IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received November 11, 2018, accepted December 5, 2018, date of publication December 19, 2018,
date of current version January 29, 2019.

Digital Object Identifier 10.1109/ACCESS.2018.2888627

TT-Miner: Topology-Transaction Miner for Mining Closed Itemset

BO LI[1], ZHENG PEI[2], KEYUN QIN[3], AND MINGMING KONG[2]
[1]School of Information Science and Technology, Southwest Jiaotong University, Chengdu 611756, China
[2]School of Computer and Software Engineering, Xihua University, Chengdu 610039, China
[3]School of Mathematics, Southwest Jiaotong University, Chengdu 611756, China

Corresponding author: Zheng Pei (pqyz@263.net)

This work was supported in part by the National Nature Science Foundation of China under Grant 61372187, Grant 61473239, and Grant 61702317 and in part by the Scientific and Technological Project of Sichuan Province under Grant 2018GZ0256.

ABSTRACT Mining frequent closed itemsets (FCIs) from transaction databases is a fundamental problem in many data mining applications. All the enumeration algorithms enumerate FCIs by adding a singleton item to an itemset and then checking whether it is closure. In order to reduce enumerations, we enumerate FCIs by adding an itemset to the existing FCI itemset. To this end, we first analyze a binary relation on the set of itemsets in transaction databases, show co-occurrence relation among items, and prove that the relation is reflexive and transitive. Next, we use the relation to construct a topology for the set of itemsets and prove that all FCIs are included in the topology. Then, we construct a topology-transaction tree (TT-tree) and provide a topology-transaction miner (TT-Miner) algorithm for enumerating FCIs in the TT-tree. Finally, an extensive experimental evaluation on a number of real and synthetic databases shows that the TT-Miner is an efficient and scalable algorithm compared with the previous methods.

INDEX TERMS Data mining (DM), frequent closed Itemsets (FCIs).

I. INTRODUCTION

Mining frequent patterns or itemsets [1] is a popular data mining task that is a fundamental and essential to a wide range of data mining applications. These applications include the discovery of correlation or causal relations [2]–[4], sequential patterns [5]–[8], multidimensional patterns or association rules [9]–[20], *etc*. The task can be described as: Given a transaction database and a user-specified minimum support, find all itemsets that occur in (frequency) no less than the minimum support. Those itemsets are called frequent itemsets (FIs), and the minimum support (*minSup*) is a percentage of the database.

The early and common way to traverse the nodes of a tree is breadth-first. The most famous and earliest implementation of FIs mining in this way is Apriori by Agrawal *et al.* [1] which is a bottom-up, breadth-first search algorithm testing candidates by scanning database many times. It uses the downward closure property to prune unfrequent itemsets, if an itemset is frequent, all its subsets must also be frequent. It uses a prefix tree data structure built level by level. Each new level is constructed by creating a child node for every frequent item set on the upper level. The support of each candidate is counted by access the transaction database. Most of the proposed FIs mining algorithms are apriori-style algorithm. They have a good performance for short FIs minging, where the transaction databases are sparse such as supermarket sale database. But they have not a satisfactory performance for long FIs mining, where the transaction databases are dense such as gene database.

Afterwards, depth-first search is popular and more efficient, known from algorithms such as Eclat [21]. Eclat is a famous algorithm best described with the concept of an enumeration tree, for the wide variation in the different strategies used by it. It first proposed prefix-based equivalence classes as a means of independent sub-lattice mining. After partitioning the candidate set into disjoint groups, Eclat uses a candidate partitioning approach, intersects tid lists recursively. There are many variations of Ecalt algorithms have been proposed [22].

Other pattern-growth mining methods proposed along the years, such as FP-growth, proposed by Han *et al.* [23] is a depth-first, partitioning-based, divide-and-conquer search algorithm and uses FP-trees to store frequency information of the original database. Its advantage is that frequent itemsets are generated in a compressed form by just scanning database two times, and it performs well.

