An Efficient Approach for Mining Fault-Tolerant Frequent Patterns based on Bit Vector Representations

Jia-Ling Koh and Pei-Wy Yo

Department of Information and Computer Education
National Taiwan Normal University
Taipei, Taiwan
Email: jlkoh@ice.ntnu.edu.tw

Abstract

*In this paper, an algorithm, called VB-FT-Mine (**V**ectors-**B**ased **F**ault–**T**olerant frequent patterns **Mining**), is proposed for mining fault-tolerant frequent patterns efficiently. In this approach, fault–tolerant appearing vectors are designed to represent the distribution that the candidate patterns contained in data sets with fault-tolerance. VB-FT-Mine algorithm applies depth-first pattern growing method to generate candidate patterns. The fault-tolerant appearing vectors of candidates are obtained systematically, and the algorithm decides whether a candidate is a fault-tolerant frequent pattern quickly by performing vector operations on bit vectors. The experimental results show that VB-FT-Mine algorithm has better performance on execution time significantly than FT-Apriori algorithm proposed previously.*

1. Introduction

Among the various data mining applications, mining association rules is an important one [1]. Several efficient algorithms have been proposed for finding frequent patterns and association rules are derived from the frequent patterns, such as the Apriori[1], DHP[3], and FP-growth[2]. When mining frequent patterns, an expected minimum support may cause only few frequent patterns are discovered because real-world data tends to be dirty. Although much more specific frequent patterns could be obtained by lowering the minimum supports, no general information about the representative frequent patterns is returned. The problem of mining fault-tolerant frequent patterns (itemsets) was defined and solved in [5] by proposing FT-Apriori algorithm. FT-Apriori algorithm was extended from Apriori Approach, in which downward closure property is applicable for mining fault-tolerant frequent patterns. Similar to Apriori-like[1] algorithms, FT-Apriori algorithm suffered from generating a large number of candidates and repeatedly scanning database. Moreover, fault toleration usually introduces much huge number of candidates when increasing the fault tolerance or decreasing the support thresholds.

In this paper, an algorithm, called VB-FT-Mine (**V**ector-**B**ased **F**ault–**T**olerant frequent patterns **Mining**), is proposed for speeding up the process of mining fault-tolerant frequent patterns. In this approach, fault–tolerant appearing vectors are designed to represent the distribution that the candidate patterns contained in data sets with fault-tolerance. VB-FT-Mine algorithm applies depth-first pattern growing method to generate candidate patterns. The fault-tolerant appearing vectors of candidate patterns are obtained systematically, and the algorithm decides whether a candidate is a fault-tolerant frequent pattern quickly by performing vector operations on bit vectors. The experimental results show that VB-FT-Mine algorithm has better performance significantly on execution time than FT-Apriori algorithm[5] proposed previously.

The remaining of this paper is organized as follows. The problem of fault-tolerant frequent pattern mining is defined in Section 2. In Section 3, the bit vector representation is introduced and applied to develop the proposed VB-FT-Mine Algorithm. The performance study of VB-FT-Mine is reported in Section 4, which shows the efficiency comparison with FT-Apriori. Finally, Section 5 concludes this paper.

2. Preliminaries

The following definitions refer to [1]. Let $I = \{i_1, i_2, \ldots, i_m\}$ be a set of literals, called *items*. A set of items is called an *itemset*. Itemsets containing k items are called *k-itemsets*. Let *DB* be a database of transactions, where each transaction T in *DB* is an itemset such that $T \subseteq I$. For a given itemset $X \subseteq I$, we say that a transaction T *contains* itemset X if and only if $X \subseteq T$. The *support count* of an itemset X in *DB*, denoted as Sup_X, is the number of transactions in *DB* containing X. Given a *minimum support threshold* s, an itemset X is called a *frequent* pattern in *DB* if $Sup_X \geq s$. Otherwise, X is named an *infrequent* pattern.

Transaction ID	Items
T1	BDEF
T2	ACDE
T3	BEFG
T4	CFG
T5	ABDEG

(a)

Item	Appearing Vector
A	<0,1,0,0,1>
B	<1,0,1,0,1>
C	<0,1,0,1,0>
D	<1,1,0,0,1>
E	<1,1,1,0,1>
F	<1,0,1,1,0>
G	<0,0,1,1,1>

(b)

Figure 1: Sample database TDB and its appearing vector table

Example 1 In the transaction database TDB shown in Figure 1(a), if the minimum support threshold is set to 4, E is the only one frequent pattern. Although lowering the minimum support threshold to 3 will get 7 frequent patterns (B, D, E, F, G, BD, and DE), the result still consists of short patterns and which is less representative (with lower support). However, observe the transactions in database TDB closely, three transactions T1, T3, and T5 contain four out of the five items: B, D, E, F, and G. Therefore, when checking whether a transaction containing a pattern with fault-tolerance(contain 4 out of 5 items), a longer "approximate" pattern (BDEFG) with support count 4 is obtained. This problem of mining *fault-tolerant frequent patterns* was defined in [5].

