Mining *N*-most interesting itemsets without support threshold by the COFI-tree

Sze-Chung Ngan, Tsang Lam,
Raymond Chi-Wing Wong and
Ada Wai-Chee Fu*

Department of Computer Science and Engineering,
The Chinese University of Hong Kong,
Shatin, Hong Kong
Fax: +852 26035024 E-mail: scngan2@cse.cuhk.edu.hk
E-mail: tlam2@cse.cuhk.edu.hk E-mail: cwwong@cse.cuhk.edu.hk
E-mail: adafu@cse.cuhk.edu.hk
*Corresponding author

Abstract: Data mining is the discovery of interesting and hidden patterns from a large amount of collected data. Applications can be found in many organisations with large databases, for many different purposes such as customer relationships, marketing, planning, scientific discovery, and other data analysis. In this paper, the problem of mining *N*-most interesting itemsets is addressed. We make use of the techniques of COFI-tree in order to tackle the problem. In our experiments, we find that our proposed algorithm based on COFI-tree performs faster than the previous approach BOMO based on the FP-tree.

Keywords: association rules; *N*-most interesting itemsets; FP-tree; COFI-tree; data mining; knowledge discovery.

Reference to this paper should be made as follows: Ngan, S-C., Lam, T., Wong, R.C-W. and Fu, A.W-C. (2005) 'Mining *N*-most interesting itemsets without support threshold by the COFI-tree', *Int. J. Business Intelligence and Data Mining,* Vol. 1, No. 1, pp.88–106.

Biographical notes: Sze-Chung Ngan is a Final-Year student studying Computer Engineering at the Chinese University of Hong Kong and expected to graduate in 2005. He is interested in research work in Data Mining. He has worked on a one year project titled 'Mining Interesting Patterns in Large Database'.

Tsang Lam is a Final-Year student studying Computer Engineering at the Chinese University of Hong Kong and she is expected to graduate in 2005. She is interested in research work in Data Mining. She has worked on a one year project titled 'Mining Interesting Patterns in Large Database'.

Raymond Chi-Wing Wong received the BSc and MPhil degrees in Computer Science and Engineering in the Chinese University of Hong Kong in 2002 and 2004, respectively. He also received Swire Scholarship from 2000 to 2002. He graduated in the Chinese University of Hong Kong with the First Honour of Bachelor of Science in 2002. Recently, he obtained the ICDM 2003 Student Travel Award and the fifth ACM Postgraduate Research Day

Merit Award. He is now working as a Research Assistant in the Chinese University of Hong Kong. His research interests include Data Mining, Database and Security.

Ada Wai-Chee Fu received her BSc degree in Computer Science from the Chinese University of Hong Kong, and both the MSc and the PhD degrees in Computer Science from the Simon Fraser University of Canada. She worked at Bell Northern Research in Ottawa, Canada from 1989 to 1993 on a wide-area distributed database project; joined the Chinese University of Hong Kong in 1993, where she is an Associate Professor at the Department of Computer Science and Engineering. Her research interests include Database Related Issues, Data Mining, Parallel and Distributed Systems.

1 Introduction

One of the important topics in the literature of data mining is Association Rule Mining. Association Rule Mining (Agrawal et al., 1993) has been proposed for understanding the relationships among items in transactions or market baskets. For instance, if a customer buys butter, what is the chance that he/she buys bread at the same time? Such information may be useful for decision makers to determine strategies in a store. The formal statement of the problem of data-mining:

Let $I = \{i_1, i_2, \ldots, i_m\}$ be a set of literals, called *items*. Let D be a set of *transactions*, where each transaction T is a set of items such that $T \subseteq I$. Each transaction is associated with a unique identifier, called its *TID*. X is a set of some items in I. If $X \subseteq T$, transaction T contains X.

An *association rule* is an implication of the form $X \Rightarrow Y$, where $X \subset I$, $Y \subset I$, and $X \cap Y = \phi$. The rule $X \Rightarrow Y$ holds in the transaction set D with *confidence c* if *c%* of transactions in D that contain X also contain Y. The rule $X \Rightarrow Y$ has *support s* in the transaction set D if *s%* of transactions in D that contain $X \cup Y$.

Given a set of transactions D, the problem of *mining association rules* is to generate all association rules that have support and confidence greater than the minimum support and minimum confidence respectively.

The problem of Association Rules Mining can be decomposed into two subproblems:

- Find all sets of items (*itemsets*) that have transaction support above minimum support. The *support* for an itemset is the number of transactions that contain the itemset. Itemsets with minimum support are called *large itemsets* while those without, are called small itemsets.
- Use the large itemsets to generate the desired rules. A *k-itemset* is a set of items containing k items.

The Apriori algorithm for discovering large itemsets make multiple passes over the data. It is very time-consuming to make multiple passes over the data to discover large itemsets. To approach real-time data-mining, we need to shorten the time to accomplish this task. There are many researchers who propose to tackle the above problem (Agrawal and Srikant, 1994; Han et al., 2000; El-Hajj and Zaiane, 2003). Agrawal and Srikant (1994) propose an Apriori algorithm. Han et al. (2000) propose an

FP-growth algorithm by using a new tree structure, the FP-tree. El-Hajj and Zaiane (2003) propose a similar data structure called COFI-tree which mines frequent itemsets.

All the above proposed algorithms focus on the problem of mining frequent itemsets with frequency greater than a support threshold. However, it is difficult for the users to determine the threshold value. If the threshold is set too large, then there are no frequent itemsets in the output. If the threshold is set too small, then there are too many frequent itemsets in the output. Thus, finding a suitable threshold is a difficult task.

Let us next introduce the problem of finding N most interesting itemsets. It is in general much easier for the users to give the parameter N compared with the threshold parameter. Before formulating our problem, we have the following definitions.

