

δ -Tolerance Closed Frequent Itemsets

James Cheng Yiping Ke Wilfred Ng
Department of Computer Science and Engineering
The Hong Kong University of Science and Technology
{csjames, keyiping, wilfred}@cse.ust.hk

Abstract

In this paper, we study an inherent problem of mining Frequent Itemsets (FIs): the number of FIs mined is often too large. The large number of FIs not only affects the mining performance, but also severely thwarts the application of FI mining. In the literature, Closed FIs (CFIs) and Maximal FIs (MFIs) are proposed as concise representations of FIs. However, the number of CFIs is still too large in many cases, while MFIs lose information about the frequency of the FIs. To address this problem, we relax the restrictive definition of CFIs and propose the δ -Tolerance CFIs (δ -TCFIs). Mining δ -TCFIs recursively removes all subsets of a δ -TCFI that fall within a frequency distance bounded by δ . We propose two algorithms, CFI2TCFI and MineTCFI, to mine δ -TCFIs. CFI2TCFI achieves very high accuracy on the estimated frequency of the recovered FIs but is less efficient when the number of CFIs is large, since it is based on CFI mining. MineTCFI is significantly faster and consumes less memory than the algorithms of the state-of-the-art concise representations of FIs, while the accuracy of MineTCFI is only slightly lower than that of CFI2TCFI.

1 Introduction

Frequent Itemset (FI) mining [1, 2] is fundamental to many important data mining tasks such as associations [1], correlations [6], sequences [3], episodes [13], emerging patterns [8], indexing [17] and caching [18], etc. Over the last decade, a huge amount of research has been conducted on improving the efficiency of mining FIs and many fast algorithms [9] have been proposed. However, the mining operation can easily return an explosive number of FIs, which not only severely thwarts the application of FIs, but also directly affects the mining efficiency.

To address this problem, Maximal Frequent Itemsets (MFIs) [4] and Closed Frequent Itemsets (CFIs) [14] are proposed as concise representations of FIs. MFIs are also FIs but none of their proper supersets is an FI. Since an FI of

size n has $(2^n - 1)$ non-empty subset FIs, mining MFIs effectively addresses the problem of too many FIs. However, most applications are not only interested in the patterns represented by the FIs, but also require their occurrence frequency in the database for further analysis. For example, we need the frequency of the FIs to compute the support and confidence of association rules. MFIs, however, lose the frequency information of most FIs.

On the contrary, the set of CFIs is a lossless representation of FIs. CFIs are FIs that have no proper superset with the same frequency. Thus, we can retrieve the frequency of the non-closed FIs from their closed supersets. However, the definition of the closure of CFIs is too restrictive, since a CFI covers its subset only if the CFI appears in every transaction that its subset appears in. This is unusual when the database is large, especially for a sparse dataset.

In this paper, we investigate the relationship between the frequency of an itemset and its superset and propose a relaxation on the rigid definition of CFIs. We motivate our approach by the following example.

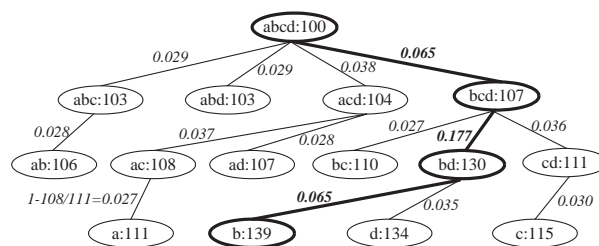


Figure 1. FIs and Their Frequency

Example 1 Figure 1 shows 15 FIs (nodes) obtained from a retail dataset, where $abcd$ is an abbreviation for the itemset $\{a, b, c, d\}$ and the number following “:” is the frequency of $abcd$.

Although we have only 1 MFI, i.e., $abcd$, the best estimation for the frequency of the 14 proper subsets of $abcd$ is that they have frequency at least 100, which is the frequency of $abcd$. However, we are certainly interested in the knowledge that the FIs b , d and bd have a frequency

significantly greater than that of other FIs. On the contrary, CFIs preserve the frequency information but all the 15 FIs are CFIs, even though the frequency of many FIs only differ slightly from that of their supersets.

We investigate the relationship between the frequency of the FIs. In Figure 1, the number on each edge is computed as $\delta = (1 - \frac{\text{frequency of } Y}{\text{frequency of } X})$, where Y is X 's smallest superset that has the greatest frequency. For CFIs, if we want to remove X from the mining result, δ has to be equal to 0, which is a restrictive condition in most cases. However, if we relax this equality condition to allow a small tolerance, say $\delta \leq 0.04$, we can immediately prune 11 FIs and retain only \mathbf{abcd} , \mathbf{bcd} , \mathbf{bd} and \mathbf{b} (i.e., the bold nodes in Figure 1). The frequency of the pruned FIs can be accurately estimated as the average frequency of the pruned FIs that are of the same size and covered by the same superset. For example, \mathbf{ab} , \mathbf{ac} and \mathbf{ad} are of the same size and covered by the same superset \mathbf{abcd} ; thus, their frequency is estimated as $\frac{106+108+107}{3} = 107$. \square

We find that a majority of the FIs mined from most of the well-known real datasets [9], as well as from the prevalently used synthetic datasets [12], exhibit the above characteristic in their frequency. Therefore, we propose to allow tolerance, bounded by a threshold δ , in the condition for the closure of CFIs, and define a new concise representation of FIs called the δ -Tolerance CFIs (δ -TCFIs). The notion of δ -tolerance greatly alleviates the restrictive definition of CFIs, as illustrated in the above example.

We propose two algorithms to mine δ -TCFIs. Our algorithm, *CFI2TCFI*, is based on the fact that the set of CFIs is a lossless representation of FIs. *CFI2TCFI* first obtains the CFIs and then generates the δ -TCFIs by checking the condition of δ -tolerance on the CFIs. However, *CFI2TCFI* becomes inefficient when the number of CFIs is large.

We study the closure of the δ -TCFIs and propose another algorithm, *MineTCFI*, which makes use of the δ -tolerance in the closure to perform greater pruning on the mining space. Since the pruning condition is a relaxation on the pruning condition of mining CFIs, *MineTCFI* is always more efficient than *CFI2TCFI*. The effectiveness of the pruning can also be inferred from Example 1 as the majority of the itemsets can be pruned when the closure definition of CFIs is relaxed.

We compare our algorithms with *FPclose* [10], *NDI* [7], *MinEx* [5] and *RPlocal* [16], which are the state-of-the-art algorithms for mining the four respective concise representations of FIs. Our experimental results on real datasets [9] show that the number of δ -TCFIs is many times (up to orders of magnitude) smaller than the number of itemsets obtained by the other algorithms. We also measure the error rate of the estimated frequency of the FIs that are recovered from the δ -TCFIs. In all cases, the error rate of *CFI2TCFI* is significantly lower than δ while that of *MineTCFI* is also

considerably lower than δ . Most importantly, *MineTCFI* is significantly faster than all other algorithms, while the memory consumption of *MineTCFI* is also small and in most cases smaller than that of the other algorithms.

