Mining Fault Tolerant Frequent Patterns using Pattern Growth Approach

Shariq Bashir, Zahid Halim, A. Rauf Baig
FAST-National University of Computer and Emerging Sciences, Islamabad, Pakistan
{shariq.bashir,zahid.halim,rauf.baig}@nu.edu.pk

Abstract

Mining fault tolerant (FT) frequent patterns from transactional datasets are very complex than mining all frequent patterns (itemsets), in terms of both search space exploration and support counting of candidate FT-patterns. Previous studies on mining FT frequent patterns adopt Apriori-like candidate set generation-and-test approach, in which a number of dataset scans are needed to declare a candidate FT-pattern frequent. First for checking its FT-pattern support, and then for checking its individual items support present in its FT-pattern which depends on the cardinality of pattern. Inspired from the pattern growth technique for mining frequent itemsets, in this paper we present a novel algorithm for mining FT frequent patterns using pattern growth approach. Our algorithm stores the original transactional dataset in a highly condensed, much smaller data structure called FT-FP-tree, and the FT-pattern support and item support of all the FT-patterns are counting directly from the FT-FP-tree, without scanning the original dataset multiple times. While costly candidate set generations are avoided by generating conditional patterns from FT-FP-tree. Our extensive experiments on benchmark datasets suggest that, mining FT frequent patterns using our algorithm is highly efficient as compared to Apriori-like approach.

Keywords: Fault Tolerant Frequent Patterns Mining, Maximal Frequent Patterns Mining, Bit-vector Representation, and Association Rules.

1. Introduction

Mining frequent patterns from transactional or relational datasets with support greater than a certain user defined threshold, plays an important role in many data mining applications such as intrusion detection, finding gene expression patterns, web log patterns etc. In recent years, a number of algorithms have been proposed for efficient mining of such frequent patterns, on the basis of Apriori property proposed by Agrawal et al. [1]. These algorithms take a transactional dataset and support threshold *(min_sup)* as an input and output those exact matching frequent patterns which contain support greater than *min_sup*, with assuming that the dataset is very well pre-processed and noise free. However, the real world datasets are dirty and contain missing and noisy values. In such situations, users face difficulties in setting this *min-sup* threshold to obtain their desired results. If *min-sup* is set too large, then there may be a small number of frequent patterns, which does not give any desirable result. If the *min-sup* is set too small, then there may be many redundant short un-useful frequent patterns, which not only take a large processing time for mining but also increase the complexity of filtering un-interesting frequent patterns. In both situations, the ultimate goal of mining interesting frequent patterns is undermined.

For handling such situations, J. Pei at el. in [6] introduced a new application of finding only interesting frequent patterns in a real world dirty datasets, instead of finding exact patterns. This approach is known as fault-tolerant (FT) frequent pattern mining. The problem of mining all FT frequent patterns from a dirty transactional dataset can be considered from the following two conditions [6].

1. Under user defined fault tolerance factor δ, a pattern X with cardinality greater than δ is called a FT frequent pattern, if it appears in at least k number of FT-transactions. A transaction t is called a FT-transaction under fault tolerance factor δ, if it contain at least $|X|$-δ number of items of X. The number k is called the frequency of X which must be greater or equal than the minimum FT support threshold (min_sup^{FT}).
2. Each individual single item i of X must be appeared in at least l number of FT-transaction of X, where l is called the minimum item support threshold under fault tolerance factor δ ($\text{item\_sup}^{FT}_{\delta}$).

978-1-4244-1968-5/08/$25.00 ©2008 IEEE

Table 1. A sample transactional dataset with 8 transactions and 12 items

Transaction ID	Items
T1	*f, e, d, c*
T2	*f, d, a*
T3	*f, e, d, b*
T4	*c, b, a*
T5	*f, e, c, b*

For example, with $\text{min\_sup}^{FT} = 3$ and $\text{item\_sup}^{FT}_{\delta} = 2$, the pattern $\langle b, c, d, e \rangle$ is a FT frequent pattern under fault tolerance factor $\delta = 1$, since 3 out of 4 items are present in FT-transaction T1, T3 and T5 which qualifies min_sup^{FT} threshold and each single item $\langle b \rangle$, $\langle c \rangle$, $\langle d \rangle$ and $\langle e \rangle$ is present in at least 2 transactions with qualifies $\text{item\_sup}^{FT}_{\delta}$ threshold. In [6] they also proposed an FT-Apriori algorithm for finding all type of such patterns. The FT-Apriori was extended from the Apriori approach, in which downward closure property is used for mining FT frequent patterns.

Although, FT-Apriori presented in [6] is very effective method for enumerating frequent FT pattern in sparse datasets if the FT support thresholds are given to be very large, but the basic algorithm of FT-Apriori encounters some difficulties and takes a very large amount of processing time if the FT support thresholds are given to be very small. We list here some of the main difficulties of FT-Apriori that do not make it an attractive solution for mining FT frequent patterns over large transaction datasets.