Definition 1: The *N-most interesting k-itemsets.* Let us sort the k-itemsets by descending support values, let S be the support of the N-th k-itemset in the sorted list. The N-most interesting k-itemsets are the set of k-itemsets having support $\geq S$.[1]

Definition 2: The *N-most interesting itemsets* is the union of the N-most interesting k-itemsets for each $1 \leq k \leq k_{\max}$, where $k_{\max}$ is the upper bound of the size of itemsets we would like to find. We say that *n* itemset in the N-most interesting itemsets is interesting.

Objective: We set our goal to find a fast algorithm to mine N-most interesting k-itemsets without a given minimum support threshold.

2 Background

Agrawal and Srikant (1994) first proposed an algorithm to tackle the problem of mining frequent itemsets with frequency greater than a given threshold. The Apriori heuristic achieves good performance gain by (possibly significantly) reducing the size of candidate sets. However, in situations with prolific frequent patterns, long patterns, or quite low minimum support thresholds, an Apriori-like algorithm may still suffer from the following two non-trivial costs: it is costly to handle a huge number of candidate sets. To discover a frequent pattern of size 100, such as $\{a1, a2, \ldots, a100\}$, it must generate more than $2^{100} \approx 10^{30}$ candidates in total. It is tedious to repeatedly scan the database and check a large set of candidates by pattern matching, especially for mining long patterns.

The bottleneck of the Apriori-like method is at the *candidate set generation* and *test*. If one can avoid generating a huge set of candidates, the mining performance can be substantially improved. A data structure called *Frequent-Pattern tree*, or *FP-tree* and an efficient algorithm called FP-growth are proposed by Han et al. (2000) to overcome the above weaknesses. The idea of FP-tree is, fetching all transactions from the database and inserting them into a compressed tree structure. Then, algorithm FP-growth reads from the structure FP-tree to mine frequent itemsets. El-Hajj and Zaiane (2003) found that FP-growth algorithm has a bottleneck of the excessive recursive call of functions. They thus proposed the COFI-tree which is similar to the FP-tree. In the COFI-tree, a vast amount of recursive calls are reduced.

As we mentioned before, it is more straightforward to mine *N* most interesting itemsets. Cheung and Fu (2004) proposed an algorithm called BOMO which mines the *N* most interesting itemsets. Algorithm BOMO is based on an FP-tree structure. Initially, by updating the threshold dynamically, BOMO can find the *N* most interesting itemsets. However, El-Hajj and Zaiane (2003) claim that the COFI-tree should have a better performance, when compared with the FP-tree. Thus, in this paper, we propose an algorithm based on the COFI-tree in order to find *N* most interesting itemsets.

3 Proposed algorithm

Our proposed algorithm adopts the techniques used in COFI-tree (co-occurrence Frequent Item Tree) in order to find *N*-most interesting itemsets. The COFI-tree approach consists of two main stages, namely *Stage 1* and *Stage 2*.

Stage 1: *the construction of a frequent pattern tree (FP-tree).*

The construction of an FP-tree is the same as the one proposed by Han et al. (2000). That is, we build an FP-tree with a given minimum support threshold is equal to 0 initially. The details can be found in the paper (Han et al., 2000). Initially, the threshold is set to be 0. However, during processing, for a number of COFI-trees in Stage 2, the threshold is updated incrementally.

Stage 2: *building the co-occurrence frequent-item-trees (COFI-trees)*

The idea of the construction of a COFI-tree is to build a number of independent and relatively small trees, called the COFI-tree, for each item. The COFI-tree has tree-structure similar to the FP-tree. It also contains a header table with the horizontal link connecting to a number of nodes. The COFI-tree for an item *X* is the one with the root node containing the item *X*, which is constructed with a *pruning* technique to remove all items which co-exist with the main frequent item *X* with frequencies less than a threshold value.[2] After the COFI-tree is constructed, all frequent itemsets with their frequencies in the COFI-tree can be found. The trees will be discarded after all frequent itemsets in the COFI-tree are mined. At any given time, only one COFI-tree exists in the main memory.

In our proposed approach, there are $k_{\max}$ result sets R_k, where each result set R_k keeps the current *N*-most interesting *k*-itemsets where $1 \leq k \leq k_{\max}$. Besides, we also keep $k_{\max}$ threshold values ξ_k , where each threshold value ξ_k is the greatest frequency of the itemsets stored in the result set R_k for $1 \leq k \leq k_{\max}$. Besides, we also store a global threshold ξ, which is the minimum of ξ_k for $1 \leq k \leq k_{\max}$.

In the COFI-tree for each item, we can, not only, find the frequent itemsets but also have to update the result sets R_k and the threshold values ξ_k. If the frequencies of the *k*-itemsets *I* mined are greater than the current threshold value ξ_k, then the result sets R_k will be updated with the *k*-itemsets *I*. If R_k does not contain enough *N*-most interesting *k*-itemsets, then *k*-itemsets *I* will be inserted into the result sets R_k. Otherwise, the itemset *I'* with the smallest frequency is removed from R_k and the *k*-itemsets *I* are inserted into R_k.

3.1 Mining the COFI-trees

The COFI-trees of all frequent items are not constructed at the same time. Each tree is built and then mined. Finally, it is discarded before the next COFI-tree is built. The mining process is repeated for each COFI-tree for item *X* independently for finding all frequent *k*-itemset patterns in which the item *X* on the root of the tree participates.

3.2 Pruning the COFI-trees

We adopt a clever way of pruning in constructing the COFI-tree. Pruning can be done after building a tree or, while building it. We adopt *pruning on the fly* because there are minimal overheads and drastic reduction in memory requirements.

Example: Let us illustrate our algorithm with an example. The example used here is similar to the one used by El-Hajj and Zaïane (2003). Suppose we have to mine 2-most interesting items and 2-most interesting 2-itemsets. That is, $k_{\max} = 2$ and $N = 2$. There are the following 18 transactions and six items in the data set.