Definition 1 (Fault-tolerant support): Given a **fault tolerance** δ ($\delta > 0$) and an itemset P. A transaction T＝(tid, S) is said to **FT-contain** itemset P under fault tolerance δ iff there exists $P' \subseteq P$ such that $P' \subseteq S$ and $|P'| \geq (|P| - \delta)$. The number of transactions in a database DB which FT-contain itemset P is called the FT-support of P under fault tolerance δ, denoted as

FT-sup$^{\delta}$(P). The set of transactions FT-containing P is called the **FT-body** of P, denoted as **FT-body$^{\delta}$(P)**. For each item p in itemset P, the number of transactions in FT-body$^{\delta}$(P) containing item x is called the **item support** of x in FT-body$^{\delta}$(P), denoted as **Item-Sup$_{P}^{\delta}$ (x)**.

Definition 2 (Fault-tolerant frequent pattern): Given a fault tolerance δ, a FT-support threshold min-sup^{FT}, and a frequent-item support threshold min-sup ^{item}. An itemset P is called a fault-tolerant frequent pattern iff

1) FT-sup$^{\delta}$ (P) $\geq$ min-sup ^{FT}; and

2) For each item x$\in$ P, Item-Sup$_{P}^{\delta}$ (x) $\geq$min-sup^{item} .

3. Bit Vector Representations

3.1 Appearing Vectors

Let |DB| denote the number of transactions in database. For each item x, the *appearing vector* of x, denoted as Appear$_{x}$, is a binary vector of |DB| dimensions. If x is contained in the ith transaction, the ith dimension in its appearing vector is set to be 1; otherwise, the dimension is set to be 0. Then, an *appearing vector table*, which consists of appearing vectors of various items, is constructed to represent the distribution of items in a transaction database.

Consider the sample database TDB shown in Table 1. There are 7 various items and 5 transactions in the database. Item A is contained in transactions T2 and T5, thus the appearing vector of A, denoted as Appear$_{A}$, is <0,1,0,0,1>. Similarly, the appearing vectors of B, C, D, E, F, and G are obtained to construct the appearing vector table of TDB, as shown in Table 2.

For each item x, the number of dimensions with value 1 in Appear$_{x}$ implies its support count in the database. This value could be obtained by performing a support counting function, Count(), which computes an inner product operation on Appear$_{x}$ and a |DB|-dimensional vector with 1s in all the dimensions (denoted as $\mathbf{I}_{|DB|}$).

An appearing vector is also applied to represent the distribution of an itemset P in transactions of a database. Suppose itemset P consists of k items: i_1,i_2 ...,and i_k . The appearing vector of P is obtained by performing AND operations on appearing vectors of its k elements. Similarly, support count of P could be obtained quickly by performing the same support counting function.

3.2 FT_appearing vectors

Given a fault tolerance δ, the *FT-appearing vector* of an itemset P is denoted as FT-Appear$_{P}$(δ). If the ith transaction FT-contains itemset P, the ith dimension in FT-Appear$_{P}$(δ) is set to be 1; otherwise, the dimension is set to be 0. The appearing vector of itemset P is regarded the FT-appearing vector of P under fault tolerance 0.

The dimensions in FT-Appear$_{P}$(δ) with 1s imply the corresponding transactions in FT-body$^{\delta}$(P). Thus, the FT-support of an itemset P under fault tolerance δ could be obtained by inputting FT-Appear$_{P}$(δ) to the support counting function, Count(), introduced in section 3.1. In addition, for each item x in P, Item-Sup$_{P}^{\delta}$ (x) equals to the number of dimensions with 1s after performing AND operation on FT-Appear$_{P}$(δ) and Appear$_{x}$. That is, Item-Sup$_{P}^{\delta}$ (x) could be obtained by performing an inner product operation on FT-Appear$_{P}$(δ) and Appear$_{x}$.

