Expert Systems With Applications 143 (2020) 113046

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

An efficient pattern growth approach for mining fault tolerant frequent itemsets

Shariq Bashir

Department of Software Engineering, Foundation University, Islamabad, Pakistan

ARTICLE INFO

Article history:
Received 13 March 2019
Revised 27 September 2019
Accepted 20 October 2019
Available online 21 October 2019

Keywords:
Fault tolerant frequent itemset mining
Frequent itemset mining
Pattern growth
Association rules mining

ABSTRACT

Mining fault tolerant (FT) frequent itemsets from transactional databases are computationally more expensive than mining exact matching frequent itemsets. Previous algorithms mine FT frequent itemsets using Apriori heuristic. Apriori-like algorithms generate exponential number of candidate itemsets including the itemsets that do not exist in the database. These algorithms require multiple scans of database for counting the support of candidate FT itemsets. In this paper we present a novel algorithm, which mines FT frequent itemsets using frequent pattern growth approach (FT-PatternGrowth). FT-PatternGrowth adopts a divide-and-conquer technique and recursively projects transactional database into a set of smaller projected transactional databases and mines FT frequent itemsets in each projected database by exploring only locally frequent items. This mines the complete set of FT frequent itemsets and substantially reduces those candidate itemsets that do not exist in the database. FT-PatternGrowth stores the transactional database in a highly condensed much smaller data structure called frequent pattern tree (FP-tree). The support of candidate itemsets are counted directly from the FP-tree without scanning the original database multiple times. This improves the processing speed of algorithm. Our experiments on benchmark databases indicates mining FT frequent itemsets using FT-PatternGrowth is highly efficient than Apriori-like algorithms.

1. Introduction

Mining frequent itemsets from transactional databases play an important role in many data mining applications, e.g., social network mining (Jiang, Leung, & Zhang, 2016; Moosavi, Jalali, Misaghian, Shamshirband, & Anisi, 2017), finding gene expression patterns (Becquet, Blachon, Jeudy, Boulicaut, & Gandrillon, 2001; Creighton & Hanash, 2003; Cremaschi et al., 2015; Mallik, Mukhopadhyay, & Maulik, 2015), web log pattern mining (Diwakar Tripathia & Edlaa, 2017; Han, Cheng, Xin, & Yan, 2007; Iváncsy, Renáta, & Vajk, 2006; Yu & Korkmaz, 2015). In recent years, many algorithms have been proposed for efficient mining of frequent itemsets (Apiletti et al., 2017; Bodon, 2003; Burdick, Calimlim, Flannick, Gehrke, & Yiu, 2005; Gan, Lin, Fournier-Viger, Chao, & Zhan, 2017; Han, Pei, & Yin, 2000; Kosters & Pijls, 2003; Liu, Lu, Yu, Wang, & Xiao, 2003; Pei, Tung, & Han, 2001; Uno, Kiyomi, & Arimura, 2004; Vo, Pham, Le, & Deng, 2017). These algorithms take a transactional database and support threshold *(minimum itemset support)* as input and mines complete set of frequent itemsets with support greater than *minimum itemset support*. Traditional frequent itemset mining (FIM) approach discovers only exact matching itemsets that are absolutely matched. This creates problem when the database contains missing items in transactions and may cause some implicit frequent itemsets not being discover (Yu, Li, & Wang, 2015). In the presence of missing items users face difficulties in setting suitable support threshold for mining desired itemsets. For example, if the support threshold is set too large then FIM discovers only a small number of frequent itemsets, which do not provide desirable output. On the other hand if the support threshold is set too small then FIM generates too many redundant short length frequent itemsets (Cheung & Fu, 2004; Huynh-Thi-Le, Le, Vo, & Le, 2015; Saif-Ur-Rehman, Ashraf, Habib, & Salam, 2016). This not only consumes large processing time but also increases the complexity of filtering interesting frequent itemsets. In both settings, the ultimate goal of mining interesting frequent itemsets is undermined.

To mine frequent itemsets in the presence of missing items, (Pei et al., 2001) proposed fault tolerant (FT) frequent itemsets mining approach. The task of mining FT frequent itemsets from a transactional database can be understand from the following conditions (Pei et al., 2001).

- Under a user-defined fault tolerant (FT) factor (δ), an itemset X with length greater than $(\delta + 1)$ is a FT frequent itemsets if it has support of at least T number of FT-transactions.

E-mail address: shariq.bashir@fui.edu.pk

https://doi.org/10.1016/j.eswa.2019.113046
0957-4174/

Table 1
A sample transactional database. Items are removed from the transactions that have item support less than 3.

TID	Items	(Ordered) Frequent Items
10	*b, c, e, f*	*e, f, c, b*
20	*a, b, e, f*	*e, f, a, b*
30	*a, d, f*	*f, d, a*
40	*e, g*	*e*
50	*a, b, c, e, h*	*e, a, c, b*
60	*a, e, f*	*e, f, a*
70	*d, c, f*	*f, d, c*
80	*d, i, k*	*d*
90	*d, j*	*d*

- A transaction t is a FT-transaction of T under FT factor if it contains at least $(|X| - \delta)$ number of items of X.
- T is the support of X which must be greater or equals to the minimum itemset support (min_sup^{δ}). Each individual item i of X must appear in at least m number of FT-transactions of X. The m is the minimum item support ($item\_sup^{\delta}$) under fault tolerant factor δ.

Table 1 shows a transactional database as our running example. It contains *eleven items* and *nine* transactions. To mine frequent itemsets if the user gives the *minimum itemset support* equals to 3 then there exists no itemset with length more than 2 items (see column 2 of Table 1). The database has many short length itemsets with low support count. To discover generalized knowledge users would be interested to mine itemsets of long length with high support count. If we further analyze the database, then we can discover some long length frequent itemsets. These are not *absolutely matched* in transactions but have support more than 2. For example, further analysis reveals that the itemset (*abcef*) has itemset support 3 as the transactions 10, 30, and 50 contain four out of five items of *abcef*, and every single item of (*abcef*) is appeared in two out of three transactions. This frequent itemset mining phenomenon is interesting in terms of that it discovers generalized frequent itemsets that are not *absolutely matched* by slightly relaxing the notion of traditional frequent itemset. This phenomenon motivates us to develop an automatic method to mine such kind of knowledge. This task is called mining fault tolerant (FT) frequent itemsets (Pei et al., 2001).

Given the definition of mining FT frequent itemset if the look again at the database of Table 1. Suppose the $(min\_sup^{\delta}) = 3$ and the $(item\_sup^{\delta}) = 2$. Suppose one mismatch is allowed, i.e., fault tolerant $(\delta = 1)$. The itemset $X = (abcef)$ is a FT frequent itemset since it's 4 out of 5 items are present in FT transactions 10,20 and 50 which qualifies $min\_sup^{\delta} = 3$, and each single item *a, b, c, e* and *f* is present in at least two transactions with qualifies $(item\_sup^{\delta}) = 2$ threshold. (Pei et al., 2001) proposed FT-Apriori algorithm for mining FT frequent itemsets from transactional databases. FT-Apriori uses candidate generation-and-test approach for mining FT frequent itemsets. Although, the performance of FT-Apriori is efficient when database is sparse and FT support thresholds are given large. However, FT-Apriori encounters difficulties and takes long processing time for dense and spare databases if the FT support thresholds are given small. We list here main limitations of FT-Apriori that do not make it an attractive solution for mining FT frequent itemsets.

- FT-Apriori is based on Apriori-like candidate generation-and-test approach. This approach is not efficient for databases having large number of items. For example, to mine complete set of FT frequent itemsets of a database with 200 items. FT-Apriori has to generate and test all the 2^{200} candidates.
- FT-Apriori applies bottom-up search mechanism and this enumerates each subset of itemset X before mining itemset X. This implies that in order to produce FT frequent itemsets of length Y, the algorithm must generate all subsets of Y which are 2^Y, since all subsets must be frequent. This exponential complexity of FT-Apriori fundamentally restricts the algorithm to mine complete set of itemsets in a reasonable time limit.
- To mine FT frequent itemsets of length Y the FT-Apriori requires full scan of database multiple times for counting support of itemsets. These scans are costly when the database is large and number of candidates to be examined are numerous.

To overcome these limitations in this paper we proposed a new approach for mining FT frequent itemsets using pattern growth approach (FT-PatternGrowth). FT-PatternGrowth adopts a divide-and-conquer technique and recursively projects a transactional database into a set of smaller projected transactional databases and mines FT frequent itemsets in each projected database by exploring only locally frequent items. This mines the complete set of FT frequent itemsets and substantially reduces those candidate itemsets that do not exist in the database (Han & Pei, 2014; Han et al., 2000; Han, Pei, Yin, & Mao, 2004). The major advantage of mining FT frequent itemsets using pattern growth approach is that it removes two costly operations of Apriori heuristic: candidate generate-and-test and repeatedly scanning of database for counting support of itemsets. The first scan of database counts the support of all frequent items of length one. The second scan of database builds a compact data structure called frequent pattern (FP)-tree. Each node of FP-tree corresponds to an item which was found frequent in first scan of database. Next, all FT frequent itemsets are mined directly from this FP-tree without scanning the database multiple times. The approach traverses search space using depth first order and during traversing each node it generates FT frequent itemsets using conditional patterns and builds compacted child FP-trees for mining FT frequent itemsets of next level. We tested our approach on several benchmarks databases and found computationally efficient than FT-Apriori.

The remainder of this paper is structured as follows. Section 2 reviews related work on mining FT frequent itemsets. In Section 3 we provide formal definition of mining FT frequent itemsets. In Section 4 we explain design and construction of pattern growth approach for mining FT frequent itemsets. Section 5 explains experimental setup and databases and analyzes the performance of algorithms on benchmark databases. Finally, Section 6 briefly summarizes the key results of our work.

2. Related work

This section provides a review on related algorithms for mining FT frequent itemsets. We start this section by first introducing some applications for FT frequent itemsets, and then we introduce related algorithms of FT frequent itemsets by providing their descriptions and limitations.

2.1. Applications of mining FT frequent itemsets

Li and Wang (2015) used the concept of FT frequent itemsets to mine FT frequent subgraphs from graph databases. They found that traditional exact matching algorithms generates only frequent subgraphs which have exact match in the graph databases. Thus, the interesting subgraphs could be left undiscovered if their are slightly different occurrences of edges in databases. They proposed algorithm using Apriori heuristic to mine FT frequent subgraphs. They also enhanced the working of algorithm by

mining non-redundant representative frequent subgraphs which further summarizes the frequent subgraphs by allowing approximate number of matches in a graph database. They performed experiments on both real as well as synthetic databases and found their approach more efficient than traditional algorithms.

Morales-González, Acosta-Mendoza, Alonso, Reyes, and Medina-Pagola (2014) used FT frequent subgraphs for image classification. They designed a classification framework in which frequent approximate subgraphs of images are utilised for classification features. They tested their approach on two real images databases and reported better classification accuracy than non mining approaches by keeping in view the fact that FT frequent subgraph mining is a better approach than exact mining approach for this particular task.

Ashraf and d. Tabrez Nafis (2017) proposed FT frequent itemset mining algorithms for both certain and uncertain composite datasets. In experiments they showed their algorithms are efficient for mining such patterns. They also discovered whenever the frequent itemset mining is done on distributed computing environment, the problem of false positive and false negative can also be handled accordingly. Kargupta, Han, Yu, Motwani, and Kumar (2008) presented an approach for mining approximate frequent sequential patterns. Through experiments they showed their approach is efficient to mine globally repeating approximate sequential patterns which could not be discovered through existing exact matching techniques.

Lee, Peng, and Lin (2009) developed algorithms for mining itemsets from biological databases by using FT frequent itemsets. They showed the number of tolerable faults occurred in a proportional FT itemsets are directly proportional to the length of the itemsets. They proposed two algorithms to solve this problem. First algorithm is based on Apriori heuristic which mines all FT itemsets with any number of faults occurred. The second algorithm divides complete set of FT itemsets in groups keeping in view a set ratio of tolerable faults which returns the mined itemsets from each group. They showed the working of their algorithms on real databases and reported epitopes of spike protein of SARS-CoV in resulting itemsets and reported FT frequent itesmets technique is better than exact matching techniques.

