ITC 4/50
Information Technology and Control
Vol. 50 / No. 4 / 2021
pp. 627-644
DOI 10.5755/j01.itc.50.4.29060

Mining Approximate Frequent Itemsets Using Pattern Growth Approach

Received 2021/05/06 | Accepted after revision 2021/10/25

http://dx.doi.org/10.5755/j01.itc.50.4.29060

HOW TO CITE: Bashir, S., Lai, D. T. C. (2021). Mining Approximate Frequent Itemsets Using Pattern Growth Approach. *Information Technology and Control*, 50(4), 627-644. https://doi.org/10.5755/j01.itc.50.4.29060

Mining Approximate Frequent Itemsets Using Pattern Growth Approach

Shariq Bashir
Institute of Applied Data Analytics (IADA), Universiti Brunei Darussalam (UBD), shariq.bashir@ubd.edu.bn

Daphne Teck Ching Lai
School of Digital Sciences (SDS), Universiti Brunei Darussalam (UBD), daphne.lai@ubd.edu.bn

Corresponding author: shariq.bashir@ubd.edu.bn

Approximate frequent itemsets (AFI) mining from noisy databases are computationally more expensive than traditional frequent itemset mining. This is because the AFI mining algorithms generate large number of candidate itemsets. This article proposes an algorithm to mine AFIs using pattern growth approach. The major contribution of the proposed approach is it mines core patterns and examines approximate conditions of candidate AFIs directly with single phase and two full scans of database. Related algorithms apply Apriori-based candidate generation and test approach and require multiple phases to obtain complete AFIs. First phase generates core patterns, and second phase examines approximate conditions of core patterns. Specifically, the article proposes novel techniques that how to map transactions on approximate FP-tree, and how to mine AFIs from the conditional patterns of approximate FP-tree. The approximate FP-tree maps transactions on shared branches when the transactions share a similar set of items. This reduces the size of databases and helps to efficiently compute the approximate conditions of candidate itemsets. We compare the performance of our algorithm with the state of the art AFI mining algorithms on benchmark databases. The experiments are analyzed by comparing the processing time of algorithms and scalability of algorithms on varying database size and transaction length. The results show pattern growth approach mines AFIs in less processing time than related Apriori-based algorithms.

KEYWORDS: Approximate Frequent Itemset Mining, Frequent Itemset Mining, Pattern Growth, Association Rules Mining.

1. Introduction

Mining frequent itemsets from databases is an important data mining task. It has many practical applications including document clustering [15, 40], social network analysis [23, 34], market basked analysis [17], fraud detection [14], bioinformatics [13, 28, 33], mining patterns from web logs [22, 38]. The concept of mining frequent itemsets and generating association rules form the frequent itemsets was proposed by [1]. In the last 20 years there have been lot of research in developing different techniques to efficiently mine frequent itemsets from transactional databases. (Han et al., 2000) proposed pattern-growth approach for mining frequent itemsets from FP-tree. Kosters et al., [26] applied depth-first search for Apriori. Bodon et al., [6] and Liu et al., [29] proposed fast implementation techniques for Apriori and pattern-growth. Uno et al., [36], Vo et al., [37], and Chen et al., [9] proposed algorithms to mine different variations of frequent itemsets such as maximal, probabilistic maximal and closed. Burdick et al., [7] proposed bit-vector technique for mapping frequent items on bit-vectors. Gan et al., [16] proposed an algorithm to mine frequent itemsets from multiple minimum supports. Chen et al., [8] proposed an algorithm to efficiently mine frequent itemsets from small-scale datasets.

The main limitation of traditional frequent itemset mining (FIM) concept is that it can be only use to discover itemsets that are absolutely matched in the database. It cannot mine useful itemsets that are noisy or have missing items due to real world data distributions [39]. From noisy databases it is difficult to provide support thresholds for mining expected set of frequent itemsets. For instance, mining frequent itemsets with high support discovers only small set of short length patterns, and small support discovers exponential set of two and three length itemsets [12, 21, 35].

For mining useful itemsets from noisy databases, Liu et al., [30] proposed the concept of mining approximate frequent itemsets (AFI). The following properties describe the concept of mining AFIs from the transactional databases [10-11].

- An itemset X is an AFI under error-tolerance percentage of row and column ($\varepsilon^r > 0\%$) and ($\varepsilon^c > 0\%$), if X appears in at least T number of approximate and $\alpha(T)$ number of absolute transactions.
- A transaction is an approximate transaction if it contains at least ε^r percent of items of X.
- X is a core pattern if the absolute support of X is equal or greater than minimum absolute support (*abs_sup*).
- Every single item of X should has support of at least ε^c percent of approximate transactions of X.

The following example describes the difference between absolute and approximate matching itemsets. Table 1 provides a transactional database with *nine transactions* containing *six items*. To discover frequent itemsets with *min_sup* = 3, the absolute matching algorithm cannot discover any itemset with length greater than two items. The algorithm mines many itemsets with length less than three items. These short length itemsets cannot be used for discovering generalized knowledge from the databases. However, if the user tries to mine itemsets using AFI concept by slightly relaxing the notion of traditional concept of FIM. The AFI mining algorithm can discover long length itemsets with high support. Even though, these itemsets are not exactly matched in the transactions, but contain high support. For example, the itemset (*efcb*) is an AFI of length four and has support 3. The transactions 10, 20, and 50 contain three out of four items of *efcb* and every single item of (*efcb*) is appeared in at least two transactions (10, 20, and 50). This approximate match mining concept is appealing in this way that it discovers long length frequent itemsets. This strategy motivates researchers to develop algorithms for mining complete AFIs [10-11, 30].

Given the AFI mining properties (presented above), if we apply the properties on the dataset of Table 1. Suppose the support threshold are (*apx_sup*) = 3, (*abs_sup*) = 1. The row and column error-tolerance percentages are (ε^r = 75%) and (ε^c = 65%). The itemset X = (*efcb*) is a core pattern since the absolute support of *efcb* is one in transaction 10. The itemset (*efcb*) is also an AFI as it is three out of four items are appeared in the approximate transactions 10, 20 and 50. This qualifies ε^r = 75%. Also, each single item *e*, *f*, *c* and *b* is appeared in at least two transactions. This qualifies ε^c = 65% threshold.

Previous state of the art AFI mining algorithms mine AFIs with two phases. In first phase, the complete set of core patterns are discovered by applying Apriori-based candidate generation and test approach.

Table 1
A sample transactional database

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

Once core patterns are available, the algorithms examine the AFI properties of core patterns for counting items and itemsets support with multiple full scans of database. Mining core patterns using Apriori-based approach are not performance efficient when the databases are dense or spare. In the following paragraph, we provide main limitations of generating core patterns using Apriori-based approach.

- *Apriori-based* algorithm mines complete set of core patterns using candidate generation and test approach. The main limitation of this approach is that if the size of database is large then this approach generates exponential combinations of candidate itemsets. For example, if the database contains 300 frequent items, the *Apriori-based* approach generates and test all 2^{300} candidate itemsets.
- Apriori-based algorithm generates candidate itemsets by applying bottom-up search space exploration on frequent items. This means the algorithm exponentially generates and tests all the 2^X subsets of an itemset X before producing X. This complexity is not suitable for the databases that have large number of frequent items.
- To examine AFI conditions of core patterns, the Apriori-based algorithm scans the original database multiple times for calculating supports of itemsets and items. These scans consume large processing time when the number of core patterns to mine are exponential and size of database is large.

This article proposes a new approach to mine complete set of AFIs using pattern growth approach (Apx-PatternGrowth). The pattern growth is a divide and conquer technique. It recursively divides the big database into small subsets and mines complete set of AFIs from the smaller subsets by generating candidate itemsets that exist only in the subsets. This prunes the combinations of itemsets that are not available in the transactions [18-20]. The major advantage of proposed approach is that it mines the core patterns and examines the approximate conditions of core patterns directly with one phase and two scans of database. Related AFI mining algorithms require two phases for obtaining AFIs. First phase generates core patterns, and second phase examines approximate conditions of core patterns. In the first scan of database, the Apx-PatternGrowth mines all one length frequent items and prunes infrequent items. In the second scan, the Apx-PatternGrowth maps all transactions of database on frequent pattern tree (FP-tree). The frequent items of database are mapped on the nodes of FP-tree, and transactions are mapped on the branches of FP-tree. If multiple transactions share common prefix, the shared items are mapped on a single branch and the support (frequency) of shared subset is mapped on the nodes of FP-tree. The pattern growth explores the AFIs using depth first search order. All subsets of an itemset are obtained by generating conditional patterns from the branches of FP-tree. The frequent items in the conditional patterns generate recursive child FP-trees. The child FP-trees generate AFIs of next level. We perform experiments on benchmarks datasets and compare the performance of our algorithm with the related algorithms. Our experiments show the Apx-PatternGrowth mines complete AFIs in less processing time than related algorithms.

The rest of article is organized as follows. In Section 2 we provide a detailed review of related algorithms on mining AFIs and explain how our approach is different to related algorithms. Section 3 provides formal definition of mining AFIs and presents how to map approximate transactions on FP-tree and how to apply pattern growth approach for mining complete set of AFIs. Section 4 presents the details of benchmark datasets and compares the performance of AFI mining algorithms. In Section 5 we present key findings of the proposed algorithm.

2. Related Work

Related state of the art algorithms mine complete set of AFIs using two phases. In first phase, the algorithms apply Apriori-based candidate generation and test approach for generating complete set of core patterns. In second phase, the approximate conditions of core patterns are examined for generating AFIs. The Apriori-based algorithms prune infrequent search space by applying anti-monotone heuristic of Apriori on the core patterns [10-11]. The main limitations of Apriori-based algorithms are: These algorithms generate large number of candidate itemsets that do not exist in the database, and require multiple scans of database for examining the approximate conditions of itemsets.

