A Tree-based Approach for Efficiently Mining Approximate Frequent Itemsets[*]

Jia-Ling Koh
Department of Computer Science and Information Engineering
National Taiwan Normal University
Taipei, Taiwan
jlkoh@csie.ntnu.edu.tw

Yi-Lang Tu
Department of Computer Science and Information Engineering
National Taiwan Normal University
Taipei, Taiwan

***Abstract*—The strategies for mining frequent itemsets, which is the essential part of discovering association rules, have been widely studied over the last decade. In real-world datasets, it is possible to discover multiple fragmented patterns but miss the longer true patterns due to random noise and errors in the data. Therefore, a number of methods have been proposed recently to discover approximate frequent itemsets. However, a challenge of providing an efficient algorithm for solving this problem is how to avoid costly candidate generation and test. In this paper, an algorithm, named FP-AFI (FP-tree based Approximate Frequent Itemsets mining algorithm), is developed to discover approximate frequent itemsets from a FP-tree-like structure. We define a recursive function for getting the set of transactions which fault-tolerant contain an itemset *P*. The patterns in the fault-tolerant supporting transactions of *P* are represented by the conditional AFP-trees of *P*. Moreover, to avoid re-constructing the tree structure in the mining process, two pseudo-projection operations on AFP-trees are provided to obtain the conditional AFP-trees of a candidate itemset systematically. Consequently, the approximate support of a candidate itemset and the item supports of each item in the candidate are obtained easily from the conditional AFP-trees. Hence, the constrain test of a candidate itemset is performed efficiently without additional database scan. The experimental results show that the FP-AFI algorithm performs much better than the FP-Apriori and the AFI algorithms in efficiency especially when the size of data set is large and the minimum threshold of approximate support is small. Moreover, the execution time of FP-AFI is scalable even when the error threshold parameters become large.**

Keywords - Approximate frequent itemset mining; FP-tree structure.

I. INTRODUCTION

Among the various data mining applications, mining association rules is an important one [2]. The strategies for mining frequent itemsets, which is the essential part of discovering association rules, have been widely studied over the last decade such as the Apriori[1], DHP[11], and FP-growth[6]. In the traditional frequent itemsets mining algorithms, a strict definition of support is used to require every item in a frequent itemset occurring in each supporting transaction. However, in real-world datasets, it is possible to discover multiple fragmented patterns but miss the longer true patterns due to random noise and errors in the data.

Motivated by such considerations, the problem of mining approximate frequent itemsets (or called fault-tolerant frequent itemsets) have been studied recently in many works [3][4][5][8][9][10][12][16]. In these approaches, it is allowed that a specified fraction of the items in an itemset can be missing when counting support of the itemset in a transaction database.

This problem was defined and solved in [12] by proposing the FT-Apriori algorithm. In this work, the number of errors allowed in a supporting transaction is a fixed constant value. Besides, an *item support threshold* is used to enforce the minimum fraction that each item of the frequent itemset occurs in its supporting transactions. The FT-Apriori algorithm was extended from the Apriori algorithm, in which the downward closure property among the fault-tolerant frequent patterns is applicable. Similar to the Apriori-like algorithms, FT-Apriori algorithm suffered from generating a large number of candidates and repeatedly scanning database. Besides, this approach does not consider that the longer itemsets should be allowed more missing items than the shorter ones.

To avoid repeated scans of the database, an algorithm, called VB-FT-Mine (Vector-Based Fault–Tolerant frequent itemsets Mining), was proposed in [8] for speeding up the process of fault-tolerant frequent patterns mining. In this approach, the fault–tolerant appearing vector was designed to represent the distribution that a candidate itemset is contained in the data set with fault-tolerance. The VB-FT-Mine algorithm applies a depth-first pattern growing method to generate candidate itemsets. The fault-tolerant appearing vectors of the candidate itemsets are obtained systematically. Besides, whether a candidate is a fault-tolerant frequent itemset could be decided quickly by performing vector operations on the appearing vectors.

Yang et. al proposed two error tolerant models for discovering the error-tolerant itemsets (ETIs) [16]. For a strong ETI, the *row error threshold* was used to place constraints on the fraction of errors permitted that an error-tolerant itemset is contained in a supporting transaction. On the other hand, without setting the row error threshold, an itemset is named a weak ETI if the fraction of errors in the set of its supporting transactions is below a certain threshold.

[*] This work was partially supported by the R.O.C. N.S.C. under Contract No. 98- 2221-E-003-017 and NSC 98-2631-S-003-002.

The problem of approximate frequent itemsets was defined in [9], which combined the requirements of a strong ETI [16] and the *item support* threshold defined in [12]. In addition to the row error threshold, another parameter named *column error threshold* was used to set criteria on the fraction of errors allowed for each item of the approximate frequent itemsets in its supporting transactions.

According to the given row error threshold, the number of errors allowed for an itemset in its supporting transaction is proportioned to the length of the itemset. Consequently, the anti-monotone property of support measure does not hold, which makes the construction of an efficient algorithm for discovering approximate frequent itemsets very challenging. Moreover, [3] considered that a large number of uninteresting patterns, which seldom occurs together in reality, may be discovered as a result of the relaxed pattern mining. Not only slowing down the mining process, it is not easy for users to distinguish the interesting patterns from the uninteresting ones. Therefore, a core pattern factor is defined in [3] to require that at least some transactions must contain all the items in an approximate frequent itemset. An algorithm called AC-Close was proposed to discover the approximate closed itemsets from the core patterns. The experimental results show that the AC-Close algorithm gets higher precision than the AFI algorithm when noisy data is introduced to the data set.

The frequent-pattern tree (FP-tree) proposed in [6] is an extended prefix-tree structure for storing compressed and crucial information in transactions about frequent itemsets. Based on the tree structure, efficient algorithms [6][15] were proposed for mining frequent itemsets However, the FP-tree structure was not applied to discover approximate frequent itemsets in the previous works.

According to the discussion of the related works, the purpose of this paper is to develop an algorithm for mining approximate frequent itemsets from a FP-tree-like structure efficiently. To prevent from discovering the uninteresting patterns, in addition to the row error threshold, column error threshold, and the minimum threshold of the approximate support, our approach adopts the core pattern factor defined in [3] to be one of the constraints.

There are two noticeable advantages of the FP-tree based mining algorithms. First, the FP-tree is a highly compact database structure which is usually substantially smaller than the original dataset. When the size of database is large, the costly database scans in the mining process can be reduced effectively. Moreover, to avoid costly candidate generation and test, the FP-tree based mining algorithms perform pattern growing by successively concatenating frequent 1-itemset found in the conditional FP-trees. This ensures that the unnecessary candidates which are not in the database are never generated.

In this paper, an algorithm, named FP-AFI (FP-tree based Approximate Frequent Itemsets mining algorithm), is proposed to discover approximate frequent itemsets, which inherits the advantages of the FP-tree based mining algorithms. The FP-AFI algorithm applies a depth-first search strategy to generate candidates of approximate frequent itemsets. We define a recursive function for the set of transactions which fault-tolerant contain an itemset P. In other words, if the set of transactions which fault-tolerant contain the prefix of P is known, the set of transactions which fault-tolerant contain P could be obtained from the defined function. The patterns in the fault-tolerant supporting transactions of P are represented in one or more FP-tree-like structures, which are named the conditional AFP-trees of P. Moreover, to avoid re-constructing the tree structure in the mining process, two pseudo-projection operations on the AFP-trees are provided to obtain the conditional AFP-trees of a candidate itemset systematically. Consequently, the approximate support of a candidate itemset and the item supports of each item in the candidate are obtained easily from the conditional AFP-trees. As a result, the constraint test of a candidate itemset is performed efficiently without additional database scan.