Another important finding of mining δ -TCFIs is when δ increases, the error rate only increases at a much slower rate. Thus, we can further reduce the number of δ -TCFIs by using a larger δ , while still attaining high accuracy.

Organization. Section 2 gives the preliminaries. Then, Section 3 defines the notion of δ -TCFIs and Section 4 presents the algorithms *CFI2TCFI* and *MineTCFI*. Section 5 reports the experimental results. Section 6 discusses related work and Section 7 concludes the paper.

2 Preliminaries

Let $\mathcal{I} = \{x_1, x_2, \dots, x_N\}$ be a set of items. An *itemset* (also called a *pattern*) is a subset of \mathcal{I} . A *transaction* is an itemset. We say that a transaction Y *supports* an itemset X if $Y \supseteq X$. For brevity, an itemset $\{x_{k_1}, x_{k_2}, \dots, x_{k_m}\}$ is written as $x_{k_1}x_{k_2} \dots x_{k_m}$ in this paper.

Let \mathcal{D} be a database of transactions. The *frequency* of an itemset X , denoted as $\text{freq}(X)$, is the number of transactions in \mathcal{D} that support X . X is called a *Frequent Itemset (FI)* if $\text{freq}(X) \geq \sigma|\mathcal{D}|$, where σ ($0 \leq \sigma \leq 1$) is a user-specified *minimum support threshold*. X is called a *Maximal Frequent Itemset (MFI)* if X is an FI and there exists no FI Y such that $Y \supset X$. X is called a *Closed Frequent Itemset (CFI)* if X is an FI and there exists no FI Y such that $Y \supset X$ and $\text{freq}(Y) = \text{freq}(X)$.

3 δ -Tolerance Closed Frequent Itemsets

In this section, we first define the notion of δ -TCFIs. Then, we discuss how we estimate the frequency of the FIs that are recovered from the δ -TCFIs. Finally, we give an analysis on the error bound of the estimated frequency of the recovered FIs.

3.1 The Notion of δ -TCFIs

Definition 1 (δ -Tolerance Closed Frequent Itemset) *An itemset X is a δ -tolerance closed frequent itemset (δ -TCFI) if and only if X is an FI and there exists no FI Y such that $Y \supset X$, $|Y| = |X| + 1$, and $\text{freq}(Y) \geq ((1 - \delta) \cdot \text{freq}(X))$, where δ ($0 \leq \delta \leq 1$) is a user-specified frequency tolerance factor.*

We can define CFIs and MFIs by our δ -TCFIs as follows.

Lemma 1 *An itemset X is a CFI if and only if X is a 0-TCFI.*

Lemma 2 *An itemset X is an MFI if and only if X is a 1-TCFI.*

Corollary 1 *The set of all CFIs and the set of all MFIs form the upper bound and the lower bound of the set of all δ -TCFIs, respectively.*

Example 2 Referring to the 15 FIs in Figure 1. Let $\delta = 0.04$, then the set of 0.04-TCFIs is $\{b, bd, bcd, abcd\}$. For example, b is a 0.04-TCFI since b does not have a proper superset that has frequency greater than $((1 - 0.04) \times 139) = 133$. The FI a is not a 0.04-TCFI since $(1 - \frac{freq(ac)}{freq(a)}) = 0.027 < 0.04$, and similarly for ac since $(1 - \frac{freq(acd)}{freq(ac)}) = 0.037 < 0.04$ and for acd since $(1 - \frac{freq(abcd)}{freq(acd)}) = 0.038 < 0.04$. Thus, they are recursively covered by their superset that has 1 more item and then finally covered by the 0.04-TCFI $abcd$.

The set of 0.07-TCFIs is $\{bd, abcd\}$, while the set of 1-TCFIs, i.e., MFIs, is $\{abcd\}$. However, the set of 0-TCFIs, i.e., CFIs, is all the 15 FIs. \square

In the rest of the paper, we use \mathcal{F} to denote the set of all FIs and \mathcal{T} to denote the set of all δ -TCFIs, for a given δ .

3.2 Frequency Estimation

Given \mathcal{T} , we can recover \mathcal{F} (when demanded by applications). The frequency of an FI $X \in \mathcal{F}$ can be estimated from the frequency of its supersets in \mathcal{T} . We discuss the frequency estimation in this subsection.

It is possible that for an FI X , there are more than one FI Y , where $Y \supset X$, $|Y| = |X| + 1$ and $freq(Y) \geq ((1 - \delta) \cdot freq(X))$. Among all these supersets of X , the one that has the greatest frequency can best estimate the frequency of X . Thus, we define this superset as the closest superset of X as follows.

Definition 2 (Closest Superset) *Given an itemset X , let $\mathcal{Y} = \{Y : Y \supset X, |Y| = |X| + 1, \text{ and } freq(Y) = MAX\{freq(Y') : Y' \supset X, |Y'| = |X| + 1\}\}$. Y is the closest superset of X if $Y \in \mathcal{Y}$ and Y is lexicographically ordered before all other itemsets in \mathcal{Y} .*

Given an itemset X , we can follow a path of closest supersets and finally reach one closest superset, which is a δ -TCFI. We define this δ -TCFI superset as the closest δ -TCFI superset of X as follows.

Definition 3 (Closest δ -TCFI Superset) *Given n itemsets, X_1, \dots, X_n , where for $1 \leq i < n$, $X_i \subset X_{i+1}$ and $|X_{i+1}| = |X_i| + 1$. X_n is the closest δ -TCFI superset of X_1 , if $X_n \in \mathcal{T}$ and for $1 \leq i < n$, $X_i \in (\mathcal{F} - \mathcal{T})$ and X_{i+1} is the closest superset of X_i .*

Example 3 Referring to Figure 1, the closest superset of a is ac , that of ac is acd and that of acd is $abcd$. For the two supersets of ab that have the same frequency, we choose abc as the closest superset of ab since abc is ordered before abd . When $\delta = 0.04$, $abcd$ is the closest δ -TCFI superset of all its subsets that contain the item a ,

while bcd is the closest δ -TCFI superset of bc , cd and c . \square

To estimate the frequency of the FIs with the same closest δ -TCFI superset Y , we group the FIs according to their size and define the frequency extension of Y as follows.

Definition 4 (Frequency Extension) *Given a δ -TCFI Y , let $\mathcal{X}_i = \{X : |X| = |Y| - i \text{ and } Y \text{ is the closest } \delta\text{-TCFI superset of } X\}$, where $1 \leq i \leq m$ and $m = MAX\{i : \mathcal{X}_i \neq \emptyset\}$. The frequency extension of Y , denoted as $ext(Y)$, is a list $(ext(Y, 1), \dots, ext(Y, m))$, where $ext(Y, i)$, for $1 \leq i \leq m$, is defined as*

$$ext(Y, i) = \frac{\sum_{X \in \mathcal{X}_i} \frac{freq(X)}{freq(Y)}}{|\mathcal{X}_i|}.$$

The size of the frequency extension of Y , denoted as $|ext(Y)|$, is defined as $|ext(Y)| = m$.

