COFI Approach for Mining Frequent Itemsets Revisited

Mohammad El-Hajj
Department of Computing Science
University of Alberta,Edmonton AB, Canada
mohammad@cs.ualberta.ca

Osmar R. Zaïane
Department of Computing Science
University of Alberta,Edmonton AB, Canada
zaiane@cs.ualberta.ca

ABSTRACT

The COFI approach for mining frequent itemsets, introduced recently, is an efficient algorithm that was demonstrated to outperform state-of-the-art algorithms on synthetic data. For instance, COFI is not only one order of magnitude faster and requires significantly less memory than the popular FP-Growth, it is also very effective with extremely large datasets, better than any reported algorithm. However, COFI has a significant drawback when mining dense transactional databases which is the case with some real datasets. The algorithm performs poorly in these cases because it ends up generating too many local candidates that are doomed to be infrequent. In this paper, we present a new algorithm COFI* for mining frequent itemsets. This novel algorithm uses the same data structure COFI-tree as its predecessor, but partitions the patterns in such a way to avoid the drawbacks of COFI. Moreover, its approach uses a pseudo-Oracle to pinpoint the maximal itemsets, from which all frequent itemsets are derived and counted, avoiding the generation of candidates fated infrequent. Our implementation tested on real and synthetic data shows that COFI* algorithm outperforms state-of-the-art algorithms, among them COFI itself.

1. INTRODUCTION

Recent days have witnessed an explosive growth in generating data in all fields of science, business, medicine, military, etc. The same rate of growth in the processing power of evaluating and analyzing the data did not follow this massive growth. Due to this phenomenon, a tremendous volume of data is still kept without being studied. Data mining, a research field that tries to ease this problem, proposes some solutions for the extraction of significant and potentially useful patterns from these large collections of data. One of the canonical tasks in data mining is the discovery of association rules. Discovering association rules, considered as one of the most important tasks, has been the focus of many studies in the last few years. Many solutions have been proposed using a sequential or parallel paradigm. However, the existing algorithms depend heavily on massive computation that might cause high dependency on the memory size or repeated I/O scans for the data sets. Association rule mining algorithms currently proposed in the literature are not sufficient for extremely large datasets and new solutions, that especially are less reliant on memory size, still have to be found.

DMKD '04 June 13, 2004, Paris, France

1.1 Problem Statement

The problem of mining association rules over market basket analysis was introduced in [1]. The problem consists of finding associations between items or itemsets in transactional data. The data could be retail sales in the form of customer transactions, text documents, or images. Association rules have been shown to be useful for other applications such as recommender systems, diagnosis, decision support, telecommunication, and even supervised classification. Formally, as defined in [2], the problem is stated as follows: Let $I = \{i_1, i_2, ...i_m\}$ be a set of literals, called items and m is considered the dimensionality of the problem. Let $\mathcal{D}$ be a set of transactions, where each transaction T is a set of items such that $T \subseteq I$. A unique identifier *TID* is given to each transaction. A transaction T is said to contain X, a set of items in I, if $X \subseteq T$. An *association rule* is an implication of the form "$X \Rightarrow Y$", where $X \subseteq I, Y \subseteq I$, and $X \cap Y = \emptyset$. An itemset X is said to be *frequent* if its *support* s is greater than or equal to a given minimum support threshold σ. Discovering association rules, however, is nothing more than an application for frequent itemset mining, like inductive databases [13], query expansion [14], document clustering [4], etc. What is relevant here is the efficient counting of some specific itemsets.

1.2 Related work

Mining for frequent itemsets is a canonical task, fundamental for many data mining applications and is an intrinsic part of many other data mining tasks. Mining for frequent itemsets is the major initial phase for discovering association rules. Associative classifiers rely on frequent itemsets. These frequent pattern are also used in some clustering algorithms. Finding frequent items is also an inherent part of many data analysis processes. Many frequent itemset mining algorithms have been reported in the last decade in the literature. The most important, and at the basis of many other approaches, is Apriori [1]. The property that is at the heart of Apriori and forms the foundation of most algorithms simply states that for an itemset to be frequent all its subsets have to be frequent. This monotone property reduces the candidate itemset space drastically. However, the generation of candidate sets, especially when very long frequent patterns exist, is still very expensive. Moreover, Apriori is heavily I/O bound. Another approach that avoids generating and testing itemsets is FP-Growth [11]. FP-Growth generates, after only two I/O scans, a compact prefix tree representing all sub-transactions with only frequent items. A clever and elegant recursive method mines the tree by creating projections called conditional trees and discovers patterns of all lengths without directly generating candidates the way Apriori does. However, the recur-