Besson, Pensa, Robardet, and Boulicaut (2005) proposed a method to mine extensions of bi-set itemsets with fault tolerant factor. They also evaluated three declared specifications of FT bi-sets by considering constraints based mining methodology. As a result, their mining framework posted a better and comprehensive understanding on the requisite trade-off between pattern extraction feasibility, ease of interpretation, relevance and completeness of these fault tolerant patterns. They showed experimental demonstration empirically on real-life medical and synthetic databases.

2.2. FT frequent itemset mining algorithms

Majority of algorithms proposed in recent years are based on candidate generation-and-test approach. We presented brief descriptions and limitations of these algorithms. These algorithms apply a top down complete search space exploration. These algorithms prune infrequent FT itemset using anti-monotone Apriori heuristic. The major drawbacks of these algorithms are repeatedly scanning of full database for counting itemset support, and generating too many candidates including those that do not exist in the database.

Pei et al. (2001) proposed `FT-Apriori` algorithm. `FT-Apriori` in based on candidate generation-and-test approach. The algorithm applies a top down complete search space exploration. The algorithm prunes infrequent FT itemset using anti-monotone Apriori heuristic: i.e., if any FT itemset of length k is discovered infrequent, then it discards all of its supersets since they too be infrequent. The major drawbacks of `FT-Apriori` is repeatedly scanning of full database for counting itemset support, and generating too many candidates including those that do not exist in the database. For example, to mine FT frequent itemsets of a database with 200 items. `FT-Apriori` has to generate and test all the 2^{200} candidates.

To avoid costly repeatedly scanning of database, Koh and Yo (2005) proposed an algorithm called `VB-FT-Mine`. `VB-FT-Mine` scans the database only once and constructs bit-vectors for each item. `VB-FT-Mine` then applies depth-first pattern generation approach to generate candidate itemsets. The bit-vectors of candidate itemsets are obtained systematically, and the `VB-FT-Mine` quickly counts the itemset support by applying bitwise operators on bit vectors. Although, bit-vectors increase the performance of `VB-FT-Mine` by quickly counting itemsets support, however, similar to `FT-Apriori` the `VB-FT-Mine` generates many non-existing candidate itemsets.

Bashir, Halim, and Baig (2008) proposed an algorithm for mining FT frequent itemset using pattern growth approach. The main limitation of their algorithm is that it constructs more than one FP-trees for each itemset to mine its supersets. For example, to mine supersets of itemset X under FT factor $\delta = 2$. The algorithm constructs three FP-tress. The algorithm constructs first FP-tree for storing all transactions of database that have mismatch factor $\delta = 0$. The algorithm then constructs second FP-tree for storing all transactions that have mismatch factor $\delta = 1$. The algorithm then constructs third FP-tree for storing all transactions that have mismatch factor $\delta = 2$. Koh and Lang (2010) proposed an algorithm for mining FT-frequent itemset using pattern growth approach. Similar to Bashir et al. (2008) approach, the algorithm constructs multiple FP-trees for each itemset to mine its supersets. For example to mine supersets of itemset $X = (ab)$ under FT factor $\delta = 2$, the algorithm constructs $2^{|X|}$ number of FP-trees. The algorithm constructs first FP-tree for storing all transactions that contain both items: a and b (ab). The algorithm constructs second FP-tree for storing all transactions that contain only item a $(a\bar{b})$. The algorithm constructs third FP-tree for storing all transactions that contain only item b $(\bar{a}b)$. Finally, it constructs fourth FP-tree to store all transactions that do not contain both items: a and b $(\overline{ab})$.

Both algorithms based on pattern growth approach construct multiple FP-trees for mining itemsets. Due to constructing multiple FP-trees the transactions that share a similar prefix are split into multiple trees. Thus, these algorithms could not gain full benefit of FP-tree for counting itemset support. For large databases and with low support thresholds both algorithms consume large main memory for mining itemsets. The pattern growth presented in this paper does not create multiple FP-trees. If an itemset has multiple mismatch transactions, our approach maps all mismatch transactions into a single FP-tree. Thus for large databases and with low support thresholds our algorithm is more space efficient than related algorithms.

To discover more interesting FT itemsets, Lee and Lin (2006) relaxed the definition of mining FT frequent itemsets by mining proportional FT frequent itemsets. The concept of mining proportional FT itemsets is similar to traditional FT itemsets, however, the fault tolerant factor in proportional FT itemsets is proportional to the length of itemset. Thus, the definition of proportional discovers much large number of itemsets than the FT itemset mining definition proposed in (Pei et al., 2001). Our proposed algorithm in based on the FT definition proposed in (Pei et al., 2001). Thus, the processing time of our proposed algorithm cannot be directly comparable with the algorithm proposed in (Lee & Lin, 2006). In (Lee et al., 2009), authors discussed the applications of proportional FT frequent itemsets in bioinformatics. To discover proportional FT frequent itemsets in a reasonable time Liu and Poon (2014, 2018) proposed efficient heuristic method to mine ap-

proximation version of the itemsets. Their study showed heuristic algorithm is much faster than the exact algorithms while the error is acceptable. In all studies on mining proportional FT frequent itemsets the authors proposed algorithms by mining itemsets on the basis of Apriori-like candidate generation-and-test property. However, no effort is made how to use FP-tree structure and pattern growth for increasing the speed of counting itemset support and reducing the number of candidate itemsets. Our work is different to this research. Our proposed algorithm utilises FP-tree for quickly counting itemset support and reduces the number of candidate itemsets using pattern growth approach.

3. Fault tolerant (FT) frequent itemset mining: Problem statement

The FT frequent itemset mining problem was first introduced by Pei et al. (2001) as *fault-tolerant frequent pattern mining: problems and challenges*.

Let $I = i_1, \ldots, i_m$ be a set of items. An itemset $X \subset I$ is a subset of items, an itemset with X items is called an itemset of length $|X|$. A transaction $T = (tid, t)$ is a tuple where tid is a transaction-id and t is a transaction of length n with set of items $t = t_1, \ldots, t_n$. A transaction $T = (tid, t)$ is said to contain itemset Y if Y is subset of t. A transaction database TDB is a set of transactions. The support of an itemset X in transaction database TDB, denoted as $sup(X)$, is the number of transactions in TDB containing X. Given a transactional database TDB and a minimum support threshold $min\_sup > 0$, X is a frequent itemset it it has $sup(X) \geq min\_sup$.

Given a user-defined fault tolerant (FT) factor (δ), a transaction t is a FT-transaction t^{δ} if it contains at least $(|X| - \delta)$ number of items of X. An itemset X with length greater than $(\delta + 1)$ is a FT frequent itemset if it satisfies the following two conditions.

- Given (min_sup^{δ}) minimum itemset support under fault tolerant (FT) factor (δ), the itemset X is FT frequent itemset if it has support of at least (min_sup^{δ}) number of FT-transactions.
- Each individual item i of X must appear in at least m number of FT-transactions of X. The m is the minimum item support $(item\_sup^{\delta})$ under fault tolerant factor δ.

Given the FT frequent itemset mining definition above if the look again at the database of Table 1. Suppose the $(min\_sup^{\delta}) = 3$ and the $(item\_sup^{\delta}) = 2$. Suppose one mismatch is allowed, i.e., fault tolerant $(\delta = 1)$. The itemset $X = (abcef)$ is a FT frequent itemset since it's 4 out of 5 items are present in FT transactions 10, 30 and 50 which qualifies $min\_sup^{\delta} = 3$, and each single item *a, b, c, e* and *f* is present in at least two transactions with qualifies $(item\_sup^{\delta}) = 2$ threshold.

4. Mining fault tolerant (FT) frequent itemsets using pattern growth: Design and construction

The algorithm mines FT frequent itemsets using two phases. In first phase, the algorithm mines all itemsets directly from the FP-tree of transactional database which have itemset length equals to $(\delta + 1)$. The second phase of algorithm mines itemsets which have itemset length greater than $(\delta + 1)$. Both phases construct FP-trees for mining itemsets.

FP-tree is a compact data structure which represents complete information of transactional database (Han & Pei, 2014; Han et al., 2000; Han et al., 2004). FP-tree avoids costly candidate generation-and-test and multiple scans of database. Each transaction of database is mapped to a branch of FP-tree. If multiple transactions share a similar set of items, the shared parts of transactions are merged into a single branch. The merging of transactions not only increases the scalability of algorithm for large databases but also improves the processing speed of algorithm for counting itemset support. To facilitate tree traversal a header table is constructed for items. This header table contains head pointers of items of FP-tree. Nodes in the tree with similar items are linked together by making linked lists of items. For mining, the head pointers and linked lists of items are used for generating candidate itemsets.

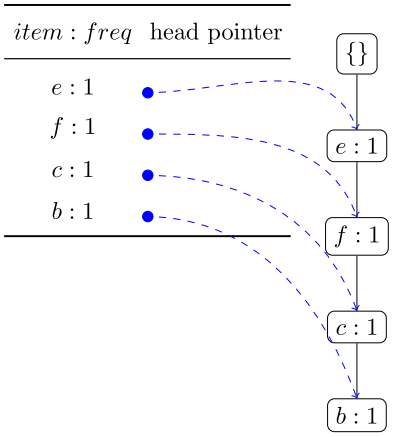


Fig. 1. FP-tree after inserting first transaction.

Example: Table 1 shows a transactional database. Let the *minimum itemset support* $(min\_sup^{\delta}) = 3$ and the *minimum item support* $(item\_sup^{\delta}) = 2$. Suppose two mismatches are allowed, i.e., FT factor $\delta = 2$.

The first scan of database derives a list of frequent items that have frequency greater or equals to $item\_sup^{\delta}$. All items that have support less than $item\_sup^{\delta}$ are removed from the transactions. This is because, if an item has support less than $item\_sup^{\delta}$ then it cannot become part of any FT frequent itemset. The items are ordered in transactions by decreasing frequency. This ordering is important since each path of FP-tree follows this order. The scan of database discovers the following frequent items, ⟨(*e*: 4), (*f*: 4), (*d*: 4), (*a*: 4), (*c*: 3), (*b*: 3)⟩, the number after *":"* indicates support of item. All transactions are mapped in the FP-tree and if multiple transactions share a similar set of items, the shared part is merged in a common branch.

In the second scan of database the algorithm constructs the FP-tree. The scan of the first transaction constructs the first branch of FP-tree ⟨(*e, f, c, b*)⟩ (see Fig. 1). The frequent items in the transaction are ordered according the order in the list of frequent items. The second transaction has ordered frequent items ⟨(*e, f, a, b*)⟩. Items *e,f* of second transaction share a common prefix with the existing path ⟨(*e, f, c, b*)⟩, the count of each shared node along the prefix is incremented by 1. One new chid node (*a*: 1) is created and linked with the parent node (*f*: 2), and another child node (*b*: 1) is created and linked with the parent node (*a*: 1) (see Fig. 2). The third transaction ⟨(*f, d, a*)⟩ does not share any common prefix with the existing tree, therefore it leads to the construction of the second branch of the tree (see Fig. 3). The fourth transaction ⟨(*e*)⟩ has only one item and it shares a common prefix with the branch (*e*: 2), the count of node (*e*: 2) is incremented by 1 (see Fig. 4). Other transactions are scanned using the same mechanism as desired for the first four transactions. If multiple transactions share a similar set of items, the common prefix of transactions is merged in a common branch. Fig. 5 shows complete FP-tree after inserting all transactions of database (Table 1).