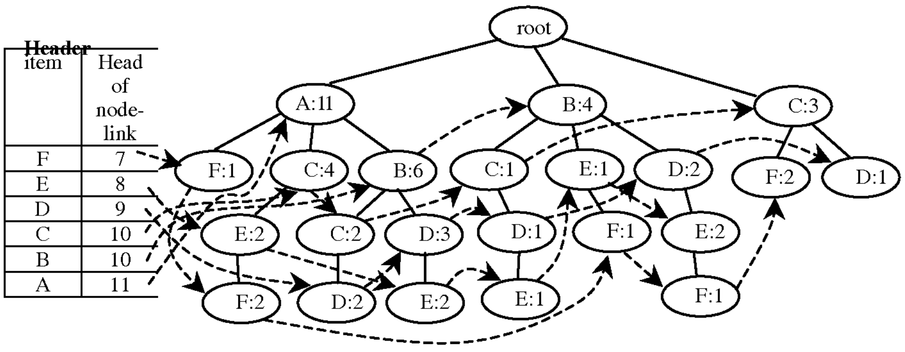
Trans. No.	*Item*			
1	A	B	C	D
2	B	C	D	E
3	A	B	D	E
4	A	C	E	F
5	A	B		
6	A	C		
7	A	C		
8	B	E	F	
9	A	F		
10	C	F		
11	A	B	D	
12	B	D	E	
13	C	D		
14	C	F		
15	B	D	E	F
16	A	B	D	E
17	A	C	E	F
18	A	B	C	D

Initially, we initialise the results sets with empty sets and the thresholds with 0.

Result set	*Itemsets*
R_1	{}
R_2	{}
Threshold Values	
$\xi_1 = 0$, $\xi_2 = 0$, $\xi = \min\{\xi_1, \xi_2\}$	

In Stage 1, we build an FP-tree from the transactions with threshold = 0 (see Figure 1).

Figure 1 Frequent pattern tree (Minimum Support Threshold $\xi = 0$)



In this FP-tree, the dotted arrow corresponds to the horizontal link of an item *X* while the solid line corresponds to the link between two nodes in the parent-and-child relationship. Each node in the FP-tree is represented by an oval containing the item name and its frequency. In the paper (El-Hajj and Zaïane, 2003), the COFI-tree for item *X* is constructed in the ascending order of the frequency of the item *X*. That is, in this example, the order of the construction is *F* COFI-tree, *E* COFI-tree, *D* COFI-tree, *C* COFI-tree and *B* COFI-tree[3]. However, we adopt the reverse order in our proposed method. That is, the construction order is *B* COFI-tree, *C* COFI-tree, *D* COFI-tree, *E* COFI-tree and *F* COFI-tree. This is because it is likely that the itemsets mined in the COFI-tree for the item *X* with higher frequency have higher frequencies. If we can mine the itemsets with higher frequencies in the initial step, then we need not update or switch the result set R_k frequently.

There is no need to construct the COFI-tree for the most frequent item *A* because all COFI-trees for other items yields all possible itemsets containing item *A*. We just need to mine the item (i.e., *A*) with frequency 11. Thus, we update the result sets and the thresholds as follows.

Result set	*Itemsets*
R_1	{<*A*:11>}
R_2	{}
Threshold values	
$\xi_1 = 0$, $\xi_2 = 0$, $\xi = \min\{\xi_1, \xi_2\} = 0$	

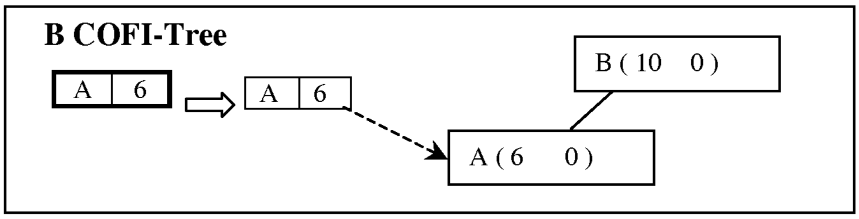
The overall threshold ξ is still equal to 0.

3.3 B COFI-tree

The first COFI-tree is built for item *B* as it is the second frequent item. In this tree for *B*, all items which are *more frequent* than *B* and share transactions with *B* participate in building the tree. This can be found by following the chain of items *B* (by the horizontal node-link) in the FP-tree structure.

The resulting *B* COFI-tree is shown in Figure 2. The two nodes on the right 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 initialised to 0 and is used by the mining algorithm. A horizontal link (dotted line) points to the next node that has the same item-name in the tree, and a child node is linked with its parent by a bi-directional vertical link.. The squares one the left are cells from the header table as in the FP-Tree. This is a list made of all frequent items that participate in building the tree structure sorted in ascending order of their global support. Each entry in this list contains the item-name, item-counter, and a pointer to the first node in the tree that has the same item-name.

Figure 2 *B* COFI-tree



The *B*-COFI-tree is constructed in the following way. We will traverse the FP-tree twice. The first traversal is to count the frequency of each item with respect to item *B*. In the header table of the FP-tree, we follow the horizontal link of the item *B*. First, we reach the node (*B*:6), called N_1. Then, we need to traverse all the nodes N_2 upwards in node N_1. In this example, the node above N_2 is (*A*:11). However, we know that *AB* has the frequency of six in this branch as node N_1 has the frequency six. In other words, the item in node N_2 (i.e., item *A*) occurs six times with item *B* only. We continue to traverse the nodes containing item *B* horizontally according to the horizontal link of item *B* and process the node similarly. The goal of this traversal is to count the frequency of each item with respect to item *B*. The frequency of each item is stored and will be used in the *pruning* step[4]. By doing so, we find that item *A* occurs six times.