Cheng et al., [10-11] proposed *AC-Close* algorithm. *AC-Close* applies top down complete search space exploration for building lattice of core patterns. Intuitively, an itemset X is a core pattern if the absolute support of $\alpha(X)$ in the noisy database is not less than minimum *abs_sup*. *AC-Close* then mines complete AFIs from the lattice of core patterns by starting with the largest pattern and proceeds level by level in the size decreasing order of core patterns. *AC-Close* discards infrequent approximate itemsets by applying anti-monotone heuristic of Apriori on the infrequent itemsets. The anti-monotone heuristic of Apriori does not generate supersets of an itemset when the itemset is found infrequent. *AC-Close* is not performance efficient for the databases having large number of transactions as it requires multiple full scans of database for examining approximate conditions of AFIs. The other drawback of *AC-Close* is it generates candidate itemsets by applying bottom-up search space exploration. This means the algorithm exponentially generates and tests all the 2^X subsets of itemset X before producing X. This generates many candidate itemsets that actually do not exist in the database. This complexity is not suitable for the databases when the number of itemsets to be mined are exponential.

Bashir et al., [3-5], and Koh et al., [25] proposed algorithms for mining fault-tolerant frequent itemsets. The concept of fault-tolerant FIM is similar to AFIs, however, fault-tolerant FIM keeps the row error-tolerance of itemsets fixed regardless of the length of itemset. Thus, fault-tolerant FIM discovers different set of approximate frequent itemsets. Our proposed algorithm discoverers AFIs using the core patterns concept as explained in [10-11]. Thus, the processing time of our algorithm cannot be directly comparable with the fault-tolerant FIM algorithms.

To avoid costly repeatedly scanning of database, Koh et al., [25] proposed a tree based approach for mining AFIs. At each iteration, the proposed algorithm constructs multiple FP-trees for mining AFIs. For example, to mine all supersets containing itemset (ab) under row error-tolerance of $\varepsilon^r = 50\%$, the algorithm constructs four FP-trees. The first FP-tree maps only those transactions of database which have both item a and item b. The second FP-tree maps all those transactions, which have item a but not item b. The third FP-tree maps transactions which have only item b. The last FP-tree maps all those transactions, which have missing both items. The main limitation of proposed algorithm is it maps the transactions on multiple FP-trees even if the transactions share similar set of items. Thus, the algorithms cannot gain actual performance of pattern growth during support counting of items and itemsets. The algorithm also consumes large memory and difficult to fit in the memory during AFI mining. Our proposed pattern growth algorithm does not construct multiple FP-trees. The proposed algorithm maps all transactions of a database on a single FP-tree even if the transactions have different percentage of row error-tolerance. The proposed algorithm is scalable on large databases and consumes less memory than the Koh et al., [25] approach.

To mine interesting AFIs in a reasonable processing time, researchers have proposed alternative heuristics (such as proportional [27, 31] and high utility [2] AFIs mining). Although, these heuristics mine interesting AFIs in less processing time, however, provide no guarantee on the completeness of the search as only imprecise mining results are obtained. Our proposed algorithm mines AFIs using the concept of core patterns [10-11] by exploring complete search space of candidate itemsets. Thus, mines complete AFIs. As the search space requirements of proportional and high utility heuristics are different to the proposed algorithm, therefore, it is not suitable to directly compare the performance of our algorithm with the AFI mining heuristics.

Lee et al., [28] applied proportional AFIs for mining patterns from bioinformatics. Liu et al., [31-32] proposed heuristics for mining proportional AFIs. In experiments they showed the heuristics quickly mine itemsets within the acceptable error than the exact matching algorithm. All studies on the proportional AFIs are Apriori-based algorithms. However, no effort has been investigated how to utilize the concept of pattern growth to mine itemsets, and how to reduce processing of itemsets support counting. In this work we investigate how to map approximate transactions on the FP-tree and how to mine complete set of AFIs by recursively generating conditional patterns from the FP-tree.

3. Mining Approximate Frequent Itemsets Using Pattern Growth: Design and Construction

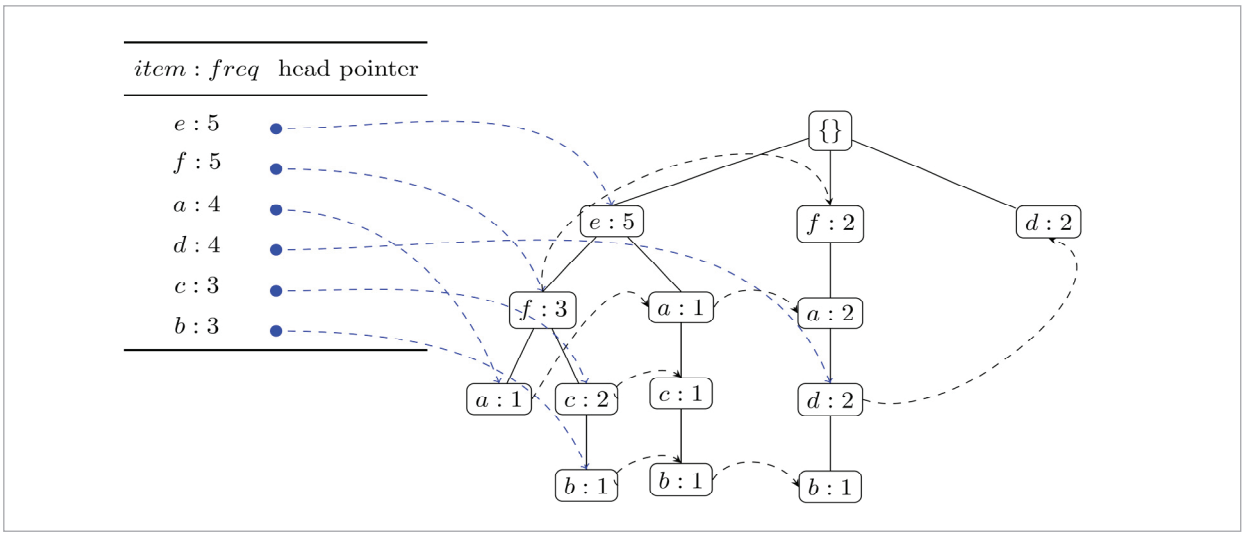
Given user defined row and column error-tolerance percentages (($\varepsilon^r > 0\%$) and ($\varepsilon^c > 0\%$)), an itemset X is an AFI if it appears in at least T number of approximate transactions and $\alpha(T)$ number of absolute transactions, and satisfies the following two conditions.

- A transaction t is an approximate transaction of T under ($\varepsilon^r > 0\%$) if it contains at least ε^r percent of items of X.
- T is the support of X, which must not be less than minimum approximate itemset support (*apx_sup*). Each individual item of X must has support of at least ε^c percent of approximate transactions of X.

Given the AFI mining conditions explained above if we look again at the database of Table 1. Suppose the ($\varepsilon^r = 75\%$ and the ($\varepsilon^c = 65\%$). The itemset $X = (efcb)$ is a frequent AFI since its 75% of items are available in transactions 10, 20 and 50. This qualifies ($\varepsilon^r = 75\%$), and each single item e, f, c and b is available in at least 65% transactions with qualifies ($\varepsilon^c = 65\%$) threshold.

The proposed algorithm mines itemsets using pattern growth approach. The pattern growth requires FP-tree to generate itemsets. FP-Tree is a tree-like data structure. It maps complete transactions of a database on the branches of tree [18-20]. Nodes of tree map items of transactions and branches map transactions of database. The transactions that share a common subset of items are mapped on the shared branches and the frequency of shared subset is mapped on the nodes of FP-tree. The pattern growth is then applied on the FP-tree for mining complete AFIs. The supersets of an itemset are obtained by generating conditional patterns from the branches of FP-tree. The conditional patterns generate recursive FP-trees. The recursive FP-trees discover candidate AFIs of next level. For generating conditional patterns of items, all nodes of items are linked together by making a linked

Figure 1
FP-tree after mapping all transactions of database



list, and a header table is constructed for storing head pointers of items. The head pointers facilitate tree traversal. One main advantage of FP-tree is it generates only those candidate itemsets that exist in the candidate patterns. Thus, it prunes the candidate itemsets that do not exist in the database. Furthermore, the approximate conditions of itemsets such as itemsets support and item supports are computed directly from the conditional patterns. This improves the scalability of algorithm on big databases.

Example: Table 1 shows a transactional database. Let the error-tolerance percentages of row and column are ($\varepsilon^r = 50\%$) and ($\varepsilon^c = 50\%$). Let the approximate and absolute itemset supports are ($apx\_sup = 3$) and ($abs\_sup = 1$).

The algorithm scans the database and removes infrequent items from the transactions that have support less than *abs sup*. The list of frequent items with their support is $\langle (e:5),(f:5),(a:4),(d:4),(c:3),(b:3) \rangle$. Items of the transactions are reordered by following the decreasing frequency order of items. The algorithm again scans the database and constructs initial FP-tree. The transactions are mapped one by one on the branches of FP-tree. If multiple transactions share a common prefix, the shared prefix is mapped only one time on the FP-tree. Figure 1 shows the FP-tree of database (Table 1).

3.1. Constructing Approximate (FP-tree)

The algorithm mines the complete set of AFIs from the Apx-FP-tree (Approximate frequent itemset tree). The Apx-FP-tree is similar to FP-tree. The only difference between Apx-FP-tree and traditional FP-tree is that Apx-FP-tree maps approximate conditional (Apx-conditional) patterns on the tree. This helps in pruning infrequent AFIs that are not available in the database.

A conditional pattern is called an Apx-conditional pattern if it contains ε^r items of an itemset X. The Apx-conditional pattern has four components. The first component stores list of items that can be used for generating supersets of X. The first component is mapped on the branch of Apx-FP-tree. The second component stores support of pattern. Third component stores row error-tolerance that the pattern contains how many items of X. Fourth component stores frequencies of all items of X. The fourth component stores error-tolerance of items. The Apx-FP-tree contains Apx-conditional pattern tables (*ApxCP-Table*) on the leaves to map components of Apx-conditional patterns. Each Apx-CP-Table contains three columns. Components second, third, and fourth of Apx-conditional patterns are mapped on the first, second and third column of Apx-CP-Tables.

