An improved algorithm for mining maximal frequent itemsets based on FP-tree

CHEN TongQing, YE FeiYue*, GE XiCong, LIU Qi
School of Computer Science and Engineering
Jiangsu Institute of Technology
Changzhou,China
yfy@jsut.edu.cn

***Abstract*—Mining maximal frequent itemsets is an important research topic in data mining. The existing FPMAX algorithm for mining maximal frequent itemsets based on FP-tree necessitate to construct conditional subset trees recursively. While the amount of frequent itemsets is large, it will cause huge consumption both in terms of runtime and memory. Therefore, this paper proposes an improved algorithm FPMAX-direct for mining maximal frequent itemsets based on FP-tree. This novel algorithm will simplify the construction of the conditional subset trees by adding directly the branch of FP-tree whose support is no less than a user-specified minimum support threshold to the maximal frequent candidate itemsets. Experimental results show that FPMAX-direct algorithm outperforms FPMAX algorithm in dense datasets.**

Keywords-maximal frequent itemsets; FP-tree; FPMAX; FPMAX-direct

I. INTRODUCTION

As an important branch of data mining, the association analysis is applied to find some interesting rules from a big scale of dataset. Mining the frequent itemsets is a basic and critical phases of the association analysis. In general, the amount of frequent itemsets is enormous and there is an inclusion relationship among the itemsets, which results in information redundancy. To some extent, maximal frequent itemsets can be used instead of frequent itemsets, because the maximal frequent itemsets encompass all the information on all frequent itemsets. Besides, in some applications of data mining, we only need to mine the maximal frequent itemsets, so how to mining them quickly and effectively has been hot in current research.

Since Agrawal proposes Apriori algorithm[1] to discover association rules in 1993, there have been many researchers putting forward lots of improved approaches based on it, whereas all of them have to scan database multiple times, which cause expensive consumption especially at the time the scale of the dataset is large. To overcome this problem, Han et al.[2] introduce a novel algorithm without candidate generation for mining the frequent itemsets over dataset. This algorithm is characterized by using an efficient and compact data structure named frequent pattern tree(FP-tree in short) to compress the dataset and only scans the database twice avoiding multiple database scans like Apriori algorithm. At present, the existing mining algorithms of the maximal frequent itemsets are mainly modifications based on Apriori or FP-trees. The Apriori-based algorithm mainly include Max-Miner algorithm[3], Pincer-Search[4], DMFI[5]. The algorithms based on FP-tree mainly include FP-MAX[6] and DMFIA[7]. The Apriori-based algorithms usually adopt a series of pruning strategies to prune the search space. However, they all require multiple database scans and will generate large quantities of alternative itemsets that is expensive consumption both in term of runtime and memory. To avoid the shortcomings of the Apriori-based algorithms, the algorithms for mining maximal itemsets over big dataset are mainly based on FP-tree.

The algorithm proposed in this paper is also based on the data structure FP-tree. After research on FP-MAX algorithm, we find that this algorithm needs traversal in the header table to obtain the conditional pattern bases, and then builds the conditional FP-trees. While the dimension of the dataset is large, constructing conditional FP-trees will cause memory overflow and cost much time. Therefore, this paper proposes the FPMAX-direct algorithm to overcome those drawbacks to a certain extent, which makes use of some strategies as follows: adding directly the itemsets whose support is no less than the predefined minimum support threshold to MFCS-tree, only using the others to build the conditional FP-trees during traversing the corresponding prefix paths according to the header table, and besides adding a mark for each node of the FP-tree. Those can simplify the conditional FP-trees constructing and superset checking. At last, we evaluate the good performance of the FPMAX-direct algorithm by comparing the performance of FPMAX-direct against FPMAX on several datasets having various characteristics.

II. RELATED KONWLEDGE

A. Frequent itemsets and maximum frequent itemsets

Let $I=\{i_1,i_2,..,i_m\}$ be a set of different items with the quantity of m and $DB=\{T_1,T_2,..,T_n\}$ be a transactional database, where T_i is a transaction which contains a set of items in I. Assume the amount of transactions in a transaction database DB is N and itemset X is a subset of $I(X \subset I)$, we can define $count(X)$ as the numbers of transactions containing X in DB, thus the support of X can be calculated by $count(X)/N$.

Definition 1. Given a minimum support threshold $minSup$ which is the occurrence frequency in the database, if $support(X) \geq minSup$, the itemset X is defined as frequent itemsets. Meanwhile, the minimum support threshold number $minCount = minSup \times N$.

Definition 2. For any frequent itemset X, if there is no frequent itemset $Y(Y \subseteq I)$ such that $X \subset Y$, the itemset X is called as the maximal frequent itemset.

Property 1. All the proper subsets of any maximal frequent itemset are not the maximal frequent itemset.

Property 2. All the subsets of any frequent itemset are frequent.

