Finding All Frequent Patterns Starting from the Closure*

Mohammad El-Hajj and Osmar R. Zaïane

Department of Computing Science,
University of Alberta, Edmonton AB, Canada
{mohammad, zaiane}@cs.ualberta.ca

Abstract. Efficient discovery of frequent patterns from large databases is an active research area in data mining with broad applications in industry and deep implications in many areas of data mining. Although many efficient frequent-pattern mining techniques have been developed in the last decade, most of them assume relatively small databases, leaving extremely large but realistic datasets out of reach. A practical and appealing direction is to mine for closed itemsets. These are subsets of all frequent patterns but good representatives since they eliminate what is known as redundant patterns. In this paper we introduce an algorithm to discover closed frequent patterns efficiently in extremely large datasets. Our implementation shows that our approach outperforms similar state-of-the-art algorithms when mining extremely large datasets by at least one order of magnitude in terms of both execution time and memory usage.

1 Introduction

Discovering frequent patterns is a fundamental problem in data mining. Many efficient algorithms have been published on this problem in the last 10 years. Most of the existing methods operate on databases made of comparatively small database sizes. Given different small datasets with different characteristics, it is difficult to say which approach would be a winner. Moreover, on the same dataset with different support thresholds different winners could be proclaimed. Difference in performance becomes clear only when dealing with very large datasets. Novel algorithms, otherwise victorious with small and medium datasets, can perform poorly with extremely large datasets. The question that we ask in this work is whether it is possible to mine efficiently for frequent itemsets in extremely large transactional databases, databases in the order of millions of transactions and thousands of items such as those for big stores and companies similar to Wal-Mart, UPS, etc. With the billions of radio-frequency identification chips (RFID) expected to be used to track and access every single product sold in the market, the sizes of transactional databases will be overwhelming even to current

* This research is partially supported by a research grant from the National Sciences and Engineering Research Council of Canada.

X. Li, S. Wang, and Z.Y. Dong (Eds.): ADMA 2004, LNAI 3584, pp. 67–74, 2005.

state-of-the-art algorithms. There is obviously a chasm between what we can mine today and what needs to be mined. It is true that new attempts toward solving such problems are made by finding the set of frequent closed itemsets (FCI) [6, 7, 8]. A frequent itemset X is closed if and only if there is no X' such that $X \subseteq X'$ and the support of X equals to the support of X'.

Finding only the closed item patterns reduces dramatically the size of the results set without loosing relevant information. Closed itemsets reduce indeed the redundancy already in the set of frequent itemsets. From the closed itemsets one can derive all frequent itemsets and their counts. Directly discovering or enumerating closed itemsets can lead to huge time saving during the mining process.

While there are myriad algorithms to discover the closed patterns, their performances are indistinguishable for small and medium size databases. Experimental results are typically reported with few hundred thousand transactions. A recent study [9] showns that with real datasets, *Apriori*, the oldest algorithm for mining frequent itemsets, outperforms the newer approaches. Moreover, when results are discovered in few seconds, performance becomes almost irrelevant. The problem of performance becomes a real issue when the size of the database increases significantly (in the order of millions of transactions) or when the dimensionality of the problem increases (i.e. the number of distinct items in the database).

We present in this paper a new algorithm for discovering closed frequent itemsets, and report on a study illustrating the importance of such algorithm when mining very large databases.

The remainder of this paper is organized as follows: To put our algorithm in the context, we explain our new traversal approach in Section 2. Since we adopt some data-structures from the literature, FP-tree and COFI-trees, we briefly describe them in Section 3 where the new COFI-closed algorithm is also explained with illustrative examples. Section 4 depicts the performance evaluation of this new algorithm comparing it with existing state-of-the-art algorithms in particular for its speed, scalability, and memory usage on dense and sparse data. Section 5 concludes and highlights our observations.

