Frequent Closed Pattern Mining Algorithm Based on COFI-Tree

Jihai Xiao[1], Xiaohong Cui[1], and Junjie Chen[2]

[1] College of Textile Engineering and Art, Taiyuan University of Technology, 030600 JinZhong, China
[2] College of Computer Science and Technology, Taiyuan University of Technology, 030024 TaiYuan, China
{xiaojihai,tycuixiaohong}@126.com, chenjj@tyut.edu.cn

Abstract. This paper proposes a frequent closed itemsets mining algorithm based on FP-tree and COFI-tree. This algorithm adopts a relatively small independent tree called COFI-tree. COFI-Tree doesn't need to construct conditional FP-Tree recursively and there is only one COFI-Tree in memory at a time, therefore this new mining algorithm reduces memory usage. Experiment shows that the new approach outperforms similar state-of-the-art algorithms when mining extremely large datasets in terms of execution time.

Keywords: Frequent Closed Itemsets, FP-Tree, COFI-Tree.

1 Introduction

Frequent pattern mining plays an important role in data mining. Now there are many mining algorithms about frequent closed itemsets, such as, CLOSET[1]. Through our study, we find this algorithm have some problems, it builds conditional FP-Tree recursively so that it requires more memory and CPU resources. In this paper, an effective frequent closed pattern mining algorithm is advanced, and it based on FP-Tree and COFI-Tree.

2 Related Work

There are many algorithms to address the problem of mining association rules [2], [6]. One of the important algorithms is the Apriori algorithm [6]. It also is the foundation of other most known algorithms. However, when mining extremely large datasets, the Apriori algorithm still suffers from two main problems of repeated I/O scanning and high computational cost. Another innovative approach of discovering frequent patterns, FP-Growth, was presented by Han et al. in [2]. It creates a compact tree-structure, FP-Tree, representing frequent patterns, that reduces the multi-scan times and improves the candidate itemset generation. The authors of FP-Tree algorithm have validated that their algorithm is faster than the Apriori. However, It needs to

H. Deng et al. (Eds.): AICI 2011, CCIS 237, pp. 175–182, 2011.

construct conditional FP-Tree recursively. This massive creation of conditional trees makes FP-Tree algorithm not scalable to mine large datasets beyond few millions. The COFI-tree[3](Co-Occurrence Frequent Item Tree) algorithm that we are presenting in this paper is based on the core idea of the FP-Growth algorithm proposed by Han et al. in [2].

3 Construction of FP-Tree

A FP-Tree[2] is constructed by scanning database twice. First, a scan of database derives a list of frequent items, Second, the FP-Tree is constructed.

Definition 1 (FP-tree). A frequent-pattern tree is a tree structure

1. It consists of one root labeled as "null", a set of item-prefix subtrees as the children of the root, and a frequent-item-header table.
2. Each node in the item-prefix subtree consists of three fields: item-name, count, and node-link, where item-name registers which item this node represents, count registers the number of transactions represented by the portion of the path reaching this node, and node-link links to the next node in the FP-tree carrying the same item-name, or null if there is none.
3. Each entry in the frequent-item-header table consists of two fields, (1) item-name and (2) head of node-link (a pointer pointing to the first node in the FP-tree carrying the item-name.

Algorithm1 (Construction of FP-tree)
Input: A transaction database DB and a minimum support threshold ξ.
Output: FP-tree.
Method: The FP-tree is constructed as follows.

1. Scan the transaction database DB once. Collect F, the set of frequent items, and the support of each frequent item. Sort F in support-descending order as L, the list of frequent items.
2. Create the root of an FP-tree, T, and label it as "null". For each transaction in DB do the following.

Select the frequent items in transaction and sort them according to the order of L. Let the sorted frequent-item list in transaction be [p | P], where p is the first element and P is the remaining list. Call insert-tree ([p | P], T).

The function insert tree ([p | P], T) is performed as follows.

If T has a child N such that N.item-name = p.item-name, then increment N's count by 1; else create a new node N, with its count initialized to 1, its parent link linked to T , and its node-link linked to the nodes with the same item-name via the node-link structure. If P is nonempty, call insert-tree(P, N) recursively.

To understand effortlessly, let's make an example.