978-1-5386-7445-1/18/$31.00 ©2018 IEEE
DOI 10.1109/DCABES.2018.00065

B. *Fp-tree and FPMAX Algorithm*

FP-tree proposed by Han in 2000 is a compact data structure which can store essential information about the frequent itemsets of a transactional database. In the FP-tree, the itemsets consisting of a set of nodes of each path from the root to each descendant is corresponding to one transactional set of the transactional database. To ensure that the tree structure is compact and informative, only frequent length-1 items will have nodes , each of which is associated with a frequent item in the tree, and the tree nodes are arranged as some sorted order of frequent items. Nodes with the same item-name are linked in sequence via such node-links.

FPMAX algorithm is an extended algorithm developed to discover maximal frequent itemsets based on the compact data structure FP-tree and FP-growth algorithm. To mine the maximal frequent itemsets, this algorithm requires to add every item in the header table into the set of head items(namely Head) according to the occurrence frequency in the database as the initial suffix and gather the corresponding conditional bases to construct the sub-FP-trees. In order to prune the process, it is necessary to check itemsets consisting of the header table of the sub-FP-tree union the suffix items whether is the subset of the maximal frequent itemsets which have been mined. If the answer is yes, skip the construction of this conditional FP-tree. Otherwise, we should gather the conditional bases to construct the conditional FP-tree and mine the sub-FP-tree till there is only single path in the sub-FP-tree. Then the itemsets consisting of the current header table of the sub-FP-tree union the corresponding suffix items are the maximal frequent candidate itemsets. Thus this method employs a MFI-tree-like data structure FP-tree to store the maximal frequent itemsets and check the candidates to identify the final maximal frequent itemsets. More details about FPM-AX algorithm can be referred to paper[6].

III. FPMAX-DIRECT ALGORITHM BASED ON FP-TREE

A. *Description of the algorithm*

Similar to FPMAX algorithm**,** FPMAX-direct algorithm also need to facilitate bottom-up-depth-first traversal. FPMAX algorithm requires to traverse the header table in item's frequency descending order to collect each item and then search the current FP-tree to gather the conditional pattern base of the item to construct the corresponding sub-FP-trees. This algorithm constructs the sub-FP-trees recursively without making the best use of some pruning strategies. Our algorithm FP-MAX-direct add one field called mark for each node in the FP-tree where mark registers whether this node has been added into the MFSC, that can prune the search space. We examine the mining process by starting from the bottom of the node-link header table. While following a node $a_i's$ node-links to collect the conditional bases, if the mark of the node is 1 and the count of the node is more than the predefined minimum support threshold, then we add directly the itemset consisting of all the items of the path from the root to the current node into the set named MFSC, Meanwhile, if the mark of the node is 1 and the count of the node is less than the threshold, we gather them for the next phrase to construct the conditional FP-trees. By employing those, we can simply the construction of the sub-FP-trees, consequently the runtime will be reduced largely.

B. *The algorithm flew*

Algorithm 1: Mining Maximal frequent candidate itemsets

Input: FP-tree(T) created from the transaction database D with all nodes of the tree with the field of the mark and predefined minimum support s.

Output: a dictionary storing the maximal frequent candidate itemset

```
Procedure MFSCALL(preFix,HeaderTable,T,s)
{
 (1)for item in T.HeaderTable.keys()
 (2)  MFSC,ConPatBase=FinMFSC_ConPatBase(pre-
Fix,HeaderTable[item][1],s)
 (3)  if MFSC!=None
 (4)     for each m in MFSC
 (5)          if subsetChecking(m,MFSC-tree) is false
 (6)              updateMFSC_tree(m)
 (7)              add m into MFSCALL
 (8)  if ConPatBase!=None
 (9)     Construct item's conditional FP-tree T_i and
initialize T_i
 (10)    MFSCALL(prefix,Header,T_i,s)
 }
```

Algorithm 2: gathering the conditional pattern bases and the MFSC

Input: the current node of the tree, predefined support threshold s and the suffix.

Output: the MFSC and the conditional pattern bases

```
Procedure FinMFSC_ConPatBase(prefix,treeNode,s)
{
 (1)  While(treeNode!=None)
 (2)  { if treeNode.count>s and treeNode.mark==1
 (3)       FindPrefixPath and treeNode.mark=0
 (4)       checkMFSC(prefixPath)
 (5)       add prefixPath into MFSC
 (6)   else if treeNode.mark==1 and treeNode.count<s
 (7)        FindPrefixPath and add PrefixPath to
       ConPatBase
 (8)  Return MFSC,ConPatBase
 (9) }}
```

Algorithm 3: identifying the final maximal frequent itemsets

Input: the maximal frequent candidate itemsets

Output: the maximal frequent itemsets

```
Procedure MFS(MFSCALL)
{
//MFSCALL is the dictionary
(1)  For key,item in MFSCALL
(2)     Deal the item which is subset of others in
      MFSCALL[key]
(3)
          Insert MFSCALL[key] into MFS
(4)  Return MFS
(5)  }
```

//FindPrefixPath is similar to the algorithm used to find the prefix path of one node in FPMAX algorithm.