It uses relatively complicated data structures for searching the FP-tree.

A major problem in applications of FIs mining is that the volume of the FIs is often extremely large. A particularly interesting form of concise representation is that of closed itemsets [24]. A frequent closed itemset (FCI) is included in no other frequent itemset included in the same transactions. Closed sets are lossless in the sense that they can be used to uniquely determine the set of all frequent itemsets and their exact frequency, which are suitable for generating rules [25].

There are several famous and efficient FCIs mining algorithms have been proposed. CHARM [25] is a FCIs mining algorithm, using the vertical representation of databases and *diffset* for efficient closure checking operation and fast support computing. It simultaneously searches both the itemset space and transaction space over a novel IT-tree search space, and uses an efficient hybrid search methods to skip many unnecessary enumerations.

FPclose [11] is a variant and improvement of FP-growth on FCIs mining. A significant part of mining time in FP-growth is spent on visiting the tree. FPclose uses a special data structure FP-array to greatly improve the need to traverse on FP-tree. If the input database is sparse, the array based technique performs well, for the array can save the traversal time. If the input database is dense, the tree based representation is more compact. FPclose has a mechanism to identify the database is sparse or dense in order to select proper data structure. FPclose counts the number of nodes in each level of the tree. If the upper quarter of the tree contains more than 15% of the total number of nodes, the database is likely sparse. Then, FPclose uses an array in the next level of mining [22]. Otherwise, it still use tree base representation. It also uses a CFI-tree for closure checking of frequent itemsets efficiently.

LCM [26] uses a purely vertical and a purely horizontal representation data structure in parallel. Its processing scheme is like a variant of the Eclat algorithm, but use a new technology named *Occurrence Deliver* to replace the intersection of transaction index lists. LCM constructs the conditional transaction database only to read memory linearly and stores the transaction indices through direct access. It proposes prefix-preserving closure extension, which combines tail-extension and closure operation to enumerate frequent closed itemsets directly [27].

All the enumeration algorithms so far, enumerate FCIs by adding a singleton item to an itemset and then checking whether it is closure. In other words, the length of an closed itemset is k, which means the generation process contains k enumerations except failure enumerations. Can we reduce the number of enumerations of a FCI to less than the length of it ? In other words, can we enumerate FCIs by adding an itemset to a known FCI itemset ? The answer is yes, if we take advantage of co-occurrence relation among items, see Fig. 1. Theoretically, the powerset of items set naturally forms a powerset lattice, also called as the itemset lattice. In the itemset lattice, items a and b are frequent items, but $\{a, b\}$ may be unfrequent, hence using frequent items to generate

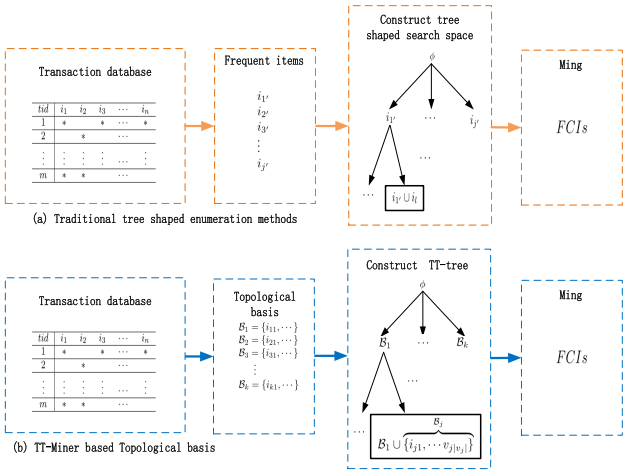


FIGURE 1. Enumeration methods comparison: (a) Traditional methods by adding one item to a itemset at a time; (b) The proposed TT-Miner by adding a itemset (*topologicalbasis*) to a itemset at a time, $k \leq j'$, $|\mathcal{B}_j| \geq 1$.