The frequency extension of Y is essentially a list of averaged frequency ratio grouped by the size of the FIs. With the frequency extension of Y , we can estimate the frequency of each $X \in \mathcal{X}_i$, as $(freq(Y) \cdot ext(Y, i))$. We illustrate the frequency estimation by Example 4.

Example 4 Referring to Example 3, let $Y = abcd$, then $\mathcal{X}_1 = \{abc, abd, acd\}$, $\mathcal{X}_2 = \{ab, ac, ad\}$ and $\mathcal{X}_3 = \{a\}$. We have $ext(abcd, 1) = (\frac{103}{100} + \frac{103}{100} + \frac{104}{100})/3 = 1.03$, $ext(abcd, 2) = (\frac{106}{100} + \frac{108}{100} + \frac{107}{100})/3 = 1.07$ and $ext(abcd, 3) = \frac{111}{100}/1 = 1.11$.

Thus, the frequency of abc , abd and acd are estimated as $(freq(abcd) \cdot ext(abcd, 1)) = 103$, the frequency of ab , ac and ad are estimated as $(freq(abcd) \cdot ext(abcd, 2)) = 107$, while the frequency of a is estimated as $(freq(abcd) \cdot ext(abcd, 3)) = 111$. \square

3.3 Error Bound of Frequency Estimation

We now analyze the error bound of the frequency estimation. We first give Lemmas 3 and 4, which we use to define the error bound.

Lemma 3 $\forall X \in (\mathcal{F} - \mathcal{T}), \exists Y \in \mathcal{T}$ such that $Y \supset X$ and $freq(Y) \geq ((1 - \delta)^{|Y| - |X|} \cdot freq(X))$.

Lemma 4 For any δ -TCFI Y , $1 \leq ext(Y, i) \leq \frac{1}{(1 - \delta)^i}$.

Lemma 5 (Error Bound of Estimated Frequency) *Given an FI X and X 's closest δ -TCFI superset Y , where $|Y| - |X| = i$. Let $freq(X)$ be the exact frequency of X and $\widetilde{freq}(X) = (freq(Y) \cdot ext(Y, i))$ be the estimated frequency of X . Then,*

$$\phi - 1 \leq \frac{\widetilde{freq}(X) - freq(X)}{freq(X)} \leq \frac{1}{\phi} - 1,$$

where $\phi = (1 - \delta)^i$.

Proof. Since $\widetilde{freq}(X) = (freq(Y) \cdot ext(Y, i))$, by Lemma 4, we have $0 \leq freq(Y) \leq \widetilde{freq}(X) \leq \frac{freq(Y)}{\phi}$. By Lemma 3, we have $0 \leq freq(Y) \leq freq(X) \leq \frac{freq(Y)}{\phi}$. Thus, $0 \leq (freq(Y) / \frac{freq(Y)}{\phi}) \leq \frac{\widetilde{freq}(X)}{freq(X)} \leq (\frac{freq(Y)}{\phi} / freq(Y))$, i.e., $\phi \leq \frac{\widetilde{freq}(X)}{freq(X)} \leq \frac{1}{\phi}$. Hence, $(\phi - 1) \leq \frac{\widetilde{freq}(X) - freq(X)}{freq(X)} \leq (\frac{1}{\phi} - 1)$. \square

Lemma 5 gives the theoretical error bound of the frequency of an FI estimated from the frequency of its closest δ -TCFI superset. However, according to Definition 4, each $ext(Y, i)$ of a δ -TCFI Y is taken as the average of the frequency ratio of the FIs in \mathcal{X}_i over the frequency of Y , while the relative difference in the frequency of any two FIs in \mathcal{X}_i is bounded by δ . Thus, in practice, the estimated frequency is highly accurate and the error bound is much smaller than the theoretical bound defined in Lemma 5, which is also verified by our extensive experiments.

4 Mining δ -TCFIs

In this section, we first present an algorithm that computes δ -TCFIs from the set of CFIs. Then, we propose a more efficient algorithm that employs pruning based on the closure of the δ -TCFIs.

4.1 Algorithm CFI2TCFI

Mining CFIs is in general much more efficient than mining FIs. Since the set of CFIs is a lossless representation of FIs, we devise an algorithm which takes advantage of the efficiency of mining CFIs. The algorithm first generates the CFIs and then computes the δ -TCFIs from the CFIs.

Algorithm 1 CFI2TCFI

1. Mine the set of all CFIs;
 2. Let \mathcal{C}_i be the set of CFIs of size i ;
 3. **for each** $i \geq 1$ **do**
 4. **for each** $X \in \mathcal{C}_i$ **do**
 5. Find X 's closest CFI superset, Y ;
 6. **if** $(\exists Y \text{ s.t. } freq(Y) \geq (1 - \delta)^{|Y| - |X|} \cdot freq(X))$
 7. Update $ext(Y)$ with $freq(X)$ and $ext(X)$;
 8. Delete X ;
 9. **else**
 10. $\mathcal{T} \leftarrow \mathcal{T} \cup \{X\}$;
 11. **return** \mathcal{T} ;
-

Our algorithm, *CFI2TCFI*, is shown in Algorithm 1. We first generate all CFIs and partition them according to the size of the CFIs. Let \mathcal{C}_i be the set of CFIs of size i . Starting from $i = 1$, we find the *closest CFI superset* of each CFI X (Line 5). Here, the *closest CFI superset* of X is defined as X 's CFI superset that has the smallest size and the greatest

frequency among all other CFI supersets of X . If X 's closest CFI superset is not found, then X is a δ -TCFI and we include X in \mathcal{T} (Line 10). If X has a closest CFI superset Y but $freq(Y) < ((1 - \delta)^{|Y| - |X|} \cdot freq(X))$, we also include X in \mathcal{T} (Line 10). Otherwise, we update $ext(Y)$ with $freq(X)$ and $ext(X)$, if any, and then delete X (Lines 7-8).

CFI2TCFI computes the exact set of all δ -TCFIs and as we show in Section 5, the estimated frequency of the FIs recovered from the δ -TCFIs obtained by CFI2TCFI is highly accurate in all cases. However, the search for the closest CFI superset of each CFI is costly when the number of CFIs is large. Thus, we propose a more scalable algorithm whose efficiency is not affected by the number of CFIs.

4.2 Algorithm MineTCFI

In this section, we discuss a very efficient algorithm, *MineTCFI*, for mining δ -TCFIs. We first describe the data structures used in *MineTCFI* in Sections 4.2.1 and 4.2.2. Then, we discuss an effective pruning in Section 4.2.3 and present the main algorithm in Section 4.2.4.

4.2.1 FP-Tree and FP-Growth

The *pattern-growth method*, *FP-growth*, by Han et al. [11] is one of the most efficient methods for mining FIs, CFIs and MFIs [10]. We adopt the pattern-growth procedure as the skeleton of our algorithm *MineTCFI*.