//subsetChecking is similar to superset-checking function in FPMAX algorithm.

IV. ANALYSIS OF AN INSTANCE

A small transaction database is given in the Table 1, and the corresponding FP-tree of this database is given in Fig. 1 when the minimum support threshold is 0.3.

TABLE I. A TRANSACTION DATABASE

Table Head	Table Column Head		
	TID	***Itemset***	***Frequent length-1 itemsets***
	T001	1,5,7,8	1,8,7
	T002	1,2,5,7	1,2,7
	T003	1,2,8	1,8,2
	T004	1,2,7,8	1,8,2,7
	T005	1,3,6	1
	T006	1,8,9	1,8
	T007	1,2,8	1,8,2

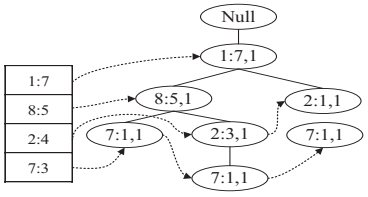

Figure 1. FP-tree with mark node

The details of mining the maximal frequent itemsets over the database given in table 1 with FPMAX-direct is as following:

(1) According the descending order of the occurrence frequency of the items in the header table, we mine the maximal frequent itemsets ending with 7 at first. As we can see from Fig 1, the marks of all the nodes of which item-name is 7 are all 1 and there are three prefix paths of the node with item-name being 7, besides the counts of two nodes are 1, less than the support threshold. Therefore, we build the sub-FP-tree for item 7 based on those three paths and the sub-FP-tree is shown in Fig 2. In the next phase, we should mine the sub-FP-tree in the Fig 2. We can find there is only one node with item-name being 1, its mark is 1 and its count is 3, more than the support threshold. So we should add $\{1,7\}$ into MFCS and meanwhile update the MFCS-tree, then set the dictionary $MFCSALL = \{6:[[1,6]]\}$, Fig 3 shows the updated MFCS-tree

Figure 2. The conditional FP-tree of the item

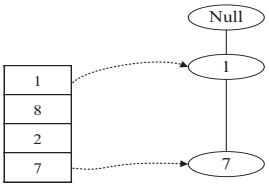

Figure 3. The updated MFCS-tree

(2) In the following phase, we should process the path ending with the item 2. As is shown from Fig 1, there are two nodes with item-name being 2 and mark being 1. Starting from the tree nodes, we can gather three sets: $\{1,8,2\}:3$, $\{1,2\}:1$. The support of set $\{1,8,2\}$ is 3, more than the minimum support threshold. So we can add this set into MFSC and set the mark of the nodes to 0, and update the MFSC-tree, then finally set the set $MFSCALL = \{7:[[1,7]],5:[[1,8,2]]\}$. The set $\{1,2\}$ of which support is less than the support threshold will be used to construct the sub-FP-tree. Because there is only one set with low support, we cannot build the conditional FP-tree and we will get the frequent set $\{1,2\}$ as the maximal frequent candidate itemsets, then use MFSC-tree for supersets checking and update MFSC-tree and MFSCALL. Fig 4 shows the updated MFSC-tree.

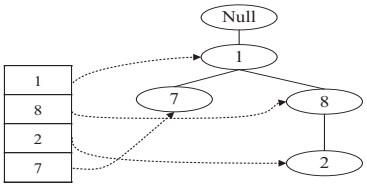

Figure 4. The updated MFCS-tree

(3) Similarly, Find the maximal frequent candidate itemsets ending with 8,1 in turn. Finally, we will get MFS$CALL = \{7: [[1, 6]], 8: [[1, 8,2]]\}$.

(4) At last, we only need to process the dictionary to select the maximal frequent itemsets according to the key of the dictionary. We get MFS=[[1, 7][1,8,2]] .

V. ANALYSIS AND COMPARISON

In order to verify the outperformance of FPMAX-direct compared with FPMAX. The experiment environment is Intel(R) Core(TM) i7-4790 CPU 3.6GHz, 8.00GB RAM and the operating system of Windows 10. The program is realized by Python. In the testing, we employ four datasets having various characteristics, two of them are mushroom and chess, dense public transaction dataset, and the rest are F1 and F2 sparse synthetic dataset. More details about those datasets are listed in Table 2.

