MMFI: an Effective Algorithm for Mining Maximal Frequent Itemsets

Shiguang Ju[1]
[1]*School of Computer Science and Telecommunication Engineering, Jiangsu University, China 212013*
jushig@ujs.edu.cn

Chen Chen[1, 2]
[2]*Jiangsu Professional Technology Institute of Finance and Economics, China 223001*
hhgmcc@163.com

Abstract

Existing algorithms for mining maximal frequent itemsets have to do superset checking, and some of them using FP-tree have to construct conditional frequent pattern trees recursively. We present a novel algorithm for mining maximal frequent itemsets from a transactional database. In the algorithm, the FP-Tree data structure is used and adapted, and a new strategy called "NBN" (Node By Node) is used for traversing the adapted FP-Tree. Neither superset checking nor constructing conditional frequent pattern trees is needed in the algorithm. We analyze the performance of the algorithm and compare our method with existing algorithms. Our technique works better for mining maximal frequent itemsets. It is also proved by experimental comparison that our algorithm is more fast and efficient.

1. Introduction

Mining association rules is a very important problem in the data mining field. Agrawal et al. proposed originally the association rule model, the support-confidence framework and the Apriori algorithm [1], [2]. The Apriori algorithm can find out all frequent itemsets through scanning the transactions database several times. The FP-growth algorithm proposed by Han et al. can find out all frequent patterns with only two times scanning based on a new data structure called "FP-tree" [3]. In many very large databases, the numbers of all frequent itemsets mined by both the Apriori and the FP-growth are very large too. To solving it, Bayardo proposed the concept of MFI (Maximal Frequent Itemsets) and the MaxMiner algorithm for mining MFI especially [4]. Since MaxMiner only looks for the MFIs, the search space can be reduced. MaxMiner performs not only subset infrequency pruning, where a candidate itemsets with an infrequent subset will not be considered, but also a "lookahead" to do superset frequency pruning. MaxMiner needs several passes of the database to find the maximal frequent itemsets.

In [5], Burdick et al. gave an algorithm called MAFIA to mine maximal frequent itemsets. MAFIA uses a linked list to organize all frequent itemsets. Each itemset I corresponds to a bitvector, the length of the bitvector is the number of transactions in the database and a bit is set if its corresponding transaction contains I, otherwise, the bit is not set. Since all information contained in the database is compressed into the bitvectors, mining frequent itemsets and candidate frequent itemset generation can be done by bitvector and-operations. Pruning techniques are also used in the MAFIA algorithm.

GenMax, another depth-first algorithm, proposed by Gouda and Zaki [6], takes an approach called progressive focusing to do maximality testing. This technique, instead of comparing a newly found frequent itemset with all maximal frequent itemsets found so far, maintains a set of local maximal frequent itemsets. The newly found FI is only compared with itemsets in the small set of local maximal frequent itemsets, which reduces the number of subset tests.

In [7], Grahne et al. presented the FPmax algorithm for mining MFIs using the FP-tree structure. FPmax is also a depth-first algorithm. It takes advantage of the FP-tree structure so that only two database scans are needed. In FPmax, a tree structure similar to the FP-tree is used for maximality testing. The experimental results in [7] showed that FPmax outperforms GenMax and MAFIA for many, although not all, cases. In [8], Grahne et al. presented the FPmax* algorithm. They introduced a new technique FP-array on the basis of the FPmax algorithm.

In this paper, we present the MMFI (Mining Maximal Frequent Itemsets) algorithm based on the adapted FP-tree. In the algorithm neither superset checking nor constructing conditional frequent pattern trees recursively is needed with adopting a new traversing method called NBN (Node By Node).

The rest of this paper is organized as follows. In Section 2, we briefly review the FP-Tree data structure and introduce our adapted FP-Tree structure. Section 3

978-0-7695-3151-9/08 $25.00 © 2008 IEEE
DOI 10.1109/ISIP.2008.60

gives algorithm MMFI for mining MFI. Here, we also introduce our approach of traversing tree and discuss the performance of our algorithm. Experimental results are presented in Section 4. Section 5 concludes and outlines directions for future research.