2 Leap-Traversal Approach

In this paper we introduce a new leap-traversal approach that looks ahead at the nature of the transactional database, and suggests a set of patterns from different sizes to test where the frequent patterns (all, closed, or maximals) are subset of this suggested set. To illustrate the traversal, we take the case of closed itemsets. Step one of this approach is to look at the nature of the distribution of frequent items in the transactional database. Figure 1.A presents a transactional database where we can see that there are only 4 distributions of frequent items. Indeed, {A, B, C, D, E, F, G, H, I} occurs 3 times, we call this as branch-support of 3; {A, B, C, D, E, J, K, L} occurs also 3 times; {A, F, G, H, I} occurs twice; and {A, J, K, L} also occurs twice. We call each one of these

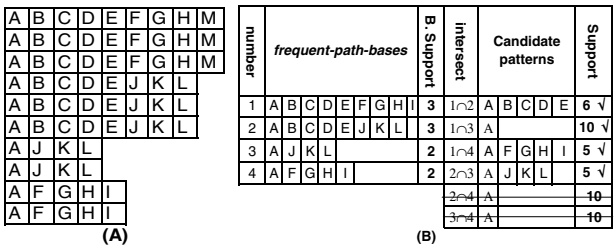

Fig. 1. (A): transactional database. (B): Steps needed to generate closed patterns using the leap-traversal approach ($\surd$ indicates a discovered closed pattern. Barred entries are the eliminated candidates)

patterns a *frequent-path-base*. Step 2 of this process intersects each one of these patterns with all other *frequent-path-bases* to get a set of potential candidates. Step 3 counts the support of each one of the generated patterns. The support of each one of them is the summation of supports of all its supersets of *frequent-path-base* patterns. Step 4 scans these patterns to remove non-frequent ones or frequent ones that already have a frequent superset with the same support. The remaining patterns can be declared as closed patterns. Figure 1.B illustrates the steps needed to generate the closed patterns of our example from Figure 1.A. The major goals of this approach are the followings: 1. Avoid the redundancy of testing patterns either from size 1 until patterns of size k, where k is the size of the longest frequent pattern or from patterns of size n until patterns of size k, where n is the size of the longest candidate pattern. 2. We only check the longest potential patterns that already exist in the transactional database, even if they are of different lengths. From Figure 1.A we can find that there is no need to test patterns such as ABJ or AFC since they never occur in the transactional database. We also do not need to test patterns such as AB since they never occur alone without any other frequent items in the transactional databases.

The main question in this approach is whether we could efficiently find the *frequent-path-bases*. The answer is yes, by using the FP-tree [4] structure to compress the database and to avoid multiple scans of the databases and COFI-trees [2] to partition the sub-transactions as we wish to do, to generate the *frequent-path-bases* as illustrate in the next section.

3 FP-Tree and COFI-Trees

The well-known FP-tree [4] data-structure is a prefix tree. The data structure presents the complete set of frequent itemsets in a compressed fashion. The construction of FP-Tree requires two full I/O scans. The first scan generates the frequent 1-itemsets. In the second scan, non-frequent items are stripped off the transactions and the sub-transactions with frequent ones are ordered based on their support, forming the paths of the tree. Sub-transactions that share the same prefix share the same portion of the path starting from the root. The FP-

tree has also a header table containing frequent items and holds the head link for each item in the FP-tree, connecting nodes of the same item to facilitate the item traversal during the mining process [4].

A COFI-tree [2] is a projection of each frequent item in the FP-tree. Each COFI-tree, for a given frequent item, presents the co-occurrence of this item with other frequent items that have more support than it. In other words, if we have 4 frequent items A, B, C, D where A has the smallest support, and D has the highest, then the COFI-tree for A presents co-occurrence of item A with respect to B, C and D, the COFI-tree for B presents item B with C and D. COFI-tree for C presents item C with D. Finally, the COFI-tree for D is a root node tree. Each node in the COFI-tree has two main variables, *support* and *participation*. *Participation* indicates the number of patterns the node has participated in at a given time during the mining step. Based on the difference between these two variables, *participation* and *support*, *frequent-path-bases* are generated. The COFI-tree has also a header table that contains all locally frequent items with respect to the root item of the COFI-tree. Each entry in this table holds the local support, and a link to connect its item with its first occurrences in the COFI-tree. A link list is also maintained between nodes that hold the same item to facilitate the mining procedure.