In the experiments, the FP-AFI algorithm is compared with two related works: the FT-Apriori [6] and the AFI algorithms[2]. For getting the same result with the FP-AFI, these two algorithms are modified to include the pruning strategy according to the constraint of core pattern factor. The experimental results show that the FP-AFI algorithm performs much better than the FP-Apriori and the AFI algorithms in efficiency especially when the size of data set is large and the minimum threshold of approximate support is small. Moreover, the execution time of the FP-AFI algorithm is scalable even when the error threshold parameters become large.

This paper is organized as follows. The problem of approximate frequent itemsets mining is defined in Section 2. The modified FP-tree structure, named AFP-tree, and the corresponding projection operations are introduced in Section 3. Section 4 gives the details of our proposed FP-AFI algorithm. The performance evaluation on the proposed algorithm and two related works is reported in Section 5. Finally, Section 6 concludes this paper.

II. PROBLEM DEFINITION

Let $I = \{i_1, i_2, \ldots, i_m\}$ denote the set of items in a specific application domain. A set of items is called an *itemset*; an itemset containing k items is called a *k-itemset*. Let *TDB* denote a database of transactions, where each transaction T_i in *TDB* is a non-empty subset of I. For a given itemset P, we say that a transaction T_i *contains* itemset P if and only if $P \subseteq T_i$. The support count of an itemset P in *TDB*, denoted as $sup(P)$, is the number of transactions in *TDB* containing P. Given a minimum support threshold $min\_sup \in [0,1]$, an itemset P is called a frequent itemset in *TDB* if $sup(P) \geq |TDB| \times min\_sup$.

Definition 1 (Fault-tolerant contain): Let ε_r and ε_c denote the *row error threshold* and *column error threshold*, respectively, where ε_r and $\varepsilon_c \in [0,1]$. Given an itemset P and a transaction $T_i = (i, S_i)$, if there is a subset of S_i, S', where $S' \subseteq P$, and there does not exist a proper superset of S', S'', such that $S' \subset S'' \subseteq P$, the transaction T_i is said to **contain** itemset P **with** $(|P|-|S'|)$ **errors**. Let μ_P denote the value of $\lfloor |P| \times \varepsilon_r \rfloor$, which is called the **maximum row error** of P. T_i is said to **fault-tolerant contain** (abbreviated as *FT-contain*) P if and only if T_i contains itemset P with k errors and $k \leq \mu_P$.

Transaction ID	Items
T1	AF
T2	AB
T3	ABCE
T4	ABCDE
T5	ACD
T6	CDE
T7	BCDE
T8	ABCDEG

Figure 1. The sample database TDB.

Definition 2 (Approximate support and **Item support):** Let $T_a^P(\mu_P)$ denote the set of transactions in a database *TDB* which *FT-contain* an itemset *P*. Besides, $|T_a^P(\mu_P)|$ denotes the number of members in $T_a^P(\mu_P)$. The **approximate support** of the itemset *P*, denoted as $sup_a(P)$, is defined to be the number of transactions which FT-contain *P*; that is, $sup_a(P)$ is equal to $|T_a^P(\mu_P)|$. For each item *x* in the itemset *P*, the number of transactions in $T_a^P(\mu_P)$ which contain item *x* is called the **item support** of *x*, denoted as $sup_{item}^P(x)$.

Definition 3 (Approximate frequent itemset): Given a *core pattern factor* α, a row error threshold ε_r, a column error threshold ε_c, and a minimum support threshold *min_sup*. An itemset *P* is called an approximate frequent itemset in the transaction database *TDB* iff

1) $sup_a(P) \geq |TDB| \times min\text{-}sup$;
2) for each item *x* in *P*, $sup_{item}^P(x) \geq \lceil sup_a(P) \times (1-\varepsilon_c) \rceil$; and
3) $sup(P) \geq |TDB| \times (min\text{-}sup \times \alpha)$.

[Example 2.1] Fig. 1 shows an example of transaction database *TDB*. Let *P* denote the itemset {A, B, C, D, E}. Since the transactions T4 and T8 contain *P*, $sup(P)$ is 2. We assume that ε_r is 0.2, so the maximum row error of *P*, μ_P, is 1. In other words, when checking whether a transaction T_i FT-contains *P*, T_i must contain at least 4 items ($|P| - \lfloor |P| \times \varepsilon_r \rfloor = 5 - \lfloor 5 \times 0.2 \rfloor = 4$) in *P*. Among the transactions in *TDB*, there are four transactions: T3, T4, T7, and T8 which FT-contain *P*. Consequently, $T_a^P(1)$ = {T3, T4, T7, T8} and $Sup_a(P)$ is 4. Among the members in $T_a^P(1)$, the item A is contained in T3, T4, and T8. Thus, $sup_{item}^P(A)$ is 3. Similarly, the item supports of the other items in *P* are $sup_{item}^P(B)=4$, $sup_{item}^P(C)=4$, $sup_{item}^P(D)=3$, and $sup_{item}^P(E)=4$. Suppose the parameters *min-sup*, ε_c, and α are set to be 0.3, 0.5, and 0.5, respectively. The itemset *P* is an approximate frequent itemset because it satisfies all the three requirements of **Definition 3**. On the other hand, let *Q* denote the itemset {B, C, D, E, G}. In this case, μ_Q is also 1. The transactions T4, T7, and T8 in *TDB* FT-contain *Q*; thus, $Sup_a(Q)$ is 3. Although $Sup_a(Q)$ is larger than $|TDB| \times min\text{-}sup (=8 \times 0.3)$, the itemset *Q* is not an approximate frequent itemset because the value of $sup_{item}^Q(G)$, 1, does not satisfy the second requirement of **Definition 3**. Furthermore, the itemset *Q* violates the third requirement in the definition because the value of $Sup(Q)$, 1, is less than $|TDB| \times (min\text{-}sup \times \alpha)$.

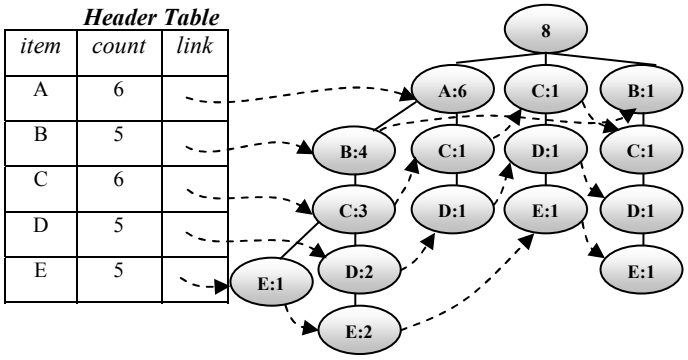


Figure 2. The constructed AFP-tree for the transaction database TDB.

III. TREE-BASED STORAGE STRUCTURE

A. Approximate Frequent Pattern Tree

The frequent-pattern tree (FP-tree) proposed in [6] is an extended prefix-tree structure for storing compressed and crucial information in transactions about frequent itemsets. In our approach, the FP-tree is modified to support the mining of approximate frequent itemsets.

By following the FP-tree structure provided in [6], in addition to the *children* links, each node except the root node in our proposed FP-tree-like structure consists of three fields: *item-name*, *count*, and *node-link*, where *item-name* records the item represented by this node, *count* records the support counts of the patterns represented by the path reaching this node, and *node-link* links to the next node in the tree structure which represents the same item. Here, the root node of the tree structure is modified to include a field named *count*, which is used to record the total number of transactions maintained in the tree. Furthermore, the header table of the tree structure consists of three fields: (1) *item-name*, (2) *count*, which is used to accumulate the support count of the item, and (3) *link*, which is a pointer pointing to the first node of the linked list for chaining all the nodes with the *item-name*.

Because of the anti-monotone property of support, if an itemset is not a core pattern, any one of its supersets can never be a core pattern. Consequently, the FP-AFI algorithm first constructs the FP-tree-like structure in two scans of the database. In the first scan, the count for each item is accumulated. In the second scan, when constructing nodes into the tree structure, the items in a transaction with support counts less than $|TDB| \times (min\text{-}sup \times \alpha)$ are ignored. Besides, the items in each transaction are sorted in lexicographically order. The constructed tree is called the **approximate frequent pattern tree** (abbreviated as AFP-tree) of the transaction database.