2. Modification to the FP-Tree

2.1. The FP-tree

In [3], Han et al. introduced a novel algorithm, known as the FP-growth method, for mining frequent itemsets. The FP-growth method is a depth-first search algorithm. In the method, a data structure called the FP-tree is used for storing frequency information of the original database in a compressed form. Only two database scans are needed for the algorithm and no candidate generation is required.

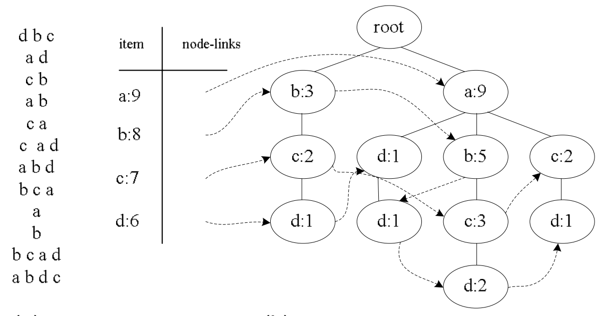

Figure 1. An FP-Tree example

Every node in the FP-tree represent an item, and every path from the root to an offspring node represent an itemset of the transactions database. Support count information of the corresponding itemset is memorized in every node. All frequent items and their support count are stored in an array called "Head Table". All nodes with the same item in the tree are linked, and the head link pointers are stored in the Head Table too. Fig. 1a shows an example of a data set (only contains frequent items) and Fig. 1b shows the FP-tree for that data set. Refer to [3] for more information about the FP-tree and FP-growth method.

2.2. The Ordered FP-Tree

The MMFI algorithm makes full use of the property that any real subset of a MFI is not MFI. After a MFI is found, we know that it's all real subsets are not MFI. We could omit judging these subsets if them were tagged. We adapt the structure of FP-Tree (called ordered FP-Tree) to ensuring that all itemsets are always judged before their subsets, and we can omit superset checking in mining process.

Definition 1. *Let T be a FP-Tree and $I = \{i_1, i_2, ..., i_n\}$ be the set of items in T.header.N_1 and N_2 be two nodes corresponding to different item in I. Say N_1 FIRST N_2 if and only if the position of the item N_1 corresponding to in T.header be under the position of the item N_2 corresponding to in T.header.*

Definition 2. *Let T be a FP-Tree and $I = \{i_1, i_2, ..., i_n\}$ be the set of items in T.header. N_1 and N_2 be two nodes corresponding to the same item in I. Node Q be root or the node which be the ancestor of N_1 and N_2 corresponding item having minimal support count in all ancestor of N_1 and N_2. Node R_1 and R_2 be the child node of Q, and R_1 be N_1 or ancestor of N_1, R_2 be N_2 or ancestor of N_2. Say $N_1 \succ N_2$ if and only if R_1 FIRST R_2.*

Definition 3. *Let T be a FP-Tree and $I = \{i_1, i_2, ..., i_n\}$ be the set of items in T.header. N_1 and N_2 be two nodes corresponding to the same item in I. When N_1 be front of N_2 in the node linked list, if $N1 \succ N_2$ always say T be an ordered FP-Tree.*

Definition 4. *Let T be an ordered FP-Tree and $I = \{i_1, i_2, ..., i_n\}$ be the set of items in T.header. N be a node in T. I_N is the set of items which are corresponded to by the nodes form root to N in T.*

The structure of nodes in an ordered FP-Tree is similar to the structure of nodes in a FP-Tree. Let *p* be a node in an ordered FP-Tree, then there are four fields in *p*. Item name or corresponding item number is stored in the field *item-name*, and the support count of I_p is stored in the field *count*. The pointer to the first child or parent of *p* is stored in the field *ahead*, and the pointer to the next sibling of *p* or the next node corresponding to the same item in the linked list from the head table is stored in the field *next*. For getting the ordered FP-Tree of the transactions database DB, we need do the followings. Scan DB one time to get all the frequent items and insert them to the head table in descending order. Scan DB again, and insert the all frequent items of each transaction in the order of head table to ordered FP-Tree. At this time the field *ahead* stored the pointer to the first child and the field *next* stored the pointer to the next sibling. When a new node is inserted to the ordered FP-Tree the insertion position of the node is not the tail but the proper position of the sibling nodes linked list. After the insertion, each node in the sibling nodes linked list is always *FIRST* the nodes in the left. After all transactions were inserted, traverse the Tree root-first. After the traversal the field *ahead* of each node stored the pointer to its parent and the field *next* of each node stored the pointer to the next node corresponding to the same item in the linked list from the head table.