1. FT-Apriori encounters difficulty in mining long FT pattern, especially for dense datasets. For example, to find a FT frequent patterns of $X = \{1 \ldots 200\}$ items. FT-Apriori has to generate-and-test all the 2^{200} candidates.
2. Similar to Apriori algorithm, FT-Apriori applies a bottom-up search that enumerates every single FT frequent pattern. This implies that in order to produce a FT frequent pattern of length l, it must produce all 2^l of its subsets, since they too must be frequent FT. This exponential complexity fundamentally restricts FT-Apriori like algorithms in discovering only useful interesting FT frequent patterns in a reasonable time limit.
3. FT-Apriori algorithm is considered to be unsuitable for handling frequency counting, which is considered to be one of the expensive tasks in frequent itemsets mining [3]. Since FT-Apriori is a level-wise candidate-generate-and-test algorithm, therefore it has to scan the dataset *200* times to find all FT frequent patterns $X = X_1 \ldots X_{200}$.
4. Even though FT-Apriori prunes the search space by removing all k FT patterns, which are infrequent before generating candidate $(k+1)$-FT patterns, but it still needs a dataset scan, to determine which candidate $(k+1)$ FT patterns are frequent and which are infrequent. Even for datasets which have more than *200* items, determining k-frequent FT patterns by repeated scanning the dataset with pattern matching takes a large amount of processing time.
5. Moreover, mining FT frequent patterns are very complex than mining all frequent patterns, in terms of both search space exploration and frequency counting of candidate patterns. In frequent pattern mining, a candidate pattern X is declared to be frequent, by checking its frequency in only one dataset scan. While in FT frequent pattern mining, a number of dataset scans are needed to declare a candidate FT-pattern X as frequent. First for checking FT-pattern support, and then for checking individual items support present in that FT-pattern, which depends on the cardinality of X.

To overcome these limitations, and inspired by the pattern growth technique present in [4] for frequent pattern mining, in this paper we have introduced a novel algorithm of mining FT frequent patterns using pattern growth method. The major advantage of FT frequent patterns mining using pattern-growth approach is that it removes the costly operation of repeatedly scanning the dataset in each iteration and generating and testing infrequent candidate itemsets. In simple words pattern growth removes the costly candidate-generate-and-test operation of FT-Apriori type technique [2], [5]. Our algorithm requires only two dataset scans for counting all FT frequent patterns. The first dataset scan collects the support of all FT frequent items of length one. The second dataset scan builds a compact data structure called FP-tree. Each node of FP-tree corresponds to an item which was found frequent in first dataset scan. Next, all frequent itemsets are mined directly from this FP-tree without concerning the transactional dataset. The main strategy of our FT pattern-growth approach is that it traverses search space in depth first order, and on each node of search space it mines FT frequent patterns on the basis of conditional patterns and creates child FP-trees (also called projected database) for mining next level FT frequent patterns.

2. Problem Defintion

The search space of FT frequent patterns mining can be considered as a lexicographical order [7] search space, where first level or root contains an empty FT pattern, and each lower level k contains all the k-FT

length patterns. Each frequent FT-pattern or node of this search space contains two fields: prefix and tail items. Prefix denotes the FT-frequent pattern of that node, and tail consists of those items which are possible extensions of new candidate FT-patterns that can be generate from that node by extending frequent tail items with node's prefix. A FT-pattern that is not checked yet with user defined FT support thresholds, that either it is a frequent FT-pattern of infrequent FT-pattern is called candidate FT-pattern. Note, this candidate FT-patterns definition is very different as J. Pei at al. have explained for FT-Apriori algorithm [6]. In FT-Apriori algorithm candidate FT-patterns are generated by simply extending the tail items with head FT-pattern, in which most of them denotes false generation i.e. not exist in the input transactional dataset. While in our algorithm candidate FT-patterns are generated only from the conditional patterns of its node's FP-tree [4], that grantee no false generation (exist in the input transactional dataset). Each node of this search space contains two data structures for representing its projected transactions, FP-tree and its associated header table. The FP-tree of any node is a compact data structure, which includes only those transactions of original transactional dataset (projected transactions), inserted with some specific sorted order, in which its node's prefix FT frequent pattern appears as a prefix. This sorted ordering among items of projected transactions in FP-tree is very important, because with sorted ordering those transactions which have some similar kind of items, shares a common path at some length. This sharing of transactions items, not only increases the scalability of algorithm on larger datasets, but also increases the speed of fast support counting of candidate FT-patterns. To facilitate tree traversal, an item header table is built in which each item points to its occurrence in the tree via a head of *node-link*. Nodes with the same item name are linked in sequence via such node links, that are further utilized in generating conditional patterns. Each node in FP-tree consists of three fields: *item_name*, *item_count* and *node_link*, where *item_name* registers which item this node represents, *item_count* registers the number of transactions represented by the portion of the path reaching this node, and *node-link* links to the next node in the FP-tree carrying the same *item_name*. Each entry in the frequent FT pattern header table consists of two fields. (1) *item_name* and (2) head of node_link, which points to the first node in the FP-tree carrying the same *item_name*.