FIs directly may be not very efficient. From the point of view of algebras, FIs are naturally used to obtain a relation R on the set of items I, *i.e.*, item a is in the relation R to item b if there exists a frequent itemset contains a and b. In other words, can we construct a more compressed relation on the set of items I, and the relation allows us to use several frequent items to generate FIs at a time. In our earlier paper [28], we propose a novel method to generate the formal concept lattice, which is based on a topology for attributes of a formal context. The topology for attributes is induced by a reflexive and transitive relation on the set of attributes. By defining an equivalent relation on the topology for attributes, we prove that the formal concept lattice and the quotient topology for attributes decided by the equivalent relation is isomorphic. In this paper, we analyze the reflexive and transitive relation on the set of items I, and construct the topology for items of transaction databases to explain co-occurrence relation among items. It is more important that we propose topology-transaction tree (TT-tree) to fast construct the topology for I, and obtain FCIs from TT-tree.

Compared with enumerating many possible subsets by singleton items in the existed methods, generating FIs, FCIs based on the base $\mathcal{B}_{\mathcal{I}}$ of the topology $T_{\mathcal{I}}$ can reduce the search space, save running time and the size of memory, efficiently.

A. CONTRIBUTIONS

We introduce Topology-Transaction Miner (TT-Miner), an efficient algorithm for enumerating all the frequent closed itemsets. The major contributions of the paper are summarized as follows:

1) We reveal a co-occurrence relation between items, described by *topologicalbasis*.
2) We use the set of *topologicalbasis* and the set of transactions to construct a novel TT-tree search space, which is condensed and contains the set of all FCIs.

3) TT-Miner mines FCIs in TT-tree, which enumerates FCIs by adding itemsets (*topologicalbasis*) to pre-existing itemsets.

The organization of this paper is as follows: In Section II, prepares the basic notion and definitions. In Section III, we analyze the reflexive and transitive relation on the set of items, and use the relation to construct the topology for the set of items, and prove that FCIs are contained in the topology. Then we use the topology and transaction pairs to construct TT-tree structure. In Section IV, we provide the design and implementation of TT-Miner algorithm, and explain some popular techniques used for reducing computation time. In Section V, we select various databases to show TT-Miner for mining FCIs and compare TT-tree method with the existed method. We conclude the paper in Section VI.

II. PRELIMINARIES

A. FREQUENT ITEMSETS MINING

Mining frequent itemsets (FIs) can be explained as following [1], [25], [26]: Let $\mathcal{I} = \{i_1, i_2, \cdots, i_m\}$ be a set of m items, the database $\mathcal{D}$ be a set of n transactions, each one identified by its unique *tid* and contains a set of items. The set of all tids of $\mathcal{D}$ is denoted as $\mathcal{T} = \{t_1, t_2, \cdots, t_n\}$. The transaction database $\mathcal{D}$ is noted as $\mathcal{D} = (\mathcal{T}, \mathcal{I})$. The set $I \subseteq \mathcal{I}$ is also called an itemset and a set $T \subseteq \mathcal{T}$ is called a tidset. If $\|I\| = k$, then I is called a *k-itemset*. Table 1 shows an example of a transaction database $\mathcal{D}$. There are five items, $\mathcal{I} = \{i_1, i_2, i_3, i_4, i_5\}$, and six transactions, $\mathcal{T} = \{t_1, t_2, t_3, t_4, t_5, t_6\}$. Each tid denotes a transaction, which is determined by a subset of $\mathcal{I}$, *i.e.*, the first transaction is t_1, which is determined by $\{i_1, i_5\} \subseteq \mathcal{I}$.