3.3 Generation of FT-appearing vectors

Given a fault tolerance δ , a transaction T FT-contains an itemset P means T contains at least |P|- δ items in P. The cost is significant to compute the appearing vectors of these $\mathbf{C}^{|P|}_{|P|-\delta}$ subsets and perform ($\mathbf{C}^{|P|}_{|P|-\delta}$ -1) OR operations among these vectors when the number of elements in P is large. For solving this problem, the following theorem provides a property for generating FT_appearing vectors recurrently.

[Theorem 1] Let P denote a nonempty itemset and P′=P∪{x}, where x is an item not in P. A transaction T FT-contains P′ under fault tolerance δ iff

1) T FT-contains P under fault tolerance (δ -1), or
2) T contains x and FT-contains P under fault tolerance δ .

Suppose itemset P′ is obtained by inserting an item x into a nonempty itemset P. According to theorem 1, the FT-appearing vector FT-Appear$_{P'}$ could be computed from FT-Appear$_P$ and Appear$_x$ according to the following definition of recurrent function.

Recurrent Function for FP-appearing vectors:
Input: Itemset P, item x($x \notin P$), FT_appearing vectors of P, Appear$_x$, and fault tolerance δ.
Output: the appearing vectors of P′, where P′=P∪{x}.

```
P′=P∪{x};
If | P′|≤$\delta$, FT-Appear$_{P'}$($\delta$) = I$_{|DB|}$;
If $\delta$=0, FT-Appear$_{P'}$($\delta$)=Appear$_{P'}$=FT-Appear$_P$(0)$\wedge$Appear$_x$;
Otherwise, FT-Appear$_{P'}$($\delta$)=FT-Appear$_P$($\delta$)-1)$\vee$(FT-Appear$_P$($\delta$))$\wedge$Appear$_x$).
```

That is, for any given pattern P and an item x, where x is not in P, FT-appearing vectors of the pattern P′=P∪{x} with various fault tolerances could be obtained when all the FT-appearing vectors of P with fault tolerance from 0 to δ are known.

3.4 VB-FT-Mine Algorithm

VB-FT-Mine algorithm is designed based on the FT-appearing vectors representation and the recurrent relation introduced in Section 3.3. First, the transactions in database are read in one by one to construct an appearing vector table. Then the candidates are generated by performing depth-first pattern growing method. For each newly generated candidate pattern, the recurrent function defined in Section 3.3 is performed to obtain its FT-appearing vectors with various tolerances. The FT-support and item-supports of a pattern are thus checked efficiently by performing inner product operations on appearing vectors as introduced in Section 3.2. The VB-FT-Mine algorithm is shown as follows.

Algorithm VB-FT-Mine
Input: Transaction database DB, min-sup^{item}, min-sup^{FT}, and fault tolerance δ.
Output: the complete set of FT-patterns.

1. Scan DB once to construct the appearing vector table.
2. Compute the support count for each item x. An item x is global frequent iff $Sup_x^{DB} \geq$ min-sup^{item}. Let the global frequent items be denoted as $x_1, x_2, \ldots, x_n$.