sive method to mine the FP-tree requires significant memory, and large databases quickly blow out the memory stack. Another innovative approach is COFI [7]. COFI in many cases is shown to be faster than FP-Growth and requires significantly less memory . The idea of COFI, which we adopt in this paper, is to build projections from the FP-tree each corresponding to sub-transactions of items co-occurring with a given frequent item. These trees are built and efficiently mined one at a time making the footprint in memory significantly small. The COFI algorithm generates candidates using a top-down approach, where its performance shows to be severely affected while mining databases that has potentially long candidate patterns that turns to be not frequent, as COFI needs to generate candidate sub-patterns for all its candidates patterns. We build upon the COFI approach to find the set of frequent patterns but after avoiding generating useless candidates, as we shall see herein.

The basic idea of our new algorithm is simple and is based on the notion of maximal frequent patterns. A frequent itemset X is said to be maximal if there is no frequent itemset X' such that $X \subseteq X'$ [3, 5]. Frequent maximal patterns are a relatively small subset of all frequent itemsets. In other words, each maximal frequent itemset is a superset of some frequent itemsets. Let us assume that we have an Oracle that knows all the maximal frequent itemsets in a transactional database. Deriving all frequent itemsets becomes trivial. All there is to do is counting them, and there is no need to generate candidates that are doomed infrequent. The oracle obviously does not exist, but we propose a pseudo-oracle that discovers this maximal patterns using the COFI-trees and we derive all itemsets from them. We argue that this process is more efficient, in most cases, than focusing solely and directly on the discovery of all frequent itemsets.

1.3 Contributions

In this paper we propose a new efficient algorithm for finding frequent patterns called COFI* based on COFI-trees proposed in [6, 7] and FP-Trees presented in [11]. This algorithm generates the frequent patterns by finding efficiently the set of local maximals for each COFI-tree. Based on these maximals we generate all their subsets that are indeed frequent. Counting the support for each of these frequent pattern is the last major step in this approach. The main differences between our new approach COFI* and COFI are the followings: (1) COFI* uses a novel technique to generate candidates and count their supports. (2) COFI* proposes a new data structure to partition the itemsets helping the handling of patterns of arbitrary length. (3) COFI* finds the set of all frequent patterns by first finding the set of maximal patterns using a novel traversal approach, and then generate from this set the set of all frequent patterns. (4) COFI uses a depth-first strategy, while COFI* uses a leap-traversal approach introduced in this paper.

The rest of this paper is organized as follows: Section 2 depicts in general what are an FP-Tree and COFI-trees. To put our algorithm in the context, we describes the existing traversal approaches and then explain our new leap-traversal approach used by our pseudo-oracle in Section 3. The complete COFI* algorithm is explained in Section 4 with illustrative examples. Section 5 depicts the performance evaluation of COFI* comparing it with existing state-of-the-art algorithms on dense and sparse data.

2. FP-TREE AND COFI-TREES

The well-known FP-Tree [11] data-structure is a prefix tree structure. 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. These sorted sub-transactions form 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. This table holds the header link for each item in the FP-Tree, and connects nodes of the same item to facilitate the item traversal during the mining process. We invite the reader to see [11] for more details.

A COFI-tree[7] 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, which are *support* and *participation* that are used in the mining step. Basically, the support represents the local support of a node, while participation represents, at a given time in the mining process, the number of times the node has participated in already counted patterns. In comparison with our COFI tree structure presented in [7], our new data structure we use here is improved by keeping only the nodes representing locally frequent items with respect to the root of the tree. This not only reduces space but facilitates the mining later. Based on the difference between the two variables, *participation* and *support*, special patterns called *frequent-path-bases* are generated. These are simply the paths from a given node x, with participation smaller than support, up to the root, i.e. nodes that did not fully participate yet in frequent patterns.