We do can get the ordered FP-Tree of DB according to the above steps. Because of the limitation of the space we omit the proof. Fig. 2 shows the ordered FP-Tree for the data set in Fig. 1(a).

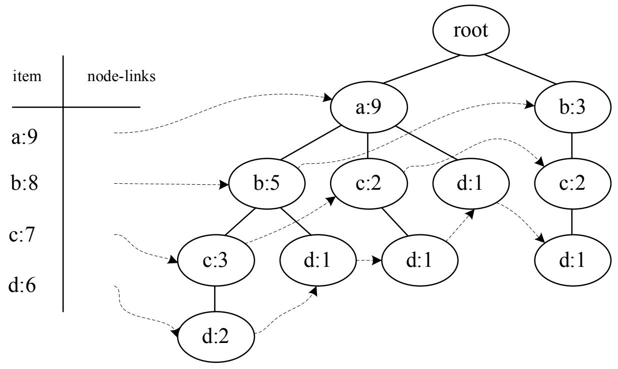

Figure 2. An ordered FP-Tree example

Lemma 1 *Let T be a FP-Tree and $I = \{i_1, i_2, \ldots, i_n\}$ be the set of items in T.header. If i_j be on the above of i_k, the itemsets with the postfix i_k can be the superset, and cannot be the subset of the itemsets with the postfix i_j.*

According to the construction of FP-Tree lemma 1 is obviously right, and for an ordered FP-Tree lemma 1 is right too.

Lemma 2 *Let T be an ordered FP-Tree, N_1 and N_2 be two nodes in T corresponding to the same item. If N_1 is on the left of N_2 in the linked list from the head table, the itemset I_{N1} can be the superset, and cannot be the subset of I_{N2}.*

Proof: according to the construction of the ordered FP-Tree I_{N1} is not the same to I_{N2}, i.e. $(I_{\text{n1}} \subseteq I_{\text{n2}}) \wedge (I_{\text{n2}} \subseteq I_{\text{n1}})$ is impossible. When there is inclusion relation of I_{N1} and I_{N2} it can only be $(I_{\text{n1}} \subset I_{\text{n2}}) \vee (I_{\text{n2}} \subset I_{\text{n1}})$. Because N_1 is on the left of N_2 we know $N_1 \succ N_2$ and R_2 *FIRST* R_1, so there must not be the item R_1 corresponding to in the offspring nodes of R_2. R_1 and R_2 are all the children of Q, so the item R_1 corresponding to must not be in I_{Q}, i.e. the item R_1 corresponding to must not be in I_{N2}. The itemset I_{N1} must not be the subset of I_{N2}, and it must be no inclusion relation of I_{N1} and I_{N2} or $I_{\text{n2}} \subset I_{\text{n1}}$.

3. MMFI Algorithm

3.1. NBN Method

The basic idea of NBN method is to dispose nodes in each "layer" from bottom to up. The concept of "layer" is different to the common concept of layer in a tree. Nodes in a "layer" mean the nodes correspond to the same item and be in a linked list from the head table. For nodes in a "layer" NBN method is apt to dispose them from left to right along the linked list.

To use NBN method, two extra fields are needed to add to each node in the ordered FP-Tree. The field *tag* of node N stores the information of whether I_{N} is maximal frequent itemset, and the field *count'* stores the support count information in the nodes at left.