To mine complete AFIs, the algorithm generates Apx-FP-Tree for all possible combinations of two length itemsets by including only frequent items. Then, the Apx-conditional patterns obtained from the Apx-FP-trees of two length itemsets are used for generating supersets of two length itemsets. The approximate itemset support and error-tolerance percentages of itemsets are counted directly from the Apx-conditional patterns of Apx-FP-trees.

Example: For example, to construct Apx-FP-tree of itemset $X = (cb)$ with error-tolerance percentage of row ($\varepsilon^r = 50\%$) and column ($\varepsilon^c = 50\%$). The algorithm generates conditional patterns of items b and c.

- Item b contains three conditional patterns: $\langle efcb : 1\rangle$, $\langle eacb : 1\rangle$, and $\langle fadb : 1\rangle$. The algorithm converts these conditional patterns into Apx-conditional patterns. The patterns $\langle efcb : 1\rangle$ and $\langle eacb : 1\rangle$ are Apx-conditional patterns with row error-tolerance of 100% because both items c and b are present in the patterns. The pattern $\langle fadb : 1\rangle$ is an Apx-conditional pattern with row error-tolerance of 50% because the pattern does not contain item c.
- The pattern $\langle\langle ef\rangle, \langle sup : 1\rangle, \langle \varepsilon^r = 100\%\rangle, \langle c : 1, b : 1\rangle\rangle$ is the Apx-conditional pattern of $\langle efcb : 1\rangle$. It has four components. The first component stores list of items that can be used for generating supersets of itemset (cb). The second component stores support of pattern. The third component contains $\langle \varepsilon^r = 100\%\rangle$ which indicates all items of (cb) are present. The fourth component stores support of all items of (cb). The pattern $\langle eacb : 1\rangle$ is converted into Apx-conditional pattern $\langle\langle ea\rangle, \langle sup : 1\rangle, \langle \varepsilon^r = 100\%\rangle, \langle c : 1, b : 1\rangle\rangle$. The pattern $\langle fadb : 1\rangle$ is converted into Apx-conditional pattern $\langle\langle fad\rangle, \langle \varepsilon^r = 50\%\rangle, \langle c : 0, b : 1\rangle\rangle$. Figure 2 shows the Apx-FP-tree of item cb.
- The initial FP-tree is again scanned for generating conditional patterns of item c. The following two conditional patterns are generated from the FP-tree: $\langle efc : 2\rangle$ and $\langle eac : 1\rangle$. If the conditional pattern (c_X) of any next item is a subset of former item, then the support of c_Y is subtracted from the support of c_X. If the support of c_X becomes zero, then the conditional pattern c_X is ignored. Following this

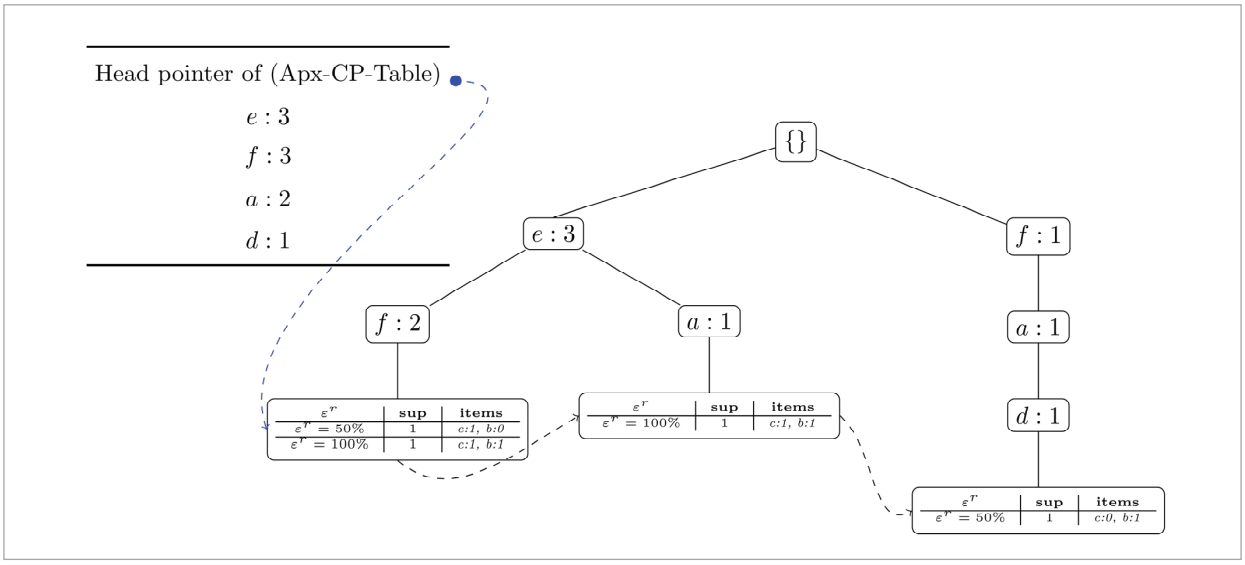
Table 2
Conditional patterns obtained from the FP-tree and Apx-conditional patterns of itemset (*cb*)

Item	Conditional Patterns	Apx-Conditional Patterns
b	*efcb* : 1	(⟨⟨*ef*⟩, ⟨*sup*:1⟩, ⟨ε^r=100%⟩, ⟨*c*:1,b:1⟩⟩)
b	*eacb* : 1	(⟨⟨*ea*⟩, ⟨*sup*:1⟩, ⟨ε^r=100%⟩, ⟨*c:1,b:*1⟩⟩)
b	*fadb* : 1	(⟨⟨*fad*⟩, ⟨*sup*:1⟩, ⟨ε^r=50%⟩, ⟨*c*:0,b:1⟩⟩)
c	*efc* : 2	(⟨⟨*ef*⟩, ⟨*sup*:1⟩, ⟨ε^r=50%⟩, ⟨*c*:1,b:0⟩⟩)
c	*eac* : 1	Ignored, as it is prefix of pattern (*eacb* : 1). The support of *eac* : 1 becomes zero after subtracting its support from the support of *eacb* : 1.

approach, the conditional patterns ⟨*efc* : 2⟩ is a prefix of pattern (⟨*efcb* : 1⟩). After subtracting its support from the conditional pattern of *b*, the new support becomes ⟨*efc* : 1⟩. The conditional pattern ⟨*eac* : 1⟩ is ignored as it is subset of conditional pattern of item *b* with similar support and it is already mapped on the Apx-FP-tree of itemset (*cb*). The pattern ⟨*efc* : 1⟩ is an Apx-conditional pattern with row error-tolerance of 50% because items *b* is missing from the pattern. The pattern ⟨*efc* : 1⟩ is converted into Apx-conditional pattern ⟨⟨*efc* : 1⟩, ⟨*sup* : 1⟩, ⟨ε^r=50%⟩, ⟨*c* : 1,*b* : 0⟩⟩.

Table 2 shows ll Apx-conditional patterns of itemset (*cb*). The itemset (*cb*) is examined for approximate and absolute itemset support thresholds and error-tolerance percentages of row and column. Two out of four Apx-conditional patterns of Table 2 have row error-tolerance of 100%. Thus, itemset (*cb*) qualifies *abs_sup*. The approximate support of itemset (*cb*) is four, which qualifies *apx_sup*. Each item of (*cb*) is appeared in at least two out of four transactions of Apx-conditional patterns, which qualifies ε^c. Thus, the itemset (*cb*) is an approximate frequent itemset of length two. The algorithm constructs Apx-FP-tree for itemset (*cb*) from the Apx-conditional patterns for mining supersets of *cb*. Figure 2 shows Apx-FP-tree. The Apx-CP-Tables of Apx-FP-tree are linked together by making linked list of Apx-CP-Tables.

Figure 2
Apx-FP-tree of itemset (*cb*)



3.2. Mining Approximate Frequent Itemsets from Apx-FP-tree

From the above section we know that Apx-FP-tree of an itemset X maps all transactions of X, therefore, complete set of supersets containing X can be mined directly from the Apx-FP-tree without scanning the original database. This section presents an approach how to mine complete set of AFIs from the Apx-FP-tree.

Example: If we examine the head table of itemset (cb) (Figure 2) the supersets containing itemset (cb) can be divided into four parts: (1) Candidate AFIs contain item d, (2) candidate AFIs contain item a, (3) candidate AFIs having item f, and (4) candidate AFIs contain item e. The algorithm mines the supersets as follows.