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. TRAVERSAL APPROACHES AND CANDIDATE GENERATION

Current algorithms generate candidate frequent patterns by using one of the two methods, namely: breadth-search or depth-search. Breadth search can be viewed as a bottom-up approach where the algorithms visit patterns of size $k+1$ after finishing the k sized patterns. The depth search approach does the opposite where the algorithm starts by visiting patterns of size k before visiting those of size $k-1$. Both methods show some efficiency while mining some databases. On the other hand, they show weaknesses or inefficiency in many other cases. To understand this fully, we will try to focus on the main advantages and drawbacks of each one of these methods in order to find a way to make use of the best of both of them, and to diminish as much as possible their drawbacks. As an example, assume we have a transactional database that is made of a large number of frequent 1-itemsets that has maximal patterns with relatively small lengths. The trees built from such a database are usually deep as they have a large number of frequent 1-itemsets. Traversing in depth-search manner would provide us with potential long patterns that end-up non-frequent ones. In such cases,

the depth-search method is not favored. However, if the longest frequent patterns generated are relatively long with respect to the depth of the tree, then the depth search strategy is favored as most of the potential long patterns that could be found early tend to be frequent. On the other hand, mining transactional databases that reveal long frequent patterns is not favored using breadth search manner, as such algorithms consume many passes to reach the long patterns. Such algorithms generate many candidate frequent patterns at level k that would be omitted once they are found to be not frequent. These generation steps become a bottleneck while mining transactional databases of long frequent patterns using the breadth-search methods. The following example demonstrates how both approaches work. If we want to mine projected transactions in Figure 1.A with support greater than 4, then the three longest frequent patterns that can be generated are (ABCDE:6) of size 5, (AFGHI:5) also of size 5 and (AJKL:5) of size 4. The number after the pattern represents its support. All sub-patterns from these three patterns are indeed frequent, but we do not mention them here for the sake of space. Discovering these patterns using the top-down approach (depth search) requires mining a tree of depth 9. Although none of the candidate patterns of size 9 to 6 are frequent, we still need to generate and test them. This generation process continues until we reach the first long frequent patterns of size 5: (ABCDE:6) and (AFGHI:5). Many pruning techniques can then be applied to reduce the remaining search space. From these patterns we will generate all sub-patterns that are frequent. The bottom-up approach needs to create all patterns from sizes 2 to 6 at which point it can detect that there are no more local frequent patterns to discover. All non-frequent patterns of sizes 2 to 6 would be removed. We propose a combination of these approaches that takes into account a partitioning of the search space à la COFI-tree. We call this method the leap-traversal approach since it selectively jumps within the lattice.

3.1 Leap-traversal approach

To find the maximal patterns, we introduce a new leap-traversal approach that looks ahead at the nature of the transactional database, and recommends a set of candidate patterns from different sizes to test where the local maximal patterns are subset of this recommended set. From these local maximals we can generate the set of all frequent patterns. Step one of this approach is to look at the nature of the distribution of frequent items in the transactional database. If we revisit the example presented above from Figure 1.A, 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; {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. These distributions are indeed similar to the *frequent-path-bases* generated from COFI-trees in the previous section. 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 superset of *frequent-path-bases* patterns. Step 4 scans these patterns to remove non-frequent ones or frequent ones that already have a frequent superset. The remaining patterns can be declared as local maximal patterns. Figure 1.B illustrates the steps needed to generate the local maximal patterns of our example from Figure 1.A.

The main question in this approach would be "can we efficiently find the *frequent-path-bases*? The answer is yes, by using the FP-tree [11] structure to compress the database and to avoid multiple scans of the database as only two full I/O scans are needed to

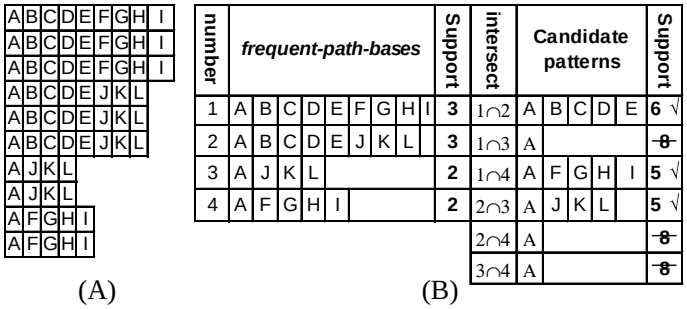

Figure 1: (A): Projected transactional database with respect to item A.
(B): Steps needed to generate maximal patterns using the leap-traversal approach ($\surd$ indicates a discovered maximal pattern. Barred entries are the eliminated candidates)

create this data-structure. COFI-trees [7] which partition the sub-transactions as we wish to do, are used to generate the *frequent-path-bases* as illustrate in the next section.