For an itemset I, we denote its corresponding tidset as $t(I)$, which is the set of all *tids* of transactions that contain I as a subset, *i.e.*, in Table 1, $t(\{i_1\}) = \{t_1, t_2, t_5\}$, $t(\{i_1, i_2\}) = \{t_5\}$. For a tidset T, we denote its corresponding itemsets as $i(T)$, which is the set of items common to all the transactions with *tids* in T, *e.g.*, $i(\{t_1\}) = \{i_1, i_5\}$, $i(\{t_1, t_2\}) = \{i_1\}$. An itemset I is contained in a transaction t', $t' \in \mathcal{T}$, if $I \subseteq i(t')$, and t' is called an occurrence of I. We note that,

$$t(I) = \bigcap_{i \in I} t(i). \tag{1}$$

$$i(T) = \bigcap_{t \in T} i(t). \tag{2}$$

The support of an itemset I , denoted by $\sigma(I)$, is the percentage of transaction in $\mathcal{D}$ containing I, *i.e.*, $\sigma(I) = |t(I)|/n$, n is the volume of transactions. Given the user defined minimum support *minSup*. If $\sigma(I) \geq minSup$, then itemset I is frequent. An frequent itemset I is closed if there exists not proper superset $I' \supset I$ with $\sigma(I') = \sigma(I)$. Alternatively, an frequent itemset I is closed, if and only if $i(t(I)) = I$ [29] .As an example in Table 1, $\sigma(\{i_1\}) = |t(\{i_1\})|/6 = 3/6 = 0.5$, $\sigma(\{i_3, i_4\}) = |t(\{i_3, i_4\})|/6 = 1/3$, $\sigma(\{i_3, i_4, i_5\}) = 1/3$. If $minSup = 1/3$, then $\{i_1\}$, $\{i_3, i_4\}$ and $\{i_3, i_4, i_5\}$ are frequent. $i(t(\{i_3, i_4, i_5\})) = i(\{t_2, t_3, t_4, t_5, t_6\} \cap \{t_5, t_6\} \cap \{t_1, t_3, t_5, t_6\}) = i(\{t_5, t_6\}) = \{i_1, i_2, i_3, i_4, i_5\} \cap \{i_3, i_4, i_5\} = \{i_3, i_4, i_5\}$,$i(t(\{i_3, i_4\})) = \{i_3, i_4, i_5\}$. So, $\{i_3, i_4, i_5\}$ is closed, and $\{i_3, i_4\}$ is not closed.

All the existing methods enumerate itemsets by adding singleton items to pre-existing itemsets, such as Apriori, FP-tree, CHARM and LCM. Our method is quite different from others, for TT-tree is constructed by elements of the *topologicalbasis*, which are itemsets itself.

TABLE 1. A transaction database.

tid	*items*				
t_1	i_1				i_5
t_2	i_1		i_3		
t_3			i_3		i_5
t_4		i_2	i_3		
t_5	i_1	i_2	i_3	i_4	i_5
t_6			i_3	i_4	i_5

III. TOPOLOGY-TRANSACTION SEARCH TREE

In this section, we first give some formal expressions on transactions and items, then we induce a binary relation on the set of items and prove that the relation is reflexive and transitive, more important, we prove all the FCIs are included in the topology, which is generated by the relation on the set of items. Finally, we give a algorithm to calculate the *topologicalbasis*.

A. THE REFLEXIVE AND TRANSITIVE RELATION ON THE SET OF ITEMS

Intuitively, $i(T)$ is the maximum itemset contained in each transaction of T, *e.g.*, $i(\{t_4, t_5\}) = \{i_2, i_3\} \cap \{i_1, i_2, i_3, i_4, i_5\} = \{i_2, i_3\}$ in Table 1, itemset $\{i_2, i_3\}$ is the maximum itemset contained in t_1 and t_2. Based on Eqs.(1) and (2), we can easily obtain the following properties about $t(I)$ and $i(T)$ [28].