Table 1. A transaction database (set the minimum support threshold as 3)

T1	T2	T3	T4
DE	ABDFEH	AGDECB	AGDFEI
T5	T6	T7	T8
AGEBFC	ADHEF	AGDBLC	ABCF
T9	T10	T11	
ADBCG	AFBCE	ABCH	

First, a scan of DB collects F , the set of frequent item and the support of each frequent item. F={(A:10),(C:7),(D:7),(E:7),(F:6),(B:8),(G:5),(H:3)},Sort F in support-descending order as L, the list of frequent items, L={(A:10),(B:8),(C:7),(D:7),(E:7),(F:6),(G:5),(H:3)}.

Second, construction FP-Tree according to Algorithm 1, in the end, FP-Tree is constructed as Fig. 1.

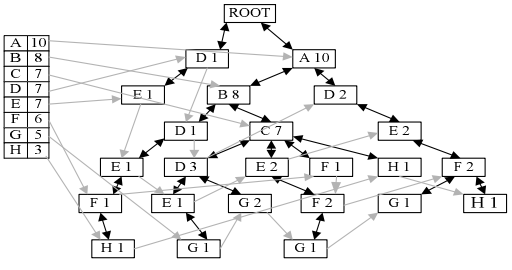

Fig. 1. FP-Tree

4 Construction of COFI-Tree

The COFI-trees are similar to the FP-Trees in general. However, the COFI-trees have bidirectional links in the tree allowing bottom-up scanning as well, and the nodes contain not only the item-name and a frequency counter, but also a participation counter. The COFI-tree for a given frequent item X contains only nodes labeled with items that are more frequent than or as frequent as X. The COFI-trees of all frequent items are not constructed together. Each tree is built, mined, and then discarded before the next COFI-tree is built. So at any given time, only one COFI-Tree is present in main memory.

Algorithm2 (construction of COFI-Tree)

Input: FP-Tree
Output: COFI-Tree
Method: The COFI-Tree is constructed as follows

1.Construct COFI-Tree for X. All branches can be found by following the chain of item X in the FP-Tree structure, and the support of each branch is equal to the support of the X node in its corresponding branch in FP-Tree.
2.The X-COFI-tree starts with the root node X. For each branch containing X do the following:

If multiple frequent items share the same prefix, they are merged into one branch and a counter for each node of the tree is adjusted accordingly, else a new branch is formed starting from the root node X and the support of this branch is equal to the support of the X node in its corresponding branch in FP-Tree. In each node, participation count is initialized to 0 and is used by the mining algorithm discussed later. Last two kinds of link are formed: a horizontal link which points to the next node that has the same item-name in the tree, and a bidirectional vertical link that links a child node with its parent and a parent with its child.

Fig.2. illustrates G- COFI-trees for frequent items of Fig.1. In Fig.2. the rectangle nodes are nodes from the tree with an item label and two counters. The first counter is a support-count for that node while the second counter, called participation-count, is initialized to 0.

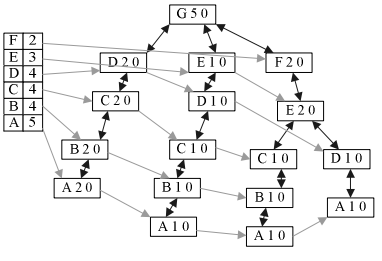

Fig. 2. G-COFI-Tree

4.1 Mining the COFI-Tree

Algorithm3 (Mining the COFI-Tree)

Input: COFI-Tree
Output: Candidate frequent patterns
Method: COFI-Tree is mined as follows:
For each frequent item in the header table of the COFI-Tree do as following:

Some branches that contain frequent item are found in the COFI-Tree, and the branch-support of each branch is the frequency of the first item in the branch minus the participation value of the same node. At the same time, the participation values for all nodes in this branch are incremented by the frequency of each branch.

For example, The G-COFI-tree in Fig.2. is mined for candidate frequent patterns. The process starts from the most globally frequent item, which is A, and then traverse all the A nodes if the support is greater than participation, the third counter on the node, then the complete path from this node to the COFI-root is built with branch-support equals to the difference between the support and participation of that node.

All values of participation for all nodes in these paths are updated with the participation of the original node . Candidate frequent patterns (A, B, C, D, G: 2), (A, B, C, E,G: 1), (A, D, E,G: 1), and (A, B, C, D, E,G: 1) are generated from this tree.

Steps to produce frequent patterns related to the G item for example, are illustrated in Fig.3.