3. Mining FT-Frequent Pattern without Candidate Generation

The algorithm that we are presenting below discovers all frequent FT patterns under user defined min_sup^{FT}, $\text{item\_sup}^{FT}_{\delta}$ and δ FT support thresholds, using pattern-growth technique or simply without candidate generation approach. Inspired by the pattern growth technique presented in [4] for mining frequent itemsets, our algorithm also uses FP-tree and header table for generation of conditional patterns at any node, while conditional patterns are further used in generation of candidate FT patterns or child nodes of that node (next level FT frequent patterns). Before studying the working of algorithm, let we introduced with some major components and definitions of this algorithm.

Multiple FP-trees at any node. In contrast to pattern growth technique presented in [4], in our algorithm there can be multiple FP-trees with their associated header tables at any node. We call them FT-FP-trees of that node. However, their number will not be greater than the fault tolerant factor δ.

Definition1. Since for any frequent FT pattern $|X|>\delta$, their can be multiple set of mismatching transactions or conditional patterns, that comes under its fault tolerant factor δ. For example at node $\langle ab \rangle$, under fault tolerant factor $\delta = 2$, one set of conditional patterns could be those, in which both items $\langle a \rangle$ and $\langle b \rangle$ are present, with mismatching factor zero. One set could be those in which either item A or B is present, denotes the set with mismatching factor *one*, and one set could be those in which both items $\langle a \rangle$ and $\langle b \rangle$ are not present, denotes the set with mismatching factor *two*. Therefore, for FT-pattern X, our algorithm stores each set of its mismatching conditional patterns in a separate FT-FP-tree. The main advantage of creating multiple FT-FP-trees on each node is that, with multiple FT-FP-trees it becomes easy to generate conditional patterns of child nodes under fault tolerant factor δ.

Information Stored in Nodes of FT-FP-tree. Each node in our algorithm FT-FP-tree contains four different types of fields. (1) *item_name* of the node, (2) *item_count* of the node, for calculating min_sup^{FT} support of child nodes, (3) *node_like* and (4) support information about each item of its frequent FT-pattern's prefix, we call it prefix support information, used in calculating $\text{item\_sup}^{FT}_{\delta}$ support of child nodes. In mining frequent itemsets using pattern growth technique, keeping this prefix support information in nodes of FP-trees is not so important; as we know for any frequent itemset y its FP-tree contains only those transactions (projected transactions) of its input

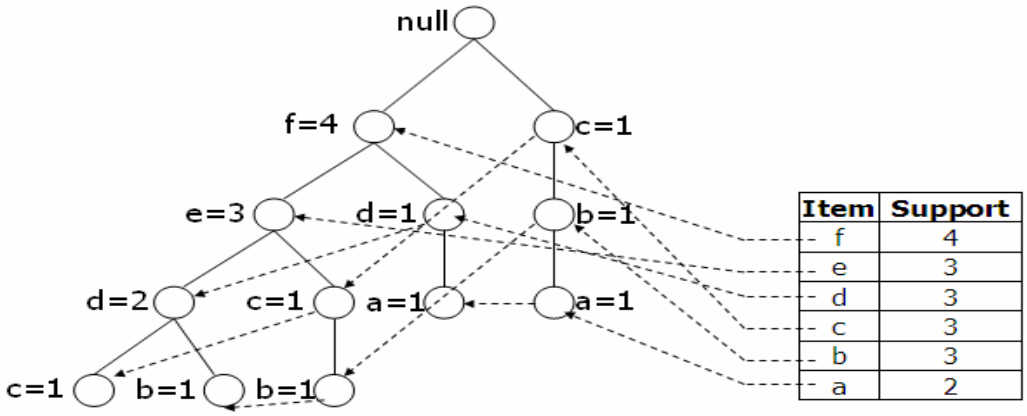


Figure 1. FT-FP-tree of root node

dataset, in which *y* appears as a prefix with a mismatch factor zero. While in mining FT frequent patterns, keeping this information is very important. As we study in defintion1, for any frequent FT-pattern, its FT-FP-trees can contained multiple mismatching set of prefix information, and sorting prefix support information about item in prefix, helps in calculating item_sup$^{FT}_{\delta}$ support. For example in FT FP-trees of node $\langle a, b, c\rangle$, each node in its FT-FP-trees contains a prefix support information about each item in its prefix $\langle a, b, c\rangle$.