To examine whether itemsets (dcb) is an AFI and to generate subsequent supersets of itemsets (dcb). The algorithm starts by generating Apx-conditional patterns of (bcd). The Apx-conditional patterns of (dcb) are obtained from the Apx-FP-tree of (cb) by traversing all nodes of Apx-CP-Table. Each node of Apx-CP-Table generates an Apx-conditional pattern. Since Apx-FP-tree of itemset (cb) include only those transactions which have items b and c but not those transactions that contain only d but not c and b. Therefore, FP-tree (shown in Figure 1) is again traversed for generating conditional patterns of d. The algorithm obtains only those conditional patterns from FP-tree which contain d but not c and b. Item d contains two conditional patterns: $\langle fad : 2\rangle$, and $\langle d : 2\rangle$. The pattern $\langle fad : 2\rangle$ is a prefix of previous conditional pattern of b ($\langle fadb : 1\rangle$). Therefore, after subtracting its support from the conditional pattern of b the new support of pattern $\langle fad\rangle$ becomes 1. Note, there is no need to include conditional pattern $\langle d : 2\rangle$ because it has row error-tolerance (33%) and

the length of pattern is one. Thus, this pattern cannot contribute in the support count of itemset (acb) and supersets of (acb). This is because, a candidate itemset of length four will make the row error-tolerance of this pattern equal to 25% which does not qualify $\varepsilon^r = 50\%$. The following list shows the Apx-conditional patterns obtained from the Apx-CP-Table and FP-tree:

- $\langle\langle ef\rangle, \langle sup : 1\rangle, \langle \varepsilon^r = 100\%\rangle, \langle c : 1, b : 1\rangle\rangle$,
- $\langle\langle ef\rangle, \langle sup : 1\rangle, \langle \varepsilon^r = 50\%\rangle, \langle c : 1, b : 0\rangle\rangle$,
- $\langle\langle ea\rangle, \langle sup : 1\rangle, \langle \varepsilon^r = 100\%\rangle, \langle c : 1, b : 1\rangle\rangle$,
- $\langle\langle fad\rangle, \langle sup : 1\rangle, \langle \varepsilon^r = 50\%\rangle, \langle c : 0, b : 1\rangle\rangle$, and
- $\langle\langle fad\rangle, \langle sup : 1\rangle, \langle \varepsilon^r = 0\%\rangle, \langle c : 0, b : 0\rangle\rangle$,

All conditional patterns of dcb are converted into Apx-conditional patterns. Table 3 shows the Apx-conditional patterns of dcb. Last conditional pattern $\langle fad\rangle$ is ignored for including in the count of itemset support. This is because this pattern has row error-tolerance $\varepsilon^r = 0\%$ and items c and b are missing from the pattern. This makes the row error-tolerance (33%) which is less than $\varepsilon^r = 50\%$. Second conditional pattern $\langle ef\rangle$ is also ignored. This is because it has only item c but b and d are missing from the pattern. This makes the row error-tolerance (33%) which does not quality $\varepsilon^r = 50\%$. All other conditional patterns qualify $\varepsilon^r = 50\%$, however, there is no Apx-conditional pattern with row error-tolerance equal to $\varepsilon^r = 100\%$. Thus, the itemset (dcb) does not qualify absolute itemset support, therefore, dcb is an infrequent approximate itemset. The algorithm backtracks to itemset (cb) and examines the approximate conditions of next superset (acb).

Similar to dcb, the Apx-conditional patterns of itemset (acb) are obtained from the Apx-CP-Table of itemset (cb) by traversing all nodes of Apx-CP-Table. FP-tree of (Figure 1) is traversed for including conditional patterns of a. This includes transactions which contain a but not c and b. Item a contains three conditional patterns: $\langle efa : 1\rangle$, $\langle ea : 1\rangle$ and $\langle fa : 2\rangle$. The patterns $\langle ea : 1\rangle$ and $\langle fa : 2\rangle$ are prefix of patterns ($\langle eacb : 1\rangle$ and $\langle fadb : 1\rangle$ (see Table 2). The pattern $\langle ea : 1\rangle$ has same support so it is ignored. The support of pattern $\langle fa : 2\rangle$ becomes one after subtracting its support from the support of pattern $\langle fadb : 1\rangle$. The following list shows the Apx-conditional patterns obtained from the Apx-CP-Table of (cb) and FP-tree:

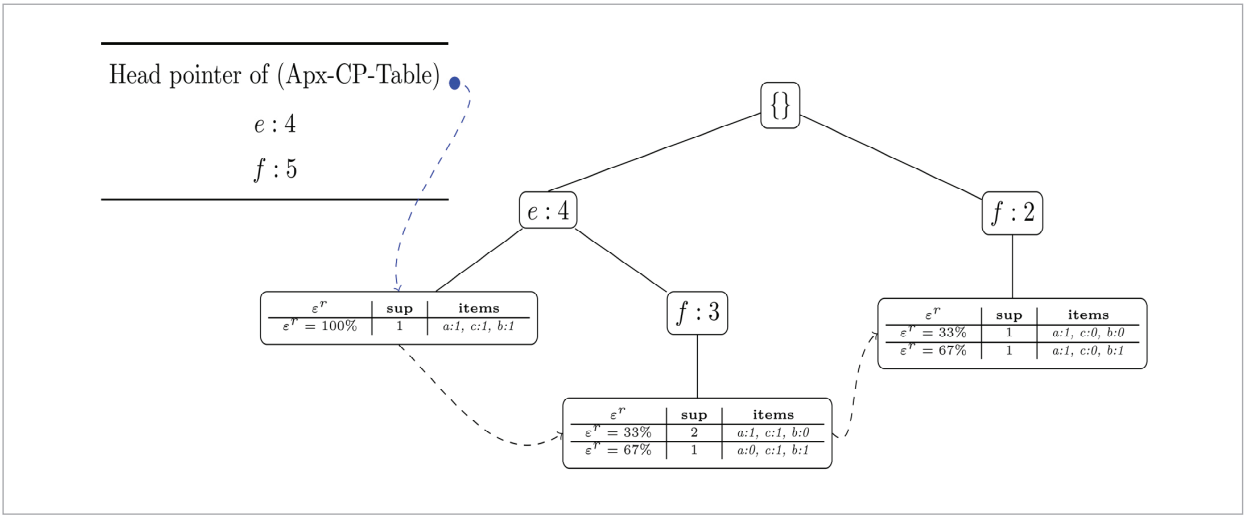
- $\langle\langle ef\rangle, \langle sup : 1\rangle, \langle \varepsilon^r = 100\%\rangle, \langle c : 1, b : 1\rangle\rangle$,
- $\langle\langle ef\rangle, \langle sup : 1\rangle, \langle \varepsilon^r = 50\%\rangle, \langle c : 1, b : 0\rangle\rangle$,
- $\langle\langle ea\rangle, \langle sup : 1\rangle, \langle \varepsilon^r = 100\%\rangle, \langle c : 1, b : 1\rangle\rangle$,
- $\langle\langle fa\rangle, \langle sup : 1\rangle, \langle \varepsilon^r = 50\%\rangle, \langle c : 0, b : 1\rangle\rangle$,
- $\langle\langle fa\rangle, \langle sup : 1\rangle, \langle \varepsilon^r = 0\%\rangle, \langle c : 0, b : 0\rangle\rangle$, and
- $\langle\langle efa\rangle, \langle sup : 1\rangle, \langle \varepsilon^r = 0\%\rangle, \langle c : 0, b : 0\rangle\rangle$,

The conditional patterns $\langle fa\rangle$ (second last) and $\langle efa\rangle$ are ignored for including in the count of itemset support. This is because both patterns have row error-tolerance $\varepsilon^r = 0\%$ and items b and c are missing from the patterns. This makes the row error-tolerance 33% which is less than $\varepsilon^r = 50\%$. The second conditional pattern $\langle ef\rangle$ is also ignored. This is because this pattern has two missing items (b and a). This makes the row error-tolerance

Table 3
Apx-conditional patterns of itemset (*dcb*)

Apx-Conditional Patterns Discovered from Apx-FP-tree of itemset (*cb*)	Apx-Conditional Patterns used for Constructing Apx-FP-tree of (*dcb*)
⟨⟨*ef*⟩, ⟨*sup*:1⟩, ⟨ε^r=100%⟩, ⟨*c*:1,*b*:1⟩⟩	⟨⟨*ef*⟩, ⟨*sup*:1⟩, ⟨ε^r=67%⟩, ⟨*d*:0,*c*:1,*b*:1⟩⟩
⟨⟨*ef*⟩, ⟨*sup*:1⟩, ⟨ε^r=50%⟩, ⟨*c*:1,*b*:0⟩⟩	⟨⟨*ef*⟩, ⟨*sup*:1⟩, ⟨ε^r=33%⟩, ⟨*d*:0,*c*:1,*b*:0⟩⟩
⟨⟨*ea*⟩, ⟨*sup*:1⟩, ⟨ε^r=100%⟩, ⟨*c*:1,*b*:1⟩⟩	⟨⟨*ea*⟩, ⟨*sup*:1⟩, ⟨ε^r=67%⟩, ⟨*d*:0,*c*:1,*b*:1⟩⟩
⟨⟨*fad*⟩, ⟨*sup*:1⟩, ⟨ε^r=50%⟩, ⟨*c*:0,*b*:1⟩⟩	⟨⟨*fa*⟩, ⟨*sup*:1⟩, ⟨ε^r=67%⟩, ⟨*d*:1,*c*:0,*b*:1⟩⟩
⟨⟨*fad*⟩, ⟨*sup*:1⟩, ⟨ε^r=0%⟩, ⟨*c*:0,*b*:0⟩⟩	⟨⟨*fa*⟩, ⟨*sup*:1⟩, ⟨ε^r=33%⟩, ⟨*d*:1,*c*:0,*b*:0⟩⟩

Figure 3
Apx-FP-tree of itemset (*acb*)



equal to 33% which do not quality ε^r = 50%. The conditional patterns ⟨*ef*⟩, ⟨*ea*⟩ and ⟨*fa*⟩ qualify ε^r = 50% which makes the approximate itemset support equal to three and the itemset qualifies *apx_sup* = 3.

Table 4 shows the Apx-conditional patterns of itemset (*acb*). Itemset (*acb*) has one Apx-conditional pattern with row error-tolerance equal to 100%. Thus, the itemset qualifies minimum absolute itemset support (*ab_sup* = 2). The supports of individual items in the Apx-conditional patterns are: ⟨*b* : 3⟩, ⟨*c* : 2⟩, and ⟨*a* : 2⟩. Each item qualifies column error-tolerance ε^c = 50%. Thus, itemset (*acb*) is an AFI of length three.

Next the algorithm generates supersets of itemset (*acb*) from the Apx-FP-tree of (*acb*). The superset of (*acb*) are partitioned into following two subsets: (1) Candidate AFIs containing item *f*, and (2) candidate AFIs containing item *e*.