Property 1: In $\mathcal{D} = (\mathcal{T}, \mathcal{I})$, let itemsets I_1 and I_2 of $\mathcal{I}$, transaction tid subsets T_1 and T_2 of $\mathcal{T}$. We have

1) If $I_1 \subseteq I_2$, then $t(I_1) \supseteq t(I_2)$;
2) If $T_1 \subseteq T_2$, then $i(T_1) \supseteq i(T_2)$;
3) $t(I_1 \cup I_2) = t(I_1) \cap t(I_2)$;
4) $i(T_1 \cup T_2) = i(T_1) \cap i(T_2)$;
5) $t(i(t(I_1))) = t(I_1)$;
6) $i(t(i(T_1))) = i(T_1)$.

Definition 1: For any transaction database $\mathcal{D} = (\mathcal{T}, \mathcal{I})$ and each item $i_k \in \mathcal{I}$, the association items set of i_k is

$$C(i_k) = \{i_{k'} | \forall i_{k'} \in \mathcal{I}, i_{k'} \in i(t(i_k))\} = \bigcap_{\forall t_k \in \mathcal{T}}^{i_k \in i(t_k)} i(t_k). \tag{3}$$

From the algebraic point of view, $C(i_k)$ is the maximal itemset which are contained in those transactions containing item i_k, *e.g.*, in Table 1, $C(i_1) = \bigcap_{\forall t_k \in \mathcal{T}}^{i_1 \in i(t_k)} i(t_k) = i(t_1) \cap i(t_2) \cap i(t_5) = \{i_1, i_5\} \cap \{i_1, i_3\} \cap \{i_1, i_2, i_3, i_4, i_5\} = \{i_1\}$, $C(i_2) = \{i_2, i_3\}$, $C(i_3) = \{i_3\}$, $C(i_4) = \{i_3, i_4, i_5\}$ and $C(i_5) = \{i_5\}$. Naturally, for each itemset I_k of $\mathcal{I}$, the association items set

of I_k is determined by

$$C(I_k) = \bigcap_{\forall t_i \in \mathcal{T}}^{I_k \subseteq i(t_i)} i(t_i).$$

Property 2: For any transaction database $\mathcal{D} = (\mathcal{T}, \mathcal{I})$ and $i_j \in \mathcal{I}$, if $i_k \in C(i_j)$, then $C(i_k) \subseteq C(i_j)$.

Proof: According to Eq.(3), if $i_k \in C(i_j)$, then

$$i_k \in C(i_j) = \bigcap_{\forall t_i \in \mathcal{T}}^{i_j \in i(t_i)} i(t_i),$$

this means that item i_k is contained in transactions containing item i_j, or transactions that contain item i_j must contain item i_k, hence,

$$C(i_k) = \bigcap_{\forall t_i \in \mathcal{T}}^{i_k \in i(t_i)} i(t_i) \subseteq \bigcap_{\forall t_i \in \mathcal{T}}^{i_j \in i(t_i)} i(t_i) = C(i_j).$$

■

Definition 2: For any transaction database $\mathcal{D} = (\mathcal{T}, \mathcal{I})$, a binary relation on $\mathcal{I}$ is called as the association relation on $\mathcal{I}$ if for any i_j and i_k in $\mathcal{I}$,

$$R_{\mathcal{I}}(i_j, i_k) = \begin{cases} 1, & \text{if} \quad i_k \in C(i_j), \\ 0, & \text{if} \quad i_k \notin C(i_j). \end{cases} \tag{4}$$

Intuitively, the association relation can be understood as co-occurrence relation among items, *e.g.*, the association relation on $\mathcal{I}$ of Table 1 is shown in Table 2, $R_{\mathcal{I}}(i_2, i_3) = 1$ means that transactions containing i_2 must contain item i_3, hence we further confirm that the support of i_3 is not less than the support of i_2, or if i_2 is a frequent item, then i_3 must be a frequent item, such information is useful to analyze FIs and FCIs.

TABLE 2. The association relation on $\mathcal{I}$ of Table 1.