Calculating FT-Pattern Support.

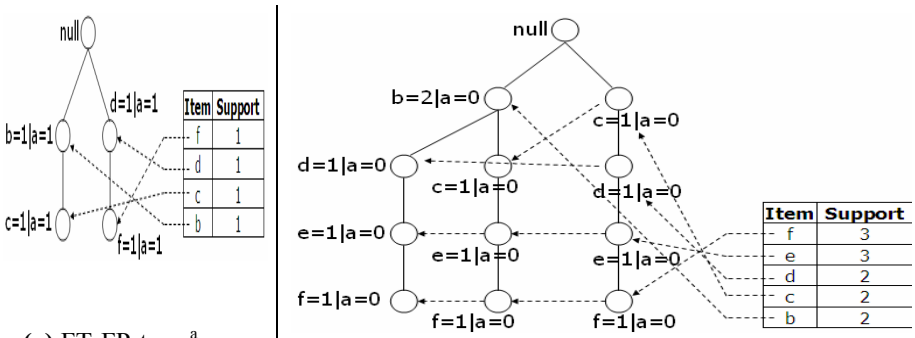
Definition2. On calculating the min_sup^{FT} of any candidate FT-pattern $m = n \cup \langle x\rangle$, from the conditional patterns of its parent node *n's* FT-FP-trees$^{n}_{\langle \delta = i\rangle}$ $\langle 0 \le i \le \delta\rangle$. The support of conditional patterns of all the tail items of node *n*, present in those *n's* FT-FP-trees with have mismatch factor $< \delta$ (FT-FP-trees$^{n}_{\langle \delta = i\rangle}$ $\langle 0 \le i < \delta\rangle$), contribute in the overall min_sup^{FT} support of *m*. For example, at node $\langle AB\rangle$ with tail $\langle C,D,E\rangle$, the support of conditional patterns of all the tail items of node $\langle AB\rangle$ present in FT-FP-trees$^{AB}_{\langle \delta = i\rangle}$ $\langle 0 \le i < \delta\rangle$, contribute in the min_sup^{FT} support of every chid extension of node $\langle AB\rangle$. However, the support of conditional patterns present in FT-FP-trees$^{AB}_{\langle \delta = i\rangle}$ $\langle i = \delta\rangle$, contribute in the min_sup^{FT} support of any child extension $\langle AB\rangle \cup \langle X\rangle$, only when *X* is present in that conditional pattern.

Calculating FT-Pattern Item Support.

Defintion3. On calculating the item_sup$^{FT}_{\delta}$ support of any candidate FT-frequent pattern $m = n \cup \langle x\rangle$ at any node *n*. The prefix support information presents in conditional patterns of all the tail items of *n* in FT-FP-trees$^{n}_{\langle \delta = i\rangle}$ $\langle 0 \le i < \delta\rangle$, contribute in the overall item_sup$^{FT}_{\delta}$ support of *m*. However, the prefix support information presents in conditional pattern *Y* in FT-FP-trees$^{n}_{\langle \delta = i\rangle}$ $\langle i = \delta\rangle$, contributes in the item_sup$^{FT}_{\delta}$ support of *m,* only if *X* is present in *Y*.

4. Candidate FT-Pattern Generation from Conditional Patterns

Scanning the Transaction dataset. Let *TDB* be our transactional dataset. For ease of algorithmic explanation let we take the fault tolerant δ threshold as *2*, but at the time of actual mining it could be any number with value greater than one. Based on the FT-Apriori property presented in [6], any *length-1* items which has support less than item_sup$^{FT}_{\delta}$ could not generate any candidate FT pattern with length greater than 1. Therefore, on the first scan of *TDB*, all the items which have support less than item_sup$^{FT}_{\delta}$ are removed from transactions, and items in transactions of *TDB* are ordered using decreasing support. After the first scan of *TDB*, transactions sorted using decreasing support are picked one by one and inserted in the FT-FP-tree of root node. For understanding, how the transactions are inserted in FP-tree and how its corresponding header table is updated, we will refer the readers to read paper [4] for better understanding. Figure 1 shows the FT-FP-tree of root node with min_sup^{FT}, item_sup$^{FT}_{\delta}$ as 2 and 1 respectively. After the transactions insertion of *TDB* in FT-FP-tree of root



(a) FT-FP-trees$^{a}_{\langle\delta=0\rangle}$ **(b)** FT-FP-trees$^{a}_{\langle\delta=1\rangle}$