To generate supersets of itemsets (*facb*) and to examine the AFI conditions of itemsets (*facb*). The algorithm obtains the error-tolerance of row and column from the Apx-FP-tree of (*acb*) by traversing all nodes of Apx-FP-table (see Figure 1). As Apx-FP-tree of itemset (*acb*) includes only those transactions which have items *a*, *c* and *b* but not those which have only *f*. Therefore, FP-tree of (Figure 1) is traversed for obtaining conditional patterns of *f*. This includes transactions which have *f* but not *a*, *c* and *b*. Item *f* contains two conditional patterns: ⟨*ef* : 3⟩ and ⟨*f* : 2⟩. Both conditional patterns are ignored as these are subsets of already discovered conditional patterns of items *a* and *b* and contain similar support. The following list shows the Apx-conditional patterns obtained from the Apx-CP-Table of itemset (*acb*) and FP-tree:

- $\langle\langle ef\rangle, \langle sup:2\rangle, \langle \varepsilon^r = 33\%\rangle, \langle a:1, c:1, b:0\rangle\rangle$,
- $\langle\langle ef\rangle, \langle sup:1\rangle, \langle \varepsilon^r = 67\%\rangle, \langle a:0, c:1, b:1\rangle\rangle$,
- $\langle\langle e\rangle, \langle sup:1\rangle, \langle \varepsilon^r = 100\%\rangle, \langle a:1, c:1, b:1\rangle\rangle$,
- $\langle\langle f\rangle, \langle sup:1\rangle, \langle \varepsilon^r = 33\%\rangle, \langle a:1, c:0, b:0\rangle\rangle$, and
- $\langle\langle f\rangle, \langle sup:1\rangle, \langle \varepsilon^r = 67\%\rangle, \langle a:1, c:0, b:1\rangle\rangle$,

All patterns are converted into the Apx-conditional patterns of itemset (*facb*) (see Table 5). All Apx-conditional patterns qualify $\varepsilon^r = 50\%$, however, there is no Apx-conditional pattern with row error-tolerance equal to 100%. The itemset (*facb*) does not qualify absolute support threshold, therefore, itemset (*facb*) is an infrequent approximate itemset. The algorithm backtracks to itemset (*acb*) and examines the approximate conditions of next superset (*eacb*).

Similar to (*facb*), the Apx-conditional patterns of itemset (*eacb*) are obtained by traversing all nodes of Apx-FP-table from the Apx-FP-tree of itemset (*acb*). FP-tree of Figure 1 is traversed for obtaining those conditional patterns of *e* which are not yet included in the Apx-FP-tree of itemset (*acb*). Only one conditional pattern $\langle e:1\rangle$ is not subset of any already included conditional pattern. All others are ignored because they are subsets of already included conditional patterns. The pattern $\langle e:1\rangle$ is ignored as the length of pattern is one and the length of candidate itemset is four. This makes the row error-tolerance equal to 25% which is less than $\varepsilon^r = 50\%$.

The following list shows the Apx-conditional patterns obtained from the Apx-CP-Table of itemset (*acb*) and FP-tree:

- $\langle\langle\rangle, \langle sup:2\rangle, \langle \varepsilon^r = 33\%\rangle, \langle a:1, c:1, b:0\rangle\rangle$,
- $\langle\langle e\rangle, \langle sup:1\rangle, \langle \varepsilon^r = 67\%\rangle, \langle a:0, c:1, b:1\rangle\rangle$,
- $\langle\langle e\rangle, \langle sup:1\rangle, \langle \varepsilon^r = 100\%\rangle, \langle a:1, c:1, b:1\rangle\rangle$,
- $\langle\langle\rangle, \langle sup:1\rangle, \langle \varepsilon^r = 33\%\rangle, \langle a:1, c:0, b:0\rangle\rangle$, and
- $\langle\langle\rangle, \langle sup:1\rangle, \langle \varepsilon^r = 67\%\rangle, \langle a:1, c:0, b:1\rangle\rangle$,

All patterns are converted into the Apx-conditional patterns of itemset (*eacb*) (see Table 6). The second last pattern $\langle\rangle$ is ignored for including in the count of itemset support. This is because it has row error-tolerance $\varepsilon^r = 33\%$ and items *e*, *c* and *b* are missing from the pattern. This makes the row error-tolerance equal to 25% which is less than $\varepsilon^r = 50\%$. All other conditional

Table 4

Apx-conditional patterns of itemset (*acb*)

Apx-Conditional Patterns Discovered from Apx-FP-tree of itemset (*cb*)	Apx-Conditional Patterns used for Constructing Apx-FP-tree of (*acb*)
⟨⟨*ef*⟩, ⟨*sup*:1⟩, ⟨ε^r=100%⟩, ⟨*c*:1,*b*:1⟩⟩	⟨⟨*ef*⟩, ⟨*sup*:1⟩, ⟨ε^r=67%⟩, ⟨*a*:0,*c*:1,*b*:1⟩⟩
⟨⟨*ef*⟩, ⟨*sup*:1⟩, ⟨ε^r=50%⟩, ⟨*c*:1,*b*:0⟩⟩	⟨⟨*ef*⟩, ⟨*sup*:1⟩, ⟨ε^r=33%⟩, ⟨*a*:0,*c*:1,*b*:0⟩⟩
⟨⟨*ea*⟩, ⟨*sup*:1⟩, ⟨ε^r=100%⟩, ⟨*c*:1,*b*:1⟩⟩	⟨⟨*e*⟩, ⟨*sup*:1⟩, ⟨ε^r=100%⟩, ⟨*a*:1,*c*:1,*b*:1⟩⟩
⟨⟨*fa*⟩, ⟨*sup*:1⟩, ⟨ε^r=50%⟩, ⟨*c*:0,*b*:1⟩⟩	⟨⟨*f*⟩, ⟨*sup*:1⟩, ⟨ε^r=67%⟩, ⟨*a*:1,*c*:0,*b*:1⟩⟩
⟨⟨*fa*⟩, ⟨*sup*:1⟩, ⟨ε^r=0%⟩, ⟨*c*:0,*b*:0⟩⟩	⟨⟨*f*⟩, ⟨*sup*:1⟩, ⟨ε^r=33%⟩, ⟨*a*:1,*c*:0,*b*:0⟩⟩
⟨⟨*efa*⟩, ⟨*sup*:1⟩, ⟨ε^r=0%⟩, ⟨*c*:0,*b*:0⟩⟩	⟨⟨*ef*⟩, ⟨*sup*:1⟩, ⟨ε^r=33%⟩, ⟨*a*:1,*c*:0,*b*:0⟩⟩

Table 5

Apx-conditional patterns of itemset (*facb*)

Apx-Conditional Patterns Discovered from Apx-FP-tree of itemset (*acb*)	Apx-Conditional Patterns used for Constructing Apx-FP-tree of (*facb*)
⟨⟨*ef*⟩, ⟨*sup*:2⟩, ⟨ε^r=33%⟩, ⟨*a*:1,*c*:1,*b*:0⟩⟩	⟨⟨*e*⟩, ⟨*sup*:2⟩, ⟨ε^r=50%⟩, ⟨*f*:2,*a*:1,*c*:1,*b*:0⟩⟩
⟨⟨*ef*⟩, ⟨*sup*:1⟩, ⟨ε^r=67%⟩, ⟨*a*:0,*c*:1,*b*:1⟩⟩	⟨⟨*e*⟩, ⟨*sup*:1⟩, ⟨ε^r=75%⟩, ⟨*f*:1,*a*:0,*c*:1,*b*:1⟩⟩
⟨⟨*e*⟩, ⟨*sup*:1⟩, ⟨ε^r=100%⟩, ⟨*a*:1,*c*:1,*b*:1⟩⟩	⟨⟨*e*⟩, ⟨*sup*:1⟩, ⟨ε^r=75%⟩, ⟨*f*:0,*a*:1,*c*:1,*b*:1⟩⟩
⟨⟨*f*⟩, ⟨*sup*:1⟩, ⟨ε^r=33%⟩, ⟨*a*:1,*c*:0,*b*:0⟩⟩	⟨⟨⟩, ⟨*sup*:1⟩, ⟨ε^r=50%⟩, ⟨*f*:1,*a*:1,*c*:0,*b*:0⟩⟩
⟨⟨*f*⟩, ⟨*sup*:1⟩, ⟨ε^r=33%⟩, ⟨*a*:1,*c*:0,*b*:1⟩⟩	⟨⟨⟩, ⟨*sup*:1⟩, ⟨ε^r=75%⟩, ⟨*f*:1,*a*:1,*c*:0,*b*:1⟩⟩

Table 6
Apx-conditional patterns of itemset (*eacb*)

Apx-Conditional Patterns Discovered from Apx-FP-tree of itemset (*acb*)	Apx-Conditional Patterns used for Constructing Apx-FP-tree of (*eacb*)
⟨⟨*e*⟩, ⟨*sup*:2⟩, ⟨ε^r=33%⟩, ⟨*a*:1,*c*:1,*b*:0⟩⟩	⟨⟨⟩, ⟨*sup*:2⟩, ⟨ε^r=50%⟩, ⟨*e*:2,*a*:1,*c*:1,*b*:0⟩⟩
⟨⟨*e*⟩, ⟨*sup*:1⟩, ⟨ε^r=67%⟩, ⟨*a*:0,*c*:1,*b*:1⟩⟩	⟨⟨⟩, ⟨*sup*:1⟩, ⟨ε^r=75%⟩, ⟨*e*:1,*a*:0,*c*:1,*b*:1⟩⟩
⟨⟨*e*⟩, ⟨*sup*:1⟩, ⟨ε^r=100%⟩, ⟨*a*:1,*c*:1,*b*:1⟩⟩	⟨⟨⟩, ⟨*sup*:1⟩, ⟨ε^r=100%⟩, ⟨*e*:1,*a*:1,*c*:1,*b*:1⟩⟩
⟨⟨*f*⟩, ⟨*sup*:1⟩, ⟨ε^r=33%⟩, ⟨*a*:1,*c*:0,*b*:0⟩⟩	⟨⟨⟩, ⟨*sup*:1⟩, ⟨ε^r=25%⟩, ⟨*e*:0,*a*:1,*c*:0,*b*:0⟩⟩
⟨⟨*f*⟩, ⟨*sup*:1⟩, ⟨ε^r=33%⟩, ⟨*a*:1,*c*:0,*b*:1⟩⟩	⟨⟨⟩, ⟨*sup*:1⟩, ⟨ε^r=50%⟩, ⟨*e*:0,*a*:1,*c*:0,*b*:1⟩⟩

patterns qualify ε^r = 50% which make the approximate itemset support equal to five, thus, the itemset (*eacb*) qualifies the minimum *apx_sup* = 3. Table 6 shows the Apx-conditional patterns of (*eacb*). Itemset (*eacb*) has one Apx-conditional pattern with row error-tolerance equal to 100%. Thus, itemset (*eacb*) qualifies minimum absolute itemset support. The supports of individual items obtained from the Apx-condi-