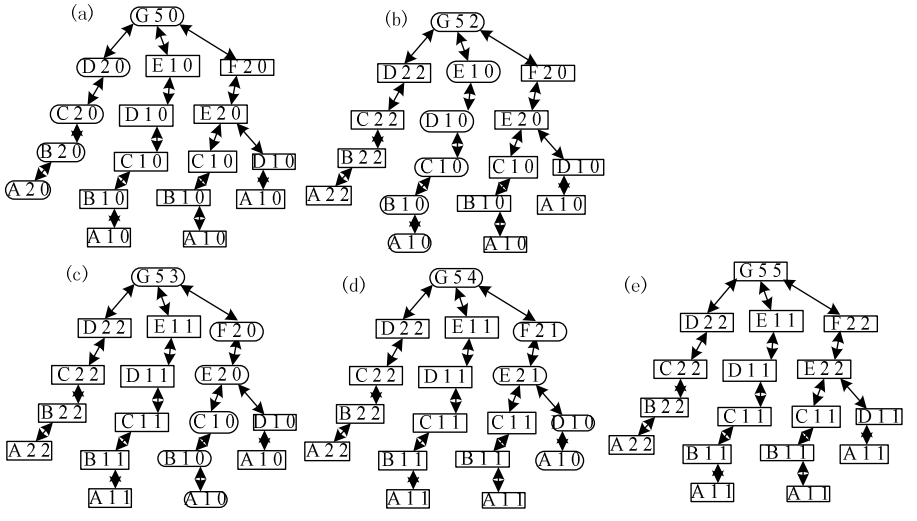

Fig. 3. The Process of finding candidate frequent patterns from G-COFI-Tree

5 Frequent Closed Itemsets

Definition 2 (Frequent closed itemsets)
An itemset Y is a frequent closed itemset if it is frequent and there exists no proper superset Y' $\supset$ Y such that sup(Y') =sup(Y).

An array of pointer that has a size equal to the length of the largest candidate frequent patterns is adopted to derive frequent closed itemsets. Each node of the connected link list is made of 4 variables which are: the pattern, a pointer to the next node, and two number variables that represent the support and branch-support of this pattern. The support records the number of times this pattern occurs in the database. The branch-support records the number of times this pattern occurs alone without other frequent items.

Algorithm 4 (Frequent closed itemsets)

Input: Candidate frequent patterns
Output: Frequent closed itemsets
Method: Steps as following:

1. Candidate frequent patterns is stored into an array of pointer, and set support is equal to branch-support.
2. Intersect each one of these patterns with all other candidate frequent patterns to get a set of potential candidates, and potential candidates are stored into array of pointer, and support and branch-support are equal to 0.
3. Count the support of each generated patterns. The support of each one of them is the summation of supports of all its supersets of candidate frequent patterns.
4. Scan these patterns to remove non-frequent ones or frequent ones that already have a frequent superset with the same support.

As an example, candidate frequent patterns in section3.1 are mined for frequent closed pattern. Candidate frequent patterns (A, B, C, D,G: 2), (A, B, C, E,G: 1), (A, D, E,G: 1), and (A, B, C, D, E,G: 1) are generated from G-COFI-Tree. According to algorithm 4, itemsets is changed into (ABCDEG:1,1), (ABCDG:2,2), (ABCEG:1,1), (ADEG:1,1) after step 1. Itemsets is changed into (ABCDEG:1,1), (ABCDG:2,2), (ABCDG:0,0), (ABCEG:1,1), (ABCEG:0,0), (ADEG:1,1), (ADEG:0,0), (ABCG:0,0), (ADG:0,0), (AEG:0,0) after step 2. Itemsets is changed into (ABCDEG:1,1), (ABCDG:2,2+1), (ABCDG:0,0+1+2), (ABCEG:1,1+1), (ABCEG:0,0+1+1), (ADEG:1,1+1), (ADEG:0,0+1+1), (ABCG:0,0+1+2+0+1+0), (ADG:0,0+1+2+0+1+0), (AEG:0,0+1+1+0+1+0) after step 3 In step 4, itemsets (ABCDEG:1,1) (ABCEG:1,2), (ABCEG:0,2) (ADEG:1,2) (ADEG:0,2), (ABCDG:0,3) are discarded. At last, frequent closed patterns (ABCDG), (ABCG), (ADG), (AEG) are generated and added to the pool of closed patterns. The G-COFI-tree and its array of pointers are cleared from memory as there is no need for them any more. The same process repeats with the remaining COFI-trees for F and E, where any newly discovered frequent closed pattern is added to the global pool of closed patterns.