[**Example 3.1**] Suppose *min-sup* is set to be 0.5 and α is set to be 0.5. Besides, the transaction database is shown as Fig. 1. After the first scan, the items with their support counts are obtained as the following: A:6, B:5, C:6, D:5, E:5, F:1, and G:1. Among the items, the supports count of items F and G are less than $|TDB| \times (min\_sup \times \alpha) = 8 \times 0.5 \times 0.5 = 2$. Therefore, items F and G in the transactions are ignored when constructing the AFP-tree. The constructed AFP-tree for the transaction database is shown as Fig. 2.

B. Approximate Support Computation Function

Let $T_e^P(i)$ denote the set of transactions in *TDB* which contain the itemset *P* with *i* errors. Besides, let *j* and *k* denote two distinct non-negative integers. If a transaction contains the itemset *P* with *j* errors, it is impossible that the transaction contains *P* with *k* errors simultaneously. In other words, for any itemset *P*, $T_e^P(j)$ and $T_e^P(k)$ are disjoint if *j* is not equal to *k*. According to this property, the approximate support of an itemset *P* in a transaction database *TDB* can be obtained by the following function:

【Approximate support of an itemset *P*】

$$\text{If } \mu_P=0,\ sup_a(P) = |T_e^P(0)| = sup(P) \quad (1)$$

$$\text{else if } \mu_P = |P|,\ sup_a(P) = |TDB| \quad (2)$$

$$\text{else } \quad sup_a(P) = \left|T_a^P(\mu_P)\right| = \left|\bigcup_{i=0}^{\mu_P} T_e^P(i)\right| = \sum_{i=0}^{\mu_P}\left|T_e^P(i)\right| \quad (3)$$

The challenge is how to get $|T_e^P(i)|$ for $0 \le i \le \mu_P$. Let *P* denote an *n*-itemset consisting of more than one item, i.e. $P=\{p_1, p_2, \ldots, p_n\}$, and *P'* denote the itemset consisting of the first *n-1* items in *P*. We call *P'* the (|*P*|-1)_prefix of *P*. When a transaction contains the itemset *P* with *i* errors, where $1 \le i \le |P|$, it is possible that the transaction contains the (|*P*|-1)_prefix of *P* with (*i*-1) errors and does not contain $\{p_n\}$. Another possibility is that the transaction contains the (|*P*|-1)_prefix of *P* with *i* errors and contains $\{p_n\}$. Therefore, the recursive function for getting $T_e^P(i)$, where $0 \le i \le |P|$, is defined as follows:

【Recursive Function of $T_e^P(i)$】

$$\text{If } i \le |P| \quad T_e^P(i) = (T_e^{P'}(i\text{-}1) \cap T_e^{\{Pn\}}(1)) \cup (T_e^{P'}(i) \cap T_e^{\{Pn\}}(0)) \quad (4)$$

for $|P|>1$ and $i>0$;

$$= T_e^{P'}(0) \cap T_e^{\{Pn\}}(0) \quad (5)$$

for $|P|>1$ and $i=0$;

$$= TDB - T_e^P(0) \quad (6)$$

for $|P|=1$ and $i=1$;

$$= T_e^P(0) \quad (7)$$

for $|P|=1$ and $i=0$;

Otherwise, $T_e^P(i) = \phi$.

[Example 3.2] Suppose ε_r is set to be 0.5 in this example. Let *P* denote the itemset {A, B}. Hence, the value of μ_P is 1. According to the definition of the function for getting the approximate support of an itemset, sup_a(AB) is computed by performing the following equation:

$$\sup_a(AB) = |T_e^{AB}(0)| + |T_e^{AB}(1)| \quad (8)$$

$$= |T_e^A(0) \cap T_e^B(0)| + |T_e^A(0) \cap T_e^B(1)| + |T_e^A(1) \cap T_e^B(0)|. \quad (9)$$

The first term $|T_e^A(0) \cap T_e^B(0)|$ in the equation indicates that the number of transactions which contain both items A and B. In the transaction database shown in Fig. 1, $|T_e^A(0) \cap T_e^B(0)|$ = |{T2, T3, T4, T8}|=4. The second term means the number of transactions which contains item A but do not contain item B, that is |{T1, T5}|=2. Similarly, the third term represents the number of transactions which contains item B but do not contain item A, i.e. |{T7}|=1. Consequently, sup_a(AB) = 4+2+1=7 is obtained.

C. Approximate Supports Counting from AFP- trees

In this section, according to the recursive function of $T_e^P(i)$, we will introduce how to get $|T_e^P(i)|$ for an itemset *P* by constructing the required conditional AFP-trees systematically.

It is worth noticing that, when constructing an AFP-tree, the items in a transaction are sorted in lexicographically order. Besides, the proposed FP-AFI algorithm adopts the prefix-based and depth-first approach to generate candidate itemsets. That is, the algorithm will generate candidate itemsets starting from a specific item. If the itemset *P* is an approximate frequent itemset, only the items whose lexicographically order are larger than the ones in *P* are appended to *P* to generate longer candidate itemsets. For reducing the storage requirement of the transactions in $T_e^P(i)$, when getting $T_e^P(i)$ for a pattern *P*, only the set of itemsets which follow *P* appearing in any transaction within $T_e^P(i)$ is maintained. It is called the **conditional pattern base** of *P* **with error** *i* and denoted as $PB_e^P(i)$. According to the definition of $PB_e^P(i)$, there is a one-to-one correspondence between the patterns in $PB_e^P(i)$ and $T_e^P(i)$. Thus, $|PB_e^P(i)|$ is equal to $|T_e^P(i)|$.

In the constructed AFP-tree of the transaction database, all the itemsets in the transactions which contain item A are maintained in the paths starting from the node of item A. Therefore, the set of patterns stored in the subtrees of the node of item A corresponding to $PB_e^A(0)$. By constructing an AFP-tree to represent the patterns in $PB_e^A(0)$, as shown in Fig. 3, the tree is called the A-conditional AFP-tree.

In order to save both memory requirement and execution time, we adopt the concept of TD-FP-growth algorithm [15], in which it is not necessary to construct additional pattern bases and sub-trees. On the contrary, only the conditional header table and the root node of the conditional AFP-tree are constructed. We call such operation a pseudo projection.

Let *P'* denote the (|*P*|-1)_prefix of *P* and *x* denote the last item in *P*. Then the *P*-conditional AFP-tree is obtained by performing a *projection* on the *P'*-conditional AFP-tree with respect to *x*. The process of performing a *projection* on the *P'*-conditional AFP-tree with respect to *x* is as follows.

1) Perform a projection *on the* P'*-conditional AFP-tree of with respect to* x*:*

Look for the entry of *x* in the *P'*-conditional header table. By following the link-list pointed by the *Link* field, all the nodes of *x* in the *P'*-conditional AFP-tree are reached. A root node R_x is constructed with $R_x.count=0$ and $R_x.children=null$ initially. For each node of *x* in the *P'*-conditional AFP-tree, denoted as N_x, the value of $N_x.count$ is added into $R_x.count$; besides, all the children of N_x are inserted into the *children* links of R_x. In addition, a *P*-conditional Header Table is constructed by accumulating the count of each item and set the

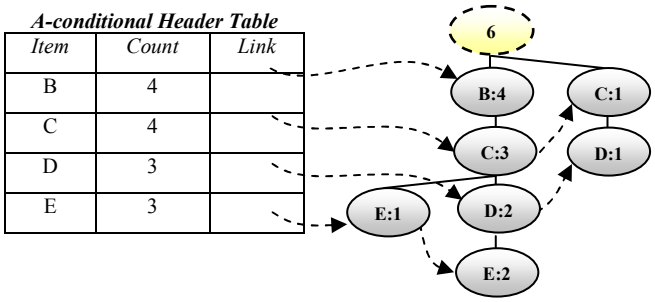


Figure 3. The A-conditional AFP-tree.