FP-growth mines FIs using an extended prefix-tree structure called the *FP-tree*. As an example, Figure 2 shows the FP-tree, T_θ , constructed from a database excerpt which generates the FIs in Figure 1.

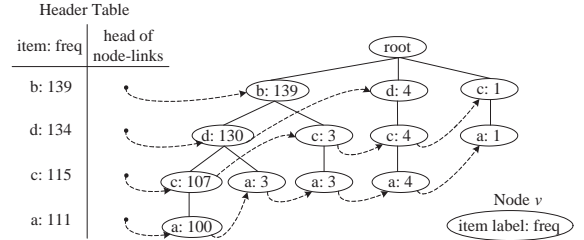


Figure 2. The FP-Tree T_θ of Figure 1

FP-growth mines the set of FIs as follows. Given an FP-tree T_X , where initially $X = \emptyset$ and T_θ is constructed from the original database. For each item x in $T_X.header$, FP-growth follows the list of pointers to extract all paths from the root to the node representing x in T_X . These paths form the *conditional pattern base* of $Y = X \cup \{x\}$, denoted as B_Y , from which FP-growth constructs a *local FP-tree*, called the *conditional FP-tree*, denoted as T_Y . First, the frequent items in B_Y form $T_Y.header$. Then, FP-growth

re-orders the frequent items in each path in B_Y (the infrequent items are discarded) and inserts the new path into T_Y . Figure 3 shows the conditional FP-tree, T_c , which is constructed from the FP-tree T_\emptyset in Figure 2.

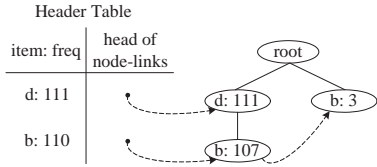


Figure 3. The Conditional FP-Tree T_c

The above procedure is applied recursively until the conditional FP-tree consists of only a single path, P , from which FP-growth generates the itemsets represented by all sub-paths of P .

4.2.2 The δ -TCFI Tree

A crucial operation in MineTCFI is the search for the supersets of an itemset in the set of δ -TCFIs already discovered. Performing a subset testing by comparing the itemset with every existing δ -TCFI is clearly inefficient. In mining CFIs, the subset testing can be efficiently processed by an FP-tree-like structure [10]. We thus develop a similar structure, called the δ -TCFI tree, to be used for mining δ -TCFIs.

To avoid testing all existing δ -TCFIs with X , a conditional δ -TCFI tree, C_X , is created corresponding to the conditional FP-tree T_X in each of the recursive pattern-growth procedure calls. Each C_X is local since it contains only δ -TCFIs that are supersets of X . Thus, this local C_X is much smaller than a global δ -TCFI tree that contains all δ -TCFIs.

Each node v in C_X has three fields: *item label*, *level* and δ -TCFI-link, where the item label indicates which item v represents, the level is the level of v in C_X (the root is at Level 0), and the δ -TCFI-link is a pointer to the δ -TCFI represented by the root-to- v path. Since each δ -TCFI has a frequency extension, we keep the δ -TCFIs in an array so that the frequency extension will not be duplicated in each of the conditional δ -TCFI trees.

Like T_X , C_X also has a header table, denoted as $C_X.header$. The items in $C_X.header$ are the same as the items in $T_X.header$ and in the same order. Each item x in $C_X.header$ is associate with an array, A_x . Each entry in A_x , $A_x[l]$, is an array of pointers to all nodes in C_X that have item label x and level l .

Example 5 If $\delta = 0.027$, we obtain seven δ -TCFIs after processing the item a . Figure 4 shows the global δ -TCFI tree, C_\emptyset , which contains the seven δ -TCFIs, and Figure 5(a) shows the conditional δ -TCFI tree, C_c , which contains only δ -TCFIs that are supersets of c . C_\emptyset and C_c correspond to

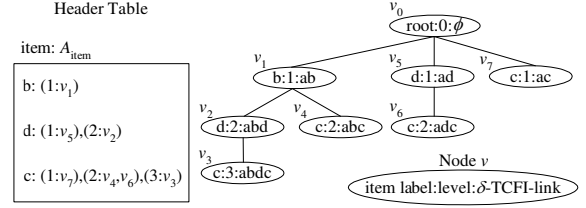


Figure 4. The Global δ -TCFI Tree C_\emptyset

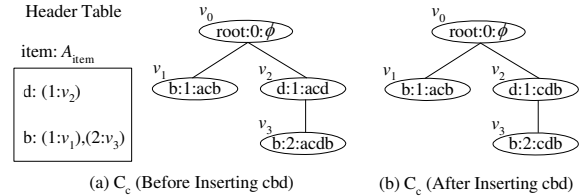


Figure 5. The Conditional δ -TCFI Tree C_c

the FP-trees T_\emptyset and T_c in Figures 2 and 3, respectively. Note that a is not in $C_\emptyset.header$ and no node in C_\emptyset represents a . This is because all δ -TCFIs containing a have already been generated and hence there is no need to include a in C_\emptyset .

In $C_\emptyset.header$ in Figure 4, “ $c: (1 : v_7), (2 : v_4, v_6), (3 : v_3)$ ” means that A_c has three entries: $A_c[1]$ has a pointer to v_7 at Level 1, $A_c[2]$ has pointers to v_4 and v_6 at Level 2, and $A_c[3]$ has a pointer to v_3 at Level 3. \square

Update and Construction of δ -TCFI Tree. To insert a δ -TCFI $Z = X \cup Y$ into C_X , we first sort the items in Y as the order of the items in $C_X.header$. Then, the sorted Y is inserted into C_X . If a prefix of the sorted Y already appears as a path in C_X , we share the prefix but change the δ -TCFI-link, *link*, of each node on the path as follows. Assume *link* currently points to W , then *link* will point to Z if either (1) $|Z| < |W|$ or (2) $|Z| = |W|$ and $freq(Z) > freq(W)$. If a new node is created for an item in Y , then its δ -TCFI-link points to Z .

To construct a conditional δ -TCFI tree, C_Y , for an item x in $C_X.header$, i.e., $Y = X \cup \{x\}$, we first initialize $C_Y.header$ based on the set of items in $T_Y.header$. Then, we access each node v in C_X via its pointer in A_x and extract the root-to- v path, P . After discarding the nodes on P that do not correspond to an item in $C_Y.header$, we re-order the remaining nodes on P according to $C_Y.header$ and then insert the path into C_Y . The insertion is the same as the way we insert Z into C_X that we just discussed above.

Example 6 To insert the δ -TCFI cbd into C_c in Figure 5(a), we first sort bd as db according to $C_c.header$ in Figure 5(a). Then, we share the path $\langle v_2, v_3 \rangle$. But the δ -TCFI-link of v_2 and v_3 will be changed to point to cdb , since $freq(cdb) > freq(acd)$ and $|cdb| < |acd|$. The δ -

TCFI tree after the insertion of `cbd` is shown in Figure 5(b), where $C_c.header$ remains unchanged as in Figure 5(a). \square

4.2.3 Closure-Based Pruning

The efficiency of CFI mining is mainly due to the pruning based on the closure of CFIs. We make use of the tolerance in the closure of the δ -TCFIs to achieve greater pruning in MineTCFI.