4.1. Mining FT frequent itemsets of length equals to $(\delta + 1)$

All itemsets that have length equals to $(\delta + 1)$ can be mine directly from FP-tree of database. To examine the support of

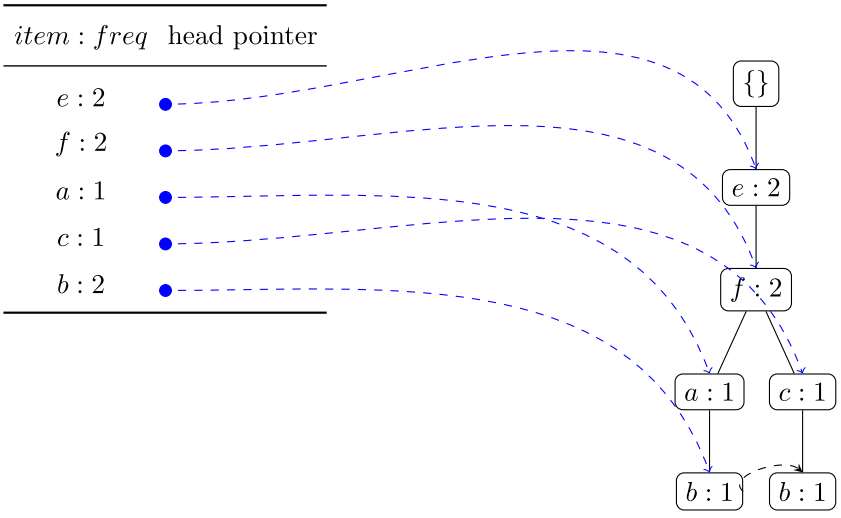


Fig. 2. FP-tree after inserting second transaction.

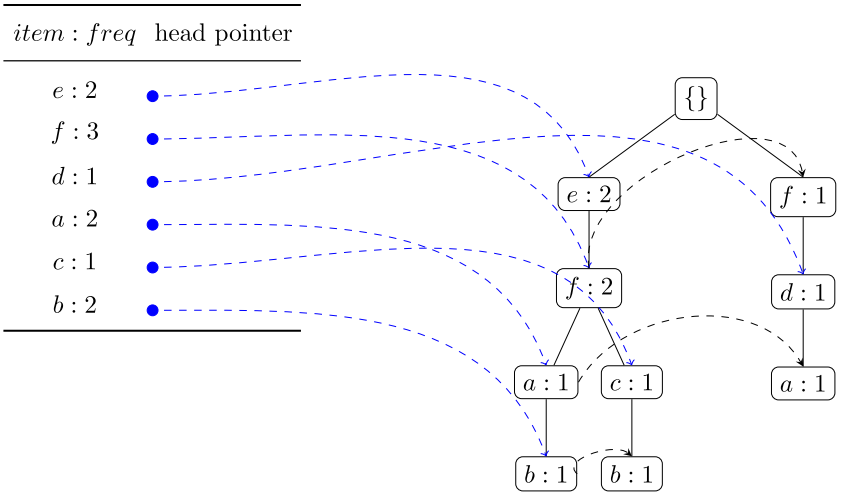


Fig. 3. FP-tree after inserting third transaction.

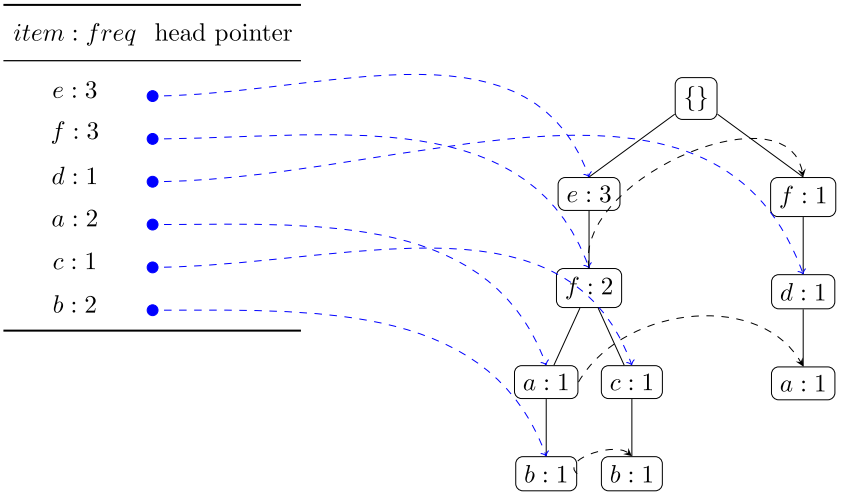


Fig. 4. FP-tree after inserting fourth transaction.

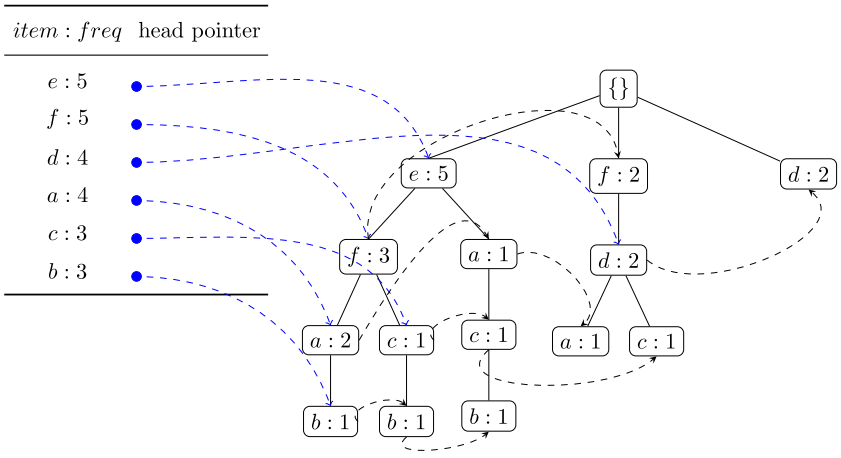


Fig. 5. Complete FP-tree after inserting all transactions.

itemsets of length equals to $(\delta + 1)$ the algorithm counts item support and itemset support directly from the conditional patterns of items stored in the FP-tree. **Example:** To examine whether itemset $X = (bca)$ is a frequent itemset. The algorithm generates conditional patterns of item *b*, item *c*, and item *a*. The algorithm ignores the conditional patterns of other items; this is because if a branch of FP-tree does not contain any item of *X* then the FT factor of the branch becomes $(\delta = 3)$, which does not qualify $(\delta = 2)$. The algorithm enumerates items of *X* in increasing frequency order, i.e., first *b*, then *c* and then *a*. For *X*, the FP-tree generates the following set of conditional patterns.

- For Item *b*, FP-tree generates conditional patterns $\langle efab: 1\rangle$, $\langle efcb: 1\rangle$, and $\langle eacb: 1\rangle$.
- FP-tree is again traversed for item *c* and the following three conditional patterns are discovered: $\langle efc: 1\rangle$, $\langle eac: 1\rangle$, and $\langle fdc: 1\rangle$. If a conditional pattern (c_B) is a subset of any already discovered conditional pattern (c_A) of previous item, then the support of c_B is subtracted from the support of c_A. If the support of c_B becomes zero, then the conditional pattern c_B is ignored. Since conditional patterns $\langle efc: 1\rangle$ and $\langle eac: 1\rangle$ are subsets of conditional patterns ($\langle efcb: 1\rangle$ and $\langle eacb: 1\rangle$) of item *b*, and after subtracting the support of $\langle efc: 1\rangle$ and $\langle eac: 1\rangle$ from the support of conditional patterns of *b*, the support of both conditional patterns become zero. Thus, both patterns are removed from the conditional patterns of *c*. After removing two patterns, the item *c* contains only conditional pattern $\langle fdc: 1\rangle$.
- The FP-tree is again traversed for item *a* and the following three conditional patterns are discovered: $\langle efa: 2\rangle$, $\langle ea: 1\rangle$, and $\langle fda: 1\rangle$. The pattern $\langle efa: 2\rangle$ is a subset of conditional pattern $\langle efab: 1\rangle$ of item *b*. The support of $\langle efa: 2\rangle$ is subtracted from the support of ($\langle efab: 1\rangle$), which makes the support of $\langle efa: 2\rangle$ equals to 1. The pattern $\langle ea: 1\rangle$ is a subset of conditional pattern $\langle eacb: 1\rangle$ of item *b*. The support of $\langle ea: 1\rangle$ is subtracted from the support of ($\langle eacb: 1\rangle$), which makes the support of $\langle ea: 1\rangle$ equals to 0. The pattern $\langle ea: 1\rangle$ is ignored. After removing, the item *a* contains two conditional patterns $\langle efa: 1\rangle$ and $\langle fda: 1\rangle$
- The three conditional patterns of *b* ($\langle efcb: 1\rangle$, $\langle efab: 1\rangle$, and $\langle eacb: 1\rangle$), one conditional pattern of *c* ($\langle fdc: 1\rangle$), and two conditional patterns of *a* ($\langle efa: 1\rangle$ and $\langle fda: 2\rangle$) are used for counting items support and itemset support. Since all items of itemset (bca) qualify $item\_sup^{\delta} = 2$, and (bca) qualifies itemset support $(min\_sup^{\delta} = 3)$. Therefore, (bca) is a FT frequent itemset of length three.

The FP-tree is continuously scan for generating conditional patterns of other itemsets of length equals to $(\delta + 1)$. The item support and itemset support are calculated by following the example of itemset (bca). Lines from 4 to 8 of Algorithm 1 show the pseudo code for mining FT frequent itemsets of length equals to $(\delta + 1)$.

Algorithm 1: Procedure for Mining Fault Tolerant Frequent Itemsets.

```
Data: Transactional database, min_sup^δ, item_sup^δ, Fault
      Tolerant factor (δ)
Result: All Fault Tolerant (FT) frequent itemests
1  Procedure FT-FP-Growth(Tree, α)
2  β = ∅;
3  if α == ∅ then
4      for each itemset X of length = |δ + 1| do
5          for each item i of X do
6              C = C ∪ i's conditional patterns from Tree;
7              if itemset − support(X) >= min_sup^δ then
8                  output X as FT frequent itemset;
9                  β = X;
10                 FT-FP-Tree(C, β);
11 else
12     if Tree contains a single path P then
13         for each item P_i in the P do
14             X = α ∪ P_i;
15             if itemset − support(X) >= min_sup^δ and each item
               i in X has item_support(i) >= item_sup^δ then
16                 output X as a FT frequent itemset;
17     else
18         for each header item h_i in the header table of Tree do
19             generate itemset X = α ∪ h_i;
20             for each path P of Tree do
21                 C = C ∪ generate conditional pattern from P;
22             if itemset − support(X) in C >= min_sup^δ and each
               item i of X in C has item_support(i) >= item_sup^δ
               then
23                 output X as a FT frequent itemset;
24                 β = X;
25                 FT-FP-Tree(C, β);
```

Table 2
Conditional patterns and FT-conditional patterns of itemset (*bca*).

Item	Conditional Patterns	FT-Conditional Patterns
b	*efcb*: 1	(⟨⟨*ef*⟩, ⟨*sup*: 1⟩, ⟨*δ*: 1⟩, ⟨*a*: 0, *c*: 1, *b*: 1⟩⟩)
b	*efab*: 1	(⟨⟨*ef*⟩, ⟨*sup*: 1⟩, ⟨*δ*: 1⟩, ⟨*a*: 1, *c*: 0, *b*: 1⟩⟩)
b	*eacb*: 1	(⟨⟨*e*⟩, ⟨*sup*: 1⟩, ⟨*δ*: 0⟩, ⟨*a*: 1, *c*: 1, *b*: 1⟩⟩)
c	*efc*: 1	Ignored, because it has been already discovered from item *b* (*efcb*: 1), and its support becomes zero after subtracting its support fromthe support of *efcb*: 1.
c	*eac*: 1	Ignored, because it has been already discovered from item *b* (*eacb*: 1), and its support becomes zero after subtracting its support fromthe support of *eacb*: 1.
c	*fdc*: 1	(⟨⟨*fd*⟩, ⟨*sup*: 1⟩, ⟨*δ*: 2⟩, ⟨*a*: 0, *c*: 1, *b*: 0⟩⟩)
a	*efa*: 2	(⟨⟨*ef*⟩, ⟨*sup*: 1⟩, ⟨*δ*: 2⟩, ⟨*a*: 1, *c*: 0, *b*: 0⟩⟩)
a	*ea*: 1	Ignored, because it has been already discovered from item *b* (*eacb*: 1), and its support becomes zero after subtracting its support fromthe support of *eacb*: 1.
a	*fda*: 1	(⟨⟨*fd*⟩, ⟨*sup*: 1⟩, ⟨*δ*: 2⟩, ⟨*a*: 1, *c*: 0, *b*: 0⟩⟩)

4.2. Mining FT frequent itemsets of length more than $(\delta + 1)$

To mine FT frequent itemsets of length greater than $(\delta + 1)$ we propose an approach for generating FT-FP-tree (fault tolerant FP-tree) and mining FT frequent itemsets from the FT-FP-tree. The FT-FP-tree is iteratively constructed for each FT frequent itemset of length $(\delta + 1)$. Similar to FP-tree, FT-FP-tree of an itemset X is a compact data structure which maps all conditional patterns of X in the tree. This helps in avoiding costly candidate generation-and-test and scanning database multiple times for generating itemsets that are supersets of X. The FT-FP-Tree of X is constructed from the FT-conditional patterns of X. The FT-conditional patterns of X are generated from the conditional patterns of items in X.