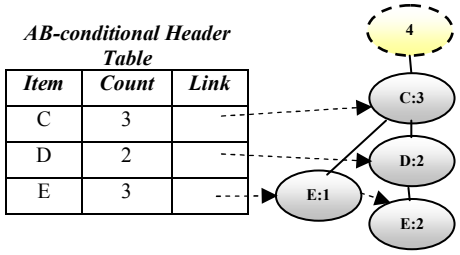


Figure 4. The AB-conditional AFP-tree.

Figure 5.

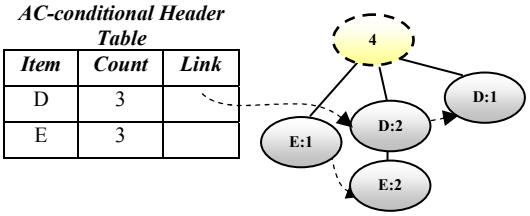


Figure 6. The AC-conditional AFP-tree.

link to point to the first node with the corresponding item in the *P*-conditional AFP-tree.

[Example 3.3] According to the A-conditional AFP-tree shown in Fig. 3, after performing a projection with respect to B, the AB-conditional AFP-tree is shown as Fig. 4. According to the count value stored in the root node of AB-conditional AFP-tree, the value of $|PB_e^{AB}(0)|$ is obtained. Similarly, the AC-conditional AFP-tree is shown in Fig. 5. In the A-conditional AFP-tree, there are two nodes representing item C. Therefore, the patterns occurring with C are maintained in the sub-trees of the two nodes of item C. To avoid re-constructing the AC-conditional AFP-tree, the two sub-trees rooted from the nodes of item D are not merged. All the children nodes of the two nodes representing item C become the *children* of the new root node. Although the result is not the most compact form of the AC-conditional AFP-tree, both the computing and storage cost of constructing a new AFP-tree is saved.

To get the values of $|PB_e^{AB}(1)|$, another pseudo projection operation named a *complement projection* is defined. A complement projection on the *P'*-conditional AFP-tree with respect to *x* is to get the patterns which co-occur with *P'* but do not occur with *x* from the *P'*-conditional AFP-tree. The result is denoted as the $P'\bar{x}$ -conditional AFP-tree.

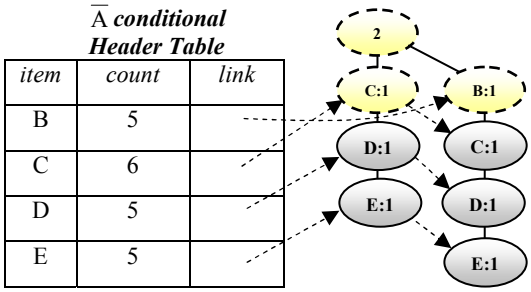


Figure 7. The $\bar{A}$-conditional AFP-tree.

For example, according to Fig. 2, after performing a complement projection on the AFP-tree with respect to A, we would get the patterns CDE and BCDE. The constructed $\bar{A}$-conditional header table and AFP-tree is shown as Fig. 6. To avoid destroying the original node links, the root node and the first node on each projected path is re-constructed as the nodes shown in dashed boundary. Moreover, the *count* field in the root node keeps the number of transactions which don't contain A. The value of the *count* field can be obtained by subtracting the *count* of A kept in the header table, 6 in this case, from the *count* stored in the root node of the FP-tree, 8 in this case. The process of performing a complement projection on the *P'*-conditional AFP-tree with respect to *x* is as follows.

2) Perform a complement projection *on the* P'*-conditional FP-tree with respect to* x*:*

A root node $R_{x'}$ is constructed with $R_{x'}.count$=0 and $R_{x'}.children$=null initially. Let y_1, y_2, …, y_k denote the items whose lexicographical order are larger than *x* and $y_1 < y_2 < \ldots < y_k$. The *Link* field of y_1 in the *P'*-conditional header table is fetched to access each node of y_1. For each node of y_1 on the *P'*-conditional AFP-tree, denoted as N_{y1}, the content of N_{y1} is copied into a new node, new_ N_{y1}, if there is not any ancestor of N_{y1} contains *x*. The node new_ N_{y1} is then assigned to be a child node of $R_{x'}$. If there is more than one node of y_1, these newly generated nodes are linked via the *node link*. For item y_i, where $2 \le i \le k$, the node of y_i, denoted as N_{yi}, is also checked one by one. However, the same process as for N_{yi} is performed on N_{yi} only when none of the ancestors of N_{yi} contains y_j for $1 \le j < i$. Let *m* denote the *count* value stored in the root node of the *P'*-conditional AFP-tree and *n* denote the support count of *x* stored in the conditional header table of *P'*. Then $R_{x'}.count$ is set to be *m*-*n*. Similarly, a $P'\bar{x}$-conditional Header Table is constructed by accumulating the count of each item and set the link points to the first node with the corresponding item in the $P'\bar{x}$-conditional AFP-tree.

[Example 3.4] For getting the value of $|PB_e^{\bar{A}B}(1)|$, a complement projection on A-conditional AFP-tree with respect to B is performed. The obtained $\bar{A}\bar{B}$-conditional AFP-tree is shown in Fig. 7. Besides, to perform a projection on the $\bar{A}$-conditional AFP-tree with respect to B will get the $\bar{A}B$-

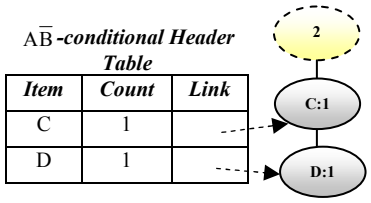


Figure 8. The $A\bar{B}$-conditional AFP-tree.

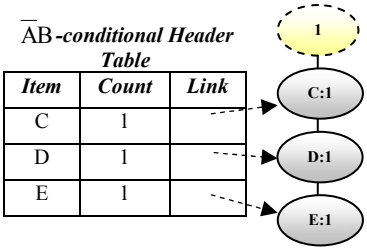


Figure 9. The $\bar{A}B$- conditional AFP-tree.

conditional AFP-tree as shown in Fig. 8. Accordingly, when ε_r is set to be 0.5, sup_a(AB) is computed by add $|PB_e^{AB}(0)|$ and $|PB_e^{AB}(1)|$, which can be obtained from the roots of the AB-conditional AFP-tree, and the sum of the values of the root nodes in the $A\bar{B}$-conditional AFP-tree and $\bar{A}B$-conditional AFP-tree, respectively. Therefore, the conditional AFP-trees of the candidate itemset AB include these three AFP-trees.

IV. THE FP-AFI ALGORITHM

A. Storage Structure for Candidate Itemsets

As the example shown in [Example 3.4], the approximate support of a candidate itemset can be obtained from one or more conditional AFP-trees. Consequently, during the mining process of the FP-AFI algorithm, a storage structure is constructed for a candidate itemset P to store the required conditional header tables and the root nodes of the conditional AFP-trees which are used to get the approximate support of P. The storage structure of a candidate itemset P consists of the following five fields:

1) *Itemset*: it stores the candidate itemset $P = \{p_1, p_2, \ldots, p_n\}$;
2) *AFP-tree list*: it maintains the list of root nodes of the required conditional AFP-trees of P;
3) *HTable list*: it keeps the header table for each conditional AFP-tree in the AFP-tree list;
4) *id vector list*: for each conditional AFP-tree in the AFP-tree list, a binary vector with length $|P|$ is assigned to identify the tree, where the ith bit is set to be 1 if the conditional AFP-tree is generated from a projection with respect to p_i; otherwise, the conditional AFP-tree is generated from a complement projection with respect to p_i and the ith bit is set to be 0.
5) *Parent*: it points to the storage structure of the ($|P|$-1) prefix of P.