The second traversal of the FP-tree is to create the COFI-tree for item *B*. Every time we visit a node, we will create a node and insert it into the COFI-tree. In the FP-tree, we traverse the first node (*B*:6). There is a branch *AB* with node N_1 (*B*:6) containing *B* and node N_2 (*A*:11) containing *A*. We just need to create a new node (*A*:6) as the frequency of this node should be equal to that in node N_1 (which is the node containing the testing item *B*). Then, we insert it into the COFI-tree for item *B* (shown in the following figure). Then, the process continues until all nodes in the horizontal link of item *B* are processed. 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. Like the FP-Tree, the COFI-tree is constructed with the header constituting a list of all frequent items to maintain the location of the first entry for each item in the COFI-tree. A link is also made for each node in the tree pointing to the next node of the same item in the tree if it exists.

Finally, we mine the itemsets in the *B* COFI-tree before we remove it and create the next COFI-tree for the next item in the header table. From the *B* COFI-tree, we obtain the following itemsets:

Item	*<B:10>*
2-itemsets	<{A,B}:6>

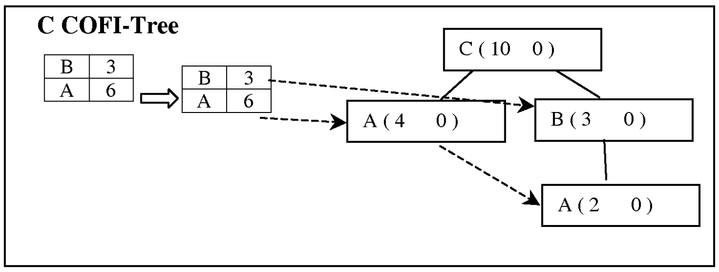
Then, we update the result sets and thresholds as follows.

Result set	*Itemsets*
R_1	{<A:11>, <B:10>}
R_2	{<{A,B}:6>}
Threshold values	
$\xi_1 = 10$, $\xi_2 = 0$, $\xi = \min\{\xi_1, \xi_2\} = 0$	

3.4 The C COFI-tree

Next we construct the *C* COFI-tree. After the first traversal we know that, with respect to item *C*, item *B* occurs three times and item *A* occurs six times. For the second traversal, we traverse the horizontal link of item *C* in the FP-tree. First, we visit the node (*C*:4). This node indicates that it shares a branch with items *A* and *C*, with support = 4. Then, a node (*A*:4) is created and is inserted into the COFI-tree. The second node we visit is (*C*:2), which shares items *A*, *B* and *C* with support = 2. So, two nodes are created for items *A* and *B* with support = 2. The third node is (*C*:1), which shares items *B* and *C* with support one. As there is a branch with node *N* containing item *B* (i.e., (*B*:2)) from the root (which was just created in the previous step), we just increment the counter in that node *N* by one. Thus, the counter in node *N* is equal to three. The fourth node we visit is (*C*:3), which shares only one item *C*. The resulting COFI-tree is shown in Figure 3.

Figure 3 *C* COFI-tree



From the *C* COFI-tree, we obtain the following itemsets:

Item	*<C:10>*
2-itemsets	<{A,C}:6>, <{B,C}:3>

We update the result sets and the thresholds as follows.

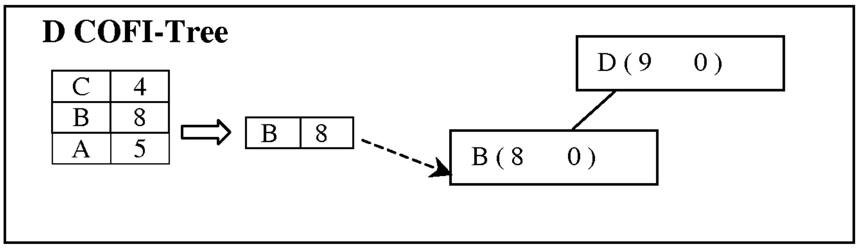
Result set	*Itemsets*
R_1	{<A:11>, <B:10>, <C:10>}
R_2	{<{A,B}:6>, <{A,C}:6>}
Threshold values	
$\xi_1 = 10$, $\xi_2 = 6$, $\xi = \min\{\xi_1, \xi_2\} = 6$	

It is noted that R_1 contains three elements, because item *B* and item *C* have the same frequency: ten. Besides, itemset {*B*, *C*} (or shortly itemset *BC*) cannot be inserted into R_1 as its frequency is smaller than six.

3.5 D COFI-tree

Recall that the overall threshold is six. In the construction of the COFI-tree for item *D*, item *C* and item *A* will thus be eliminated since they do not have enough support when appearing together with *D* (see Figure 4).

Figure 4 *D* COFI-tree



From *D* COFI-tree, we obtain the following itemsets:

Item	*<D:9>*
2-itemsets	<{B, D}:8>

We update the result sets and the thresholds as follows

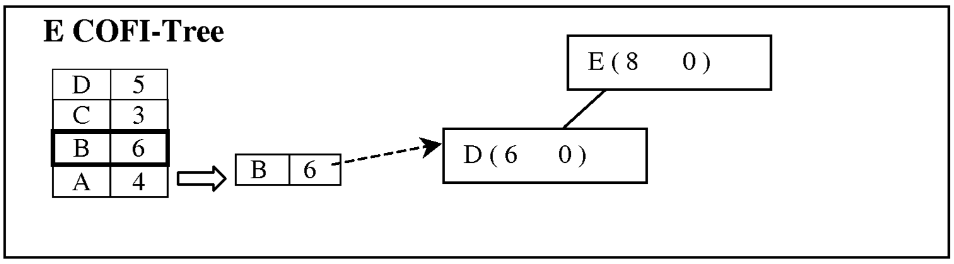
Result set	*Itemsets*
R_1	{<A:11>, <B:10>, <C:10>}
R_2	{<{B,D}:8>, <{A,B}:6>, <{A,C}:6>}
	Threshold values
	$\xi_1 = 10$, $\xi_2 = 6$, $\xi = \min\{\xi_1, \xi_2\} = 6$

It is noted that R_2 contains three elements because itemset {*A*, *B*} and itemset {*A*, *C*} have the same frequency: six; *D* is not inserted into R_1 as its frequency is smaller than ten.