The pruning is described as follows. Given an FI X and X 's conditional FP-tree T_X . Let $Y = X \cup \{x\}$, where x is an item in $T_X.header$, and $\mathcal{F}(T_Y)$ be the set of FIs to be generated from Y 's conditional FP-tree T_Y . We say that Y is covered if there exists a δ -TCFI Z such that $Z \supset Y$ and $freq(Z) \geq ((1 - \delta)^{|Z|-|Y|} \cdot freq(Y))$. At the time when we generate Y , if Y is already covered, then we prune all FIs in $\mathcal{F}(T_Y)$ and thus T_Y will not be constructed.

The above pruning can be directly applied to mine 0-TCFIs (i.e., CFIs), since the FIs in $\mathcal{F}(T_Y)$ must already be covered by some 0-TCFIs that are found before we generate Y . However, when $\delta > 0$, a minority of FIs in $\mathcal{F}(T_Y)$ may not be covered by any existing δ -TCFI due to the frequency tolerance in the closure. Some of this minority of FIs may later become δ -TCFIs. However, only a very small number of these FIs will become δ -TCFIs. Missing these δ -TCFIs will only slightly degrade the accuracy of the estimated frequency of the recovered FIs, while we can still recover all FIs from their other δ -TCFI supersets. But to improve the accuracy of the estimated frequency, we apply an additional checking to prevent pruning these potential δ -TCFIs, as described by the following heuristic.

Heuristic 1 *Let H be the set of frequent items in Y 's conditional pattern base, B_Y , and $Y' = Y \cup H$. If Y is covered and Y' is also covered, then we prune all FIs in $\mathcal{F}(T_Y)$.*

Heuristic 1 is based on the proximity of frequency of the itemsets found in most datasets: if Y is covered and Y' , which is the largest possible superset of Y that can be generated from T_Y , is also covered, then most likely other FIs in-between Y and Y' are also covered.

However, at the time when we generate Y , the frequency of Y' has not been determined and hence we cannot check the condition whether Y' is covered. However, we find that if there exists a δ -TCFI, Z' , which is a superset of Y' , then in most cases Y' is covered (due to the proximity of frequency). Thus, we obtain the following heuristic.

Heuristic 2 *If Y is covered and there exists a δ -TCFI, Z' , such that $Z' \supset Y'$, then we prune all FIs in $\mathcal{F}(T_Y)$.*

Heuristic 2 implies that we only need to check the subset-superset condition without knowing the frequency of Y' . To further increase the probability that other FIs in $\mathcal{F}(T_Y)$ are also covered, we can add one more level of checking that

$|ext(Z')| \geq (|Z'| - |Y| - 1)$, which means that Z' has already covered subsets of size from $(|Y| + 1)$ to $(|Z'| - 1)$. Let U and V be any two such subsets covered by Z' , where $|V| = |U| + 1$, then the difference between the frequency of U and that of V is bounded by δ . Since the FIs in $\mathcal{F}(T_Y)$ also share the same superset Z' , this proximity of frequency of other subsets of Z' implies a high probability that the FIs in $\mathcal{F}(T_Y)$ are also covered. Thus, we obtain Heuristic 3.

We first define that Y' is *conditionally covered* by a δ -TCFI Z' if $Z' \supset Y'$ and $|ext(Z')| \geq (|Z'| - |Y| - 1)$.

Heuristic 3 *If Y is covered and there exists a δ -TCFI, Z' , such that Y' is conditionally covered by Z' , then we prune all FIs in $\mathcal{F}(T_Y)$.*

Example 7 Based on T_\emptyset in Figure 2, if $\delta=0.07$, we first find `abdc` is a 0.07-TCFI after processing `a`. Then, when we process `c` in $T_\emptyset.header$, there are two frequent items $\{d, b\}$ in B_c , from which we can generate `cb`, `cbd` and `cd`. Since `c` is covered by `abdc`, $(c \cup \{d, b\}) \subset abdc$ and $|ext(abdc)| = 3 > (|abdc| - |c| - 1) = 2$, we can be sure that the frequency of `cb`, `cbd` and `cd` can be estimated with $ext(abdc)$. Thus, we can prune `cb`, `cbd` and `cd`.

Note that if $\delta=0.04$, then `abdc` does not cover `c`. Hence, we will continue from `c` and find the δ -TCFI `cbd`. \square

Coverage Testing. We now discuss how Heuristic 3 can be efficiently processed using the δ -TCFI tree.

Given an FI X and X 's δ -TCFI tree C_X , let $Y = X \cup \{x\}$, where x is an item in $C_X.header$. We find the superset of Y in C_X as follows. We access A_x in $C_X.header$ and follow the pointers in $A_x[i]$ ($i \geq 1$, starting from $i = 1$) to visit the nodes that have item label x and are at Level i of C_X . For each node v visited, let v 's δ -TCFI-link point to Z , we check if Y is covered by Z by testing $freq(Z) \geq ((1 - \delta)^{|Z|-|Y|} \cdot freq(Y))$.

If Y is covered by Z , then Heuristic 3 requires us to check if $Y' = Y \cup H$ is conditionally covered, where H is the set of frequent items in B_Y . To check this, we first sort the items in H as their order in $C_X.header$. Let the sorted H be $H = x_1x_2 \cdots x_k$. We access A_{x_k} of the item x_k in $C_X.header$.

We first process $A_{x_k}[k]$, which contains the pointers to the nodes at Level k in C_X . For each node v accessed via a pointer in $A_{x_k}[k]$, we check if the root-to- v path represents a superset of H . The checking starts from v 's parent up to the root and we compare both the item label and the level of each node along the path. When we compare x_i ($1 \leq i \leq k - 1$) with a node u , if u 's level is smaller than i , we stop the comparison and move on to process the next node pointer in $A_{x_k}[k]$, and then the pointers in $A_{x_k}[k + 1]$ when we finish $A_{x_k}[k]$ and so on.

Since C_X is a local δ -TCFI tree containing only δ -TCFIs that are supersets of X , the number of comparisons is usually small. In addition, those δ -TCFIs that are accessed via

pointers in $A_{x_k}[i]$ ($\forall i < k$) are not compared, since the paths from the root to those nodes have less nodes than the number of items in H and hence cannot be supersets of H . In the same way, the level of a node also helps terminate many of the subset testings earlier.

When a root-to- v path is found to be a superset of H , let v 's δ -TCFI-link point to Z' , we check if Y' is conditionally covered by Z' by testing $|ext(Z')| \geq (|Z'| - |Y| - 1)$. If Y' is conditionally covered by Z' , Heuristic 3 is then applied and all FIs in $\mathcal{F}(T_Y)$ are pruned.