The storage structure of an itemset P is maintained only when it is used to generate longer candidate itemsets. After the test on all the candidate itemsets with P as their prefix is finished, the storage structure of P will be deleted.

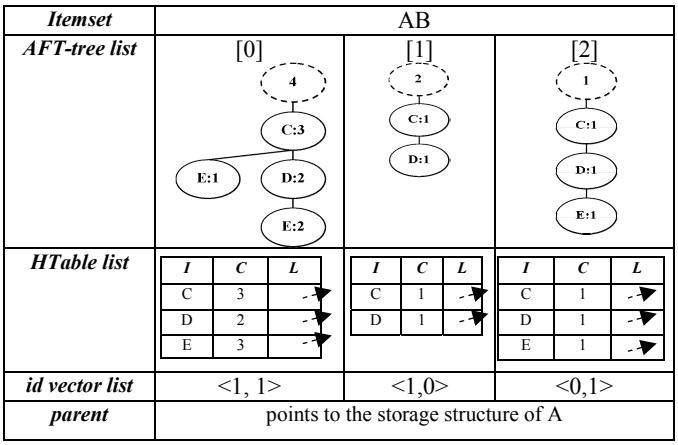


Figure 10. The storage structure of itemset AB.

[Example 4.1] To continue the running example, when ε_r is set to be 0.5, the maximum row error of the candidate itemset AB is 1. Therefore, the storage structure of AB is constructed as Fig. 9, where the AB-conditional AFP-tree identified by <1,1> represents the patterns in PB_e^{AB} (0). The $A\bar{B}$-conditional AFP-tree and $\bar{A}B$-conditional AFP-tree are identified by <1,0> and <0,1>, respectively; these two conditional AFP-trees jointly represent the patterns in $PB_e^{AB}(1)$. Let i denote the number of bits with value 0 in the *id vector* of a conditional AFP-tree. It is indicated that the corresponding transactions in the tree contain the candidate itemset with i errors. Moreover, the lost items which cause the errors are implied by the positions of the bits with value 0. As a result, for the items in the candidate itemset AB, their item supports can be obtained easily by the following formula: <1,1>×4+<1,0>×2+<0,1>×1=<6,6>, where each term in the summation corresponds to the multiplication of the count value in the root node and the *id vector* of a conditional AFP-tree in the *AFP-tree list* of AB. The result shows that both the sup_{item}^{AB}(A) and sup_{item}^{AB}(B) are 6.

B. Constructing the Conditional AFP-Trees Systematically

In the proposed FP-AFI algorithm, if an approximate frequent itemset P' is discovered. The next candidate itemset P is generated by appending an item x to P, where x is in the P'-conditional header table with $count \geq |TDB| \times min\_sup \times \alpha$. It is indicated that P is a core pattern. Then, for checking the other requirements of an approximate frequent itemset, the conditional AFP-Trees of P are constructed from the ones of P' as the following.

When the maximum row error of P is equal to the one of P' ($\mu_p = \mu_{p'}$), case 1 is performed; otherwise, case 2 is performed.

- <case 1>: For computing $sup_a(P)$, according to equation (3), we have to get $|T_e^P(i)|$ for $i = 0$ to μ_p. The conditional AFP-trees corresponding to $PB_e^{P'}(i)$ ($0 \leq i \leq \mu_{p'}$) have been constructed in the storage structure of P'. According to the recursive function defined in

equation (4), the conditional AFP-trees corresponding to $PB_e^P(i)$ are constructed by performing a projection on the conditional AFP-trees corresponding to $PB_e^{P'}(i)$ with respect to p_n, as well as a complement projection on the conditional AFP-trees corresponding to $PB_e^{P'}(i\text{-}1)$ with respect to p_n.

- <case 2>: In this case, the maximum row error of P is 1 larger than the one of P'($\mu_p=\mu_{p'}+1$). Since only the conditional AFP-trees corresponding to $PB_e^{P'}(i)$ ($0\le i\le\mu_{p'}$) were available for computing $sup_a(P')$, the conditional AFP-trees corresponding to $PB_p^{P'}(\mu_p)$ was not constructed previously. In order to get the conditional AFP-trees corresponding to $PB_e^P(\mu_p)$, we have to go back to the storage structure of the prefix P_k of P with $|P_k| = \mu_p$ *for getting* $PB_e^{Pk}(\mu_p)$. Let P_l denote the $(|P_k|\text{-}1)$_prefix of P_k and y denote the last item in P_k. The conditional AFP-trees corresponding to $PB_e^{Pk}(\mu_p)$ are obtained by performing a complement projection on the conditional AFP-trees corresponding to $PB_e^{Pl}(\mu_p - 1)$ with respect to y. We don't have to perform a projection on the conditional AFP-trees corresponding to $PB_e^{Pl}(\mu_p)$ with respect to y because $PB_e^{Pl}(\mu_p)$ must be an empty set when $|P_l|$ is less than μ_p. The similar process goes forward until the conditional AFP-trees corresponding to $PB_e^{P'}(\mu_p)$ is generated. Finally, the conditional AFP-trees corresponding to $PB_e^P(i)$ (i=0, …, μ_p) are constructed by performing the same operations described in <case 1>.

According to the conditional AFP-trees constructed in the storage structure of P, $Sup_a(P)$ is obtained by adding the count values in these conditional AFP-trees. The item support of each item p_i in P is also obtained easily from the conditional AFP-trees corresponding to P. For each conditional AFP-tree corresponding to P, suppose the count value stored in the root node is c and its id vector is $<b_1, b_2, \ldots, b_n>$; consequently, the frequency of item p_i appearing in this tree is $b_i \times c$. By summarizing the frequencies of p_i in all the conditional AFP-trees of P, $sup_{item}^P(p_i)$ is obtained.

C. FP-AFI Algorithm

The FP-AFI algorithm for mining approximate frequent itemsets using AFP-tree is proposed as the following.

Algorithm FP-AFI:
Input: A transaction database *TDB*, a *core pattern factor* α, a row error threshold ε_r, a column error threshold ε_c, and a minimum support threshold *min_sup*.
Output: The complete set of approximate frequent itemsets.
begin Initialization(*P'*);
Pattern_growth(*P'*);
end.

Transaction ID	Items
T1	AE
T2	AB
T3	ABC
T4	ABCD
T5	BCD

Figure 11. Sample database TDB2.

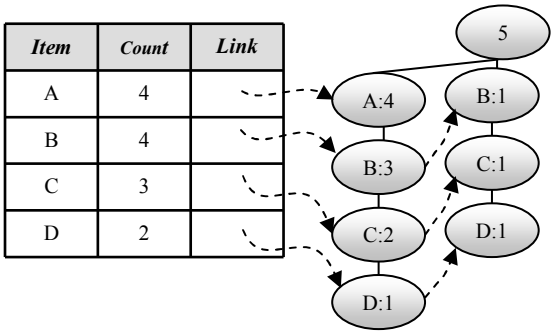


Figure 12. The constructed AFP-tree of TDB2.

Procedure Initialization(*P'*)
(1) {Scan *TDB* to accumulate the support count of each item;
(2) The items with support count less than $|TDB|\times min\_sup\times\alpha$ are ignored;
(3) Scan TDB to construct the AFP-tree and Header Table of *TDB*;
(4) $P' = \phi$;}

Procedure Pattern_growth(*P'*)
(1) { For each item x in P'-conditional header table
(2) If $x.count \ge |TDB|\times min\_sup\times\alpha$
(3) then $\{P = P' \cup \{x\}$;
(4) Construct the conditional AFP-trees required by P;
(5) Get $Sup_a(P)$ from the conditional AFP-trees;
(6) If $Sup_a(P) \ge |TDB|\times min\text{-}sup$
(7) then {Get $sup_{item}^P(p_i)$ for each item p_i in P;
(8) If (each item p_i in P $sup_{item}^P(p_i) \ge \lceil sup_a(P)\times(1-\varepsilon_c)\rceil$)
(9) then {output(P);
(10) $P'=P$;
(11) **Pattern_growth(*P'*)**;}
(12) }
(13) }
(14) }

[Example 4.2] The transaction database TDB_2 shown in Fig. 10 is used to explain the mining process of the FP-AFI algorithm. Here, *min_sup* is set to be 0.8, both ε_r, ε_c are set to be 0.5, and α is set to be 0.4.