3.1 COFI-Closed Algorithm

The COFI-Closed algorithm is explained by a running example. The transactional database in Figure 2. A needs to be mined using a support greater or equal to 3. The first step is to build the FP-tree data-structure in Figure 2.B. This FP-tree data structure reveals that we have 8 frequent 1-itemsets. These are (A:10, B:8, C:7, D:7, E:7, F:6, G:5, H:3). COFI-trees are built after that, one at a time starting from the COFI-tree of the frequent item with lowest support, which is H. Since, in the order imposed, no other COFI-tree has item H then any closed pattern generated from this tree is considered globally closed. This COFI-tree generates the first closed pattern HA: 3. After that, H-COFI-tree is

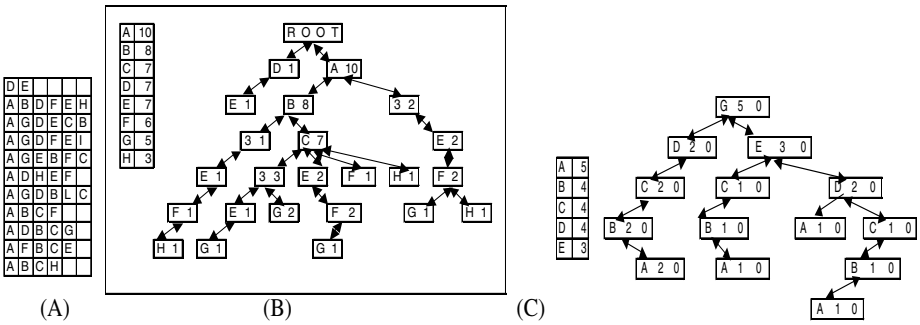

Fig. 2. (A) A Transactional database. (B) FP-Tree built from (A). (C) G-COFI-tree pointers from header tables are not presented

discarded and G-COFI-tree, in Figure 2.C, is built and it generates (GABCD:3, GABC:4, GAE:3, GAD:4, and GA:5), a detailed explanation of the steps in generating these frequent patterns are described later in this section. F-COFI-tree is created next and it generates all its closed patterns using the same method explained later.

Mining a COFI tree starts by finding the *frequent-path-bases*. As an example, we will mine the G-COFI-tree in Figure 2.C for closed patterns. We start 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 A. *Frequent-path-bases* (A, B, C, D: 2), (A, B, C, E: 1), (A, D, E: 1), and (A, B, C, D, E: 1) are generated from this tree. From these bases we create a special data structure called Ordered-Partitioning-Bases (OPB). The goal of this data structure is to partition the patterns by their length. Patterns with the same length are grouped together. This, on one hand allows dealing with patterns of arbitrary length, and on the other hand allows traversing the pattern space from the longest ones to the shortest ones and directly prunes the short ones if a frequent superset with same support is discovered as a candidate closed pattern.

This OPB structure is an array of pointers that has a size equal to the length of the largest *frequent-path-base*. Each entry in this array connects all *frequent-path-bases* of the same size. The first entry links all *frequent-path-bases* of size 1, the second one refers to all *frequent-path-bases* of size 2, the n^{th} one points to all *frequent-path-bases* of size n. 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* reports 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, i.e. not part of any other superset of frequent patterns. This *branch-support* is used to identify the *frequent-path-bases* from *non-frequent-path-bases* as *non-frequent-path-bases* have *branch-support* equal to 0, while a *frequent-path-base* has *branch-support* equal to the number of times this pattern occurs independently. The *branch-support* is also used to count the support of any pattern in the OPB. The support of any pattern is the summation of the *branch-supports* of all its supersets of *frequent-path-bases*. For example, to find the support for pattern X that has a length of k, all what we need to do is to scan the OPB from $k+1$ to n where n is the size of OPB, and sum the *branch-supports* of all supersets of X that do not have a *branch-support* equal to 0, i.e. the *frequent-path-bases*. The superset of X, as explained before are easily found using the prime codes.