5.1 Frequent Closed Pattern Mining Algorithm Based on COFI-Tree

Algorithm 5 (Frequent closed pattern mining algorithm based on COFI-Tree)

Input: A transactional database
Output: Frequent closed pattern
Method:

1.construction of FP-Tree
2.construction of COFI-Tree
3.mining COFI-Tree to find candidate frequent patterns
4.Travel candidate frequent patterns to derive frequent closed pattern.

6 Experiment

Some papers[4] had validated that FP-closed is better than CLOSET, so here a performance study is presented to evaluate the new approach against the state-of-art algorithm that mine closed patterns which is FP-Closed. The experiment is conducted on an HP Intel Core i5-2300 CPU 2.80GHz with 4GB memory. Experiment database is T10I4D100K[5].

6.1 Compare with FP-Closed

Experiment analyzing contains two aspects: First, analyze performance of two algorithms at spatial complexity by changing the support of transaction database. The comparing result is shown in Fig.4. Second, analyze performance of two algorithms at time complexity by changing the size of transaction database. The comparing result is shown in Fig.5.

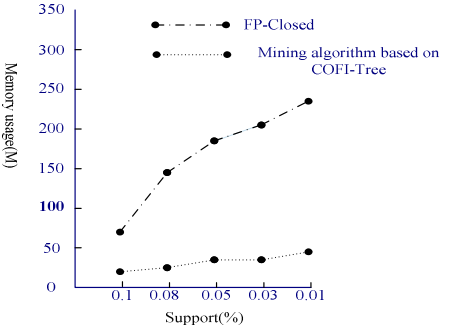

Fig. 4. Spatial complexity of two algorithms

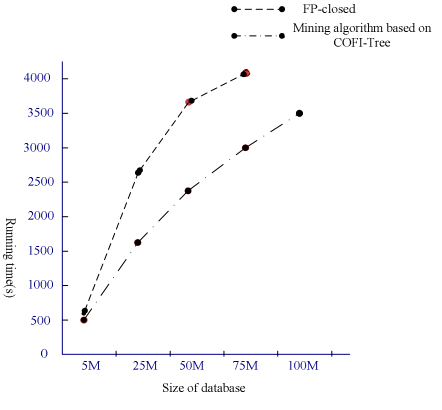

Fig. 5. Two complexity of two algorithms

6.2 Performance Evaluation

Fig.4. reveals the applicability and scalability of the COFI-tree algorithm. It shows that memory usages of FP-closed algorithm tends to grow exponentially when support threshold reduces from 0.1% to 0.01%. However, as support threshold reduces smaller and smaller, there is subtle change in memory usages of COFI-tree algorithm.

From Fig.5, it is easy to notice that COFI-tree algorithm has better performance for large data set. It is because this algorithm is no need to construct conditional sub-trees recursively. The experiments we conducted showed that our algorithm is scalable to mine tens of millions of transactions.

7 Conclusion

In this paper, a frequent closed pattern mining algorithm based on COFI-Tree is advanced. In this algorithm, COFI-Tree is adopted. COFI-Tree doesn't need to build conditional FP-Tree recursively and there is only one COFI-Tree in memory at a time, therefore this new mining algorithm reduces memory usage. The result of experiment shows that this algorithm is better than others on running time.

References

1. Pei, J., Han, J., Mao, R.: An efficient algorithm for mining frequent closed itemsets. In: Gunopulos, D., et al. (eds.) Proc. of the 2000 ACM SIGMOD Int'l. Workshop on Data Mining and Knowledge Discovery. ACM Press, Dallas (2000)
2. Han, J., Pei, J., Yin, Y.: Mining frequent patterns without candidate generation. In: 2000 ACM SIGMOD Intl. Conference on Management of Data, pp. 1–12 (2000)
3. El-Hajj, M., Zaiane, O.R.: Non recursive generation of frequent k-itemsets from frequent pattern tree representations. In: Proc. of 5th International Conference on Data Warehousing and Knowledge Discovery, DaWak 2003 (2003)
4. Leung, C.K.-S., Mateo, M.A.F., Brajczuk, D.A.: A Tree-Based Approach for Frequent Pattern Mining from Uncertain Data. In: Washio, T., Suzuki, E., Ting, K.M., Inokuchi, A. (eds.) PAKDD 2008. LNCS (LNAI), vol. 5012, pp. 653–661. Springer, Heidelberg (2008)
5. `http://fimi.cs.helsinki.fi/data/`
6. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. In: Proc.1994 Int. Conf. Very Large Data Bases, Santiago, Chile, pp. 487–499 (1994)