Figure 2. FT-FP-tree of node $\langle a\rangle$ with prefix $\langle a\rangle$ and tail $\langle b, c, d, e, f\rangle$

node, all the length-1 items in header table of FT-FP-tree are ordered by increasing support, and first header item X from the ordered header list is picked and a new node $n = \langle\ \rangle \cup \langle X\rangle$, with prefix $\langle X\rangle$ and tail $\langle$*frequent length-1 items -X*$\rangle$ is constructed for mining those length *2* frequent FT-patterns in which *n's* prefix appear as a subset. Before iterating to next recursion level, the algorithm first generates the conditional patterns of item X from FT-FP-trees$^{root}_{\langle\delta\ =\ i\rangle}$ $\langle i=0\rangle$, for the construction of n's FT-FP-trees$^{n}_{\langle i\ =\ \delta\rangle}$ $\langle 0\le i\le\delta\rangle$. Since at root level, there is only one FT-FP-tree exist which has mismatch factor $< \delta$. Therefore, according to definition 1, the conditional patterns of all the tail items of node *n*, present in FT-FP-trees$^{root}_{\langle i\ =\ \delta\rangle}$ $\langle 0\le i\le\delta\rangle$, contribute in the construction of FT-FP-trees of node n. All those conditional patterns in which item X is present, are inserted in FT-FP-trees$^{n}_{\langle\delta\ =\ 0\rangle}$, and all those conditional patterns in which item X is not present, are inserted in FT-FP-trees$^{n}_{\langle\delta\ =\ 1\rangle}$. Therefore, at node *n*, there would be two FT-FP-trees, one which has a mismatch factor zero, and one which has a mismatch factor one. Figure 2 shows FT-FP-trees of node $\langle a\rangle$ with prefix $\langle a\rangle$ and tail $\langle b, c, d, e, f\rangle$. After constructing the FT-FP-trees of node *n*, the algorithm iterates to next recursion level for mining those *2*-length frequent FT-patterns in which prefix of node *n* appears as a prefix.

Mining Frequent FT-Patterns from FT-FP-Trees.

1. At node *n*, firstly the conditional patterns of all the tail items of node *n* are generated one by one from the FT-FP-trees$^{n}_{\langle i\ =\ \delta\rangle}$ $\langle 0\le i\le\delta\rangle$ for generating child extensions (candidate FT-patterns) of node n. For understanding how candidate FT-pattern are generated from conditional patterns, we will refer the readers to reader paper [4] for better understanding.
2. The conditional patterns present in FT-FP-trees$^{n}_{\langle i=\delta\rangle}$ $\langle 0\le i<\delta\rangle$, will contribute in the support of every candidate FT-pattern of node *n*. While conditional patterns present in FT-FP-trees$^{n}_{\langle i\ =\ \delta\rangle}$ $\langle i=\delta\rangle$, will contribute in the support of any candidate FT-pattern, only when its corresponding tail item of node n is present in that conditional pattern.
3. After generating candidate FT-pattern, the support thresholds min_sup^{FT} and item_sup$^{FT}_{\delta}$ of all the *2*-length candidate FT patterns are checked, and those *2*-length candidate FT-patterns which have support thresholds less than the user defined thresholds, their associated tail items are removed from the list of frequent tail items of node *n*.
4. After filtering infrequent tail items, all tail items are ordered by increasing support, and a first item X from the *n's* tail items is picked and a new node $m = n \cup \langle X\rangle$ is constructed with prefix $\langle n \cup X\rangle$ and tail items $\langle$*n's tail-X*$\rangle$.
5. For constructing FT-FP-trees of node *m*, different sets of conditional patterns of all the tail items of node *m* are generated from the FT-FP-trees of node *n* according to their mismatching factor. For example, if X is not present in any conditional pattern that is generated from FT-FP-trees$^{n}_{\langle\delta\ =\ 0\rangle}$, it will denote the conditional pattern of *m* with mismatching factor one, while if X is present in that conditional pattern, it will denote the conditional pattern of *m* with mismatching factor zero. Similarly, if X is not present in any conditional pattern that is generated from FT-FP-trees$^{n}_{\langle\delta\ =\ 1\rangle}$, it will denote the conditional pattern of *m* with mismatching factor two, while if X is present in that conditional pattern, it will denote the conditional pattern of *m* with mismatching factor one.