In MineTCFI, if Y is covered by Z and Y' is conditionally covered (by Z'), we need to determine if Z is the closest δ -TCFI superset of Y in order to update the frequency extension of Z . To do this, we need to check whether the size of Z is the smallest among all δ -TCFIs that are supersets of Y . But this does not mean that we need to process all δ -TCFIs that are supersets of Y . We do not process any of the δ -TCFIs that are accessed via the pointers in $A_x[j]$, $\forall j > |Z| - |X|$, because the pointers in $A_x[j]$ link to δ -TCFIs of size at least $(|X| + j) > |Z|$. In most cases, Y 's closest δ -TCFI superset is found via a pointer in $A_x[1]$ and rarely do we go through many entries of A_x .

4.2.4 Algorithm MineTCFI

We now present our algorithm, *MineTCFI*, as shown in *Algorithm 2*. After constructing the global FP-tree T_\emptyset , MineTCFI invokes the recursive pattern-growth procedure *GenTCFI*, which is shown in *Procedure 1*.

In Procedure 1, the processing of *IsCovered* (Lines 4 and 14), *IsCondCovered* (Line 15) and the search for the closest δ -TCFI superset (Lines 5 and 16) are discussed in Coverage Testing in Section 4.2.3. Procedure 1 can be divided into two parts: when the input conditional FP-tree, T_X , consists of only one single path (Lines 1-9), and when T_X has more than one path (Lines 10-23).

When T_X consists of only one single path P , GenTCFI generates all itemsets which satisfy locally the condition of a δ -TCFI. Then, for each local δ -TCFI Y , GenTCFI checks if Y is covered. If Y is not covered, then Y is a δ -TCFI and we add it to \mathcal{T} (Line 8). GenTCFI also inserts Y into all the conditional δ -TCFI trees which are constructed along the path of the previous recursive calls of GenTCFI (Line 9), so that the future recursive calls can construct their conditional δ -TCFI trees correctly. If Y is covered, GenTCFI finds Y 's closest δ -TCFI superset Z from C_X and updates Z 's frequency extension with the frequency of Y (Lines 5-6).

When T_X consists of more than one path, GenTCFI processes each item x in $T_X.header$ as follows. First, GenTCFI constructs the conditional pattern base B_Y of $Y = X \cup \{x\}$. Let H be the set of frequent items in B_Y . If Y is covered and $(Y \cup H)$ is conditionally covered, by Heuristic 3, GenTCFI prunes all supersets of Y that are to

Algorithm 2 MineTCFI

1. Construct the global FP-tree, T_\emptyset ;
 2. Initialize the global δ -TCFI tree, C_\emptyset ;
 3. $\mathcal{T} \leftarrow \emptyset$;
 4. Invoke **GenTCFI**($T_\emptyset, C_\emptyset, \mathcal{T}$);
 5. Return \mathcal{T} ;
-

Procedure 1 GenTCFI(T_X, C_X, \mathcal{T})

1. **if**(T_X is a single path, P)
 2. Generate all local δ -TCFIs from P ;
 3. **for each** local δ -TCFI, Y , generated **do**
 4. **if**(*IsCovered*(Y, C_X) = *true*)
 5. Find Y 's closest δ -TCFI superset, Z ;
 6. Update *ext*(Z) with *freq*(Y);
 7. **else**
 8. $\mathcal{T} \leftarrow \mathcal{T} \cup \{Y\}$;
 9. Insert Y into all C_X 's predecessor δ -TCFI trees in the recursive-call stack;
 10. **else**
 11. **for each** x in $T_X.header$ **do**
 12. $Y \leftarrow X \cup \{x\}$;
 13. Let H be the set of frequent items in B_Y ;
 14. **if**(*IsCovered*(Y, C_X) = *true*)
 15. **if**(*IsCondCovered*($Y \cup H, C_X$) = *true*)
 16. /* Prune all supersets of Y */
 17. Find Y 's closest δ -TCFI superset, Z ;
 18. Update *ext*(Z) with *freq*(Y);
 19. **else**
 20. Construct Y 's conditional FP-tree, T_Y ,
 21. and Y 's conditional δ -TCFI tree, C_Y ;
 22. **GenTCFI**(T_Y, C_Y, \mathcal{T});
 23. **else** /* *IsCovered*(Y, C_X) = *false* */
 24. Construct Y 's conditional FP-tree, T_Y ,
 25. and Y 's conditional δ -TCFI tree, C_Y ;
 26. **GenTCFI**(T_Y, C_Y, \mathcal{T});
-

be generated from T_Y (Lines 14-17). Otherwise, GenTCFI constructs Y 's conditional FP-tree T_Y and conditional δ -TCFI tree C_Y (Lines 19 and 22). The recursive procedure is then called to process on T_Y and C_Y (Lines 20 and 23).

5 Experimental Results

We now evaluate our approach of mining δ -TCFIs. We run all experiments on a PC with an Intel P4 3.2GHz CPU and 2GB RAM, running Linux 64-bit.

Datasets. We use the real datasets from the popular FIMI Dataset Repository [9]. We choose three datasets with the following representative characteristics. For a wide range of values of σ :

- **pumsb***: the number of CFIs is orders of magnitude smaller than that of FIs, but is orders of magnitude larger than that of MFIs.

- accidents: the number of CFIs is almost the same as that of FIs, and is orders of magnitude larger than that of MFIs.
- mushroom: the number of CFIs is orders of magnitude smaller than that of FIs, but is only a few times larger than that of MFIs.

Algorithms for Comparison. We compare our algorithms CFI2TCFI and MineTCFI with the following algorithms:

- *FPclose* [10]: the winner of *FIMI* 2003 [9] and one of the fastest public implementations for mining CFIs.
- *NDI* [7]: the algorithm (the faster DFS approach) for computing the set of *non-derivable FIs (NDIs)*.
- *MinEx* [5]: the algorithm for mining the set of frequent δ -free-sets.
- *RPlocal* [16]: the faster algorithm (than *RPglobal*) for computing the *representative patterns of the δ -clusters*.

5.1 Performance at Different Minimum Support Thresholds

We first study the performance of the different algorithms by varying the minimum support threshold σ . We fix $\delta=0.05$ for both *pumsb** and *accidents*. For *mushroom*, since the difference between the number of CFIs and that of MFIs is much smaller than the other two datasets, we set a larger $\delta = 0.2$ to obtain a greater reduction for the algorithms with the parameter δ .

We use the same δ for CFI2TCFI, MineTCFI and *RPlocal*. However, the δ defined in *MinEx* is an absolute value. Thus, in each case we compare with *MinEx*, we find a δ for *MinEx* such that the error rate of *MinEx* approximately matches that of MineTCFI.

5.1.1 Number of Itemsets and Error Rate

We compare the size of each of the concise representations of FIs. For simplicity, we use $Num(alg)$ to denote the number of itemsets obtained by the algorithm *alg*.