4. COFI* ALGORITHM

We will explain the COFI* algorithm by a running example. A simplified pseudo-code can be found in Figure 4. The transactional database in Figure 2.A needs to be mined using*support* greater than 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, which 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. This COFI-tree generates the first frequent pattern HA: 3. After that G-COFI-tree, in Figure 2.C, is built and it generates all frequent patterns with respect to item G, a detailed explanation of the steps in generating these frequent patterns are described later in this section. The remaining COFI trees are built independently. Each one of them generates its corresponding frequent patterns.

4.1 Mining a COFI-tree

Mining COFI trees starts by finding the *frequent-path-bases.* As an example, we will mine the G-COFI-tree in Figure 2.C for all frequent 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* 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, allows dealing with patterns of arbitrary length.

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-pattern-bases* of the same size. The first entry links all *frequent-pattern-bases* of size 1, the second one refers to all *frequent-pattern-bases* of size 2, the n^{th} one points to all *frequent-pattern-bases* of size n. An illustrative example can be found in Figure 3. Each node of the connected link list is made of

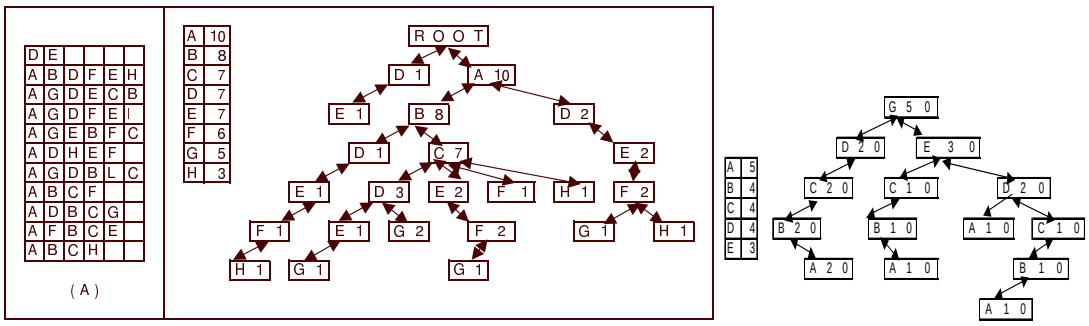

Figure 2: (A) A Transactional database. (B) FP-Tree built from (A). (C) G-COFI-tree

4 variables which are: the pattern, a pointer to the next node, and two number variables that present 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 *frequent-path-bases* have *branch-support* equal to the number of times the pattern occurs independently together. 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 the 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-bath-bases.*

In our example above, 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. The Second entry in this array points to all *frequent-path-bases* of size 2. A null pointer is being linked to this node since no *frequent-path-bases* of size 2 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 second step in the mining process is to find the global *support* of each one of the local *frequent-path-bases.* Applying a top-down traversal between these nodes does this. If node A is a subset of node B then its *support* is incremented by the *branch-support* of node B. By doing this, we can find that ABCD is a subset of ABCDE, which has a *branch-support* equal to 1. The ABCD *support* becomes 3 (2+1). ABCE *support* becomes 2, as it is a subset of ABCDE. ADE *support* becomes 2. At level 3 we find that ADE is a subset of only ABCDE so its *support* becomes 2. From this we can find that ABCD is a frequent pattern. We put this pattern aside as a potential frequent maximal pattern. A leap-traversal approach is applied on the 3 remaining *frequent-path-bases*, which are (ABCDE: 1, ABCE: 1, and ADE: 1). Intersecting ABCDE with ABCE gives ABCE, which already exists, so nothing needs to be done since we already have the global frequency of this pattern. Intersecting ABCDE with ADE gives back ADE, which also already exists. Intersecting, ABCE with ADE gives AE. AE is a new node of size 2. It is be added to the OPB data structure and linked from the second pointer as it has a pattern size of 2. The *support* and the *branch-support* of this node equals 0. *branch-support* equals 0 indicates that this pattern is a result of intersecting between *frequent-path-bases* and not a *frequent-path-base* per se. The global *support* of this node is then computed which is equal to the summation of all *branch-supports* of all superset of *frequent-path-bases.* The *support* of this node becomes 3, as it is a subset of (ADE, 1,1), (ABCE: 1: 1), and (ABCDE: 1:1). At this stage all non-frequent patterns, and frequent patterns that have a local frequent superset except the *frequent-path-bases* are removed from OPB. The remaining frequent nodes are locally maximal. These steps are presented in Figure 3.