Algorithm 1: Procedure for Mining Approximate Frequent Itemsets.

```
Data: Transactional database, apx_sup, abs_sup, ε^r, ε^c
Result: All approximate frequent itemests
Procedure Apx-FP-Growth(Tree,δ)
β = ∅;
if δ == ∅ then
    for each itemset X of length two do
        for each item i of X do
            C_i = i's conditional patterns from Tree;
            if C_i contains at least ε^r items then
                C = C ∪ C_i;
                if sup(X) in C >= apx_sup and α(X) in C >= abs_sup and each item i of X
                 appears at least ε^c times in C then
                    output X as AFI;
                    β = X;
                    Apx-FP-Tree(C,β);
else
    if Tree contains a single path P and P contains at least ε^r items then
        for each item P_i in the P do
            X = δ ∪ P_i;
            if sup(X) >= apx_sup and α(X) >= abs_sup and each item i in X appears at least ε^c
             times in P then
                output X as an AFI;
    else
        for each header item h_i in the header table of Tree do
            generate itemset X = δ ∪ h_i;
            for each path P of Tree and P contains at least ε^r items do
                C = C ∪ generate conditional pattern from P;
            if sup(X) >= apx_sup and α(X) >= abs_sup and each item i of X appears at least ε^c
             times in P then
                output X as an AFI;
                β = X;
                Apx-FP-Tree(C,β);
```

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

```
Data: Conditional patterns of itemset X
Result: Construct Apx-FP-Tree of X
Procedure Apx-FP-Tree(C,X)
for each conditional pattern c in C do
    convert c into Apx-conditional pattern c^α;
    if c^α contains at-least ε^r items of X then
        C^α = C^α ∪ c^α;
construct Apx-FP-Tree of X using C^α;
if Apx-FP-Tree of X != ∅ then
    Apx-FP-Growth( Apx-FP-Tree of X, X);
```

tional patterns of (*eacb*) are: $\langle b:3\rangle$, $\langle c:3\rangle$, $\langle a:3\rangle$, and $\langle e:4\rangle$. All items qualify column error-tolerance ε^c = 50%. Thus, (*eacb*) is an AFI of length four. All candidate supersets of itemset (*acb*) have been generated. The algorithm now backtracks to itemset (*cb*) and examines the approximate conditions of next superset *fcb*. Similar to the working example of itemset (*acb*) presented in above section, the algorithm generates remaining AFIs by generating recursive Apx-FP-trees, and then performs mining on them, respectively. Algorithm 1 and Algorithm 2 present pseudo code to mine AFIs using pattern growth approach.

4. Experiments

This section compares the performance of (*Apx-PatternGrowth*) with the best related AFI mining algorithms on one synthetic and three real benchmark databases. The databases are *T10I4D100K, FoodMart,* and *BMSWebView1*. These databases have been frequently in FIM articles for performance analysis of algorithms. The databases are freely available to download from the FIM repository (http://fimi.ua.ac.be). Table 7 shows the properties of databases. The columns of Table 7 show size of databases, average transaction length, and the number of frequent items.

The performance of *Apx-PatternGrowth* is compared with *CoreApriori* [10-11], *AC-Close* [10-11] and *Apx-MultiTree* [25]. *CoreApriori* is similar to *AC-Close*, however, it applies Apriori-based candidate generation and test approach for generating lattice of core patterns. *CoreApriori* then examines the AFI conditions of core patterns in lattice with top-down approach starting from the largest pattern and proceeds level by level, in the size decreasing order of core patterns. *CoreApriori* is not efficient since it generates exponential number of candidate itemsets and requires multiple scan of database for counting itemsets support. *AC-Close* also mines AFIs by generating lattice of core patterns. However, *AC-Close* mines core patterns using pattern growth. The pattern growth reduces the number of candidate itemsets. However, both *CoreApriori* and *AC-Close* examine the support of itemsets and error-tolerance of row and column with multiple scans of database.

Apx-MultiTree is a tree based approach. *Apx-MultiTree* is not efficient in terms of calculating support of itemsets from the FP-tree as it maps similar set of transactions on multiple FP-trees when the trans-

Table 7
Properties of 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 8
Characteristics of experiment settings

Database	Number of Transactions	ε^r	ε^c	apx_sup	abs_sup
Retail	88,162	50%	50%	0.01% *to* 0.11%	$\alpha = \frac{apx\_sup}{4}$
BMSWebView1	59,601	50%	50%	0.03% *to* 0.23%	$\alpha = \frac{apx\_sup}{4}$
FoodMart	4,141	50%	50%	0.01% *to* 0.07%	$\alpha = \frac{apx\_sup}{4}$
T10I4D100K	100,000	50%	50%	0.05% *to* 0.55%	$\alpha = \frac{apx\_sup}{4}$

actions have different percentages of missing items. The implementations of *CoreApriori*, *AC-Close* and *Apx-MultiTree* are not available from the authors, therefore, we implement all algorithms in C++. All experiments are executed on MacBook Pro 3.2 GHz processor with processor speed of 2.7 GHz and 8GB of RAM. Various values of approximate and absolute itemset supports are used for performance analysis with the error-tolerance percentages of row and column $\varepsilon^r = 50\%$ and $\varepsilon^c = 50\%$. Table 8 shows the settings of parameters.

We analyze the performance of AFI mining algorithms with the following three aspects.

- In first aspect, we compare all algorithms in term of how much processing time the algorithms consume for mining complete set of AFIs.
- In second aspect, we compare the performance of algorithms on varying database size. This helps us to understand the scalability analysis of algorithms.
- In third aspect, we compare the performance of algorithms on subsets of databases with varying transaction size.

To analyze the performance of AFI algorithms on varying database size, we generate subsets of database containing 10,000 to 90,000 transactions. For building subsets, the *T10I4D100K* and *Retail* are partitioned into five subsets. The size of each subset is 30*k* transactions by including random 30,000 transactions from the original database. The first subset contains only the transactions that have length between 1 to 10. The second subset contains transactions of length between 11 to 20. Third, fourth and fifth subsets contain transaction of length between 21 to 30, 31 to 40 and 41 to 50 respectively. We perform experiments with second and third aspects only on *Retail* and *T10I4D100K* databases. The reason to select only these databases is both databases are sparse and have varying length of transactions. All algorithms are executed over the subsets with similar values of approximate support thresholds.

Figures 4-7 show the performance of algorithms in terms of processing time. Note that the processing time of an algorithm is the total execution time from providing input to the algorithm and mining complete set of AFIs. On low (support) thresholds all algorithms consume very long execution time. We stop the execution of algorithm when it takes more than 2300 seconds.

Figures 4-7 show the processing time of algorithms on *Retail*, *BMSWebView1*, *FoodMart* and *T10I4D100K*. Figures 9 and 11 show the performance of algorithms on varying database size. Figures 8 and 10 show the performance of algorithms on varying average transaction length.

Figures 4, 8 and 9 provide performance of all algorithms on *Retail* database. The results show *Apx-PatternGrowth* mines complete set of itemsets in less processing time than *CoreApriori*, *AC-Close*, and *Apx-MultiTree*. On databases with long transactions, the performance of *Apx-PatternGrowth* is scalable which is challenging for the *CoreApriori* and *AC-Close* to complete the processing within a reasonable time. The *Apx-PatternGrowth* is scalable as it maps transactions on FP-tree and applies pattern growth for generating candidate itemsets that exists on the branches of FP-tree. FP-tree maps the transactions on similar branches when the transactions share similar set of items. This explains, why the *Apx-PatternGrowth* computes the approximate conditions of itemsets in less processing time even when the databases are large and support thresholds are low.

If we compare the results on *T10I4D100K*,

FoodMart and *BMSWebView1* databases, then similar to *Retail* database, the *Apx-PatternGrowth* mines AFIs in less processing time than other algorithms on different support thresholds. We noted, on low support thresholds the *CoreApriori*, *AC-Close*, and *Apx-MultiTree* could not mine all AFIs within 2300 seconds. On large databases, *CoreApriori* generates large combinations of candidate itemsets including those that do not available in the database. The combinations become exponential when the number of frequent items of database are very large. Also, *CoreApriori* does not provide any facility to compress the transactions when transactions share a common set of items. Therefore, *CoreApriori* spends large amount of processing time on examining approximate conditions of itemsets. *Apx-MultiTree* takes less processing time than *CoreApriori*. This is because *Apx-MultiTree* applies pattern growth approach for generating itemsets and counting itemsets support. Whereas *CoreApriori* is an Apriori-based candidate generate approach and it generates exponential number of candidate itemsets. *AC-Close* takes less processing time than *CoreApriori* because it generates lattice of core patterns using pattern growth approach.