First, the item E is ignored because its support does not satisfy the requirement of the given core pattern factor. The constructed AFP-tree of TDB_2 is shown as Fig. 11. From the header table of the AFP-tree, the candidate itemset P={A} is

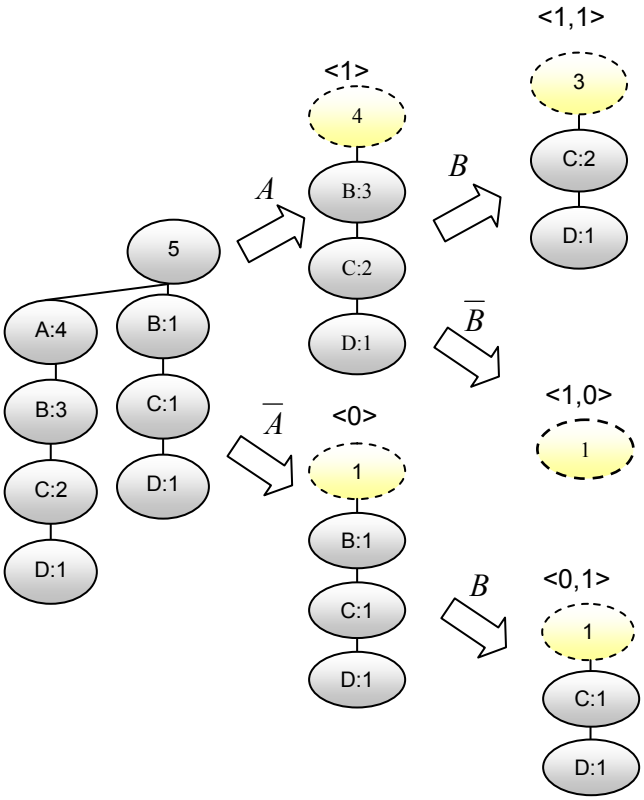


Figure 13. The constructing process of the required conditional AFP-trees of AB

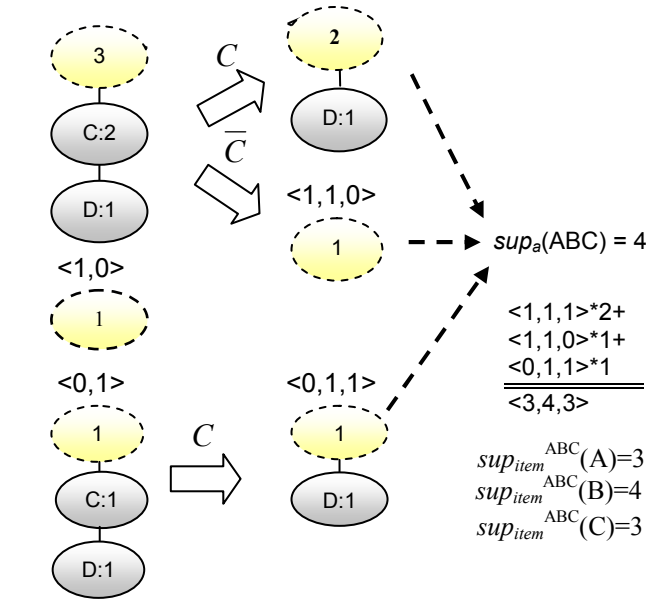


Figure 14. The constructing process of the required conditional AFP-trees of ABC.

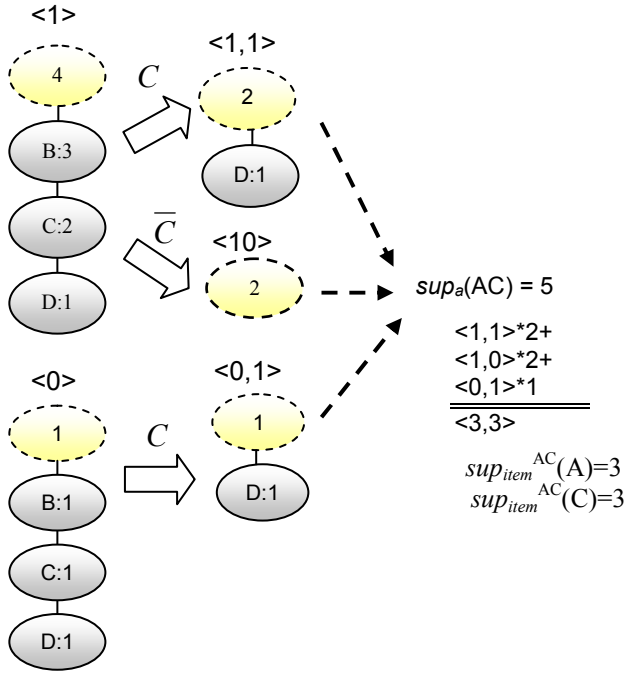


Figure 15. The constructing process of the required conditional AFP-trees of AC.

generated with μ_p=0. Consequently, by performing a projection on A from the original AFP-tree, the A-conditional AFP-tree, whose id vector is <1>, is obtained as shown in Fig. 12. According to the count value stored in the root node of the A-conditional AFP-tree, $sup_a(A)=|T_e^A(0)|= |PB_e^A(0)|=4$; A is an approximate frequent itemset. In the A-conditional header table, B is the first item with support larger than or equal to $|TDB_2|\times\alpha$, so P is set to be the next generated candidate itemset {A, B}. The maximum row error allowed by {A, B} is different from the error allowed by {A}. Thus, the <case 2> for constructing the conditional AFP-Trees of the candidate itemset is performed. In order to get $sup_a(AB) = |T_e^{AB}(0)|+|T_e^{AB}(1)| = |PB_e^{AB}(0)|+|PB_e^{AB}(1)|$, a complement projection on the original AFP-tree with respect to A is performed to get the conditional AFP-tree corresponding to $PB_e^A(1)$ which is identified by vector <0> in Fig. 12. From the conditional AFP-tree identified by vector <1>, a projection with respect on B is performed to get the AFP-tree identified by vector <1,1> which corresponds to $PB_e^{AB}(0)$. On the other side, the conditional AFP-trees with vectors <1,0> and <0,1>, which correspond to $PB_e^{AB}(1)$, are obtained from the AFP-trees with vectors <1> and <0> by performing a complement projection and a projection with respect to B, respectively. By adding the count values in the root nodes of these three AFP-trees, the value of $sup_a(AB)$, 5, is obtained. By summarizing the scalar product of the id vector and the count value stored in the root node of the three conditional AFP-trees: <1,1>×3 +<1,0>×1+<0,1>×1=<4,4>, $sup_{item}^{AB}(A)=4$ and $sup_{item}^{AB}(B)=4$ are obtained. Consequently, AB is discovered to be an approximate frequent itemset.

Next, *P'* is set to be AB and the **Pattern_growth()** procedure is performed recursively. Item C is the first item in the conditional header table of AB with support larger than or equal to $|TDB_2|\times\alpha$. Therefore, the next generated candidate itemset *P* is {A, B, C}. As shown in Fig. 13, the conditional AFP-trees required by candidate itemset {A, B, C} are constructed from the ones corresponding to {A, B}. The conditional AFP-trees with id vector <1,1,1>, as shown in Fig. 13, corresponds to $PB_e^{ABC}(0)$. Moreover, the conditional AFP-trees with id vectors <1,1,0> and <0,1,1> corresponds to $PB_e^{ABC}(1)$. According to the obtained result, ABC is an approximate frequent itemset.