$R_{\mathcal{I}}$	i_1	i_2	i_3	i_4	i_5
i_1	1	0	0	0	0
i_2	0	1	1	0	0
i_3	0	0	1	0	0
i_4	0	0	1	1	1
i_5	0	0	0	0	1

Property 3: For any transaction database $\mathcal{D} = (\mathcal{T}, \mathcal{I})$, the association relation $R_{\mathcal{I}}$ on $\mathcal{I}$ is such that

1) Reflexivity: $\forall i_j \in \mathcal{I}, R_{\mathcal{I}}(i_j, i_j) = 1$;
2) Transitivity: $\forall i_j, i_k, i_l \in \mathcal{I}$, if $R_{\mathcal{I}}(i_j, i_k) = 1$ and $R_{\mathcal{I}}(i_k, i_l) = 1$, then $R_{\mathcal{I}}(i_j, i_l) = 1$.

Proof: Reflexivity is obvious. We only prove transitivity, according to Property 2 and Eq.(4), if $R_{\mathcal{I}}(i_j, i_k) = 1$ and $R_{\mathcal{I}}(i_k, i_l) = 1$, then $i_l \in C(i_k) \subseteq C(i_j)$, hence $R_{\mathcal{I}}(i_j, i_l) = 1$. ■

In Table 2, we notice that $R_{\mathcal{I}}$ may be not symmetry, *e.g.*, $R_{\mathcal{I}}(i_2, i_3) = 1$ but $R_{\mathcal{I}}(i_3, i_2) = 0$, this means that $R_{\mathcal{I}}$ is not necessary an equivalence relation on $\mathcal{I}$.

B. THE TOPOLOGY FOR THE SET OF ITEMS

In this section, we construct the topology for the set of items based the association relation $R_{\mathcal{I}}$ on $\mathcal{I}$ and prove that FCIs are contained in the topology, then we provide Algorithm 1 to calculate the *topologicalbasis* $\mathcal{B}_{\mathcal{I}}$.

Algorithm 1 *Calc-Topologicalbasis*($\mathcal{D}$, *minSup*)

Input: $\mathcal{D} = (\mathcal{T}, \mathcal{I})$, *minSup*.
Output: $\mathcal{B}_{\mathcal{I}}$.
Method:
1) Read $\mathcal{D}$, save it as horizontal format H and vertical format V, where $|H| = n$, $|V| = m$.
2) **for** $i := 1 : m$ **do**
3) **if** $|V[i]|/n \geq minSup$ **do**
4) $C(i) = H[V[i, 1]]$
5) **for** $j := 2 : |V_i|$ **do**
6) $C(i) := C(i) \cap H[V[i, j]]$
7) **end**
8) insert $C(i)$ to $\mathcal{B}_{\mathcal{I}}$
9) **end for**
10) **sort** $\mathcal{B}_{\mathcal{I}}$ with the descending order of $|C(i)|$
11) return $\mathcal{B}_{\mathcal{I}}$

The binary relation, especially the equivalence relation, has been widely used in rough set theory [30]–[39]. In [28], we proved that a reflexive and transitive relation on $\mathcal{I}$ can be used to determine lower approximation of subset of $\mathcal{I}$ and construct a topology for $\mathcal{I}$. According to Property 3, the association relation $R_{\mathcal{I}}$ on $\mathcal{I}$ can be used to construct a topology for $\mathcal{I}$, *i.e.*, for any itemset $I \subseteq \mathcal{I}$, $R_{\mathcal{I}}$ can be used to determine lower approximation of I as follows:

$$\begin{aligned} \underline{R_{\mathcal{I}}}(I) &= \{i_j | \forall i_j, i_k \in \mathcal{I}, R_{\mathcal{I}}(i_j, i_k) = 1 \text{ implies } i_k \in I\} \\ &= \{i_j | \forall i_j \in \mathcal{I}, C(i_j) \subseteq I\}. \end{aligned}$$

Theorem 1 [28]: For any transaction database $\mathcal{D} = (\mathcal{T}, \mathcal{I})$, 1) $T_{\mathcal{I}} = \{\underline{R_{\mathcal{I}}}(I) | \forall I \subseteq \mathcal{I}\}$ is a topology for $\mathcal{I}$;
2) $\mathcal{B}_{\mathcal{I}} = \{C(i) | \forall i \in \mathcal{I}\}$ is a base for the topology $T_{\mathcal{I}}$.