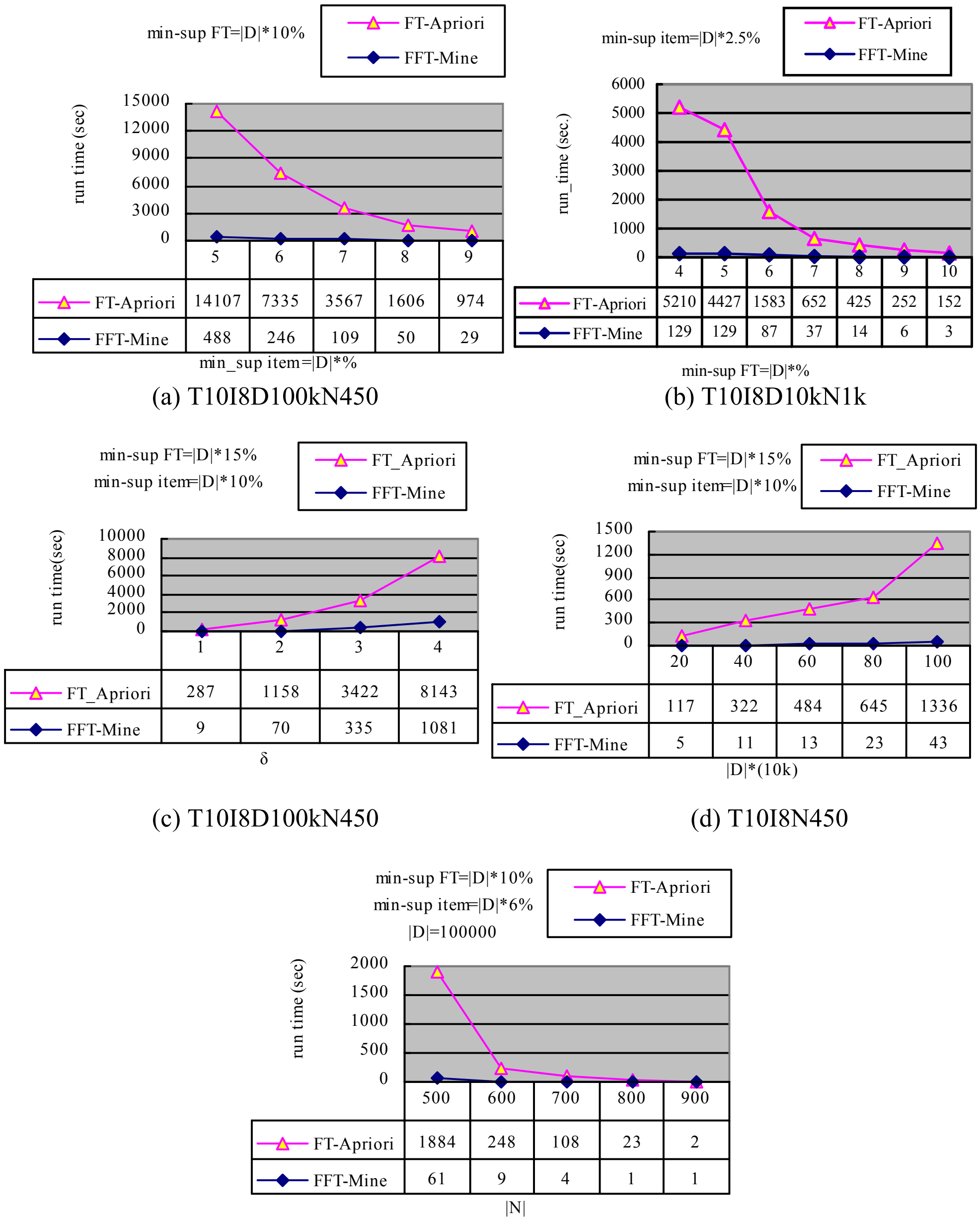
```
3. For i=1 to n {
  (a) Initialization:  Set P = {$x_i$};  FT-$\text{Appear}_P(0)$ =$\text{Appear}_{xi}$ ;
                       Set j= i+1;  Push (P, FT-appearing vectors of P, j) into stack;
  (b) While (stack is not empty){
    (b-1) Generate a candidate pattern $P' = P \cup \{x_i\}$;
          FT-$\text{Appear}_{P'}(0)$ = FT-$\text{Appear}_P(0) \wedge \text{Appear}_{x;}$
          For (k = 1 to $\delta$)
           { If $|P'| \leq k$, FT-$\text{Appear}_{P'}(k) = I_{|DB|}$ ;
               else FT-$\text{Appear}_{P'}(k)$ =FT-$\text{Appear}_P(k-1) \vee$(FT-$\text{Appear}_P(k) \wedge \text{Appear}_{xj}$)}
    (b-2) Compute FT-$\text{sup}^{\delta}(P')$ = FT-$\text{Appear}_C(\delta) \cdot I_{|DB|}$;
    (b-3) If FT-$\text{sup}^{\delta}(P') \geq$min-$\text{sup}^{FT}$,
            for (each item x in P′)  performs FT-$\text{Appear}_{P'}(\delta) \cdot \text{Appear}_X$ to obtain Item-
    $\text{Sup}_{P'}^{\delta}(x)$;
    (b-4) If Item-$\text{Sup}_{P'}^{\delta}(x) \geq$min-$\text{sup}^{\text{Item}}$
            for (each x in P′) { Output P′ as a result;
              Set P = P′ and  j=j+1;
              If $j \leq n$  Push(P, FT-appearing vectors of P, j) into stack;}
            else Repeat
             { Pop(P, P’s FT-appearing vectors, j) from stack;  j=j+1;}
          until (($j \leq n$) or (stack is empty))
    } /* end while
 } /* end for
```

4. Performance Evaluation

In order to show the efficiency and effectiveness of our approach by comparing with FT-Apriori[5] algorithm, both algorithms are implemented using Microsoft Visual C++ 6.0. The experiments are performed in a PC with an Intel Pentium4 2.4GHz CPU and 256MB main memory, running Microsoft Windows XP Professional. The experiments were performed on synthetic data generated by the IBM synthetic market-basket data generator.