A conditional pattern is a FT-conditional pattern of mismatch factor f if has f number of items missing in the pattern. The value of f should be less or equals to δ. If multiple FT-conditional patterns share similar set of items, then all are merged into a single FT-conditional pattern. A FT-conditional pattern of itemset (X) contains four segments. The first segment contains items that can generate itemsets contain X. The second segment contains support of FT-conditional pattern. The third segment contains number of missing items of X in FT-conditional pattern. The fourth segment contains item support of each item of X, which is useful for counting support of items. To map segments of FT-conditional patterns on FT-FP-Tree the algorithm creates FT-conditional pattern table (*FT-CP-Table*) at the end of each branch of FT-FP-Tree. The first segment of pattern is directly mapped on the nodes of FT-FP-Tree. The other segments of patterns are mapped on the FT-CP-Tree. Each FT-CP-Table contains three columns. The first column maps first segment (δ) of FT-conditional pattern. The second column maps second segment (support of pattern). The third column maps fourth column (items support) of FT-conditional pattern.

For example, to construct FT-FP-tree of itemset $X = (bca)$. The algorithm generates conditional patterns using items *b, c*, and *a*.

- Item *b* contains three conditional patterns: ⟨*efab*: 1⟩, ⟨*efcb*: 1⟩, and ⟨*eacb*: 1⟩. These conditional patterns are converted into FT-conditional patterns. The pattern ⟨*efab*: 1⟩ is a FT-conditional pattern of FT factor $\delta = 1$ because item *c* is missing from the pattern. The pattern ⟨*efcb*: 1⟩ is a FT-conditional pattern of FT factor $\delta = 1$ because item *a* is missing from the pattern. The pattern ⟨*eacb*: 1⟩ is a FT-conditional pattern of FT factor $\delta = 0$ because no item is missing from the pattern. The pattern ⟨*efab*: 1⟩ is converted into FT-conditional pattern ⟨⟨*ef*⟩, ⟨*sup*: 1⟩, ⟨*δ*: 1⟩, ⟨*a*: 1, *c*: 0, *b*: 1⟩⟩. The FT-conditional pattern ⟨*efab*: 1⟩ has four segments. The first segment contains the items that generate supersets of itemset ((*bca*)). The second segment contains support of pattern. The third segment says one item of itemset ((*bca*)) is missing in the pattern. The fourth segment contains item support of each item of (*bca*). The pattern ⟨*efcb*: 1⟩ is converted into FT-conditional pattern ⟨⟨*ef*⟩, ⟨*sup*: 1⟩, ⟨*δ*: 1⟩, ⟨*a*: 0, *c*: 1, *b*: 1⟩⟩. The pattern ⟨*eacb*: 1⟩ is converted into FT-conditional pattern ⟨⟨*e*⟩, ⟨*sup*: 1⟩, ⟨*δ*: 0⟩, ⟨*a*: 1, *c*: 1, *b*: 1⟩⟩. All FT-conditional patters of item *c* are mapped into FT-FP-tree. The frequency ordering of items (that is discovered from the first scan of database) is followed for mapping items on branches.
- Item *c* generates three conditional patterns: ⟨*efc*: 1⟩, ⟨*eac*: 1⟩, and ⟨*fdc*: 1⟩. Conditional patterns ⟨*efc*: 1⟩ and ⟨*eac*: 1⟩ are ignored as both are subsets of conditional patterns of item *b* with similar support, and both are already mapped on FT-FP-tree of itemset (*bca*). ⟨*fdc*: 1⟩ is a FT-conditional pattern of FT factor $\delta = 2$ because items *b* and *a* are missing from the pattern. The pattern ⟨*fdc*: 1⟩ is converted into FT-conditional pattern ⟨⟨*fd*: 1⟩, ⟨*sup*: 1⟩, ⟨*δ*: 2⟩, ⟨*a*: 0, *c*: 1, *b*: 0⟩⟩.
- Item *a* generates conditional patterns ⟨*efa*: 2⟩, ⟨*ea*: 1⟩ and ⟨*fda*: 1⟩. The pattern ⟨*efa*: 2⟩ is a subset of pattern ⟨*efab*: 1⟩ of item *b*. Its support becomes 1 after subtracting its support from the support of pattern ⟨*efab*: 1⟩. The pattern ⟨*ea*: 1⟩ is a subset of pattern ⟨*eacb*: 1⟩ of item *b*. This pattern is ignored because its support becomes zero after subtracting its support from the support of pattern ⟨*eacb*: 1⟩. The pattern ⟨*efa*: 1⟩ and pattern ⟨*fda*: 1⟩ are FT-conditional patterns of FT factor $\delta = 2$ because items *c* and *b* are missing from the patterns. The pattern ⟨*efa*: 1⟩ is converted into FT-conditional pattern ⟨⟨*ef*⟩, ⟨*sup*: 1⟩, ⟨*δ*: 2⟩, ⟨*a*: 1, *c*: 0, *b*: 0⟩⟩. The pattern ⟨*fda*: 1⟩ is converted into FT-conditional pattern ⟨⟨*fd*⟩, ⟨*sup*: 1⟩, ⟨*δ*: 2⟩, ⟨*a*: 1, *c*: 0, *b*: 0⟩⟩.

All FT-CP-Tables at leaf nodes are linked to each other through pointers. Table 2 lists all conditional patterns and FT-conditional patterns of itemset (*bca*), and Fig. 6 shows FT-FP-tree of (*bca*).

4.3. Mining FT frequent itemsets from FT-FP-tree

The compact FT-FP-tree provides facility that subsequent mining of itemsets can be performed directly on the FT-FP-tree without scanning the database multiple times. In this section, we will show how to explore information stored on the branches of FT-FP-tree, and develop a mining approach for generating all FT frequent itemsets. Since FT-FP-tree of itemset X maps all transactions of X on tree that are needed for obtaining the possible FT frequent itemsets that contain X. Therefore, we observe the following interesting property from the FT-FP-tree for mining FT frequent itemsets.

For any FT frequent itemset X, all the possible FT frequent itemsets that contain X can be generated by traversing all conditional patterns of FT-FP-tree, staring from head of FT-CP-Table in the header table.

Example: Let us examine the mining method from the FT-FP-tree of itemset (*bca*) shown in Fig. 6. According to the list of

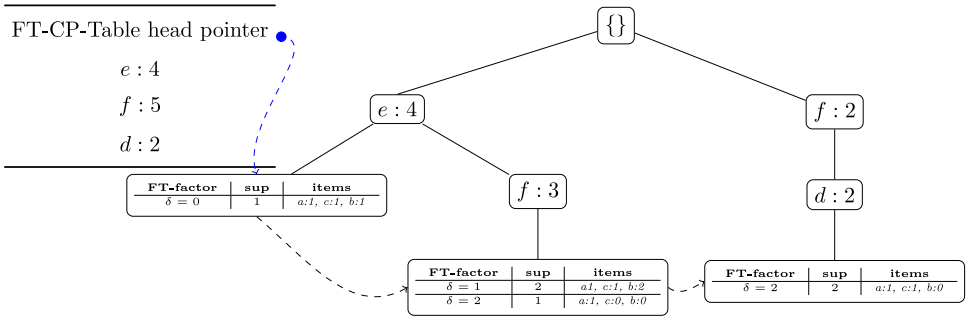


Fig. 6. FT-FP-tree of itemset (*bca*).

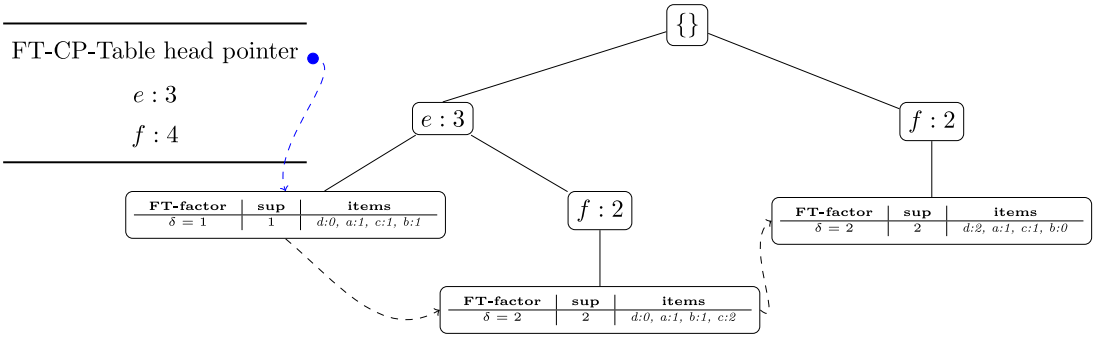


Fig. 7. FT-FP-tree of itemset (*bcad*).

frequent items in the header table the set of frequent itemsets contain itemset (*bca*) are divided into three subsets without overlap: (1) FT frequent itemsets having item *d*, (2) FT frequent itemsets having item *f*, and (3) FT frequent itemsets having item *e*. The algorithm discovers all these itemsets as follows.

To examine whether itemsets (*bcad*) is a FT frequent itemset and to generate supersets of itemsets (*bca*) having item *d*, the algorithm first examines the itemset support and items support of (*bcad*) form the FT-conditional patterns of itemset (*bcad*). The algorithm then generates supersets of (*bcad*) from the FT-FP-tree of itemset (*bcad*). FT-FP-tree of (*bcad*) is constructed from the conditional patterns of (*bcad*). The FT-conditional patterns of (*bcad*) are collected from the FT-FP-tree of itemset (*bca*) by traversing all pointers of the FT-CP-Table starting from the head pointer of FT-CP-Table. Note each row of FT-CP-Table generates an independent FT-conditional pattern. The pointers of FT-CP-Table drive following FT-conditional patterns:

- $\langle\langle fd\rangle, \langle sup\colon 2\rangle, \langle\delta\colon 2\rangle, \langle a\colon 1, c\colon 1, b\colon 0\rangle\rangle$,
- $\langle\langle ef\rangle, \langle sup\colon 2\rangle, \langle\delta\colon 1\rangle, \langle a\colon 1, c\colon 1, b\colon 2\rangle\rangle$,
- $\langle\langle ef\rangle, \langle sup\colon 1\rangle, \langle\delta\colon 2\rangle, \langle a\colon 1, c\colon 0, b\colon 0\rangle\rangle$, and
- $\langle\langle e\rangle, \langle sup\colon 1\rangle, \langle\delta\colon 0\rangle, \langle a\colon 1, c\colon 1, b\colon 1\rangle\rangle$.