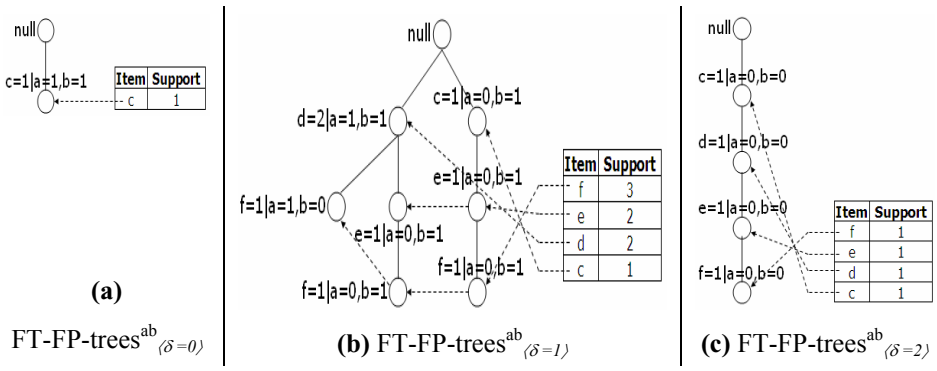


Figure 3. FT-FP-tree of node ⟨*ab*⟩ with prefix ⟨*ab*⟩ and tail ⟨*c, d, e, f*⟩

6. Since, the algorithm is contracting FT-FP-trees of that node, which has pattern length *2*. Therefore at maximum their can be three sets of mismatching type of conditional patterns. After generating conditional patterns, all the conditional patterns are inserted in their respected FT-FP-tree according to their mismatching factor. Figure 3 shows FT-FP-trees of node ⟨*ab*⟩ with prefix ⟨*ab*⟩ and tail ⟨*b, c, d, e, f*⟩. After constructing the node *m* and its associated FT-FP-trees the algorithm iterates to node *m* for mining all those 3-length frequent FT-patterns in which node *m's* prefix appears as a subset.

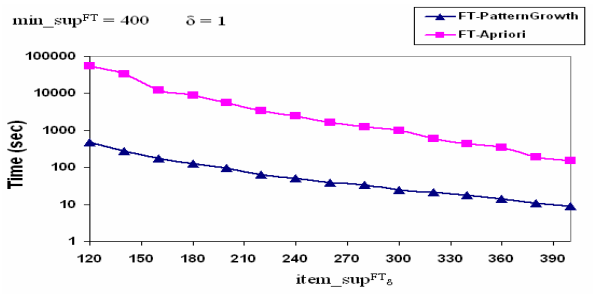
At node *m* or any other higher level node of search space, the same sequence of candidate FT-pattern generation, filtering infrequent tail items and construction of new node operations are performed as we explained for node *n* in step 1 to 6. The only one main difference that would happen at node *m* or any other higher level node of search node, at the time of creating FT-FP-trees of its child nodes, will be that. If any of conditional pattern contains a mismatching factor greater than ⟨*δ*⟩, then according to FT-Apriori property [6], it is removed from the FT-FP-trees of its child nodes. If at any node, all of its tail items are found infrequent, then the algorithm backtracks to previous level and constructs a new node from its remaining tail items, and iterates to next recursion level for mining its candidate FT-patterns. Figure 4 shows the pseudo code of mining FT frequent patterns using pattern growth approach.

Table 2. Datasets use in our experimental results.

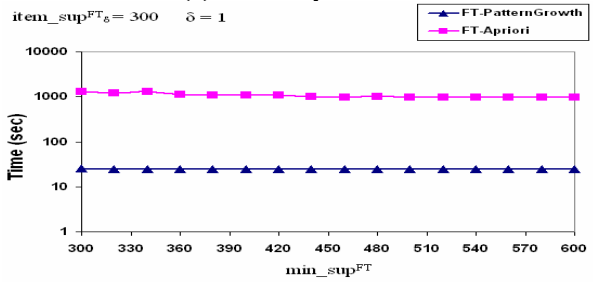
Dataset	Items	Average Length	Records
T10I4D100K	1000	10	100,000
BMS-WebView2	3341	5.6	77,512

```
FT-PatternMining ( )
(1) Scan the TDB and count the support all the length-1 items.
(2) for each transaction t in TDB
(3)         remove the infrequent items from t
(4)         reorder the items by decreasing support
(5)         insert the t in FT-FP-trees^root ⟨δ=0⟩
(6) for each item i in header table of FT-FP-trees^n ⟨δ=0⟩
(7)         n = ⟨i⟩ ∪ ⟨frequent length-1 items – i ⟩
(8)         FT-PatternGrowth (n)

FT-PatternGrowth (node n)
(1) for each FT-FP-trees^n ⟨δ=i⟩ ⟨0≤ i < δ⟩
(2)         for each tail item X in n's tail
(3)             CP-List = CP_List ∪ generate X's conditional pattern P
(4)                 for each item i in P
(5)                     child_nodes = child_nodes ∪ i (generate candidate FT-pattern for n)
(6)         for each conditional pattern P of prefix's n in FT-FP-trees^n ⟨δ=i⟩ ⟨i = δ⟩
(7)             CP-List = CP_List ∪ generate prefix's n conditional pattern P
(8)                 for each item i in P
(9)                     child_nodes = child_nodes ∪ i (generate candidate FT-pattern for n)

(10) for each candidate-FT-pattern X in child_nodes
(11)        S = calculated FT-pattern support of X from CP_List
(12)        if ( S ≥ min_sup^FT)
(13)                for each item i in candidate-FT-pattern X
(14)                    SupItem = calculated item support of i from CP_List
(15)                    if ( SupItem < item_sup^FT_δ)
(16)                        remove X from child_nodes

(17) reorder child_nodes by increasing support
(18) for each item X in child_nodes
(19)        m's prefix = ⟨n⟩ ∪ ⟨X⟩
(20)        m's tail = ⟨child_nodes – X⟩
(21)        child_nodes = child_nodes – X
(22)        generate FT-FP-trees of node m from CP_List
(23)        FT-PatternGrowth (m)
```