Figures 6 to 8 report the number of itemsets returned by each algorithm. In most cases, $Num(CFI2TCFI)$ and $Num(MineTCFI)$ are about an order of magnitude smaller than $Num(FPclose)$ and $Num(NDI)$, many times smaller than $Num(MinEx)$, and on average 2 times smaller than $Num(FPclose)$. In all cases, the number of δ -TCFIs obtained by both MineTCFI and CFI2TCFI is very close to the number of MFIs.

Table 1 shows the error rate of the estimated frequency of the FIs recovered from the δ -TCFIs. We can see the error rate of CFI2TCFI is much lower than δ in all cases. The error rate of MineTCFI is higher but still lower than δ , especially that for *mushroom* is only 1/10 of δ . The error rate

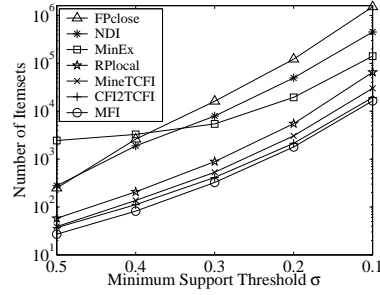


Figure 6. Number of Itemsets (*pumsb**)

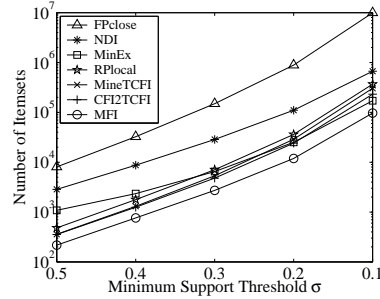


Figure 7. Number of Itemsets (*accidents*)

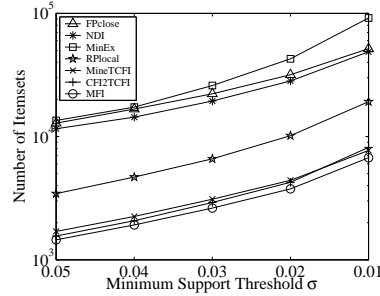


Figure 8. Number of Itemsets (*mushroom*)

of MineTCFI is higher than CFI2TCFI because MineTCFI is only able to include partially the frequency of the subsets of a δ -TCFI in its frequency extension, as some of the subsets are pruned. The error rate of *MinEx* is the same as that of MineTCFI. NDIs and CFIs are lossless representations of FIs, while the error rate of *RPlocal* is bounded by δ .

	<i>pumsb*</i> ($\delta = 0.05$)	<i>accidents</i> ($\delta = 0.05$)	<i>mushroom</i> ($\delta = 0.2$)
CFI2TCFI	0.01	0.01	0.01
MineTCFI	0.03	0.04	0.02

Table 1. Error Rate of Estimated Frequency

5.1.2 Running Time and Memory Consumption

Figure 9 reports the running time and memory consumption of the algorithms. We truncate the time and memory that are orders of magnitude larger than the largest points presented

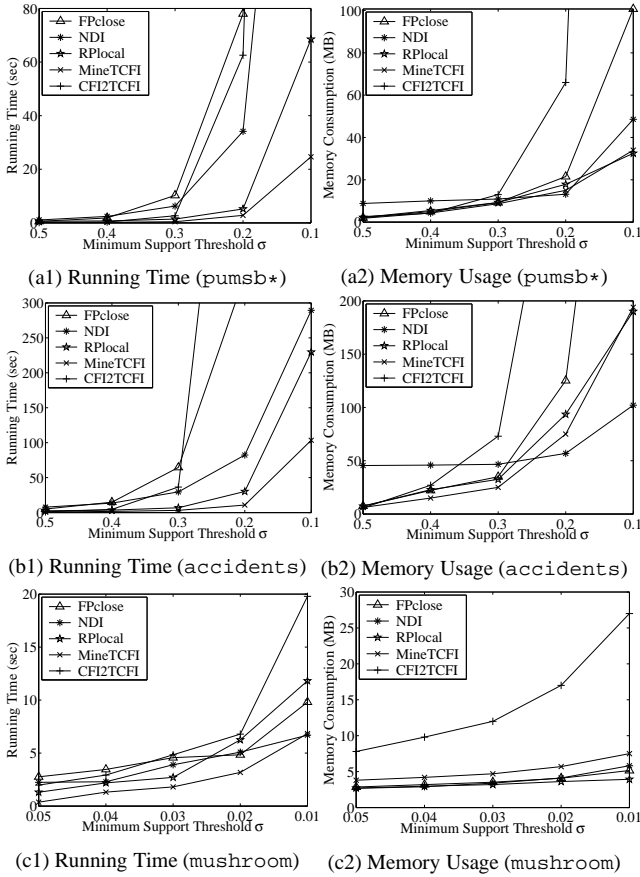


Figure 9. Time and Memory for Varying σ

in the respective figures, since most of the time and memory usage are small and will be squeezed into a single line if we use a logarithmic scale.

It is obvious from Figures 9 (a1), (b1) and (c1) that MineTCFI, which is the lowest line in all figures, is significantly faster than all other algorithms. The running time of RPlocal is the closest to that of MineTCFI but still about 3 times longer on average. CF12TCFI is also fast in most of the cases, except when the number of CFIs is large.

The memory consumption of the algorithms is small in most cases, except that CF12TCFI and FPclose use more memory when the number of CFIs is large. For mushroom as shown in Figure 9 (c2), MineTCFI consumes more memory than other algorithms but the difference is only 2MB. However, in most of the other cases, MineTCFI has the lowest memory consumption among all algorithms, as shown in Figures 9 (a2) and (b2).

5.2 Effect of Different Values of δ

We now study the effect of different values of δ on mining δ -TCFIs. We test on the two larger datasets pumsb*

and accidents. We fix σ at 0.3 and vary δ from 0.001 (a sufficiently low error rate in our opinion) to 0.2 (a δ at which the set of δ -TCFIs is almost the set of MFIs).

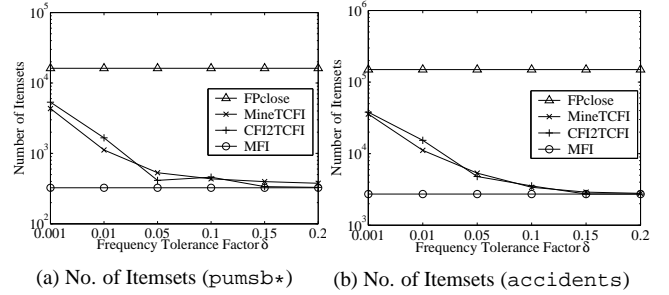


Figure 10. Different Values of δ

Figures 10 (a) and (b) show the number of δ -TCFIs obtained by CF12TCFI and MineTCFI, as well as the number of CFIs and MFIs as references. The number of δ -TCFIs is about 4 to 5 times smaller than that of CFIs at $\delta = 0.001$ and already becomes over an order of magnitude smaller at $\delta = 0.01$. The number of δ -TCFIs is within 2 times of that of MFIs at $\delta = 0.05$ and is almost the same as that of MFIs at $\delta = 0.2$.