The conditional pattern ($\langle\langle ef\rangle, \langle\delta\colon 2\rangle\rangle$) is ignored because it contains FT factor $\delta = 2$ and item *d* is missing from the pattern, this makes the FT factor equals to $\delta = 3$. All other conditional patterns qualify FT factor $\delta = 2$ which make the support count of itemset (*bcad*) equals to 5. Table 3 shows the FT-conditional patterns of (*dacb*). The support of items are collected from the FT-conditional patterns of (*dacb*) which makes the support of items: $\langle d\colon 2\rangle$, $\langle a\colon 3\rangle$, $\langle c\colon 3\rangle$, and $\langle b\colon 3\rangle$. All items qualify $item\_sup^{\delta} = 2$. Thus (*bcad*) is a FT frequent itemset of length four. The Fig. 7 shows the FT-FP-tree of (*bcad*).

To generate and examine the support of supersets of itemsets (*bcad*) the algorithm generates itemsets and FT-conditional patterns from the FT-FP-tree of (*bcad*). The itemsets contain (*bcad*) are divided into two subsets: (1) FT frequent itemsets having item *f*, and (2) FT frequent itemsets having item *e*.

Table 3
FT-conditional patterns of itemset (*bcad*).

FT-Conditional Patterns Discovered from FT-FP-tree of itemset (*bca*)	FT-Conditional Patterns used for Constructing FT-FP-tree of (*bcad*)
$\langle\langle fd\rangle, \langle sup\colon 2\rangle, \langle\delta\colon 2\rangle, \langle a\colon 1, c\colon 1, b\colon 0\rangle\rangle$	$\langle\langle f\rangle, \langle sup\colon 2\rangle, \langle\delta\colon 2\rangle, \langle d\colon 2, a\colon 1, c\colon 1, b\colon 0\rangle\rangle$
$\langle\langle ef\rangle, \langle sup\colon 2\rangle, \langle\delta\colon 1\rangle, \langle a\colon 1, c\colon 1, b\colon 2\rangle\rangle$	$\langle\langle ef\rangle, \langle sup\colon 2\rangle, \langle\delta\colon 2\rangle, \langle d\colon 0, a\colon 1, c\colon 1, b\colon 2\rangle\rangle$
$\langle\langle ef\rangle, \langle sup\colon 1\rangle, \langle\delta\colon 2\rangle, \langle a\colon 1, c\colon 0, b\colon 0\rangle\rangle$	Ignored because it has $\langle\delta\colon 2\rangle$ and item *d* is missing in the pattern
$\langle\langle e\rangle, \langle sup\colon 1\rangle, \langle\delta\colon 0\rangle, \langle a\colon 1, c\colon 1, b\colon 1\rangle\rangle$	$\langle\langle e\rangle, \langle sup\colon 1\rangle, \langle\delta\colon 1\rangle, \langle d\colon 0, a\colon 1, c\colon 1, b\colon 1\rangle\rangle$

To examine whether itemsets (*bcadf*) is a FT frequent itemset and to generate supersets of itemsets (*bcad*) having item *f*, the algorithm first examines the itemset support and items support of (*bcadf*) form the FT-conditional patterns of itemset (*bcadf*). The algorithm then generates supersets of (*bcadf*) from the FT-FP-tree of itemset (*bcadf*). FT-FP-tree of (*bcadf*) is constructed from the conditional patterns of (*bcadf*). The FT-conditional patterns of (*bcadf*) are collected from the FT-FP-tree of itemset (*bcad*) by traversing all pointers of the FT-CP-Table starting from head of FT-CP-Table (see Fig. 7). The pointers of FT-CP-Table drive FT-conditional pattern ($\langle\langle f\rangle, \langle sup\colon 2\rangle, \langle\delta\colon 2\rangle, \langle d\colon 2, a\colon 1, c\colon 1, b\colon 0\rangle\rangle$), FT-conditional pattern ($\langle\langle ef\rangle, \langle sup\colon 2\rangle, \langle\delta\colon 2\rangle, \langle d\colon 0, a\colon 1, c\colon 1, b\colon 2\rangle\rangle$), and FT-conditional pattern ($\langle\langle e\rangle, \langle sup\colon 1\rangle, \langle\delta\colon 2\rangle, \langle d\colon 0, a\colon 1, c\colon 1, b\colon 1\rangle\rangle$). Table 4 shows the FT-conditional patterns of (*bcadf*). All patterns qualify FT factor $\delta = 2$ thus make the support count of itemset (*bcadf*) equals to 5. The support of items collected from the FT-conditional patterns of (*bcadf*) are: $\langle f\colon 4\rangle$, $\langle d\colon 2\rangle$, $\langle a\colon 3\rangle$, $\langle c\colon 3\rangle$, and $\langle b\colon 3\rangle$. The item supports of all items of (*bcadf*) qualify $item\_sup^{\delta} = 2$. Thus, (*bcadf*) is a FT frequent itemset of length five. The Fig. 8 shows the FT-FP-tree of (*bcadf*).

In next iteration, the algorithm generates and examines the support of supersets of itemsets (*bcadf*). The algorithm generates itemsets and FT-conditional patterns from the FT-FP-tree of (*bcadf*). Since, FT-FP-tree of (*bcadf*) has only one item *e*, therefore,

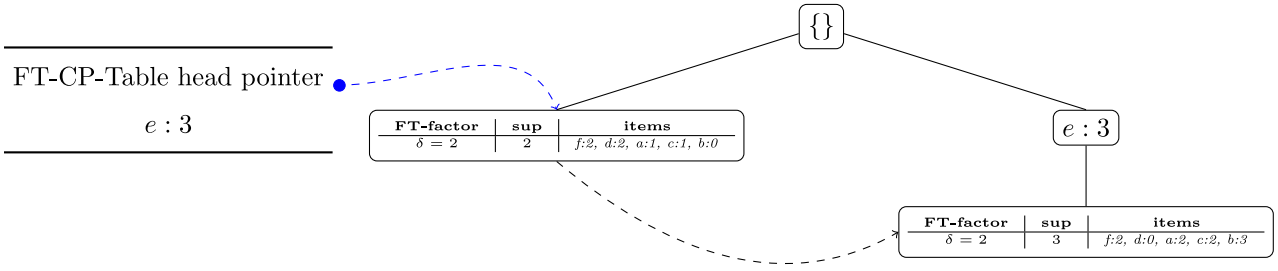


Fig. 8. FT-FP-tree of itemset (*bcadf*).

Table 4
FT-conditional patterns of itemset (*bcadf*).

FT-Conditional Patterns Discovered from FT-FP-tree of itemset (*bcad*)	FT-Conditional Patterns used for Constructing FT-FP-tree of (*bcadf*)
$\langle\langle f\rangle, \langle sup: 2\rangle, \langle\delta: 2\rangle, \langle d: 2, a: 1, c: 1, b: 0\rangle\rangle$	$\langle\langle\rangle, \langle sup: 2\rangle, \langle\delta: 2\rangle, \langle f: 2, d: 2, a: 1, c: 1, b: 0\rangle\rangle$
$\langle\langle ef\rangle, \langle sup: 2\rangle, \langle\delta: 2\rangle, \langle d: 0, a: 1, c: 1, b: 2\rangle\rangle$	$\langle\langle e\rangle, \langle sup: 2\rangle, \langle\delta: 2\rangle, \langle f: 2, d: 0, a: 1, c: 1, b: 2\rangle\rangle$
$\langle\langle e\rangle, \langle sup: 1\rangle, \langle\delta: 1\rangle, \langle d: 0, a: 1, c: 1, b: 1\rangle\rangle$	$\langle\langle e\rangle, \langle sup: 1\rangle, \langle\delta: 2\rangle, \langle f: 0, d: 0, a: 1, c: 1, b: 1\rangle\rangle$

the algorithm examines the itemset support and items support of itemsets (*bcadfe*) from the FT-conditional patterns of (*bcadfe*). The FT-conditional patterns are collected from the FT-FP-tree of (*bcadf*). The pointers of FT-CP-Table drive FT-conditional pattern ($\langle\langle\rangle, \langle sup: 2\rangle, \langle\delta: 2\rangle, \langle f: 2, d: 2, a: 1, c: 1, b: 0\rangle\rangle$) and FT-conditional pattern ($\langle\langle e\rangle, \langle sup: 3\rangle, \langle\delta: 2\rangle, \langle f: 2, d: 0, a: 2, c: 2, b: 3\rangle\rangle$). The conditional pattern ($\langle\langle\rangle, \langle sup: 2\rangle, \langle\delta: 2\rangle, \langle f: 2, d: 2, a: 1, c: 1, b: 0\rangle\rangle$) is ignored because it contains FT factor $\delta = 2$ and item e is missing in the pattern, this makes the FT factor equals to $\delta = 3$. The second FT-conditional pattern has item support for item d equals to 0. The total item support of d in second FT-conditional patterns does not qualify $item\_sup^{\delta} = 2$, thus, the itemset (*bcadfe*) is a FT infrequent itemset.

The algorithm backtracks to FT-FP-tree of itemset (*bcad*) and examines the FT conditions of itemset (*bcade*). Similarly, the algorithm mines the remaining FT-frequent itemsets by generating their corresponding FT-conditional patterns and FT-FP-trees, and then performs mining on them, respectively. Lines from 18 to 25 of Algorithm 1 and Algorithm 2 show pseudo code of mining FT frequent itemsets of length greater than $(\delta + 1)$.

Algorithm 2: Procedure for constructing FT-FP-Tree of itemset.

Data: Conditional patterns of itemset X
Result: Construct FT-FP-Tree of X
1 **Procedure** FT-FP-Tree(C, X)
2 **for** *each conditional pattern c in C* **do**
3 convert c into FT-conditional pattern c^{α};
4 **if** c^{α} *contains at-least* $(|X| - \alpha)$ *items of* X **then**
5 $C^{\alpha} = C^{\alpha} \cup c^{\alpha}$;
6 construct FT-FP-Tree of X using C^{α};
7 **if** *FT-FP-Tree of X != ∅* **then**
8 FT-FP-Growth(FT-FP-Tree of X, X);

5. Experiments

To test the performance of algorithms we used three real databases and one synthetic database. The four databases are *Retail, BMSWebView1, FoodMart*, and *T10I4D100K*, which are frequently used in previous studies on mining frequent itemsets. The *Retail, BMSWebView1, FoodMart* and *T10I4D100K* are downloaded from FIMI repository (http://fimi.ua.ac.be) and (http://www.kdd.org/kdd-cup/view/kdd-cup-2000). Table 5 shows the characteristics of these databases, where columns of table show the average transaction length, the number of items and the number of transactions of each database.

We compare the performance of our algorithm `FT-PatternGrowth` with `FT-Apriori` (Pei et al., 2001), `VB-FT-Mine` Koh & Lang, 2010), and `FT-TreeBased` (Bashir et al., 2008). `FT-Apriori` mines the FT frequent itemsets using candidate generation-and-test approach. The limitation of `FT-Apriori` is it generates many candidate itemsets including those that do not exist in the database. `FT-Apriori` counts the support of itemset using costly full scan of database. `VB-FT-Mine` also mines itemsets using candidate generation-and-test approach. It also generates many nonexisting candidate itemsets. However, `VB-FT-Mine` improves the speed of counting itemset support by storing transactions in bit-vectors. The support of itemsets are counted efficiently by performing bitwise-AND operators on bit-vectors (Burdick et al., 2005). `FT-TreeBased` mines FT frequent itemsets using pattern growth approach. The major limitation of `FT-TreeBased` is it constructs multiple FP-trees for each discovered itemset to mine its supersets. In `FT-TreeBased` multiple transactions of database that share a common prefix of items are distributed into multiple FP-trees if they have different fault tolerant mismatch. Thus `FT-TreeBased` does not gain full advantage of FP-tree for counting itemset support. All algorithms are implemented in C++[1] The experiments are performed on MacBook Pro-3.2 GHz processor with main memory of size 8GB. We analyze the performance of all algorithms on two FT factors ($\delta = 1$ and $\delta = 2$) with various values of minimum item supports ($item\_sup^{\delta}$). Table 6 explains the setting of experiments. For *Retail* and *T10I4D100K* datasets we compare the performance of algorithms with FT factors $\delta = 1$, $\delta = 2$, $\delta = 3$, $\delta = 4$ and $\delta = 5$.