In our example above, the first step is to build the OPB structure. The first pointer of this OPB structure points to 5 nodes which are (A, 5, 0), (B, 4, 0), (C, 4, 0), (D, 4, 0), and (E, 3, 0) which can be taken from the local frequent array

of the G-COFI-tree (Figure 2.C). The first number after the pattern presents the *support* while the second number presents the *branch-support*. The Second entry in this array points to all *frequent-path-bases* of size two. A null pointer is being linked to this node since no *frequent-path-bases* of size two are created. The third pointer points to one node which is (ADE, 1,1), the fourth points to (ABCD: 2: 2) and (ABCE: 1, 1), the fifth and last points to (ABCDE: 1:1). The leap-traversal approach is applied in the second step on the 4 *frequent-path-bases*, which are (ABCDE: 1: 1, ABCD:2 :2, ABCE: 2: 1, and ADE: 2: 1). Intersecting ABCDE with ABCD gives ABCD, which already exists, so nothing needs to be done. Same occurs when interesting ABCDE with ABCE. Intersecting ABCDE with ADE gives back ADE, which also already exists. Intersecting ABCD with ABCE gives ABC. ABC is a new node of size three. It is added to the OPB data structure and linked from the third pointer as it has a pattern size of 3. The *support* and the *branch-support* of this node equal 0. *Branch-support* equals 0 indicates that this pattern is a result of intersecting between *frequent-path-bases* and a *non-frequent-path-base*. Intersecting ABCD with ADE gives AD. AD is a new node of size two. It is added to the OPB data structure and linked from the second pointer. The *support* and the *branch-support* of this node equal 0. Intersecting, ABCE with ADE gives AE. AE is also a new node of size two and is also added to the OPB structure, at this stage we can detect that there is no need to do further intersections. The third step in the mining process is to find the global support for all patterns. Applying a top-down traversal between these nodes does this. If node X is a subset of a *frequent-path-base* Y then its support is incremented by the *branch-support* of node Y. By doing this we can find that ABCD is a subset of ABCDE, which has a *branch-support* equals to 1. The ABCD support becomes 3 (2+1). ABCE support becomes 2, as it is a subset of ABCDE. At level 3 we find that ADE is a subset of only ABCDE so its support becomes 2. ABC support equals to 4. AD support equals to 4, and AE support equals to 3. At this stage all non-frequent patterns and frequent patterns that have a local frequent superset with same support are removed from OPB. The remaining nodes (ABCD:3, ABC:4, AE:3, AD:4, and A:5) are potentially global closed. We test to see if they are a subset of already discovered closed patterns with the same support from previous COFI-trees. If not then we declare them as closed patterns and add them to the pool of closed patterns. The G-COFI-tree and its OBP data structure 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 closed pattern is added to the global pool of closed patterns.

4 Performance Evaluations

We present here a performance study to evaluate our new approach COFI-Closed against most of the state-of-art algorithms that mine closed patterns which are FP-Closed [3] and MAFIA-closed [1], CHARM [8]. Their respective authors provided us with the source code for these programs. All our experiments were conducted on an IBM P4 2.6GHz with 1GB memory running Linux

2.4.20-20.9 Red Hat Linux release 9. Timing for all algorithms includes the preprocessing cost such as horizontal to vertical conversions. The time reported also includes the program output time. We have tested these algorithms using synthetic datasets [5] on very large datasets. All experiments were forced to stop if their execution time reached our wall time of 5000 seconds. We made sure that all algorithms reported the same exact set of frequent itemsets on each dataset.

4.1 Experiments on Large Datasets

Mining extremely large databases is the main objective of this research work. We used five synthetic datasets made of 5M, 25M, 50M, 75M, 100M transactions, with a dimension of 100K items, and an average transaction length of 24 items. To the best our knowledge, these data sizes have never been reported in the literature before. CHARM could not mine these datasets. MAFIA could not mine the smallest dataset 5M in the allowable time frame. Only FP-Closed and COFI-Closed algorithms participated in this set of experiments. All results are depicted in Figure 3.A. From these experiments we can see that the difference between FP-Closed implementations and the COFI-Closed algorithm become clearer once we mine extremely large datasets. COFI-Closed saves at least one third of the execution time and in some cases goes up to half of the execution time compared to FP-Growth approach. The last recorded time for FP-Closed was mining 50 millions transactions while COFI-Closed was able to mine up to 100 millions transactions in less than 3500 seconds.