In the following experiments, the three run-time parameters (min-sup^{item},min-sup^{FT} , and δ) are controlled individually for observing their effects on execution time of the two mining algorithms. In addition, the execution times on different data sets with various setting on database size and number of items are evaluated in the other two experiments.

Figure 5(a) shows that the execution time of VB-FT-Mine is much less than the time of FT-Apriori. When the setting on min-sup $^{\text{item}}$ is decreased, more candidate itemsets are generated. Therefore, the execution time of both algorithms increases as min-sup^{item} is decreased. Moreover, the increasing rates of both algorithms are similar. However, VB-FT-Mine is about 30 times faster than FT-Apriori with the same support threshold settings. Figure 5(b) shows, when the setting on min-sup^{FT} is decreased, there are also more candidate itemsets generated. Thus, the execution time of both algorithms increases as min-sup $^{\text{FT}}$ is decreased. The execution time of FT-Apriori increases significantly when min-sup^{FT} is lower than |D|*6%. However, the execution time of VB-FT-Mine keeps stable when min-sup^{FT} is



	5	6	7	8	9
FT-Apriori	14107	7335	3567	1606	974
FFT-Mine	488	246	109	50	29

(a) T10I8D100kN450

	4	5	6	7	8	9	10
FT-Apriori	5210	4427	1583	652	425	252	152
FFT-Mine	129	129	87	37	14	6	3

(b) T10I8D10kN1k

	1	2	3	4
FT_Apriori	287	1158	3422	8143
FFT-Mine	9	70	335	1081

	20	40	60	80	100
FT_Apriori	117	322	484	645	1336
FFT-Mine	5	11	13	23	43

(c) T10I8D100kN450

(d) T10I8N450

	500	600	700	800	900
FT-Apriori	1884	248	108	23	2
FFT-Mine	61	9	4	1	1

(e) T10I8D100k

Figure 5: Experimental Results

below |D|*5%. When the fault tolerance δ increases, much more candidates are generated. As Figure 5(c) shows, a minor increase of fault tolerance increases the execution time of FT-Apriori algorithm dramatically. The growing ratio of VB-FT-Mine is less than FT-Apriori, which indicates the scalability of VB-FT-Mine with respect to fault tolerance.VB-FT-Mine algorithm scans the database only once. The result in Figure 5(d) shows, as expected, the execution time of VB-FT-Mine glow more slowly than the one of FT-Apriori as the size of database increases. As the number of various items in database |N| increases, the average size of the transactions remains 10, the data distribution in the database becomes sparser. Figure 5(e) shows that, execution times of both algorithms decrease as N increases because less frequent patterns are generated in the mining process. As the figure shows, for FT-Apriori algorithm, the costs to handle a huge number of candidate patterns are more and more significant when |N| is below 600. However, this factor does not influence the execution time of FT-Apriori very much. From the above experiments, in general, VB-FT-Mine algorithm outperforms FT-Apriori on execution time with respect to various parameters setting. Especially, the VB-FT-Mine has better scalability on execution time with respect to support thresholds and fault tolerance than FT-Apriori.

5. Conclusion

In this paper, an algorithm named VB-FT-Mine is proposed for mining fault-tolerant frequent patterns efficiently. VB-FT-Mine algorithm is designed based on the bit vector representations. According to the depth-first pattern growing strategy, the FT_supports and item supports of candidate patterns could be obtained by performing vector operations on FT_appearing vectors efficiently. The experimental results show our approach has significant improvement on execution time than FT-Apriori algorithm. To extend VB-FT-Mine algorithm by applying the strategies for mining frequent patterns in data streams provide a good solution for this problem, which is under our study currently.

References

[1] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules," in Proc. of Int. Conf. on Very Large Data Bases, 1994.

[2] J. Han, J. Pei, Y. Yin and R. Mao, "Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach", Data Mining and Knowledge Discovery, 8(1):53-87, 2004.

[3] J.S. Park, M.S. Chen, and P.S. Yu, "An Effective Hash-based Algorithm for Mining Association Rules," in Proc. of the ACM SIGMOD International Conference on Management of Data (SIGMOD'95), May, pages 175-186, 1995.

[4] J. Pei, A.K.H. Tung, and J. Han, "Fault-Tolerant Frequent Pattern Mining: Problems and Challenges," in Proc. of ACM-SIGMOD Int. Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD'01), 2001.

[5] S.-S. Wang and S.-Y. Lee, "Mining Fault-Tolerant Frequent Patterns in Large Database," in Proc. of Workshop on Software Engineering and Database Systems, International Computer Symposium, Taiwan, 2002.