1) and 2) in Theorem 1 means that the topology $T_{\mathcal{I}}$ for $\mathcal{I}$ can be generated by the base $\mathcal{B}_{\mathcal{I}}$, *i.e.*, for any $\underline{R_{\mathcal{I}}}(\mathcal{I}') \in T_{\mathcal{I}}$, $\underline{R_{\mathcal{I}}}(\mathcal{I}') = \cup_{i_j \in \mathcal{I}'} C(i_j)$, *e.g.*, according to Table 2, Table 3 is a *topologicalbasis* for the topology, the topology for $\mathcal{I}$ is $T_{\mathcal{I}} = \{\emptyset, \{i_1\}, \{i_3\}, \{i_5\}, \{i_1, i_3\}, \{i_1, i_5\}, \{i_2, i_3\}, \{i_3, i_5\}, \{i_1, i_2, i_3\}, \{i_1, i_3, i_5\}, \{i_2, i_3, i_5\}, \{i_3, i_4, i_5\}, \{i_1, i_3, i_4, i_5\}, \{i_1, i_2, i_3, i_5\}, \{i_2, i_3, i_4, i_5\}, \{i_1, i_2, i_3, i_4, i_5\}\}$.

The pseudocode for calculating *topologicalbasis* $\mathcal{B}_{\mathcal{I}}$ appears in algorithm 1. The algorithm starts by reading transaction database $\mathcal{D}$, and saving it as horizontal format H (itemsets) and vertical format V (tidset) in Line 1. The $C(i)$ is initialized by the row of $V[i, 1]$ in H in Line 3, where $V[i, 1]$ is the first tid of the *ith* vertical list. Line 6 means $C(i)$ is calculated by set intersection operation of all the horizontal format lists corresponding to the tidset of item i. We then insert $C(i)$ to $\mathcal{B}_{\mathcal{I}}$, it should be noted that all the members of $\mathcal{B}_{\mathcal{I}}$ are identical, and $|\mathcal{B}_{\mathcal{I}}| \leq \mathcal{I}$. Then sort $\mathcal{B}_{\mathcal{I}}$ with the descending order of each *topologicalbasis*'s cardinality

in Line 10. After we return, the *topologicalbasis* $\mathcal{B}_{\mathcal{I}}$ has already been generated. As an example calculating $C(i_1)$ of i_1 in Table 1, $V_1 = \{1, 2, 5\}^T$, so $C(i_1) = H[V[1, 1]] \cap H[V[1, 2]] \cap H[V[1, 3]] = H[1] \cap H[2] \cap H[5] = \{i_1, i_5\} \cap \{i_1, i3\} \cap \{i_1, i_2, i_3, i_4, i_5\}\} = \{i_1\}$.

What's more, we find that *topologicalbasis* has some important properties as follows:

Property 4: In $\mathcal{D} = (\mathcal{T}, \mathcal{I})$, for any item $i \in \mathcal{I}$, its corresponding *topologicalbasis* is $C(i)$, the set of *topologicalbasis* is $\mathcal{B}_{\mathcal{I}}$,

1) $\forall i \in \mathcal{I}, i \in C(i)$;
2) $\forall i \in \mathcal{I}, |C(i)| \geq 1$;
3) $|\mathcal{B}_{\mathcal{I}}| \leq |\mathcal{I}|$.