3.6 *E COFI-tree*

The next frequent item is *E*. The COFI-tree for item *E* indicates that items *A*, *C* and *D* are eliminated as their frequencies of appearances with *E* are smaller than the threshold six (see Figure 5).

Figure 5 *E* COFI-tree



From the *E* COFI-tree, we obtain the following itemsets:

Item	*<E:8>*
2-itemsets	<{D,E}:6>

We update the result sets and thresholds as follows.

Result set	*Itemsets*
R_1	{<A:11>, <B:10>, <C:10>}
R_2	{<{B,D}:8>, <{A,B}:6>, <{A,C}:6>, <{D, E}:6>}
Threshold values	
$\xi_1 = 10$, $\xi_2 = 6$, $\xi = \min\{\xi_1, \xi_2\} = 6$	

It is noted that R_2 contains four elements because itemsets {*A*, *B*}, {*A*, *C*} and {*D*, *E*} have the same frequency six. Besides, item *E* cannot be inserted into R_1 as its frequency is less than ten.

3.7 *F COFI-tree*

We can construct the following *F* COFI-tree. However, all items (i.e., items *A*, *B*, *C*, *D* and *E*) are locally not frequent with respect to item *F* as the support for all these items (with respect to item *F*) are not greater than the global threshold: six. So, they are not involved in the generation of the *F* COFI-tree. The *F* COFI-tree has only one node for *F*.

From the F COFI-tree, we obtain the following itemsets.

Item	*<F:7>*
2-itemsets	

As the frequencies of all itemsets mined are smaller than the correspondence threshold, we need not update the results and the thresholds.

Next we present the pseudo code for the algorithm.

Algorithm

In general, our proposed algorithm called *COFI + BOMO algorithm* runs as follow:

1 Declare k result sets. For each result set, set maximum size as N. Initialise each with empty set. Also, keep a threshold value ξ_k or each result set.

2 Scan through transaction database D and count the support of each item. Sort them in order.

3 Build a *FP-tree* with *threshold* as 0.

4 Do COFI-tree building for each item in header table. Mine the COFI-tree.

```
4.1 Whenever a pattern is generated,
    4.1.1 If the result set of the corresponding size is not full, add the pattern to
          the result set, update the thresholds.
    4.1.2 Else if support of that pattern > corresponding threshold, add the
          pattern to the result set, update the thresholds.
```

The pseudo-code of the proposed algorithm is shown as follows.

```
Pseudo Code of COFI + BOMO:
1. Let $result_k$ be the resulting set of interesting $k$-itemsets. Set $result_k = \phi$.
2. Scan through transaction database $D$. Count the support of each item.
3. Sort the items by their supports in descending order, denoted as sorted-list.
4. $\xi_1 = \xi_2 = \xi_3 = \xi_4 = \cdots = \xi_{k\max} = \xi = 0$
5. Create FP-tree, $T$, with root node only labeled as 'NULL'.
6. Build ($T$, $D$, sorted-list, $\xi$)
7. COFI ($T$, $\xi$, $\xi_1$, $\xi_2$, …, $\xi_{k\max}$)
```

```
Function Build (T, D, sorted-list, ξ):
Build (T, D, sorted-list, ξ)
1.  sorted-list ← sorted 1-itemsets list whose support ≥ξ.
2.  For each Trans in D,
    If Trans consists of some items in sorted-list,
    Select and sort the items in Trans according to the order of sorted-list.
    Let [p|P] be the sorted frequent item list in Trans, where p is the first element and P is the remaining
    list.
    Call Insert ([p|P], T)
    {
    If T has a child C such that C.item-name= p.item-name,
    Then increment C's count by one,
    Else create a new node C, and let its count be one, it's parent link be linked to T, and its node-link be
    linked to nodes with the same item-name via the node-link structure.
    If P is non-empty, Call insert (P, C).
```

Modified COFI Algorithm for Mining N-most interesting itemsets

```
Algorithm COFI: Creating, Pruning and Mining COFI-trees
Input: FP-Tree, a minimum support threshold ξ
Output: Full set of frequent patterns
 Method:
1.  A = the most frequent item on the header table of FP-Tree
2.  While (There are still frequent items) do {
    2.1  count the frequency of all items that share item (A) in a path. Frequencies of all items
         that share the same path are the same as of the frequency of the (A) items
    2.2  Remove all non-locally frequent items for the frequent list of item (A)
    2.3  Create a root node for the (A)-COFI-tree with both frequency-count and
         participation-count = 0
         2.3.1  C is the path of locally frequent items in the path of item A to the root
         2.3.2  Items on C form a prefix of the (A)-COFI-tree.
         2.3.3  If the prefix is new then
                Set frequency-count=frequency of (A) node and participation-count=0 for all
                nodes in the path
                Else
         2.3.4  Adjust the frequency-count of the already existing part of the path.
         2.3.5  Adjust the pointers of the Header list if needed
         2.3.6  find the next node for item A in the FP-tree and go to 2.3.1
    2.4  MineCOFI-tree (A)
    2.5  Release (A) COFI-tree
    2.6  A = next frequent item from the header table}
```

```
Function: MineCOFI-tree (A)
1   nodeA = select_next_node
(Selection of nodes starting with the node of the most locally frequent item and following its
chain, until we reach the least frequent item in the Header list of the (A)-COFI-tree)
2   While there are still nodes do {
    2.1 D = set of nodes from nodeA to the root
    2.2 F = nodeA.frequency-count – nodeA.participation-count
    2.3 Generate all Candidate patterns X from items in D. Patterns that do not have A will be
        discarded.
    2.4 Patterns in X that do not exist in the A-Candidate List will be added to it with
        frequency = F otherwise just increment their frequency with F
    2.5 Increment the value of participation-count by F for all items in D
    2.6 nodeA = select_next_node}
3   Based on support threshold $\xi$ remove non-frequent patterns from A-Candidate List.
4   For each pattern S in the A-Candidate List,
    4.1 L = number of items in pattern S
    4.2 If ($L < k_{max}$) {
        4.2.1 If frequency of pattern S ≥ $\xi L$ then
            4.2.1.1 Insert pattern S into resultL; update $\xi L$; update $\xi$ if necessary.
                    {$\xi = \min(\xi_1, \xi_2, \ldots, \xi_{k_{max}})$}
        4.2.2 If ($\xi_L \neq 0$) and (result$_L$ contains itemset X with support < frequency of
              pattern S),
        4.2.3 Remove X from result$_L$}
```