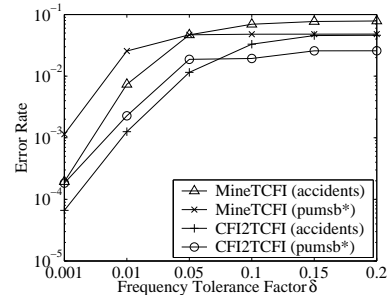


Figure 11. Error Rate of Different δ

Figure 11 shows the error rate of CF12TCFI and MineTCFI for pumsb* and accidents. At $\delta = 0.001$, the error rate of CF12TCFI and MineTCFI is significantly (up to 20 times) smaller than δ , except that of MineTCFI for pumsb* which is approximately 0.001. The error rate increases only slightly for large values of δ . For pumsb* at $0.05 \leq \delta \leq 0.2$ and accidents at $0.1 \leq \delta \leq 0.2$, the error rate increases only within the range of 0.01.

This result shows that the actual error rate does not grow with the theoretical error bound given in Lemma 5, but remains to be small when δ becomes large. This is an important finding since for many applications the user is allowed to specify a large δ , while we can still achieve high accuracy, which is not largely affected by δ , and obtain a very concise set of δ -TCFIs. The small error rate also demon-

strates the need for the frequency extension of a δ -TCFI in maintaining high accuracy of the estimated frequency.

6 Related Work

In addition to MFIs [4] and CFIs [14] we have discussed in Section 1, we are aware of the work by Xin et al. [16] which uses a similar notion of closeness measure of frequency by δ . They define a set of itemsets, S , to form a cluster if $\exists Y$ (called a *representative pattern*), such that $\forall X \in S, X \subseteq Y$ and $(1 - \frac{freq(Y)}{freq(X)}) \leq \delta$. However, this definition is non-recursive, while the definition of our δ -TCFIs removes the redundant subsets recursively. Thus, our approach is able to achieve better compression as evidenced by experimental results. We note that the number of δ -TCFIs can be significantly reduced when a relaxed minimum support threshold is used as in [16]. However, in our experiments, we do not relax the minimum support threshold for both RPlocal and our algorithms, as to be fair to other algorithms under comparison.

Boulicaut et al. [5] define an itemset X as a δ -free-set if $\forall X' \subseteq X, \nexists Y \subset X'$ such that $(freq(Y) - freq(X')) \leq \delta$. The frequency of an FI X is estimated from its subsets; thus, an extra set of border itemsets is required in order to determine whether X is frequent. Calders and Goethals [7] utilize the inclusion-exclusion principle to deduce the lower bound and the upper bound for the frequency of an itemset and define an itemset as non-derivable if the lower bound and the upper bound are not equal. The set of NDIs is a lossless representation of FIs but can be still too large in some cases. Pei et al. [15] propose two types of condensed FI bases to approximate the frequency of itemsets with a user-defined error bound k . The frequency of an FI can be derived from either its subsets or supersets in the FI base.

7 Conclusions

We propose δ -TCFIs as a concise and flexible representation of FIs. The notion of δ -tolerance allows us to flexibly tune δ to enjoy the benefits of both MFIs and CFIs: we can prune a great amount of redundant patterns from the mining result as do MFIs, while we can retain the frequency information of the recovered FIs as do CFIs. Experimental results verify that in all cases, the number of δ -TCFIs is very close to the number of MFIs and much smaller than all other existing concise representations of FIs [10, 7, 5, 16]. The results also show that the actual error rate of the estimated frequency of the recovered FIs is much lower than the theoretical error bound. In particular, our algorithm CFI2TCFI attains an error rate significantly lower than δ in all cases. CFI2TCFI is also shown to be very efficient in most cases except when the number of CFIs is large. Our second algorithm MineTCFI attains an accuracy slightly lower than that

of CFI2TCFI; however, MineTCFI is significantly faster than all other algorithms [10, 7, 5, 16] in all cases and also consumes less memory in most cases.

Acknowledgement. This work is partially supported by RGC CERG under grant number HKUST6185/03E. The authors would like to thank Prof. Gösta Grahne for providing us FPclose, Dr. Bart Goethals for providing us NDI, Prof. Christophe Rigotti for providing us MinEx, and Prof. Jiawei Han and Mr. Dong Xin for providing us RPlocal.

References

- [1] R. Agrawal, T. Imielinski, and A. N. Swami. Mining Association Rules between Sets of Items in Large Databases. In *Proc. of SIGMOD*, 1993.
- [2] R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules. In *Proc. of VLDB*, 1994.
- [3] R. Agrawal and R. Srikant. Mining Sequential Patterns. In *Proc. of ICDE*, 1995.
- [4] R. Bayardo. Efficiently Mining Long Patterns from Databases. In *Proc. of SIGMOD*, 1998.
- [5] J.F. Boulicaut, A. Bykowski and C. Rigotti. Free-Sets: a Condensed Representation of Boolean Data for the Approximation of Frequency Queries. In *DMKD* 7(1):5-22, 2003.
- [6] S. Brin, R. Motwani, and C. Silverstein. Beyond Market Basket: Generalizing Association Rules to Correlations. In *Proc. of SIGMOD*, 1997.
- [7] T. Calders and B. Goethals. Mining All Non-derivable Frequent Itemsets. In *Proc. of PKDD*, 2002.
- [8] G. Dong and J. Li. Efficient Mining of Emerging Patterns: Discovering Trends and Differences. In *Proc. of KDD*, 1999.
- [9] B. Goethals and M. Zaki. FIMI 2003 workshop. In *Proc. of the ICDM Workshop on Frequent Itemset Mining Implementations*, 2003.
- [10] G. Grahne and J. Zhu. Efficiently Using Prefix-trees in Mining Frequent Itemsets. In *IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI 03)*, 2003.
- [11] J. Han, J. Pei, and Y. Yin. Mining Frequent Patterns without Candidate Generation. In *Proc. of SIGMOD*, 2000.
- [12] IBM Quest Data Mining Project. The Quest retail transaction data mining generator. <http://www.almaden.ibm.com/software/quest/>, 1996.
- [13] H. Mannila, H. Toivonen, and A. I. Verkamo. Discovery of Frequent Episodes in Event Sequences. In *DMKD*, 1:259-289, 1997.
- [14] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering Frequent Closed Itemsets for Association Rules. In *Proc. of ICDT*, 1999.
- [15] J. Pei, G. Dong, W. Zou, and J. Han. Mining Condensed Frequent-Pattern Bases. In *Knowl. Inf. Syst.* 6(5): 570-594, 2004.
- [16] D. Xin, J. Han, X. Yan, and H. Cheng. Mining Compressed Frequent-Pattern Sets. In *Proc. of VLDB*, 2005.
- [17] X. Yan, P. Yu, and J. Han. Graph Indexing: A frequent Structure-Based Approach. In *Proc. of SIGMOD*, 2004.
- [18] L. H. Yang, M. L. Lee, W. Hsu. Efficient Mining of XML Query Patterns for Caching. In *Proc. of VLDB*, 2003.