In Fig. 2, the first node to be disposed is "d:2". If the min_sup is equal to or less than 2 we know $\text{I}_{\text{d:2}}$ (abcd) is a maximal frequent itemset. Firstly we output $\text{I}_{\text{d:2}}$ and set the field *tag* of "c:3" FALSE(the field *tag* of each node is TRUE initially). Next we check whether the right four itemsets $I_{\text{d:1}}$ be the subset of $I_{\text{d:2}}$. If the itemset one node "d:1" corresponding to is the subset of $I_{\text{d:2}}$ we set the field *tag* of the node FALSE. In the following process when the field *tag* of the disposed node is FALSE we can omit the node after the same tagging. If the min_sup is more than 2 we also need to check whether the right four $I_{\text{d:1}}$ is the subset of $I_{\text{d:2}}$. If the itemset one node "d:1" corresponding to is the subset of $I_{\text{d:2}}$ we set the field *count'* of the node with the sum of the former *count'* and 2. In spite of the min_sup, we next dispose the first "d:1" node. After all the nodes "d" disposed we begin to dispose the node "c:3".

By lemma 1 and lemma 2 we can conclude that the NBN method returns all and only the maximal frequent itemsets in the given dataset.

3.2. MMFI

Fig. 3 gives the MMFI algorithm. Line 16 in Fig. 3 is not complete because of space limitation. The full text as follows: If p point to the first child of the root currently set p null, else if p point to the last node of one layer set p to point to the first node of the nearest upper layer, else set p to point to the next node in the same layer linked list.

3.3. Discussion

In the MMFI algorithm, each frequent itemset about to be inserted to *MFS* is maximal frequent itemset without fail. Superset checking is entirely avoided. The algorithm introduces NBN method to traverse the ordered FP-Tree, and construction of conditional frequent pattern trees is avoided accordingly. Though getting the ordered FP-Tree needs more time than getting the FP-Tree, the total time for finding out all maximal frequent itemsets is effectively reduced. Since we introduce two extra fields, it seems that we are using the space to exchange the time. In despite of the space of conditional frequent pattern trees, the two extra fields are not necessary in fact. The necessary information in the field *tag* and *count'* only involve the nodes of current layer at one time. We can use two single dimension arrays to replace the field *tag* and *count'*. We introduced the two fields is just to make the algorithm understandable. Length of the array is equal to the max nodes number of a layer in the ordered FP-Tree, and the number is easy to know in course of constructing the ordered FP-Tree.

```
Procedure MMFI($T$)
Input : T: an ordered FP-Tree
Global: MFS: a set of MFI.
Output: The MFS that contains all MFI.
Methods:
1.  get the pointer $p$ to the first node of the
    bottom layer linked list
2.  While $p \neq$ Null Do{
3.    if $p \rightarrow$tag=F {
4.      for each q in the right list of the same layer
5.      if $q \rightarrow$tag=T and $I_q \subset I_p$  $q \rightarrow$tag=F;
6.  $p \rightarrow$ahead$\rightarrow$tag=F;}
7.  else if $p \rightarrow$count+$p \rightarrow$count'$\geq$min_sup{
8.    for each q in the right list of the same layer
9.      if $q \rightarrow$tag=T and $I_q \subset I_p$  $q \rightarrow$tag=F;
10.   $p \rightarrow$ahead$\rightarrow$tag=F;
11.   $MFS = MFS \cup I_p$;}
12. else{
13.   for each q in the right list of the same layer
14.     if $q \rightarrow$tag=T and $I_q \subset I_p$
15.   $q \rightarrow$count'=$q \rightarrow$count'+$p \rightarrow$count;}
16. $p$=$p \rightarrow$next;}
```

Figure 3. Algorithm MMFI

Searching the right nodes in the same layer linked list of one node to check whether the itemsets they correspond to are the subset of the itemset the node corresponds to (called subset checking) is the primary work in the MMFI algorithm. We adopt the bitvectors method of the MAFIA algorithm, so the work can be done by bitvector and-operations.

4. Experimental results

To see the performance of the ordered FP-Tree and NBN method, we implemented the FPmax* algorithm on the basis of the paper [8]. We ran the programs on real datasets downloaded from [9]. We used datasets *chess* and *mushroom*. The *chess* dataset is compiled from game state information, and the *mushroom* dataset consists of records describing the characteristics of various mushroom species.