4.2 Memory Usage

We also tested the memory usage by FP-Closed, MAFIA and our approach. In many cases we noticed that our approach consumes one order of magnitude less memory than FP-Closed and two orders of magnitude less memory than MAFIA. Figure 3.B illustrates these results. We conducted experiments with the database size, the dimension and the average transaction length 1 million transactions, 100K items and 12 items respectively. The support was varied from 0.1% to 0.01%.

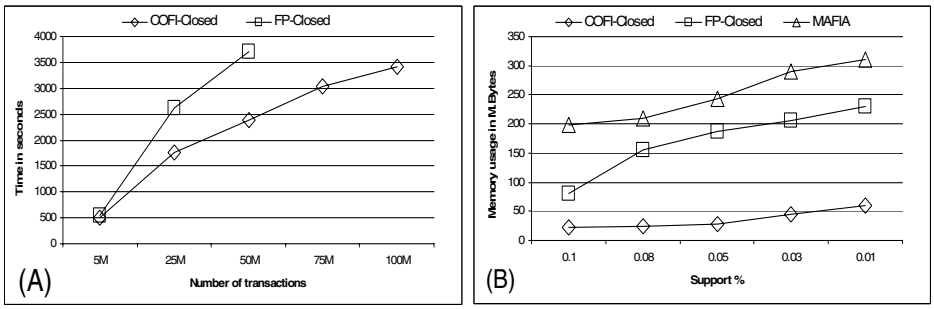

Fig. 3. (A) Scalability testing, (B) Disparity in memory usage T = 1000K, D = 100K

5 Conclusion

Mining for frequent itemsets is a canonical task, fundamental for many data mining applications. Many frequent itemset mining algorithms have been reported in the literature, some original and others extensions of existing techniques. They either model transactions horizontally or vertically and traverse the pattern space either bottom-up or top-down. However, most of the solutions assume to work on relatively small datasets in comparison to what we are expected to deal with in real applications such as affinity analysis in very large web sites, basket analysis for large department stores, or analysis of tracking data from radio-frequency identification chips on merchandize.

In this work we presented COFI-Closed, an algorithm for mining frequent closed patterns. This novel algorithm is based on existing data structures FP-tree and COFI-tree. Our contribution is a new way to mine those existing structures using a novel traversal approach. Using this algorithm, we mine extremely large datasets, our performance studies showed that the COFI-Closed was able to mine efficiently 100 million transactions in less than 3500 seconds on a small desktop while other known approaches failed.

References

1. D. Burdick, M. Calimlim, and J. Gehrke. Mafia: A maximal frequent itemset algorithm for transactional databases. In *ICDE*, pages 443–452, 2001.
2. M. El-Hajj and O. R. Zaïane. Non recursive generation of frequent k-itemsets from frequent pattern tree representations. In *In Proc. of 5th International Conference on Data Warehousing and Knowledge Discovery (DaWak'2003)*, September 2003.
3. G. Grahne and J. Zhu. Efficiently using prefix-trees in mining frequent itemsets. In *FIMI'03, Workshop on Frequent Itemset Mining Implementations*, November 2003.
4. J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *2000 ACM SIGMOD Intl. Conference on Management of Data*, pages 1–12, 2000.
5. IBM_Almaden. Quest synthetic data generation code. http://www.almaden.ibm.com/cs/quest/syndata.html.
6. N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. In *International Conference on Database Theory (ICDT)*, pages pp 398–416, January 1999.
7. J. Wang, J. Han, and J. Pei. Closet+: Searching for the best strategies for mining frequent closed itemsets. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'03)*, Washington, DC, USA, 2003.
8. M. Zaki and C.-J. Hsiao. ChARM: An efficient algorithm for closed itemset mining. In *2nd SIAM International Conference on Data Mining*, April 2002.
9. Z. Zheng, R. Kohavi, and L. Mason. Real world performance of association rule algorithms. In *7th ACM SIGKDD International Conf. on Knowledge Discovery and Data Mining*, pages 401–406, 2001.