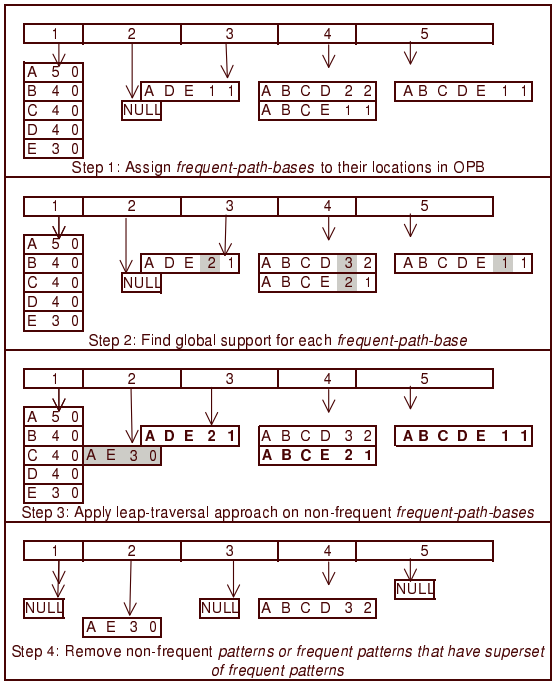

Figure 3: Steps needed to generate local maximal frequent patterns using OPB

4.2 Generating frequent patterns

Frequent patterns are generated from each local maximal. A direct step is applied to find all subset of each locally maximal patterns. The next step after finding each one of these patterns is to find their *support.* The *support* of each pattern is the summation of the *branch-support* of all its superset of *frequent-path-bases.* In our previous example for mining the G-COFI-tree we find that we have 2 local maximals which are (AE:3, and ABCD:3) and 4 *frequent-path-bases* which are (ADE:1: 1), (ABCD:2:2),

(ABCE:1:1), (ABCDE:1:1). The first local maximal AE:3 generates two patterns which are A and E. A has a *support* of 5 and E has a *support* of 3. This *support* is taken from the header list of the G-COFI-tree. The second local frequent maximal is ABCD:3 that generates A, B, C, D, AB, AC, AD, ABC, ABD, BC, BD, CD, and BCD. The *support* of ABC for example is 4 as it is a subset of three *frequent-path-bases* which are ABCD, ABCE, and ABCDE which have *support*-branches 2,1 and 1 respectively. The same process is executed to find the *support* for all frequent patterns that are of size greater than 1. The *support* of frequent patterns of size 1 can be derived directly from the COFI-tree header table.

5. PERFORMANCE EVALUATIONS

We tested our approach against FP-Growth[11], Eclat[15], and COFI algorithms. Eclat was provided to us by its original authors. We used FP-Growth written by Bart Goethals [8] as we found that this implementation is faster than the one provided by the original authors of FP-Growth, which we initially used to compare with COFI [7]. 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.

5.1 Experiments on UCI datasets

We tested COFI* and the contenders COFI, Eclat and FP-Growth using 4 databases, *chess, connect, pumsb, and mushroom*, downloaded from [9]. These datasets are commonly used for testing such algorithms and are described in detail in many previous work [10]. In our experiments, we set a time limit to 120 seconds. Beyond 2 minutes, if a program did not return the answer, we terminated the program and considered it a failure. COFI did not perform well in these datasets as it could not mine *connect and pumsb* with support less than 80%, and *chess* with support less than 70%. COFI*, on the other hand, reported the best result in all test cases with *connect, pumsb, and chess*. FP-growth was able to compete with COFI* in some of the test cases while mining the *mushroom* dataset. Figure 5 depicts the results of these experiments.