4 Empirical study

After implementing the COFI + BOMO in C++, we would like to do some experiments to evaluate its performance. Before this, we have to generate some data sets. We have used the IBM synthetic data set generator for generating various synthetic data sets. We use the Pentium IV 2.2GHz PC to conduct our experiment.

We label our dataset by *Dx.Ty.Mz*, where *D* refers to the Number of transactions, *T* is the average number of items in a transaction, *M* is the number of different items, for dataset *Dx.Ty.Mz*, there are *x* transactions, each transaction has *y* items on average and there are *z* different possible items.

We aim to find the *N*-most interesting itemsets.

4.1 Performance

First, let us look at the performance of '*COFI + BOMO*' *itself only*

Figure 6 Fix k = (4, 6), varying N = (10, 20, 40, 100) using *D*1k.*T*10.*M*1k

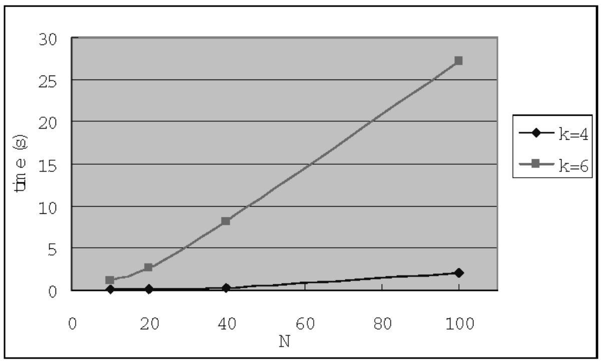
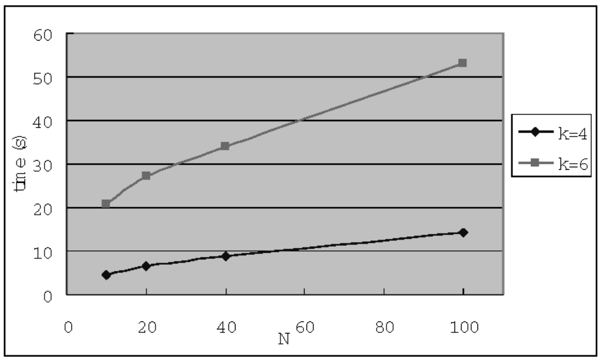
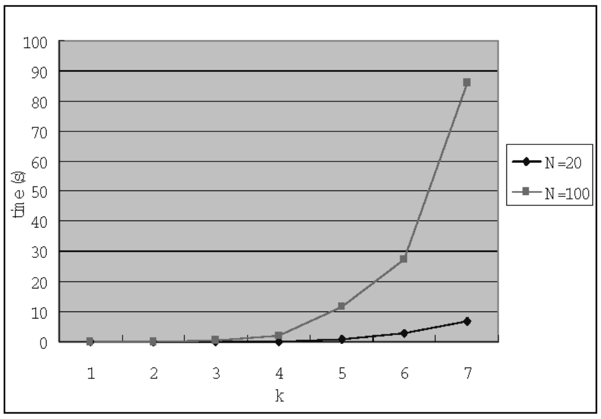


Figure 7 Fix k = (4, 6), varying N = (10, 20, 40, 100) using *D*100k.*T*10.*M*1k



The above two figures show that for a fix *k,* the total time needed to mine a result set grows almost *linearly* with *N.*

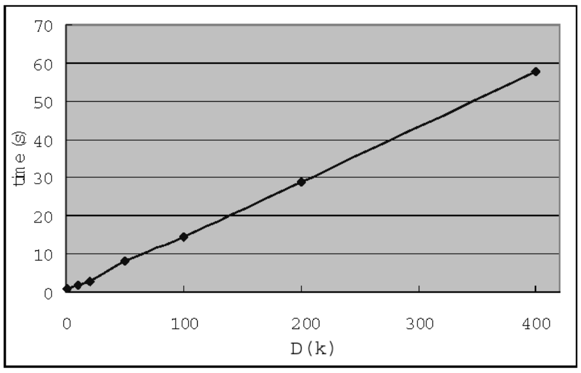
Figure 8 Fix N = (20, 100), various k = (1, 2, 3, 4, 5, 6, 7), using *D*1k.*T*10.*M*1k



This figure reveals that for $k < 4$, the mining time grows *linearly* for a wide range of N. However, as k grows larger and larger, especially when $k > 6$, the mining time tends to grow *exponentially*.