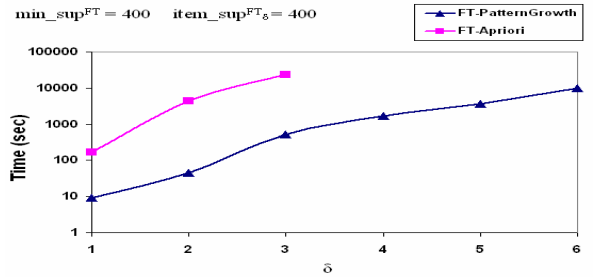
Figure 4. Pseudo cod mining FT frequent pattern using pattern growth approach



(a) First Experiment

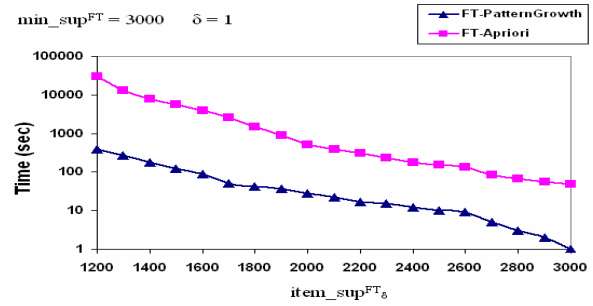


(b) Second Experiment

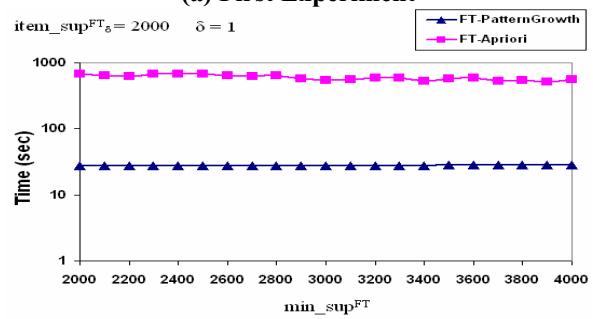


(c) Third Experiment

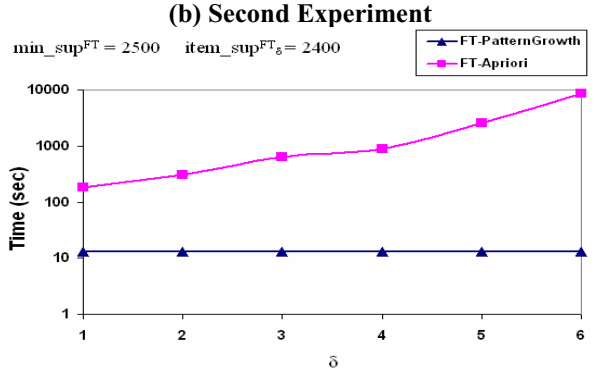
Figure 5. FT-PatternGrowth performance results on BMS-WebView2 dataset.



(a) First Experiment



(b) Second Experiment



(c) Third Experiment

Figure 6. FT-PatternGrowth performance results on T10I4D100K dataset.

Calculating Candidate FT-Pattern Support. Before considering, how our algorithm counts the support of any candidate FT-pattern from the FT-FP-trees$^{n}_{\langle i=\delta\rangle}$ $\langle 0 \le i < \delta \rangle$ of its parent node n. Let us consider the following definition.

Definiton4. Let $m = n \cup \langle X \rangle$ denotes a candidate FT-pattern at node n, where n is the parent node and X is its associated tail item of node n. A conditional pattern P comes under the fault tolerant factor δ for tail item X, if at least one of following two properties holds.

- P contains FT under fault tolerance factor $(\delta\text{-}1)$ with pattern m, or
- P contains X and FT under fault tolerance factor (δ) with pattern n.

From the definition 1, since every conditional pattern P in FT-FP-trees$^{n}_{\langle i=\delta\rangle}$ $\langle 0 \le i < \delta \rangle$ of all the tail items of node n, contribute in the support of every candidate FT-pattern of node n. Therefore, the property1 of definition1 can be obtained by generating conditional patterns from FT-FP-trees$^{n}_{\langle i=\delta\rangle}$ $\langle 0 \le i < \delta \rangle$. Where the property2 of definition1 can be obtained by generating conditional patterns of only item X from n's FT-FP-trees$^{n}_{\langle i=\delta\rangle}$ $\langle i=\delta \rangle$. Since the conditional patterns present in n's FT-FP-trees$^{n}_{\langle i=\delta\rangle}$ $\langle i=\delta \rangle$, states that they have already fulfill the maximum fault tolerance factor δ. Therefore, they could increase the support of any n's candidate FT-pattern, if and only if, it associated tail item is present in that conditional pattern.