TABLE II. CHARACTERISTICS OF THE DATASETS

Table Head	Table Column Head		
	Average length of a transaction	*Numbers of transaction*	*Numbers of item*
Mush-room[a]	23	8124	120
Chess	37	3196	76
F1	5	60798	7360
F2	5	60810	17910

In this paper, we carry out the comparison experiments of the two algorithms under the conditions of different minimum supports and datasets. Fig 5 shows the comparison of the runtime of the two algorithms in the condition of different minimum supports on the dataset mushroom. Fig 6 shows the comparison on the dataset chess. Fig 7 shows the comparison on the dataset F1. Fig 8 shows the comparison on the dataset F2.

Obviously, from the results, we can conclude that FPMAX-direct algorithm outperforms against FPMAX algorithm for mining the maximal frequent itemsets, especially on the dense dataset.

VI. CONCLUSION

This paper proposes a novel algorithm FPMAX-direct for mining the maximal frequent itemsets based on FP-tree. This algorithm employs some strategies that are adding filed of mark and branch-checking to simply the construction of the sub-FP-trees, which can improve the efficiency of mining the maximal frequent itemsets on dense datasets. The results of the experiments prove the outperformance of FPMAX-direct compared with FPMAX on dense datasets or under the condition of low support. Therefore, this novel algorithm can be applied in data mining task over dense datasets.

ACKNOWLEDGMENT

This work was supported by the National Natural Science Foundation of China under Grant Nos. 61472166.

REFERENCES

[1] Imielinski T, Swami A, Agrawal R. Mining association rules between sets of items in large database[C]. In Proceedings of 1993 ACM SIGMOD conference on management of data.New York: ACM, 1993: 207-216.

[2] Han J, Pei J, Yin Y. Mining frequent patterns without candidate generation[C]. In Proceedings of the 2000 ACM-SIGMOD international conference on management of data.New York: ACM, 2000: 1-12.

[3] Hur J, Noh D K. Attribute-Based Access Control with Efficient Revocation in Data Outsourcing Systems[J]. IEEE Transactions on Parallel and Distributed Systems, 2011, 22(7).

[4] Lin D, Kedem Z. Pincer-search: a new algorithm for discovering the maximum frequent set[C]. In Proceedings of the 6th European conference on extending database technology, Valencia, Spain, 1998: 432-444.

[5] Lu Songfeng, Lu Zhengding. Fast Mining Maximum Frequent Itemsets[J]. Journal of Software, 2001, 12(2): 293-297.

[6] Grahne G, Zhu J. High performance mining of maximal frequent itemsets[C]. In Proceedings of the 6th International Workshop on High Performance Data Mining, 2003: 1-10.

[7] Song Yuqing, Zhu Yuquan, Sun Zhihui, Chen Geng. An Algorithm and Its Updating Algorithm Based on FP-Tree for Mining Maximum Frequent Itemsets[J]. Journal of Software, 2003, 14(09): 1586-1592.

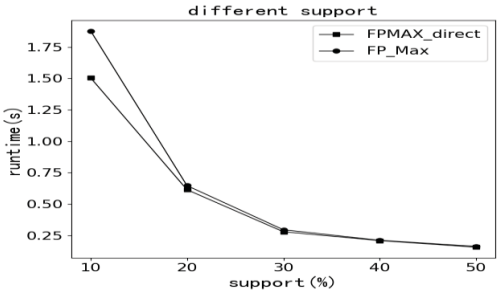

Figure 5 comparison on mushroom

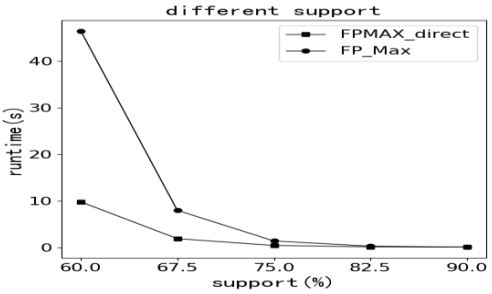

Figure 6 comparison on chess

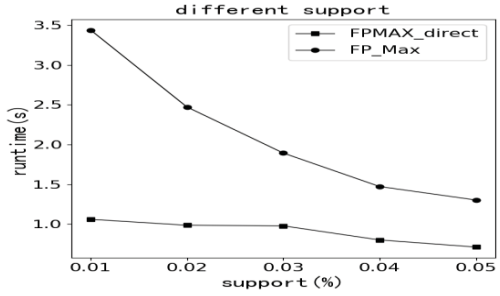

Figure 7 comparison on F1

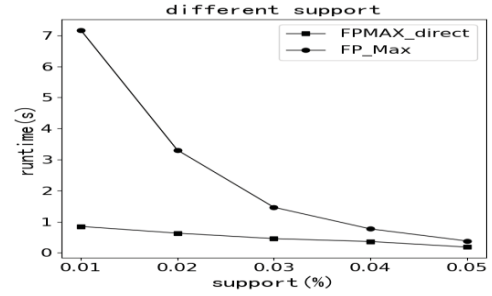

Figure 8 comparison on F2