Figure 4

Performance of AFI mining algorithms on *Retail* with ε^r = 50% and ε^c = 50%

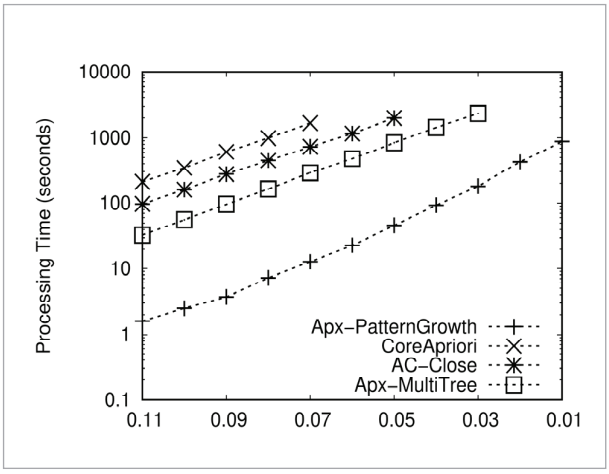


Figure 5

Performance of AFI mining algorithms on *BMSWebView1* with ε^r = 50% and ε^c = 50%

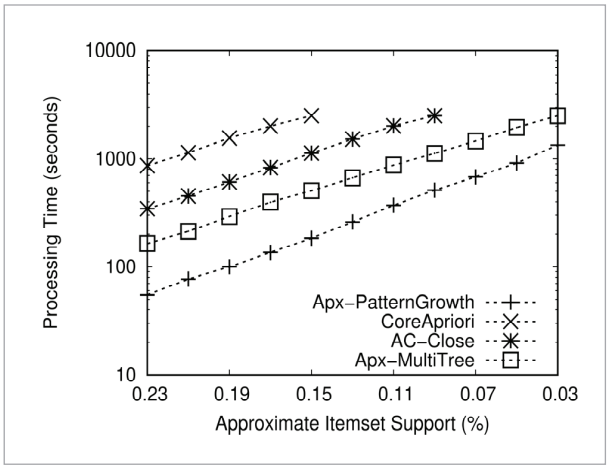


Figure 6

Performance of AFI mining algorithms on *T10I4D100K* with ε^r = 50% and ε^c = 50%

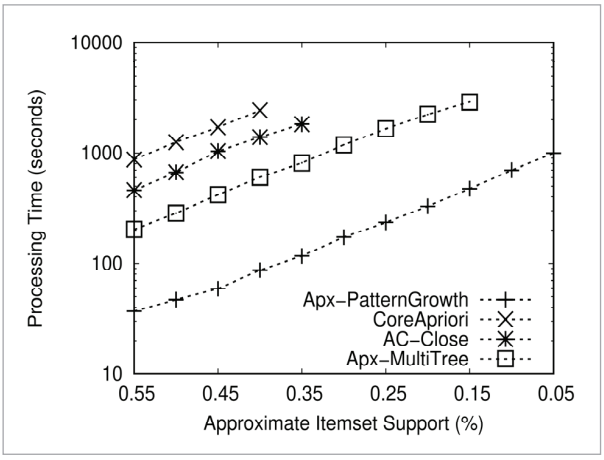


Figure 7

Performance of AFI mining algorithms on *FoodMart* with ε^r = 50% and ε^c = 50%

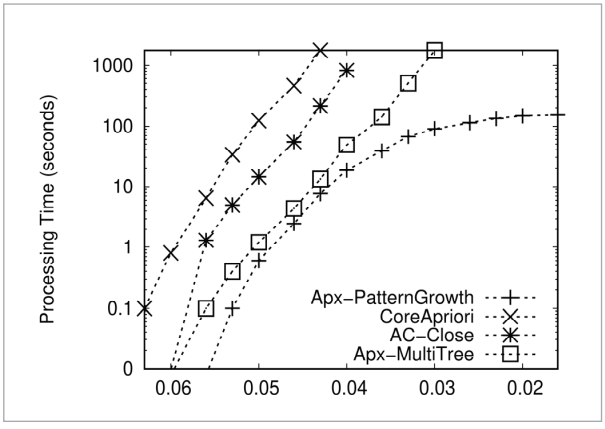


Figure 8
Scalability of AFI mining algorithms on *Retail* with varying transaction length and ε^r = 50% and ε^c = 50% and apx_sup = 0.09%

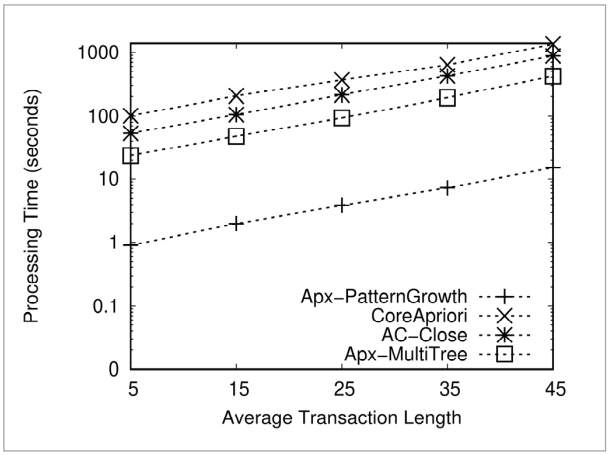


Figure 9
Scalability of AFI mining algorithms on *Retail* with varying transaction size and ε^r = 50% and ε^c = 50% and apx_sup = 0.09%

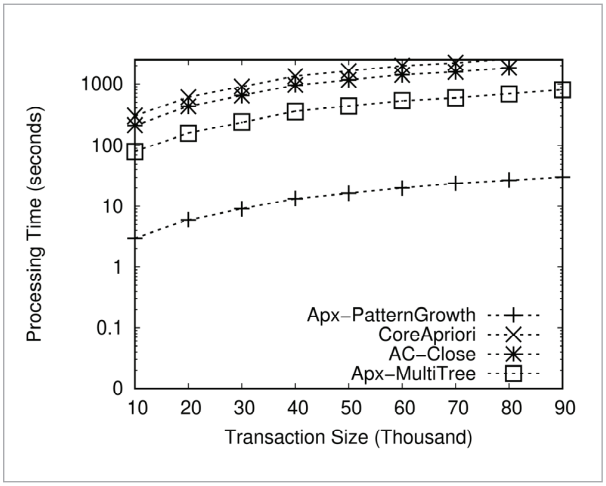


Figure 10
Scalability of AFI mining algorithms on *T10I4D100K* with varying transaction length and ε^r = 50% and ε^c = 50% and apx_sup = 0.50%

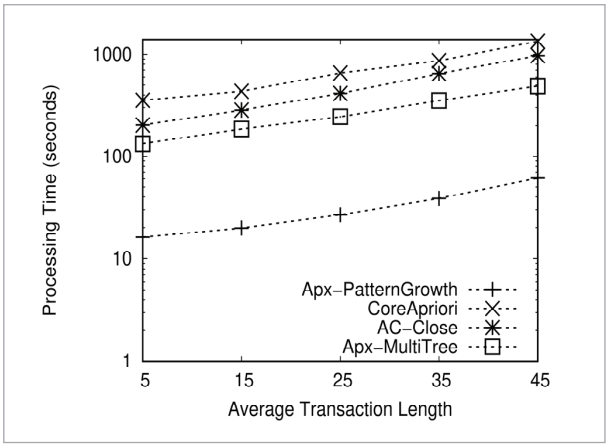
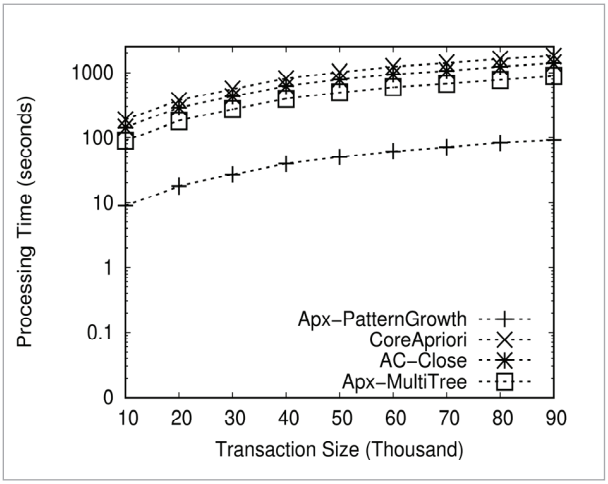


Figure 11
Scalability of AFI mining algorithms on *T10I4D100K* with varying transaction size and ε^r = 50% and ε^c = 50% and apx_sup = 0.50%



5. Conclusion

In this article, we presented a novel algorithm *Apx-PatternGrowth* for storing approximate transactions and mining approximate frequent itemsets (AFI) in noisy databases. The *Apx-PatternGrowth* has several advantages over existing Apriori-based AFI mining algorithms: (1) It maps approximate transactions on a highly compressed data structure, approximate FP-tree which maps transactions on common branches when multiple transactions share similar set of items. Thus, the size of database during itemset mining becomes substantially small which helps in efficiently computing support of itemsets. (2) The mining approach of proposed algorithm is not Apriori-based because the proposed algorithm generates

candidate itemsets from conditional patterns of Apx-FP-tree by applying pattern-growth approach. This substantially reduces the combinations of candidate itemsets as the pattern-growth makes sure the algorithm never generates any candidate itemset which does not exist in the transactions of database. We implemented and analyzed the performance of *Apx-PatternGrowth* with the existing AFI mining algorithms on several benchmark databases of varying database size and transaction length. Our results show the proposed algorithm mines complete AFIs in less processing time and scalable on large databases. For future work, there any many interesting directions to explore including mining only top k approximate itemsets without support threshold, mining maximal AFIs and mining sequential AFIs using pattern growth.