Calculating FT Item Support. The procedure that we describe above only calculates the support of FT transactions contain pattern m under fault tolerance factor δ. If this frequency is greater than min_sup^{FT}, then the item_sup$^{FT}_{\delta}$ support of each individual item i present in the candidate FT-pattern m can be obtained from those conditional patterns of m, which the

algorithm has generated in (calculating candidate FT-pattern support). As we state earlier, each node in FT-FP-tree, contains support information about each item of its prefix. Therefore, each conditional pattern in our algorithm contains not only the support information about how many transactions are merge into it, but it also contains support information about each item of its prefix. The prefix support of any conditional pattern will be only that node's prefix support from where that conditional pattern is generated. For example, if the conditional pattern P is generated from tail item X and node t of the FT-FP-tree, then what will be the prefix support of $\langle AB \rangle$ at node t will also be the prefix support of conditional pattern P.

5. Experiments

All the source code of FT-PatternGrowth and FT-Apriori is written in C language. The experiments are performed on 3.2 GHz processor with main memory of size 512 MB, running windows XP 2005 professional. For experiments, we used the benchmark datasets available at http://fimi.cs.helsinki.fi/data/. These datasets are frequently used in many frequent pattern mining algorithms. Unfortunately, due to lack of space we could not show our experiments that we have performed on all the available datasets, Table 2 shows the description of our datasets, which we have used in our experiments. For each dataset we performed three different experiments to check the performance of both algorithms. In first experiment, we fixed the values of min_sup^{FT} and δ while the value of $item\_sup^{FT}_{\delta}$ are varied on different support thresholds. In second experiment, the values of $item\_sup^{FT}_{\delta}$ and δ are kept fixed, while the value of min_sup^{FT} is varied. In third experiment the values of min_sup^{FT}, $item\_sup^{FT}_{\delta}$ are kept fixed, while the value of δ is varied on different thresholds. As clear from the Figure 5 and 6 results, the FT-PatternGrowth outperforms the FT-Apriori on almost all the datasets and on all the mining threshold values. This is due to its fast support counting of FT-patterns from FT-FP-trees, while the costly candidate set generations are avoided by generating conditional patterns from FT-FP-trees.

6. Conclusion

Mining fault-tolerant (FT) frequent pattern in a real world datasets is considered to be a fruitful direction for future data mining research. However, mining frequent FT-patterns are considered more complex than mining frequent itemsets in terms of both search space exploration and support counting of candidate FT-patterns. Previous studies on mining frequent FT-patterns adopt FT-Apriori approach, which is not considered an efficient approach due to their candidate-generate-and-test approach. To increase the efficiency of mining, in this paper we present a new approach of mining frequent FT-patterns using pattern growth approach. Our approach shows that the supports of all frequent FT-patterns are counted directly from the FT-FP-trees without scanning the transactional dataset multiple times. While costly candidate set generations are avoided by generating conditional patterns from FT-FP-trees. For checking the efficiency of our FT-pattern growth approach, we performed several different experiments on two benchmark datasets. Our experiments show that FT-pattern growth outperforms the FT-Apriori algorithm on all datasets and on all the mining thresholds. This shows the effectiveness of our approach.

Reference

[1] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules," *In Proc. Int'l Conf. Very Large Data Bases*, pp. 487-499, Sept. 1994.

[2] S. Bashir, A. Rauf Baig, "Max-FTP: Mining Maximal Fault-Tolerant Frequent Patterns from Databases", *In the proceedings of 24th British National Conference on Databases, Springer (LNCS),* Volume 4587, July 3-5, Glasgow, UK, 2007.

[3] Proc. IEEE ICDM Workshop Frequent Itemset Mining Implementations, B. Goethals and M.J. Zaki, eds., CEUR Workshop Proc., vol. 80, Nov. 2003, http://CEUR-WS.org/Vol-90.

[4] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation", *In SIGMOD*, pages 1–12, 2000.

[5] J. L. Koh, P. Yo, "An Efficient Approach for Mining Fault-Tolerant Frequent Patterns based on Bit Vector Representations", *in the proceedings of 10th International Conference, DASFAA 2005*, Beijing, China, April 17-20, 2005.

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

[7] R. Rymon: Search through Systematic Set Enumeration, *In Proc. Of Third Int'l Conf. On Principles of Knowledge Representation and Reasoning*, 539 –550. 1992.