Figure 4 shows the experimental results on *chess*, and figure 5 shows the experimental results on *mushroom*. We can draw the conclusion that the MMFI algorithm outperforms the FPmax* algorithms.

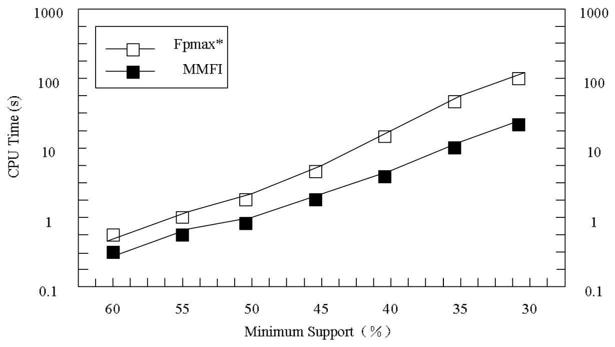

Figure 4. Dataset chess

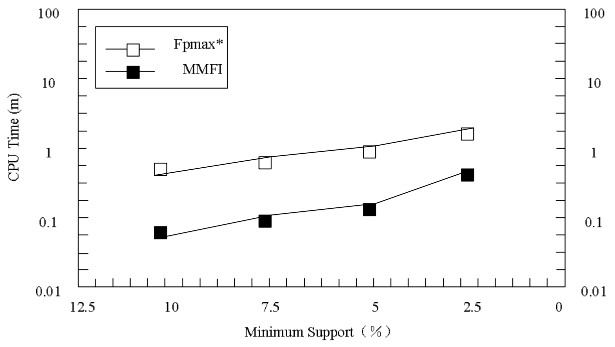

Figure 5. Dataset mushroom

5. Conclusions

In this paper we present the MMFI algorithm for mining maximal frequent itemsets. Neither superset checking nor construction of conditional frequent pattern trees is needed. We are currently investigating techniques to reduce the time of subset checking and adapt the MMFI algorithm to mining frequent itemsets and frequent closed itemsets.

Acknowledgments

This work was supported by the National science Foundation of China (No: 60773049) and the Science Foundation of Jiangsu (No: BK2006073).

Reference

[1] R. Agrawal, T. Imielinski, and A. Swami, "Mining Association Rules between Sets of Items in Large Databases," *Proc. ACM-SIGMOD Int'l Conf. Management of Data*, pp. 207-216, May 1993.

[2] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules," *Proc. Int'l Conf. Very Large Data Bases*, pp. 487-499, Sept. 1994.

[3] J. Han, J. Pei, and Y. Yin, "Mining Frequent Patterns without Candidate Generation," *Proc. ACM-SIGMOD Int'l Conf.*

Management of Data, pp. 1-12, May 2000.

[4] R.J. Bayardo, “Efficiently Mining Long Patterns from Databases,” *Proc. ACM-SIGMOD Int’l Conf. Management of Data*, pp. 85-93,1998.

[5] D. Burdick, M. Calimlim, J. Flannick, J. Gehrke, and T. Yiu, “MAFIA: a maximal frequent itemset algorithm,” *IEEE Transactions on Knowledge and Data Engineering*, pp. 1490–1504, Nov 2005.

[6] K. Gouda, and M.J. Zaki, “Efficiently Mining Maximal Frequent Itemsets,” *Proc. IEEE Int’l Conf. Data Mining*, pp. 163-170, 2001.

[7] G. Grahne and J.F. Zhu, “High Performance Mining of Maximal Frequent Itemsets,” *Proc. SIAM Int’l Conf. High Performance Data Mining*, pp. 135−143, 2003.

[8] G. Grahne, and J.F. Zhu, “Fast Algorithms for Frequent Itemset Mining Using FP-Trees,” IEEE Transactions on Knowledge and Data Engineering, pp. 1347-1362, Oct 2005.

[9]http://www.almaden.ibm.com/cs/people/bayardo/resources.html, 2003.