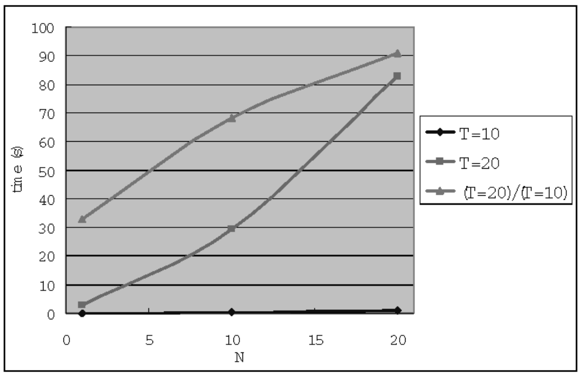
Consider the *Mine*-COFI-tree function. In the step of adding a pattern in a result set, ξ is updated by $\xi = \min(\xi_1, \xi_2, \ldots, \xi_{k_{\max}})$. If k is large, ξ will probably grow slower. When ξ is small, the pruning power when constructing the COFI-tree will be weaker. And thus, more items will be added to the COFI-tree, and more candidate patterns will be generated in COFI-mining. For example, suppose result$_1$ to result$_{k\max-1}$ are full and their corresponding thresholds are high; as long as result$_{k \max}$ is not full, $\xi_{k\max}$ will be 0. Since $\xi = \min(\xi_1, \xi_2, \ldots, \xi_{k_{\max}})$, ξ will also be 0. As a result, every item in the COFI header table will be taking part in the construction and mining of COFI-tree and this is very time-consuming.

Figure 9 Varying number of transactions, D ($N = 20$, $k = 5$, $T = 10$, M1k)



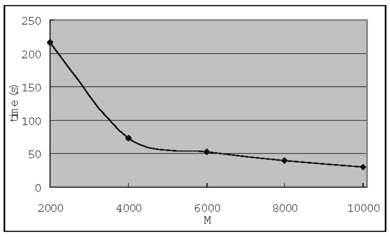
This figure clearly shows that the mining time grows *linearly* with the number of transactions.

Figure 10 Varying average number of items in a transaction**,** T ($N = 10$, $k = 4$, varying D)



As the average number of items in a transaction increases, the total mining time grows geometrically by a factor that increases linearly with the N value.

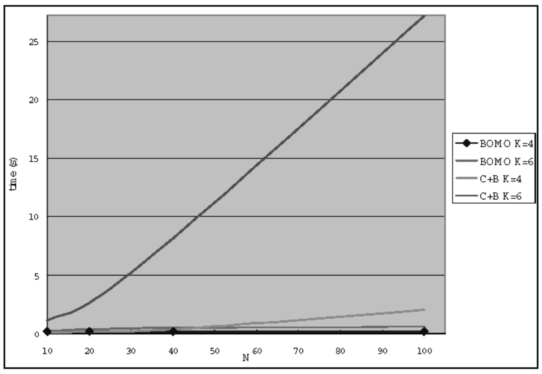
Figure 11 Varying number of items, M = (2k, 4k, 6k, 8k, 10k) (D = 100k, T = 10) Fix N = 100 Varying k = 7



The plot shows that for a fixed transaction number and average transaction size the mining time *decreases* with the number of items, *with large* N and k.

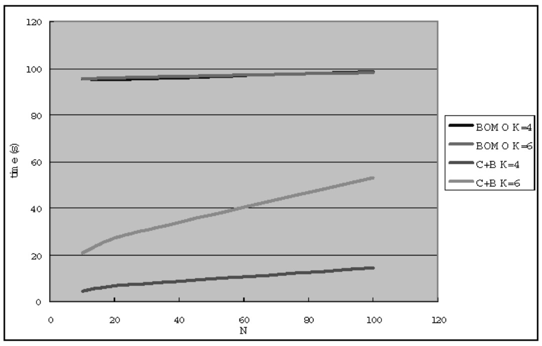
Next we compare the performance of 'COFI + BOMO' with that of BOMO.

Figure 12 Fix k = (4, 6), various N = (10, 20, 40, 100) using D1k.T10.M1k



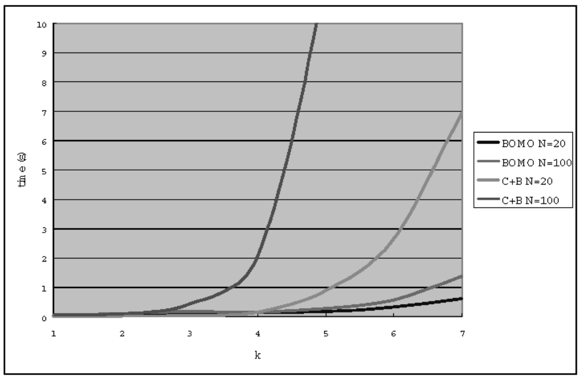
For small data set, It is clear that BOMO has better performance for most N and k. However, COFI+BOMO is better than BOMO for $N \leq 20$ and $k \leq 4$, Hence COFI+BOMO performs well for small N and k.

Figure 13 Fix k = (4, 6), various N = (10, 20, 40, 100) using D100k.T10.M1k



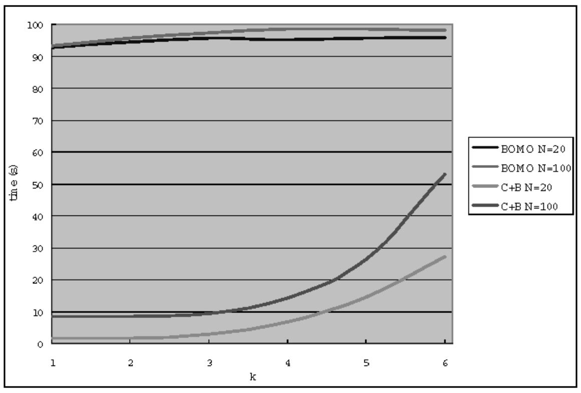
It is easy to notice that COFI+BOMO has better performance for all test cases for large data set. It is because COFI+BOMO has a good pruning power and there is no need to construct conditional sub-trees recursively.