The runtime comparisons of all algorithms are shown in Figs. 9–12. Note that, execution time here means the total execution time of algorithm, which is the period between providing input and mining all FT frequent itemsets. On low support thresholds the algorithms take very long processing time, therefore, we finish the execution of an algorithm when it takes more than 23000 seconds.

Fig. 9 shows the processing time of all algorithms on the *Retail* database. Results shows `FT-PatternGrowth` is efficient than `FT-Apriori`, `VB-FT-Mine`, and `FT-TreeBased` for each minimum item support. We observe that when the minimum item support is given very small, the `FT-PatternGrowth` finish its execution in less processing time than other algorithms. `FT-TreeBased` is efficient than `VB-FT-Mine` and `FT-Apriori`.

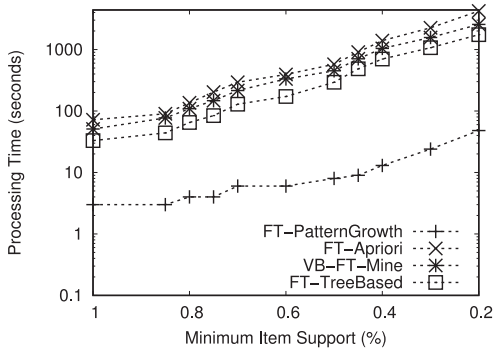
[1] The C++ Implementation of our algorithm (`FT-PatternGrowth`) is available on the following link to download (https://sites.google.com/site/drshariqbashir/shariqpublications/FTFPPatternGrowth.zip).

Table 5
Characteristics of transactional databases.

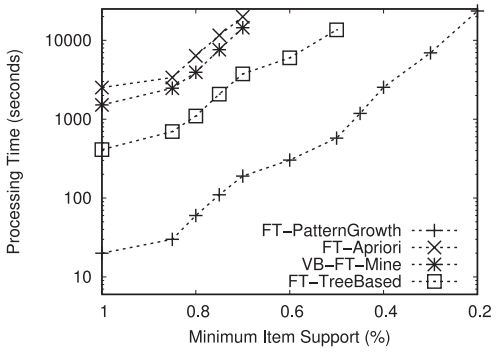
Database	Number of Transactions	Number of Items	Avg. Transaction Length
Retail	88,162	16,470	10
BMSWebView1	59,601	497	3
FoodMart	4,141	1,559	4
T10I4D100K	100,000	870	11

Table 6
Characteristics of experiment settings.

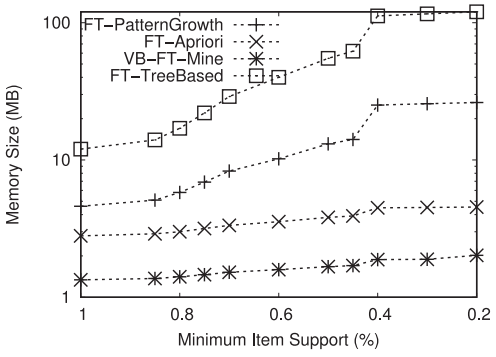
Database	Number of transactions	δ	$item\_sup^{\delta}$	min_sup^{δ}
Retail	88,162	1 *and* 2	0.2% *to* 1%	1%
BMSWebView1	59,601	1 *and* 2	0.05% *to* 0.35%	0.4%
FoodMart	4,141	1 *and* 2	0.01% *to* 0.06%	0.06%
T10I4D100K	100,000	1 *and* 2	1% *to* 2%	2%



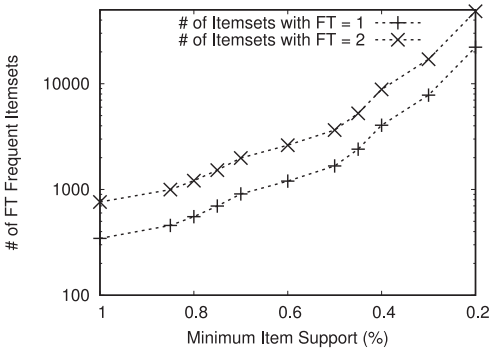
(a) **Retail with** $\delta = 1$ **and** $min\_sup^{\delta} = 1\%$



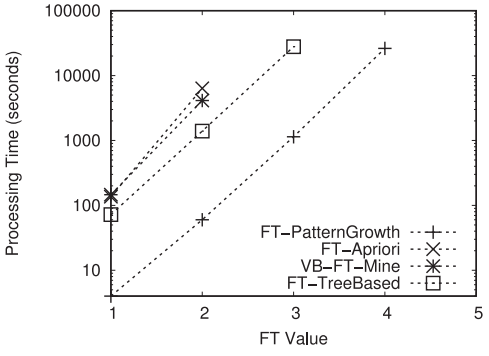
(b) **Retail with** $\delta = 2$ **and** $min\_sup^{\delta} = 1\%$



(c) **Memory size with** $\delta = 1$ **and** $min\_sup^{\delta} = 1\%$

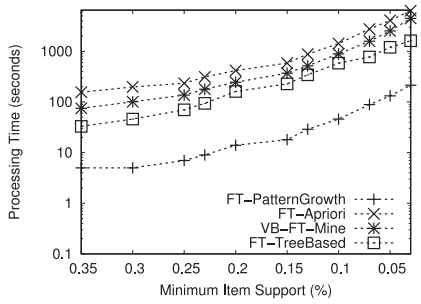


(d) **Number of FT frequent itemsets**

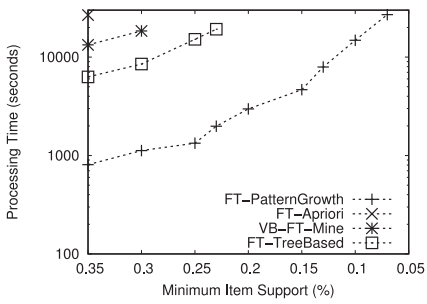


(e) **Scalability of algorithms on varying** δ **values with** $min\_sup^{\delta} = 1\%$ **and** $item\_sup^{\delta} = 0.8\%$

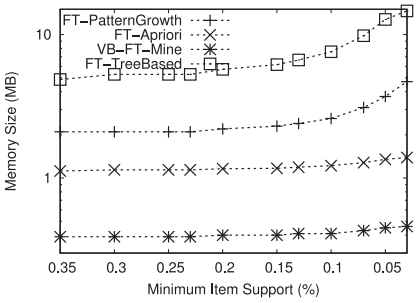
Fig. 9. The performance of FT frequent itemset mining algorithms on *Retail* database. (d) Number of FT frequent itemsets discovered with $\delta = 1$, $\delta = 2$ and $min\_sup^{\delta} = 1\%$.



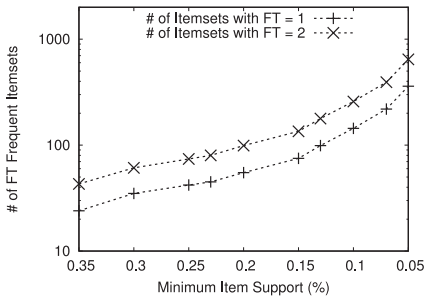
(a) **BMSWebView1 with $\delta = 1$ and $min\_sup^{\delta} = 0.4\%$**



(b) **BMSWebView1 with $\delta = 1$ and $min\_sup^{\delta} = 0.4\%$**



(c) **Memory size with $\delta = 1$ and $min\_sup^{\delta} = 0.4\%$**



(d) **Number of FT frequent itemsets**

Fig. 10. The performance of FT frequent itemset mining algorithms on *BMSWebView1* database. (d) Number of FT frequent itemsets discovered with $\delta = 1$, $\delta = 2$ and $min\_sup^{\delta} = 0.4\%$.

This is because FT-PatternGrowth generates itemsets using pattern growth approach. Whereas the FT-Apriori and VB-FT-Mine generates itemsets using candidate generation-and-test approach, and this approach generates many candidate itemsets including those that do not exist in the database. VB-FT-Mine is efficient than FT-Apriori because it counts the support the itemset using efficient bit-vectors technique. On $\delta = 2$, the FT-TreeBased, VB-FT-Mine and FT-Apriori could not finish their execution within 23000 seconds when the minimum item support is given less than 0.4%. Fig. 9(d) shows complete set of mined FT frequent itemsets with FT $\delta = 1$ and $\delta = 2$.

Figs. 10 –12 show the runtime of all algorithms on the other databases (*BMSWebView1, T10I4D100K* and *FoodMart*). Once again, FT-PatternGrowth mines itemsets more efficiently than other algorithms on each minimum item support. FT-PatternGrowth consistently outperforms FT-Apriori, VB-FT-Mine and FT-TreeBased. Similar to *Retail* database FT-Apriori, VB-FT-Mine, and FT-TreeBased could not finish their execution within 23000 seconds when the minimum item support is given very small. The FT-Apriori and VB-FT-Mine are slower on all support thresholds because both algorithms generate candidate itemsets using Apriori property. The FT-PatternGrowth is efficient because it generates candidate itemsets using pattern growth approach. Moreover, FT-PatternGrowth counts the support of candidate FT itemsets efficiently from few branches of FP-trees as multiple transactions of database that share a common set of items are grouped into common branches of FP-tree. FT-Apriori examines the support of candidate FT itemsets using all transactions of database as it does not group transactions that share common set of items. This increases the processing time of FT-Apriori.

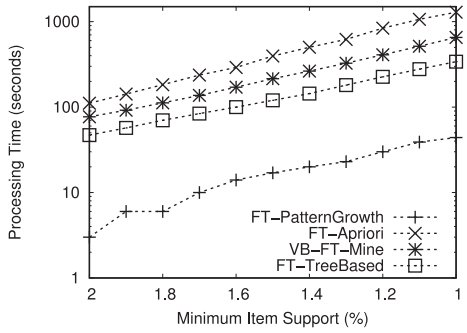
Figs. 9 (c), 10(c), 11(c) and 12(c) compare the performance in terms of how much different algorithms consume memory during execution. VB-FT-Mine memory consumption is very small as compared to all other algorithms. This is because VB-FT-Mine saves transactions in bit-vectors, and multiple transactions are compressed in a single element of bit-vectors. FT-Apriori memory consumption is second best. FT-Apriori creates linked list for each frequent item, and transactions are mapped in the linked lists of frequent items. FT-PatternGrowth memory consumption is higher than VB-FT-Mine and FT-Apriori. FT-PatternGrowth maps transactions in the FP-tree and large part of memory is consumed for creating nodes of tree and connecting parent and child nodes. FT-TreeBased memory consumption is higher than all algorithms due to creating multiple FP-trees. In FT-TreeBased multiple transactions of database that share a common prefix of items are distributed into multiple FP-trees if they have different fault tolerance mismatch.

5.1. Scalability analysis

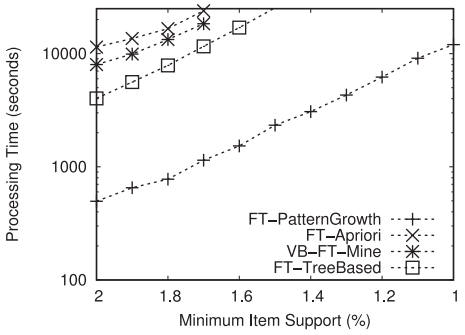
In above experiments we examine the performance of algorithms on various minimum item support ($item\_sup^{\delta}$). However, further analyses are required to analyze the scalability of algorithms on varying number of transactions and transaction length. To test the scalability of FT-PatternGrowth against the number of transactions, a set of random transactions are selected ranges from 10*k* to 90*k*. We select only *Retail* and *T10I4D100K* databases as both databases are sparse and have transactions of varying length. All algorithms are tested over them using the similar values of support thresholds.

The advantage of FT-PatternGrowth is dramatic in databases with long patterns, which is challenging to the algorithms that mine the complete set of FT frequent patterns. The results on mining the real databases *Retail* and *T10I4D100K* are shown in Figs. 14 and 16, which show the linear increase of runtime with the number of transactions.