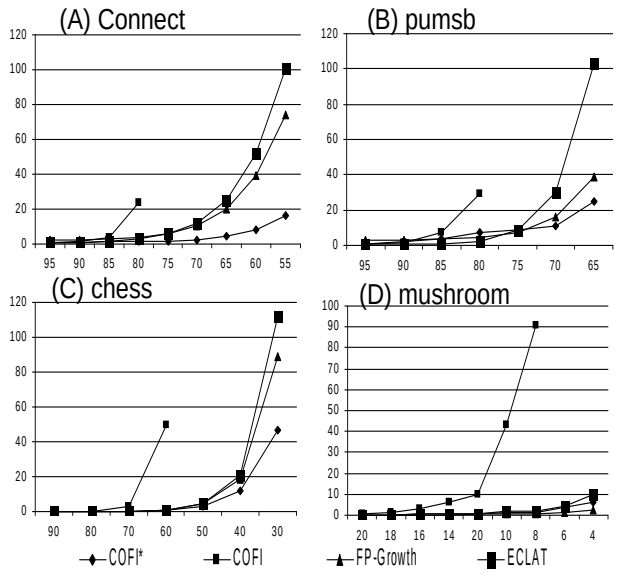

Figure 5: Mining UCI databases

5.2 Experiments on small and large synthetic datasets

We generated synthetic datasets using [12]. We report results here only for FP- Growth, COFI, and COFI* since Eclat, inexplicably, did not generate any frequent patterns in most cases. A similar run time limit was set as before. In this set of experiments we focused on studying the effect of changing the support while testing three parameters: transaction length, database size, and dimension of the database. We have created databases with transaction length averaging 12 and 24 items per transaction. Our datasets also have as a two dimension sizes of different values: 5000 and 10000 items. The database size varies from 10,000 to 250,000 transactions per database. Notice that these datasets are considered relatively sparse. COFI* and COFI outperformed FP-Growth almost in all test cases. the difference between COFI* and COFI in many cases was negligible. Both were 2 to 3 times faster than FP-Growth. These experiments are depicted in Figure 6.

In this set of experiments we found that COFI* and COFI are competing while mining datasets with relatively long transactions. This can be justified by the fact that if long candidate patterns in long transaction are indeed frequent then depth-search strategy used by COFI algorithm is favored. However, if long candidate patterns from long transactions are infrequent, then the leap-traversal approach adopted by COFI* is the winner because it generates significantly less candidates to test.

We do not have access to very large real datasets since none are publicly available, but we know, based on the experiments on the UCI datasets, that COFI* has an advantage. We generated very large synthetic datasets with 100K items and an average of 12 items per transaction. On datasets with 5M, 10M, 15M and 20M transactions COFI and COFI* were very close ending respectively with support of 0.002 in 64, 140, 205, 309 seconds and 69, 149, 221, 321 seconds. FP-Growth mined only the 5 and 10 million collections in twice as much time: 126 and 376 seconds.

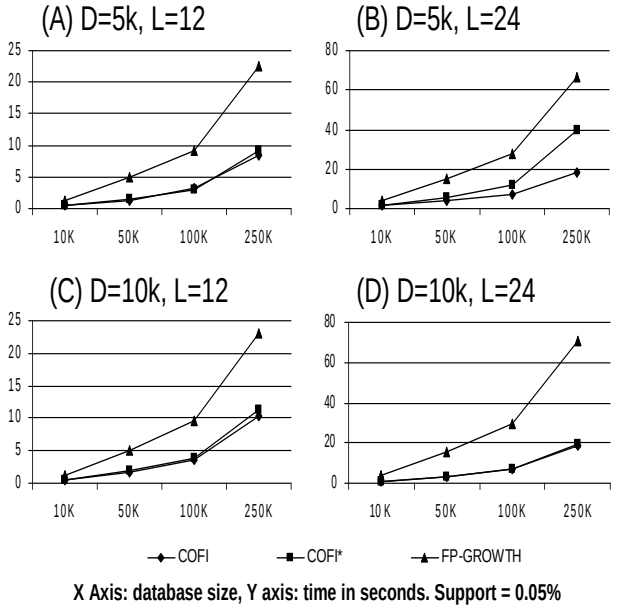

Figure 6: Mining synthetic databases

Algorithm : COFI* algorithm
Input : Transactional Database D, and minimum support σ
Output : A set of frequent patterns
Method :

