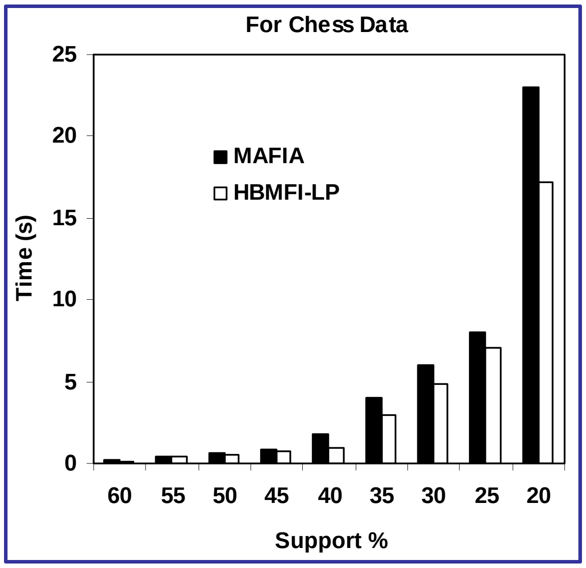

Fig. 5: Time Comparison of MAFIA and HBMFI-LP on Chess dataset

For Connect-4, the increased efficiency of itemset generation and support counting in MAFIA and HBMFI-LP explains the improvement. Connect-4 contains an order of magnitude more transactions than the other two datasets, amplifying the advantage in generation and counting.
For Mushroom, the improvement is best explained by how the MFI is computed at each level and found directly without waiting for FI completely. This leads to a much greater reduction in the overall search space than for the other datasets, since the reductions is so great at highest levels.
The percentage in the improvement of the performance of the proposed algorithm, HBMFI-LP is less in Chess dataset when compared to other datasets. The extremely low number of transactions and small number of frequent items at low supports muted the factors that HBMFI relies on to improve over MAFIA.
Both the algorithms scale linearly with the database size, but HBMFI-LP is about 2 to 3times faster than MAFIA. Thus we see that HBMFI-LP performs better with large number of transactions and long itemsets.

5. Conclusions

We presented HBMFI-LP, an algorithm for finding maximal frequent itemsets. Our experimental results demonstrate that HBMFI-LP consistently outperforms MAFIA by a factor of 2 to 3 on average. The vertical data format representation of the database, the easy manipulations on hash data structure and directly computing MFI are the added advantages of this algorithm and hence HBMFI-LP shows better performance in terms of time taken to generate MFI when compared to MAFIA.

6. References

[1]. Roberto Bayardo, "Efficiently mining long patterns from databases", in ACM SIGMOD Conference 1998.
[2]. R. Agarwal, C. Aggarwal and V. Prasad, "A tree projection algorithm for generation of frequent itemsets", Journal of Parallel and Distributed Computing, 2001.
[3]. K. Gouda and M.J.Zaki, "Efficiently Mining Maximal Frequent Itemsets", in Proc. of the IEEE Int. Conference on Data Mining, San Jose, 2001.
[4]. R. Agrawal, T. Imielienski and A. Swami, "Mining association rules between sets of items in large databases. In P. Bunemann and S. Jajodia, editors, Proceedings of the 1993 ACM SIGMOD Conference on Management of Data, Pages 207-216, Newyork, 1993, ACM Press.
[5]. Gosta Grahne and Jianfei Zhu, "Efficiently using prefix-trees in Mining Frequent Itemsets", in Proc. of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations Melbourne, Florida, USA, November 19, 2003.
[6]. Burdick, D., M. Calimlim and J. Gehrke, "MAFIA: A maximal frequent itemset algorithm for transactional databases", In International Conference on Data Engineering, pp: 443 – 452, April 2001, doi = 10.1.1.100.6805
[7]. D. Gunopulos, H. Mannila, and S. Saluja, " Discovering all the most specific sentences by randomized algorithms", In Intl. Conf. on Database Theory, Jan. 1997.
[8]. J. Han, J. Pei, and Y. Yin. "Mining frequent patterns without candidate generation", In ACM SIGMOD Conf., May 2000.
[9]. Heikki Mannila, Hannu Toivonen, and A. Inkeri Verkamo, "Efficient Algorithms for discovering association rules", in Usama M. Fayyad and Ramasamy Uthurusamy, editors, AAAI Workshop on Knowledge Discovery on Databases (KDD-94), pages 181-192, Seattle, Washington, 1994, AAAI Press.
[10]. R. Agrawal and R. Srikant, "Fast algorithms for mining association rules", in Proceedings of the 20th International Conference on Very Large Databases (VLDB'94), Santiago de Chile, September 12-15, pages 487-499, Morgan Kaufmann, 1994.
[11]. R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo, "Fast discovery of association rules", Advances in Knowledge Discovery and Data Mining, pages 307-328, MIT Press, 1996.

[12]. M. J. Zaki, "Scalable Algorithms for Association Mining", IEEE Transactions on Knowledge and Data Engineering, Vol. 12, No. 3, pp 372-390, May/June 2000.

[13]. P. Shenoy, J. R. Haritsa, S. Sudarshan, G. Bhalotia, M. Bawa, and D. Shah, "Turbo-charging Vertical Mining of Large Databases", SIGMOD Conference 2000: 22-33

[14]. V. Ganti, J. E. Gehrke, and R. Ramakrishnan, "DEMON: Mining and Monitoring Evolving Data", ICDE 2000: 439-448

[15]. M. Holsheimer, M. L. Kersten, H. Mannila, and H.Toivonen, "A Perspective on Databases and Data Mining", KDD 1995: 150-155.

[16]. A. Savasere, E. Omiecinski, and S. Navathe, "An efficient algorithm for mining association rules in large databases", 21st VLDB Conference, 1995.

[17]. Aggarwal, C.C. and P.S. Yu, "Mining large itemsets for association rules", in Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, 1998, pp: 23-31. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.48.306

[18]. Aggarwal C.C and P.S. Yu, "Online generation of association rules", in proceedings of the fourteenth International Conference on Data Engineering, 1998, pp:402-411.

[19]. Park J.S, M.S. Chen, P.S. Yu, "An Effective Hash Based Algorithm for Mining Association Rules", ACM SIGMOD Record, Vol. 24, Issue 2, May 1995, pp: 175-186, ISSN: 0163-5808.

[20]. Dunkel B. and N. Soparkar, "Data Organization and access for efficient data mining", in the proceedings of the 15th International Conference on Data Engineering, pp: 522-529, 1999, ISBN: 0-7695-0071-4