When trying to generate a longer candidate from itemset ABC, D is the only item in the conditional header table of ABC. However, the support of D in the conditional header table of ABC is less than $|TDB_2|\times\alpha$. As a result, the process returns back to generate another longer pattern from AB. Since the support of D in the conditional header table of AB is less than $|TDB_2|\times\alpha$, the process returns back to append item C to pattern A for generating candidate itemset AC. The conditional AFP-trees correspond to $PB_e^{AC}(0)$ and $PB_e^{AC}(1)$ are constructed as shown in Fig. 14. According to the obtained value of sup_a(AC), that is 5, AC is an approximate frequent itemset.

Since the supports of D in the conditional header table of AC and A are both less than $|TDB_2|\times\alpha$, the process of discovering the approximate frequent itemsets starting with prefix A is finished. The same process is performed repeatedly to discover the approximate frequent itemsets starting with other items.

V. PERFORMANCE EVALUATION

A systematic study is performed to evaluate the performance efficiency of the FP-AFI algorithm. In the experiments, the FP-AFI algorithm is compared with two related works: the FT-Apriori[6] and AFI algorithm[2]. The FT-Apriori[6] and AFI algorithms[2] were proposed previously for mining the approximate frequent itemsets, which do not need to satisfy the requirement of core patterns. For getting the same result with the FP-AFI algorithm, these two algorithms are modified to include the pruning strategy according to the threshold value of a core pattern factor. All of these algorithms were implemented using Microsoft Visual C++ 6.0. The experiments have been performed on a 3.4GHz Intel Pentium IV machine with 2G megabytes main memory and running Microsoft XP Professional. Moreover, the data sets are generated from the IBM data generator [1].

In each experiment, one of the setting among the given threshold values at run time and the parameters of the data generator is varied to observe the changing trend of execution time. A dataset T10I50D20K [1], which is generated with 20K transactions, 50 distinct items and an average of 10 items per transaction, is used in the first part of experiments.

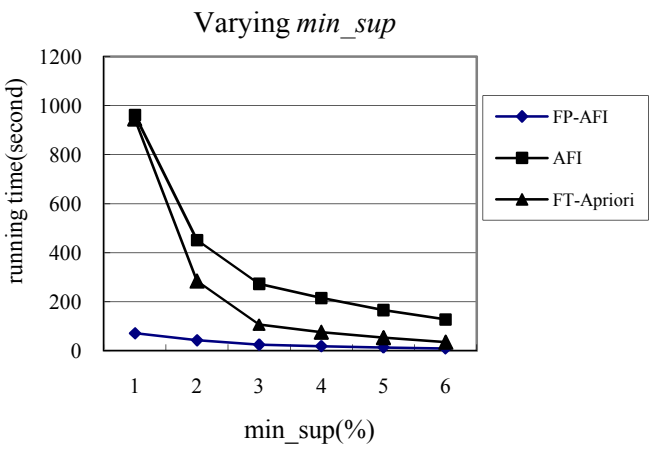
Fig. 15(a) shows the running time of the three algorithms by varying *min_sup* with ε_r=0.2, ε_c=0.5, and α=0.8. As shown in the figure, FP_AFI runs much faster than both AFI and FP-Apriori especially when *min_sup* is smaller. As *min_sup* decreases, more candidates are generated in the process of approximate itemset mining such that the computation time increases. Nevertheless, the execution time of the FP-AFI algorithm is shown to be very efficient when *min_sup* is set to as low as 0.01, which is about 1/13 of the ones of the AFI and FT-Apriori. The reason is that the FP-AFI avoids illustrating the candidate itemsets exhaustively.

Fig. 15(b) presents the run-time performance of the algorithms by varying the core pattern factor α with *min_sup* =0.02, ε_r=0.2, and ε_c=0.5. The number of the set of core patterns becomes larger as α decreases. Therefore, more candidates are generated in the process of approximate frequent itemsets mining; the execution time also increases. Nevertheless, the FP-AFI algorithm is shown to be much more efficient than the other two algorithms when α is set to as low as 0.3. When α becomes larger, the number of candidates satisfying the constraint of the core pattern factor α will reduce. By adopting the anti-monotone property of support to prune candidate itemsets, FT-Apriori is better than AFI when α is set to as high as 0.5.

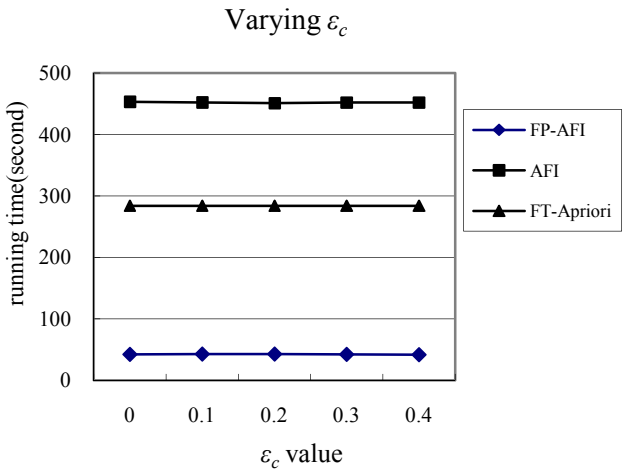
We also tested the scalability of the algorithms as the parameter ε_r is varied with *min_sup* =0.02, α=0.8, and ε_c=0.5. The result is shown in Fig. 15(c). When the setting of ε_r becomes larger, a larger maximum row error is allowed when checking whether a transaction FT-contains a transaction. Accordingly, more approximate frequent itemsets are discovered. As shown in Fig. 15(c), the running time of AFI increases very quickly as ε_r increases while the efficiency of both the FT-Apriori and FP-AFI is not affected much by ε_r. When the maximum row error changes frequently as the length of candidate itemsets glow, more cost is required in the FP-AFI to perform the back-and-forward process for constructing the required conditional AFP-trees. That is the reason why the execution time of the FP-AFI increases slightly from ε_r =0.3. However, the running time of the FP-AFI is still less than FT-Apriori when ε_r is set to as high as 0.6.

The running time of the three algorithms by varying ε_c with *min_sup* =0.02, α=0.8, and ε_r=0.2 is shown in Fig. 15(d). The result indicates that the effect of changing ε_c on the running time of the three algorithms is nearly unnoticed.

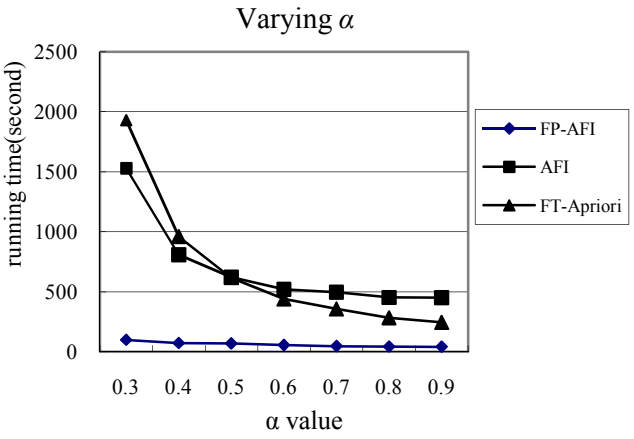
On the other hand, the scalability tests by varying the size and the number of distinct items in the transaction database are performed. With 50 distinct items and an average of 10 items per transaction when generating the transaction dataset, Fig. 15(e) shows the running time of the three algorithms by varying the number of transactions, where *min_sup* =0.02, ε_r =0.2, ε_c = 0.5 and α= 0.8 are used at rum time. All the running time of the three algorithms increases as the number of transactions increases. The running time of the FT-Apriori algorithm increase most significantly because the FT-Apriori suffers from the disadvantage of repeatedly scanning the data set. Although it is unnecessary for the AFI to scan the database repeatedly, the cost of performing the union or intersection operations on the transaction id lists also increases. The reason is that the transaction id lists used in the AFI algorithm will become longer as the number of transactions increase. On the other hand, the AFP-tree structure provides a compact representation of the transaction database which helps to