1. Scan the database D twice the build the FP-Tree based on σ
2. For each frequent item F (starting with the item with lowest support)
   2.1. Build condensed ***F-COFI-tree***
   2.2 Find frequent-path-bases
   2.3 Built Ordered-Partitioning Bases (OPB structure)
   2.4 Find local maximal patterns
3. For each local maximal pattern M
   3.1 Generate all sub-patterns of M
   3.2 For each sub-pattern X of M
      3.2.1 Count support of X using OPB

Figure 4: COFI* pseudocode: The oracle part and the pattern counting are merged.

6. CONCLUSION

Mining for frequent itemsets is a canonical task, fundamental for many data mining applications. Especially for dense data and when fairly long patterns exist, it becomes expensive, if not impossible, to enumerate candidate patterns. For instance the enumeration of the search space for patterns in the order of 40 items or more is practically computationally unfeasible. In these cases, a promissing direction is to mine closed or maximal itemsets.

We presented COFI* a new algorithm for mining frequent patterns. This new algorithm is based on existing data structures FP-tree and COFI-trees and initiates the process by first identifying maximal patterns using a novel lattice traversal approach. Our contribution is a new way to mine those structures and a set of prunning methods to accelerate the discovery process.

Our performance studies show that COFI* outperforms most of the state of the art methods. This algorithm finds the set of exact maximal patterns using only 2 I/O scans of the database then generates all frequent patterns with their respective support. It also presents a new way of traversing the pattern space to search for candidates. This new traversing approach dramatically minimizes the size of the candidate list. It also introduces a new method of counting the supports of candidates based on the supports of other candidate patterns.

7. REFERENCES

[1] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proc. 1993 ACM-SIGMOD Int. Conf. Management of Data*, pages 207–216, Washington, D.C., May 1993.

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. 1994 Int. Conf. Very Large Data Bases*, pages 487–499, Santiago, Chile, September 1994.

[3] R. J. Bayardo. Efficiently mining long patterns from databases. In *ACM SIGMOD*, 1998.

[4] F. Beil, M. Ester, and X. Xu. Frequent term-based text clustering. In *Proc. 8th Int. Conf. on Knowledge Discovery and Data Mining (KDD '2002), Edmonton, Alberta, Canada*, 2002.

[5] D. Burdick, M. Calimlim, and J. Gehrke. Mafia: A maximal frequent itemset algorithm for transactional databases. In *ICDE*, pages 443–452, 2001.

[6] M. El-Hajj and O. R. Zaïane. Inverted matrix: Efficient discovery of frequent items in large datasets in the context of interactive mining. In *In Proc. 2003 Int'l Conf. on Data Mining and Knowledge Discovery (ACM SIGKDD)*, pages 109–118, August 2003.

[7] 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)*, pages 371–380, September 2003.

[8] B. Goethals. Frequent pattern mining implementations. http://www.cs.helsinki.fi/u/goethals/software/index.html.

[9] B. Goethals and M. Zaki. Advances in frequent itemset mining implementations: Introduction to fimi03. In *Workshop on Frequent Itemset Mining Implementations (FIMI'03) in conjunction with IEEE-ICDM*, 2003.

[10] K. Gouda and M. J. Zaki. Efficiently mining maximal frequent itemsets. In *ICDM*, pages 163–170, 2001.

[11] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *ACM-SIGMOD*, Dallas, 2000.

[12] IBM_Almaden. Quest synthetic data generation code. http://www.almaden.ibm.com/cs/quest/syndata.html.

[13] H. Mannila. Inductive databases and condensed representations for data mining. In *International Logic Programming Symposium*, 1997.

[14] A. Rungsawang, A. Tangpong, P. Laohawee, and T. Khampachua. Novel query expansion technique using apriori algorithm. In *TREC, Gaithersburg, Maryland*, 1999.

[15] M. J. Zaki. Scalable algorithms for association mining. *Knowledge and Data Engineering*, 12(2):372–390, 2000.