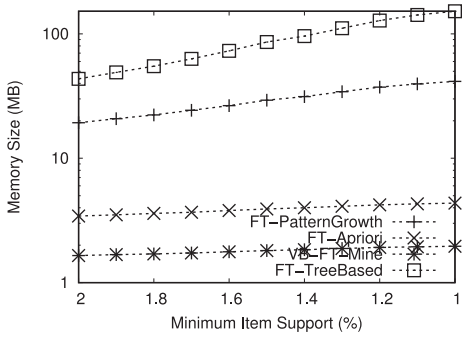
From the figure, one can see that performance of FT-PatternGrowth is scalable even when the number of transactions are increased. To deal with large number of transactions FT-Apriori has to generate many candidates itemsets,



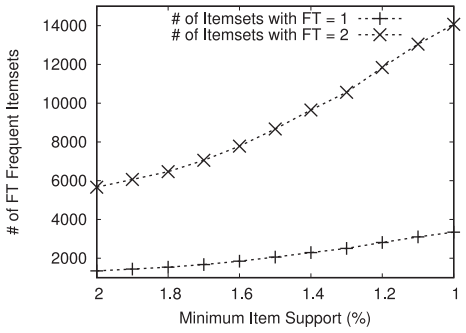
(a) **T10I4D100K with $\delta = 1$ and $min\_sup^{\delta} = 2\%$**



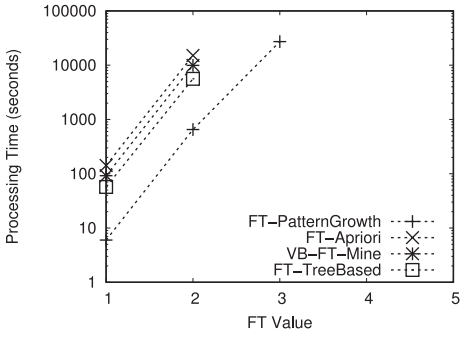
(b) **T10I4D100K with $\delta = 2$ and $min\_sup^{\delta} = 2\%$**



(c) **Memory size with $\delta = 1$ and $min\_sup^{\delta} = 2\%$**



(d) **Number of FT frequent itemsets**



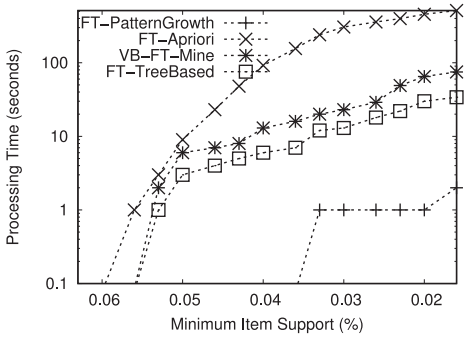
(e) **Scalability of algorithms on varying δ values with $min\_sup^{\delta} = 2\%$ and $item\_sup^{\delta} = 1.8\%$**

Fig. 11. The performance of FT frequent itemset mining algorithms on *T10I4D100K* database. (d) Number of FT frequent itemsets discovered with $\delta = 1$, $\delta = 2$ and $min\_sup^{\delta} = 2\%$.

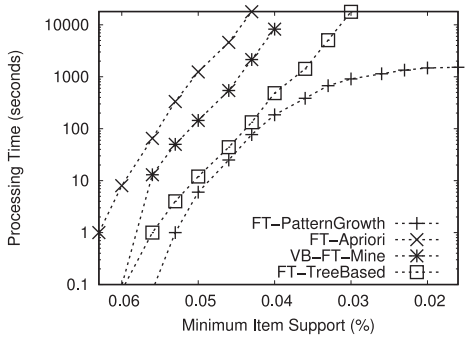
even those that do not exist in the database. We also found that large amount of processing time of `FT-Apriori` is spend on counting itemset support. This is because `FT-Apriroi` does not provide any functionality to compress the transactions which share a common set of items. This number of candidates itemsets becomes tremendous large when the database contains large number of frequent items. In contrast, the `FT-PatternGrowth` is efficient because it generates only those candidate itemsets which exist in the branches of FP-trees. This mines all FT frequent itemsets in less processing time as it substantially eliminates those candidate itemsets that do not exist in the transactions. `FT-PatternGrowth` provides much better speed for counting support of itemsets as it compresses transactions in the similar branches of FP-tree if they share a common set of items. This explains why `FT-PatternGrowth` is more efficient than `FT-Apriori` when the support threshold is low and when the number of transactions is large.

To analyze the performance of `FT-PatternGrowth` on transaction length, we partitioned the *Retail* and *T10I4D100K* databases into five groups. For each group we construct database by including random 30,000 transactions. For first group, we construct the database by including transactions that have transaction length between 1 to 10. For second group, we construct database by including transactions that have transaction length between 11 to 20. For third group, we construct database by including transactions that have transaction length between 21 to 30. For fourth group, we construct database by including transactions that have transaction length between 31 to 40, and last group contains all transactions that have transaction length between 41 to 50.

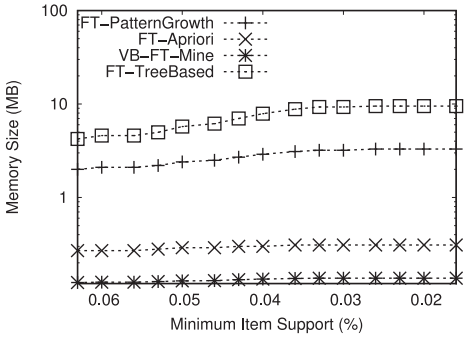
Figs. 13 and 15 show the processing time of all algorithms on the *Retail* and *T10I4D100K* databases. Results show `FT-PatternGrowth` is more efficient than `FT-Apriori`, `VB-FT-Mine`, and `FT-TreeBased` on varying transaction length. From the experiments we can observe that when the transaction length increases all algorithms take more processing time to mine complete FT frequent itemsets due to large number of frequent items. Also, from the experiments we can observe the `FT-PatternGrowth` finishes its execution in less processing time than other algorithms. `FT-TreeBased` is efficient than `VB-FT-Mine` and `FT-Apriori`.



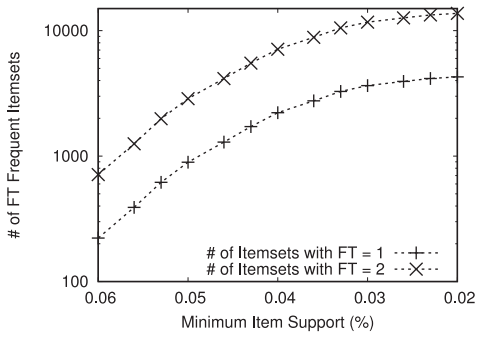
(a) **FoodMart with** $\delta = 1$ **and** $min\_sup^{\delta} = 0.06\%$



(b) **FoodMart with** $\delta = 1$ **and** $min\_sup^{\delta} = 0.06\%$

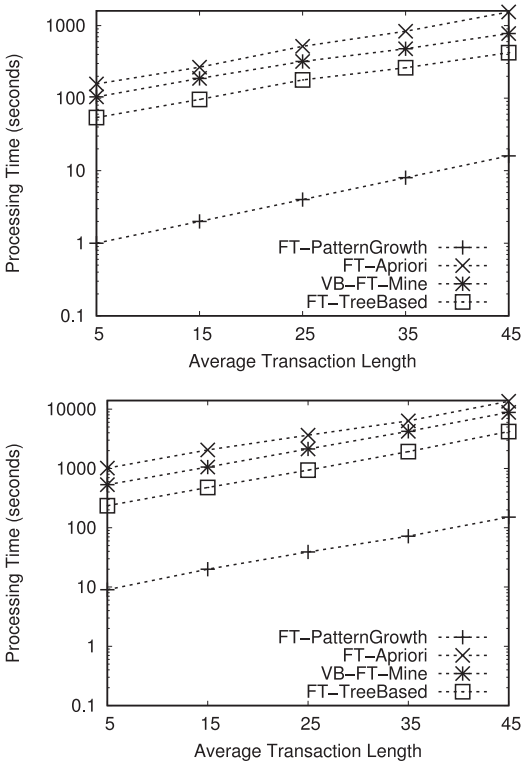


(c) **Memory size with** $\delta = 1$ **and** $min\_sup^{\delta} = 0.06\%$



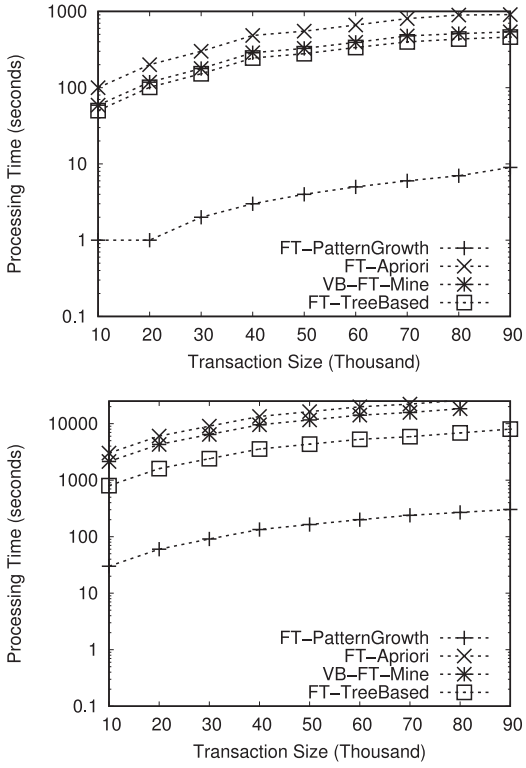
(d) **Number of FT frequent itemsets**

Fig. 12. The performance of FT frequent itemset mining algorithms on *FoodMart* database. (d) Number of FT frequent itemsets discovered with $\delta = 1$, $\delta = 2$ and $min\_sup^{\delta} = 0.06\%$.



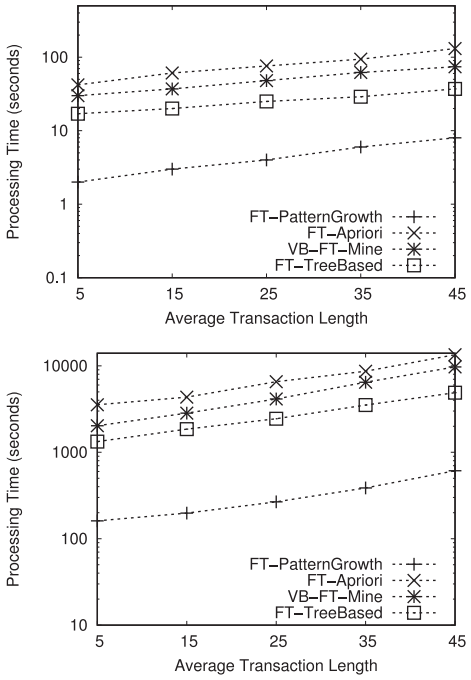
(a) **Retail with** $\delta = 1$, $min\_sup^{\delta} = 1\%$ **and** $item\_sup^{\delta} = 0.8\%$
(b) **Retail with** $\delta = 2$, $min\_sup^{\delta} = 1\%$ **and** $item\_sup^{\delta} = 0.8\%$

Fig. 13. Scalability of FT frequent itemset mining algorithms on various transaction length for *Retail* database.



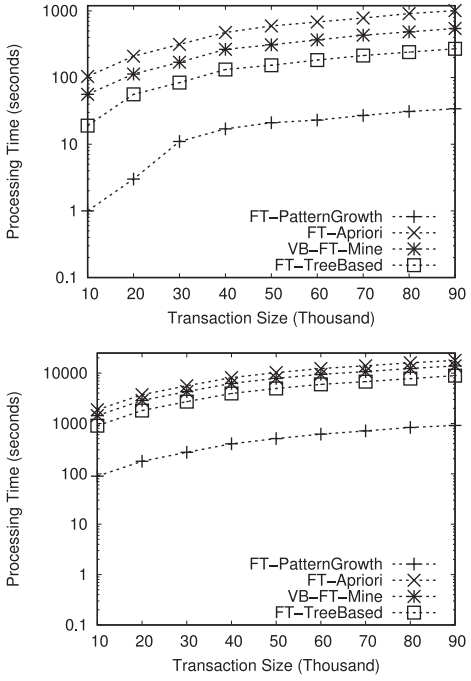
(a) **Retail with** $\delta = 1$, $min\_sup^{\delta} = 1\%$ **and** $item\_sup^{\delta} = 0.8\%$
(b) **Retail with** $\delta = 2$, $min\_sup^{\delta} = 1\%$ **and** $item\_sup^{\delta} = 0.8\%$

Fig. 14. Scalability of FT frequent itemset mining algorithms on various transaction size for *Retail* database.