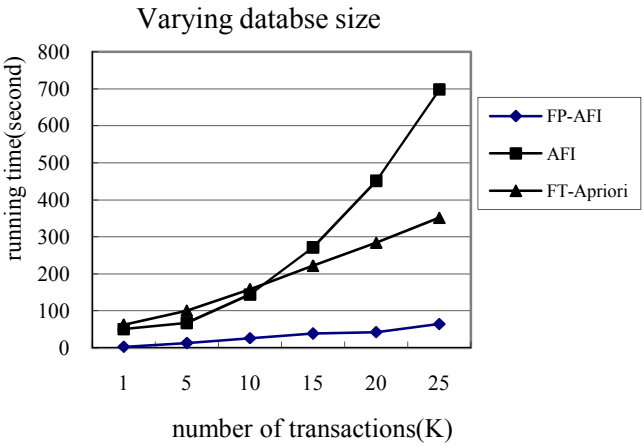
(a)



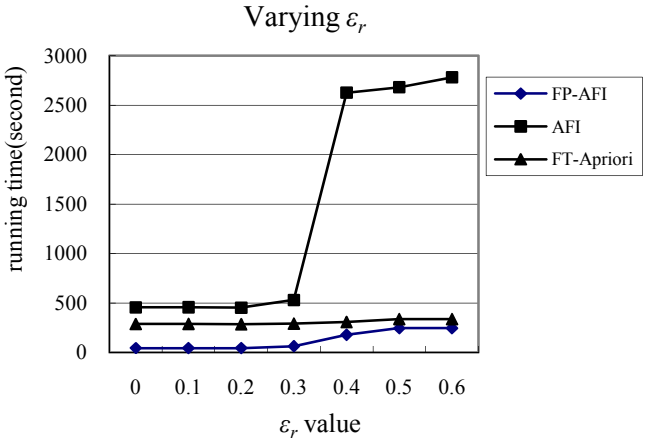
(d)



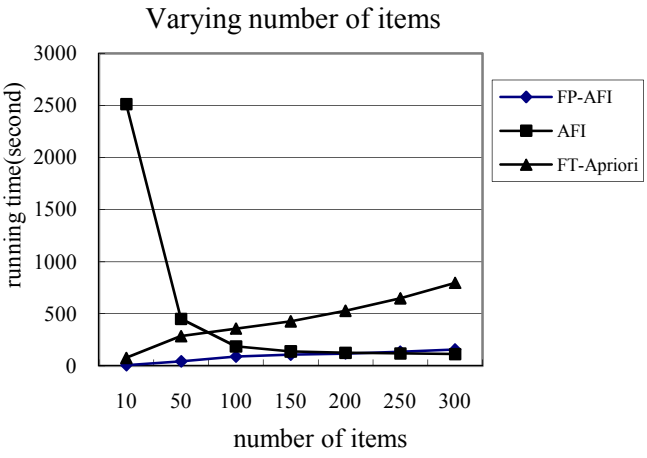
(b)



(e)



(c)



(f)

Figure 16. The results of performance evaluation.

efficiently generate candidate itemsets and perform approximate support counting. Therefore, the execution efficiency of the FP-AFI is much better than the other two algorithms when the size of the database is set to as high as 25K.

With an average of 10 items per transaction, the number of distinct items in the database is varied to generate 10 transaction databases of size 20K. The running time of the three algorithms is shown in Fig. 15(f). When the number of distinct items in the database increases, the database becomes sparse. The case is good for the AFI. Therefore, the AFI becomes the most efficient one when the number of distinct items is set to be 250. For the FP-AFI algorithm, the constructed AFP-tree becomes bushier when the number of distinct items in the database increases. Consequently, the performance of FP-AFI degrades slightly because it requires more time to traverse the AFP-tree and construct the conditional AFP-trees. However, the running time of the FP-AFI keeps comparable with the AFI when number of distinct items is set as high as 300.

VI. CONCLUSION

In this paper, the FP-AFI algorithm is developed to discover approximate frequent itemsets which inherits the advantages of the FP-tree based mining algorithms. We extended the FP-tree structure to be the AFP-tree for storing compressed and crucial information in transactions about approximate frequent itemsets. By deriving the recursive relationship between the set of supporting transactions of an itemset and the one of its prefix, two pseudo-projection operations on AFP-trees are provided to obtain the conditional AFP-trees of an itemset systematically. Consequently, the approximate support of a candidate itemset and the item supports of each item in the candidate are obtained easily from the conditional AFP-trees. As a result, the constraint test of a candidate itemset is performed efficiently without additional database scan. The experimental results show that the FP-AFI algorithm performs much better than the FP-Apriori and AFI algorithms in efficiency especially when the size of data set is large and the minimum threshold of approximate support is small. Moreover, the execution time of FP-AFI algorithm is scalable even when the error threshold parameters become large.

The FP-tree-like structure is suitable for maintaining the transactions within a sliding window in a data stream environment. Therefore, to apply the FP-AFI algorithm for mining recently approximate frequent itemsets in data stream provides a good solution for this problem, which is under our investigation currently.

REFERENCES

[1] R. Agrawal and R. Srikant, "Fast Algorithm for Mining Association Rule in Large Databases," in Proc. of the 20th International Conference on Very Large Data Bases, 1994.

[2] M. S. Chen, J. Han and P. S. Yu, "Data Mining: An Overview from a Database Perspective," in Proc. of the IEEE Transactions on Knowledge and Data Engineering, Volume 8(6): pages 866-883, 1996.

[3] H. Cheng, P. S. Yu and J. Han, "AC-Close: Efficiently Mining Approximate Closed Itemsets by Core Pattern Recovery," in Proc. of the 6th IEEE International Conference on Data Mining (ICDM 2006), 2006.

[4] H. Cheng, P. S. Yu and J. Han, "Approximate Frequent Itemset Mining In the Presence of Random Noise," Soft Computing for Knowledge Discovery and Data Mining. Oded Maimon and Lior Rokach. Springer, pages 363-389, 2008.

[5] R. Gupta, G. Fang, B. Field, M. Steinbach and V. Kumar, "Quantitative evaluation of approximate frequent pattern mining algorithms," in Proc. of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2008), 2008.

[6] J. Han, J. Pei and Y. Yin, "Mining Frequent Patterns without Candidate Generation, " in Proc. of the 2000 ACM SIGMOD International Conference on Management of Data, 2000.

[7] IBM inc. http://www.almaden.ibm.com/cs/quest/syndata.html.

[8] J. L. Koh and P. W. Yo, "An Efficient Approach for Mining Fault-Tolerant Frequent Patterns Based on Bit Vector Representations," in Proc. of the 10th International Conference, Database Systems for Advanced Applications, (DASFAA 2005), 2005.

[9] J. Liu, S. Paulsen, W.Wang, A. Nobel, and J. Prins, "Mining Approximate Frequent Itemsets from Noisy Data," in *ICDM*, pages 721–724, 2005. 721–724, 2005.

[10] J. Liu, S. Paulsen, X. Sun, W. Wang, A. Nobel and J. Prins, "Mining approximate frequent itemsets in the presence of noise: Algorithm and analysis," in Proc. of the 6th SIAM International Conference on Data Mining (SDM 2006), 2006.

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

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

[13] J. K. Seppänen and H. Mannila, "Dense itemsets," in Proc. of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2004), 2004.

[14] UCI machine learning repository. (http://www.ics.uci.edu/mlearn/ML Summary.html)

[15] K. Wang, L. Tang, J. Han and J. Liu, "Top Down FP-Growth for Association Rule Mining," in Proc. of Advances in Knowledge Discovery and Data Mining, 6th Pacific-Asia Conference (PAKDD 2002), 2002.

[16] C. Yang, U. M. Fayyad and P. S. Bradley, "Efficient discovery of error-tolerant frequent itemsets in high dimensions," in Proc. of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2001), 2001.Minin Min Mining (KDD 2001), 2001.