References

1. Agrawal, R., Srikant, R. Fast Algorithms for Mining Association Rules. Proceedings of 20th International Conference on VLDB, 1994, pages 487-499. https://dl.acm.org/doi/10.5555/645920.672836
2. Baek, Y., Yun, U., Kim, H., Kim, J., Vo, B., Truong, T. C., Deng, Z. H. Approximate High Utility Itemset Mining in Noisy Environments. Knowledge Based Systems, 2021, 212, 106596. https://doi.org/10.1016/j.knosys.2020.106596
3. Bashir, S. An Efficient Pattern Growth Approach for Mining Fault Tolerant Frequent Itemsets. Expert Systems with Applications, 2020, 143, 113046. https://doi.org/10.1016/j.eswa.2019.113046
4. Bashir, S., Baig, A. R. Max-FTP: Mining Maximal Fault-Tolerant Frequent Patterns from Databases. Proceedings of 24th British National Conference on Databases, BNCOD 24, Glasgow, UK, July 3-5, 2007, 4587 of Lecture Notes in Computer Science, 235-246, Springer. https://doi.org/10.1007/978-3-540-73390-4_26
5. Bashir, S., Halim, Z., and Baig, A. R. Mining Fault Tolerant Frequent Patterns Using Pattern Growth Approach. Proceedings of 6th ACS/IEEE International Conference on Computer Systems and Applications, AICCSA 2008, Doha, Qatar, March 31 - April 4, 2008, 172-179. https://doi.org/10.1109/AICCSA.2008.4493532
6. Bodon, F. A Fast APRIORI Implementation. Proceedings of FIMI '03, Frequent Itemset Mining Implementations, ICDM 2003 Workshop on Frequent Itemset Mining Implementations, 19th December 2003, Florida, USA.
7. Burdick, D., Calimlim, M., Flannick, J., Gehrke, J., Yiu, T. MAFIA: A Maximal Frequent Itemset Algorithm. IEEE Transactions on Knowledge and Data Engineering, 2005, 17, 1490-1504. https://doi.org/10.1109/TKDE.2005.183
8. Chen, R., Zhao, S., Liu, M. A Fast Approach for Up-Scaling Frequent Itemsets. IEEE Access, 2020, 8, 97141-97151. https://doi.org/10.1109/ACCESS.2020.2995719
9. Chen, S., Nie, L., Tao, X., Li, Z., Zhao, L. Approximation of Probabilistic Maximal Frequent Itemset Mining Over Uncertain Sensed Data. IEEE Access, 2020, 8, 97529-97539. https://doi.org/10.1109/ACCESS.2020.2997409
10. Cheng, H., Yu, P. S., Han, J. AC-Close: Efficiently Mining Approximate Closed Itemsets by Core Pattern Recovery. Proceedings of the 6th IEEE International Conference on Data Mining (ICDM 2006), 18-22 December 2006, Hong Kong, China, 839-844. https://doi.org/10.1109/ICDM.2006.10
11. Cheng, H., Yu, P. S., Han, J. Approximate Frequent Itemset Mining in the Presence of Random Noise. Soft Computing for Knowledge Discovery and Data Mining, Springer 2008, 363-389. https://doi.org/10.1007/978-0-387-69935-6_15
12. Cheung, Y.-L., Fu, A. W.-C. Mining Frequent Itemsets Without Support Threshold: With and Without Item Constraints. IEEE Transactions on Knowledge and Data Engineering, 2004, 16, 1052-1069. https://doi.org/10.1109/TKDE.2004.44
13. Creighton, C., Hanash, S. Mining Gene Expression Databases for Association Rules. Bioinformatics (Oxford, England), 2003, 19(1), 79-86. https://doi.org/10.1093/bioinformatics/19.1.79
14. Tripathia, D., Nigam, B., Edla, D. R. A novel Web Fraud Detection Technique Using Association Rule Mining. Procedia Computer Science, 2017, 115, 274-281. https://doi.org/10.1016/j.procs.2017.09.135
15. Fung, B. C. M., Wang, K., Ester, M. Hierarchical Document Clustering Using Frequent Itemsets. Proceedings of the Third SIAM International Conference on Data Mining, San Francisco, CA, USA, May 1-3, 2003, 59-70. https://doi.org/10.1137/1.9781611972733.6
16. Gan, W., Lin, J. C.-W., Fournier-Viger, P., Chao, H.-C., Zhan, J. Mining of Frequent Patterns with Multiple Minimum Supports. Engineering Applications of Artificial

Intelligence, 2017, 60, 83-96. https://doi.org/10.1016/j.engappai.2017.01.009

17. Han, J., Cheng, H., Xin, D., Yan, X. Frequent Pattern Mining: Current Status and Future Directions. Data Mining and Knowledge Discovery, 2007, 15, 55-86. https://doi.org/10.1007/s10618-006-0059-1
18. Han, J., Pei, J. Pattern-Growth Methods. Frequent Pattern Mining Springer, 2014, 65-81. https://doi.org/10.1007/978-3-319-07821-2_3
19. Han, J., Pei, J., Yin, Y. Mining Frequent Patterns Without Candidate Generation. Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA., 1-12. https://doi.org/10.1145/335191.335372
20. Han, J., Pei, J., Yin, Y., Mao, R. Mining Frequent Patterns Without Candidate Generation: A Frequent-Pattern Tree Approach. Data Mining and Knowledge Discovery, 2004, 8, 53-87. https://doi.org/10.1023/B:DAMI.0000005258.31418.83
21. Huynh-Thi-Le, Q., Le, T., Vo, B., Le, B. An Efficient and Effective Algorithm for Mining Top-Rank-k Frequent Patterns. Expert Systems with Applications, 2015, 42, 156-164. https://doi.org/10.1016/j.eswa.2014.07.045
22. Iváncsy, R., Vajk, I. Frequent Pattern Mining in Web Log Data. Acta Polytechnica Hungaria, 2006, 3. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.101.4559
23. Jiang, F., Leung, C. K., Zhang, H. B-Mine: Frequent Pattern Mining and Its Application to Knowledge Discovery from Social Networks. Proceedings of Web Technologies and Applications, 18th Asia-Pacific Web Conference, Suzhou, China, September 23-25, 2016, 316-328. https://doi.org/10.1007/978-3-319-45814-4_26
24. Koh, J.-L., Yo, P.-W. An Efficient Approach for Mining Fault-Tolerant Frequent Patterns Based on Bit Vector Representations. Proceedings of Database Systems for Advanced Applications, 10th International Conference, Beijing, China, April 17-20, 2005, 568-575. https://doi.org/10.1007/11408079_51
25. Koh, J.-L., Tu, Y.-L. A Tree-Based Approach for Efficiently Mining Approximate Frequent Itemsets. Proceedings of Fourth International Conference on Research Challenges in Information Science (RCIS), 2010. https://doi.org/10.1109/RCIS.2010.5507360
26. Kosters, W. A., Pijls, W. Apriori, a Depth First Implementation. Proceedings of FIMI '03, Frequent Itemset Mining Implementations, ICDM 2003 Workshop on Frequent Itemset Mining Implementations, 19th December 2003, Florida, USA.
27. Lee, G., Lin, Y.-T. A Study on proportional Fault-Tolerant Data Mining. Proceedings of 2006 Int. Conf. Innovations in Information Technology, 2006, Dubai, UAE. https://doi.org/10.1109/INNOVATIONS.2006.301951
28. Lee, G., Peng, S.-L., Lin, Y.-T. Proportional Fault-Tolerant Data Mining with Applications to Bioinformatics. Information Systems Frontiers, 2009, 11, 461-469. https://doi.org/10.1007/s10796-009-9158-z
29. Liu, G., Lu, H., Yu, J. X., Wang, W., Xiao, X. AFOPT: An Efficient Implementation of Pattern Growth Approach. Proceedings of FIMI '03, Frequent Itemset Mining Implementations, ICDM 2003 Workshop on Frequent Itemset Mining Implementations, 19th December 2003, Florida, USA.
30. Liu, J., Paulsen, S., Sun, X., Wang, W., Nobel, A., Prins, J. Mining approximate Frequent Itemsets in the Presence of Noise: Algorithm and Analysis. Proceedings of the Sixth SIAM International Conference on Data Mining, April 20-22, 2006, Bethesda, MD, USA, 407-418. https://doi.org/10.1137/1.9781611972764.36
31. Liu, S., Poon, C. K. On Mining Proportional Fault-Tolerant Frequent Itemsets. Proceedings of 19th International Conference on Database Systems for Advanced Applications, Bali, Indonesia, April 21-24, 2014, 342-356. https://doi.org/10.1007/978-3-319-05810-8_23
32. Liu, S., Poon, C. K. On Mining Approximate and Exact Fault-Tolerant Frequent Itemsets. Knowledge and Information Systems, 2018, 55, 361-391. https://doi.org/10.1007/s10115-017-1079-4
33. Mallik, S., Mukhopadhyay, A., Maulik, U. Ranwar: Rank-Based Weighted Association Rule Mining from Gene Expression and Methylation Data. IEEE Transactions on NanoBioscience, 2015, 14, 59-66. https://doi.org/10.1109/TNB.2014.2359494
34. Moosavi, S. A., Jalali, M., Misaghian, N., Shamshirband, S., Anisi, M. H. Community Detection in Social Networks Using User Frequent Pattern Mining. Knowledge and Information Systems, 2017, 51, 159-186. https://doi.org/10.1007/s10115-016-0970-8
35. Rehman, S.-R., Ashraf, J., Habib, A., Salam, A. Top-k Miner: Top-k identical Frequent Itemsets Discovery Without User Support Threshold. Knowledge and Information Systems, 2016, 48, 741-762. https://doi.org/10.1007/s10115-015-0907-7
36. Uno, T., Kiyomi, M., Arimura, H. LCM ver. 2: Efficient Mining Algorithms for Frequent/Closed/Maximal Itemsets. Proceedings of FIMI '04, IEEE ICDM Workshop on Frequent Itemset Mining Implemen-

tations, Brighton, UK, November 1, 2004. https://doi.org/10.1145/1133905.1133916

37. Vo, B., Pham, S., Le, T., Deng, Z.-H. A Novel Approach for Mining Maximal Frequent Patterns. Expert System with Applications, 2017, 73, 178-186. https://doi.org/10.1016/j.eswa.2016.12.023
38. Yu, X., Korkmaz, T. Heavy Path Based Super-Sequence Frequent Pattern Mining on Web Log Dataset. Artificial Intelligence Research, 2015, 4. https://doi.org/10.5430/air.v4n2p1
39. Yu, X., Li, Y., Wang, H. Mining Approximate Frequent Patterns from Noisy Databases. Proceedings of 10th International Conference on Broadband and Wireless Computing, Communication and Applications (BWCCA), 2015. https://doi.org/10.1109/BWCCA.2015.29
40. Zhang, W., Yoshida, T., Tang, X., Wang, Q. Text Clustering Using Frequent Itemsets. Knowledge Based Systems, 2010, 23, 379-388. https://doi.org/10.1016/j.knosys.2010.01.011