Figure 14 Fix $N = (20, 100)$, various $k = (1, 2, 3, 4, 5, 6, 7)$, using *D*1k.*T*10.*M*1k



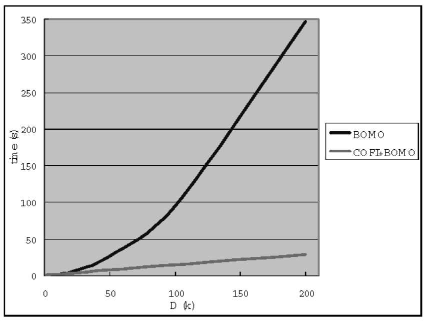
Again, for small data set, It is clear that BOMO has better performance for most *N* and *k*. However, COFI+BOMO is better than BOMO for $N \leq 20$ and $k \leq 4$, hence COFI+BOMO performs well for small *N* and *k*.

Figure 15 Fix $N = (20, 100)$, various $k = (1, 2, 3, 4, 5, 6, 7)$, using *D*100k.*T*10.*M*1k



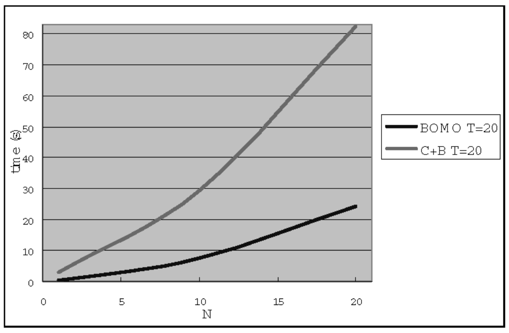
Again, it is easy to notice that COFI+BOMO shows better performance for all test cases for large data set. It is because COFI+BOMO has good pruning power and there is no need to construct conditional sub-trees recursively.

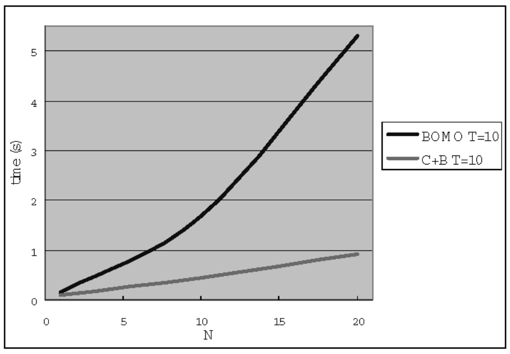
Figure 16 Varying number of transactions, *D* (*N = 20, k = 5, T = 10 M = 1k*)



From the result above, we observe that except for the case of $D < 10$k, i.e., small data set, COFI+BOMO has much better performance for all data set sizes.

Figure 17 Varying average number of items in a transaction, T ($N = 10$, $k = 4$, varying D)

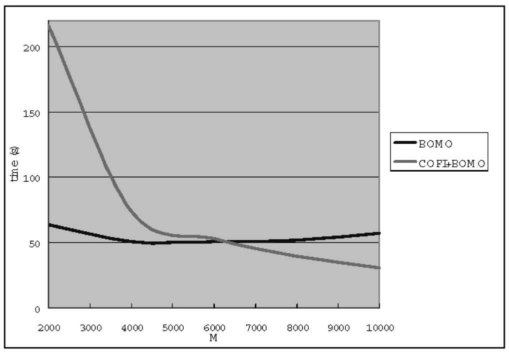




From the above two graphs, it is trivial that for data sets with, on average, more (e.g., 20) items in a transaction, BOMO is better than COFI+BOMO. However, with smaller average transaction size, COFI+BOMO out-performs BOMO.

The graph shows that for a fixed number of transactions and average transaction size, the mining time of COFI+BOMO decreases with the number of items, with large N and k while that of BOMO remains constant. Two lines intersect at the point when M is about 6000.

Figure 18 Varying number of items, M = (2k, 4k, 6k, 8k, 10k) (D = 100k, T = 10) Fix N = 100 Various k = 7



4.2 Performance conclusion:

COFI + BOMO Algorithm

Strength:	Good performance for large data set Excellent for small k value ($k \leq 4$) Works well with small average items in a transaction
Weakness:	Weak performance for large k because of the slow growth of threshold. With small data set, poor performance for $k \geq 4$ Not as good as BOMO for large average items in a transaction

5 Conclusion

We have implemented an algorithm to solve the problem of mining *N* most interesting *k*-itemsets without giving a minimum support threshold. The algorithm performs especially well with small *k*.

References

Agrawal, R., Imilienski, T. and Swami (1993) 'Mining association rules between sets of items in large databases', *Proceedings of the ACM SIGMOD*, pp.207–216.

Agrawal, R., and Srikant, R. (1994) 'Fast algorithms for mining association rules', *Proceedings of the 20th VLDB Conference*, pp.487–499.

Cheung, Y.L. and Fu, A.W-C. (2004) 'Mining association rules without support threshold: with and without item constraints', *IEEE Transactions on Knowledge and Data Engineering,* (*TKDE*), Vol. 16, No. 9, September, pp.1052–1069.

El-Hajj, M. and Zaïane, O.R. (2003) 'COFI-tree mining: a new approach to pattern growth with reduced candidacy generation', *Workshop on Frequent Itemset Mining Implementations* (*FIMI'03*) *in conjunction with IEEE-ICDM.*

El-Hajj, M. and Zaïane, O. R. (2003), 'COFI Tree Mining: A New Approach to Pattern Growth with Reduced Candidacy Generation', Workshop on Frequent Itemset Mining Implementations (FIMI'03) in conjunction with IEEE-ICDM 2003.

Han, J., Pei, J. and Yin, Y. (2000) 'Mining frequent patterns without candidate generations', *Proceeding of the ACM SIGMOD*, pp.1–12.

Notes

[1]If multiple itemsets have the same support *S*, we pick all of them according to Definition. Therefore, the resulting set may contain more than *N* itemsets. There is an extreme case where too many patterns exist with the same support *S*; in this case, we suggest that the scenario be reported to the user, instead of returning all the patterns.

[2]The threshold value is not fixed, but is updated incrementally.

[3]There is no need to construct the COFI-tree for the most frequent item *A*.

[4]The pruning step occurs when the global threshold is not equal to 0.