(a) **T10I4D100K with** $\delta = 1$, $min\_sup^{\delta} = 2\%$ **and** $item\_sup^{\delta} = 1.8\%$
(b) **T10I4D100K with** $\delta = 2$, $min\_sup^{\delta} = 2\%$ **and** $item\_sup^{\delta} = 1.8\%$

Fig. 15. Scalability of FT frequent itemset mining algorithms on various transaction length for *T10I4D100K* database.



(a) **T10I4D100K with** $\delta = 1$, $min\_sup^{\delta} = 2\%$ **and** $item\_sup^{\delta} = 1.8\%$
(b) **T10I4D100K with** $\delta = 2$, $min\_sup^{\delta} = 2\%$ **and** $item\_sup^{\delta} = 1.8\%$

Fig. 16. Scalability of FT frequent itemset mining algorithms on various transaction size for *T10I4D100K* database.

6. Conclusion

Mining fault tolerant frequent itemsets from transactional databases are computationally more expensive than mining traditional exact matching frequent itemsets. Previous algorithms on mining FT frequent itemsets are based on Apriori-like candidate generation-and-test approach. These algorithms generate many candidate itemsets including those that do not exist in the database and require multiple scans of database for counting the support of each candidate itemsets. In this paper we present a novel algorithm for mining FT frequent itemsets using pattern growth approach (FT-PatternGrowth). FT-PatternGrowth adopts a divide-and-conquer technique and recursively projects a transactional database into a set of smaller projected transactional databases and mines FT frequent itemsets in each projected database by exploring only locally frequent items. This mines the complete set of FT frequent itemsets in less processing time and substantially reduces those candidate itemsets that do not exist in the database. FT-PatternGrowth also stores the transactional database in a highly condensed much smaller data structure called FT-FP-tree. The support of candidate itemsets and the support of items are calculated directly from the FT-FP-tree without scanning the database multiple times. This reduces the processing time of algorithm for counting support of itemsets. Our experiments on benchmark databases indicate mining FT frequent itemsets using pattern growth approach is highly efficient than Apriori-like algorithms.

Declaration of Competing Interes

None.

References

Apiletti, D., Baralis, E., Cerquitelli, T., Garza, P., Pulvirenti, F., & Venturini, L. (2017). Frequent itemsets mining for big data: A comparative analysis. *Big Data Research, 9*, 67–83. doi:10.1016/j.bdr.2017.06.006.

Ashraf, S. M. A., & Tabrez Nafis, d. (2017). Fault tolerant frequent patterns mining in large datasets having certain and uncertain records. *Advances in Computational Sciences and Technology, 10*.

Bashir, S., Halim, Z., & Baig, A. R. (2008). Mining fault tolerant frequent patterns using pattern growth approach. In *6th ACS/IEEE international conference on computer systems and applications, AICCSA 2008, Doha, Qatar, march 31, - april 4, 2008* (pp. 172–179). doi:10.1109/AICCSA.2008.4493532.

Becquet, C., Blachon, S., Jeudy, B., Boulicaut, J.-F., & Gandrillon, O. (2001). Strong association rule mining for large-scale gene-expression data analysis: a sase study on human sage data. *Genome Biology, 3*.

Besson, J., Pensa, R. G., Robardet, C., & Boulicaut, J. (2005). Constraint-based mining of fault-tolerant patterns from boolean data. In *Knowledge discovery in inductive databases, 4th international workshop, KDID 2005, Porto, Portugal, october 3, 2005, revised selected and invited papers* (pp. 55–71). doi:10.1007/11733492_4.

Bodon, F. (2003). A fast APRIORI implementation. *FIMI '03, frequent itemset mining implementations, proceedings of the ICDM 2003 workshop on frequent itemset mining implementations, 19 december 2003, Melbourne, Florida, USA*.

Burdick, D., Calimlim, M., Flannick, J., Gehrke, J., & Yiu, T. (2005). MAFIA: A maximal frequent itemset algorithm. *IEEE Transactions on Knowledge and Data Engineering, 17*(11), 1490–1504. doi:10.1109/TKDE.2005.183.

Cheung, Y.-L., & Fu, A. W.-C. (2004). Mining frequent itemsets without support threshold: With and without item constraints. *IEEE Transactions on Knowledge and Data Engineering, 16*(9), 1052–1069. doi:10.1109/TKDE.2004.44.

Creighton, C., & Hanash, S. (2003). Mining gene expression databases for association rules. *Bioinformatics (Oxford, England), 19*, 79–86.

Cremaschi, P., Carriero, R., Astrologo, S., Col, C., Lisa, A., Parolo, S., & Bione, S. (2015). An association rule mining approach to discover lncrnas expression patterns in cancer datasets. *BioMed Research International, 2015*.

Diwakar Tripathia, B. N., & Edlaa, D. R. (2017). A novel web fraud detection technique using association rule mining. *Procedia Computer Science, 51*.

Gan, W., Lin, J. C., Fournier-Viger, P., Chao, H., & Zhan, J. (2017). Mining of frequent patterns with multiple minimum supports. *Engineering Applications of Artificial Intelligence, 60*, 83–96. doi:10.1016/j.engappai.2017.01.009.

Han, J., Cheng, H., Xin, D., & Yan, X. (2007). Frequent pattern mining: Current status and future directions. *Data Mining and Knowledge Discovery, 15*, 55–86.

Han, J., & Pei, J. (2014). Pattern-growth methods. In *Frequent pattern mining* (pp. 65–81). doi:10.1007/978-3-319-07821-2_3.

Han, J., Pei, J., & Yin, Y. (2000). Mining frequent patterns without candidate generation. In *Proceedings of the 2000 ACM SIGMOD international conference on man-*

agement of data, may 16–18, 2000, Dallas, Texas, USA. (pp. 1–12). doi:10.1145/342009.335372.

Han, J., Pei, J., Yin, Y., & Mao, R. (2004). Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Mining and Knowledge Discovery, 8*(1), 53–87. doi:10.1023/B:DAMI.0000005258.31418.83.

Huynh-Thi-Le, Q., Le, T., Vo, B., & Le, B. (2015). An efficient and effective algorithm for mining top-rank-k frequent patterns. *Expert Systems with Applications, 42*(1), 156–164. doi:10.1016/j.eswa.2014.07.045.

Jiang, F., Leung, C. K., & Zhang, H. (2016). B-mine: Frequent pattern mining and its application to knowledge discovery from social networks. In *Web technologies and applications - 18th asia-pacific web conference, apweb 2016, Suzhou, China, september 23–25, 2016. proceedings, part I* (pp. 316–328). doi:10.1007/978-3-319-45814-4_26.

(2008). Next Generation of Data Mining. In H. Kargupta, J. Han, P. S. Yu, R. Motwani, & V. Kumar (Eds.). *Data Mining and Knowledge Discovery Series*. CRC Press / Chapman and Hall / Taylor & Francis. doi:10.1201/9781420085877.

Koh, J., & Yo, P. (2005). An efficient approach for mining fault-tolerant frequent patterns based on bit vector representations. In *Database systems for advanced applications, 10th international conference, DASFAA 2005, Beijing, China, april 17–20, 2005, proceedings* (pp. 568–575). doi:10.1007/11408079_51.

Koh, J.-L., & Lang, t. (2010). A tree-based approach for efficiently mining approximate frequent itemsets. In *Fourth international conference on research challenges in information science (RCIS)* (pp. 25–36).

Kosters, W. A., & Pijls, W. (2003). Apriori, A depth first implementation. *FIMI '03, frequent itemset mining implementations, proceedings of the ICDM 2003 workshop on frequent itemset mining implementations, 19 december 2003, Melbourne, Florida, USA.*

Lee, G., & Lin, Y.-T. (2006). A study on proportional fault-tolerant data mining. *In proc. 2006 int. conf. innovations in information technology, Dubai.*

Lee, G., Peng, S.-L., & Lin, Y.-T. (2009). Proportional fault-tolerant data mining with applications to bioinformatics. *Information Systems Frontiers, 11*(4), 461–469. doi:10.1007/s10796-009-9158-z.

Li, R., & Wang, W. (2015). REAFUM: Representative approximate frequent subgraph mining. In *Proceedings of the 2015 SIAM international conference on data mining, vancouver, bc, canada, april 30, - may 2, 2015* (pp. 757–765). doi:10.1137/1.9781611974010.85.

Liu, G., Lu, H., Yu, J. X., Wang, W., & Xiao, X. (2003). AFOPT: An efficient implementation of pattern growth approach. *FIMI '03, frequent itemset mining implementations, proceedings of the ICDM 2003 workshop on frequent itemset mining implementations, 19 december 2003, Melbourne, Florida, USA.*

Liu, S., & Poon, C. K. (2014). On mining proportional fault-tolerant frequent itemsets. In *19th international conference on database systems for advanced applications, bali, indonesia, april 21–24, 2014* (pp. 342–356). doi:10.1007/978-3-319-05810-8_23.

Liu, S., & Poon, C. K. (2018). On mining approximate and exact fault-tolerant frequent itemsets. *Knowledge and Information Systems, 55*(2), 361–391. doi:10.1007/s10115-017-1079-4.

Mallik, S., Mukhopadhyay, A., & Maulik, U. (2015). Ranwar: Rank-based weighted association rule mining from gene expression and methylation data. *IEEE Transactions on NanoBioscience, 14*(1).

Moosavi, S. A., Jalali, M., Misaghian, N., Shamshirband, S., & Anisi, M. H. (2017). Community detection in social networks using user frequent pattern mining. *Knowledge and Information Systems, 51*(1), 159–186. doi:10.1007/s10115-016-0970-8.

Morales-González, A., Acosta-Mendoza, N., Alonso, A. G., Reyes, E. B. G., & Medina-Pagola, J. E. (2014). A new proposal for graph-based image classification using frequent approximate subgraphs. *Pattern Recognition, 47*(1), 169–177. doi:10.1016/j.patcog.2013.07.004.

Pei, J., Tung, A. K. H., & Han, J. (2001). Fault-tolerant frequent pattern mining: Problems and challenges. *ACM SIGMOD workshop on research issues in data mining and knowledge discovery, Santa Barbara, CA, USA, may 20, 2001.*

Iváncsy, Renáta, & Vajk, I. (2006). Frequent pattern mining in web log data. *Acta Polytechnica Hungarica, 3.*

Saif-Ur-Rehman, Ashraf, J., Habib, A., & Salam, A. (2016). Top-k miner: Top-k identical frequent itemsets discovery without user support threshold. *Knowledge and Information Systems, 48*(3), 741–762. doi:10.1007/s10115-015-0907-7.

Uno, T., Kiyomi, M., & Arimura, H. (2004). LCM ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets. *FIMI '04, proceedings of the IEEE ICDM workshop on frequent itemset mining implementations, brighton, uk, november 1, 2004.*

Vo, B., Pham, S., Le, T., & Deng, Z. (2017). A novel approach for mining maximal frequent patterns. *Expert System with Applications, 73*, 178–186. doi:10.1016/j.eswa.2016.12.023.

Yu, X., & Korkmaz, T. (2015). Heavy path based super-sequence frequent pattern mining on web log dataset. *Artificial Intelligence Research, 4*(2), 1–12. doi:10.5430/air.v4n2p1.

Yu, X., Li, Y., & Wang, H. (2015). Mining approximate frequent patterns from noisy databases. *10th international conference on broadband and wireless computing, communication and applications (BWCCA).*