1) and 2) in Property 4 show that all items are included in their corresponding *topologicalbasis*, and the volume of each *topologicalbasis* is greater than or equal to 1. It should be noted that, if item i occurrence in an itemset X and $|C(i)| > 1, i \in \mathcal{I}$, then all the items in $C(i)$ besides item i will occurrence in X, in other words, $C(i) \setminus i$ is co-occurrence with item i, *i.e.*, shown in Table 3, the *topologicalbasis* of item i_2 is $C(i_2) = \{i_2, i_3\}$, i_2 occurrences in $\{i_2, i_3\}, \{i_1, i_2, i_3\}, \{i_2, i_3, i_5\}, \{i_1, i_2, i_3, i_5\}, \{i_2, i_3, i_4, i_5\}, \{i_1, i_2, i_3, i_4, i_5\}$, then i_3 occurrence in those itemsets as well.

TABLE 3. A base for the topology $T_{\mathcal{I}}$ of Table 1.

$C(\mathcal{I})$	$\mathcal{B}_{\mathcal{I}}$
$C(i_1)$	$\{i_1\}$
$C(i_2)$	$\{i_2, i_3\}$
$C(i_3)$	$\{i_3\}$
$C(i_4)$	$\{i_3, i_4, i_5\}$
$C(i_5)$	$\{i_5\}$

3) in Property 4 shows that the *topologicalbasis* space is no bigger than the item set space. In other words, different item may have a same *topologicalbasis*, *e.g.*, in Table 1, if add a new item i_6 into $\mathcal{I}$, its *tidset* is $\{t_1, t_2, t_5\}$, then $t(i_6) = \{t_1, t_2, t_5\}$, and $C(i_6) = \{i_1, i_6\} = C(i_1)$, so item i_1 and i_6 have a same *topologicalbasis*. If two or more items have a same *topologicalbasis*, we only save one copy of it to *topologicalbasis* set $\mathcal{B}_{\mathcal{I}}$. From this point of view, this can be viewed as a attribute reduction method.

Property 5: For any transaction database $\mathcal{D} = (\mathcal{T}, \mathcal{I})$.

1) $\forall i \in I$, supports of $i \in I$ and $C(i)$ are equal, *i.e.*,
$$\frac{|t(i)|}{n} = \frac{|t(C(i))|}{n};$$
2) $\forall i_k \in C(i_j)$, the support of i_k is more than or equal to the support of i_j, *i.e.*,
$$\frac{|t(i_k)|}{n} \geq \frac{|t(i_j)|}{n};$$
3) if I is a frequent itemset such that $i \in I$, then
$$\frac{|t(I \cup C(i))|}{n} = \frac{|t(I)|}{n}.$$

According to Eqs.(1)-(3) and Property 1, the property can be easily proved. Such as in Table 1, $i_3 \in C(i_2)$ and $\frac{|t(i_3)|}{6} = \frac{|\{t_2,t_3,t_4,t_5,t_6\}|}{6} = \frac{5}{6} > \frac{1}{3} = \frac{|t(t_2)|}{6}$, if we fix $minSup = \frac{1}{3}$, i_2 is a frequent item due to $\frac{|t(i_2)|}{6} = \frac{|\{t_4,t_5\}|}{6} = \frac{1}{3} = \frac{|t(C(i_2))|}{6} = \frac{|t(\{i_2,i_3\})|}{6} = \frac{|\{t_4,t_5\}|}{6}$, then $\{i_2, i_3\}$ is also a FIs, these can help us to fast generate FIs.

To detail all FCIs are in the topology, we need an equivalence relation $\cong$ on the topology $T_{\mathcal{I}}$, the equivalence relation $\cong$ is determined by support of element of $T_{\mathcal{I}}$, *i.e.*, for any two elements I_j and I_k of $T_{\mathcal{I}}$, $I_j \cong I_k$ if and only if $t(I_j) = t(I_k)$, then we can obtain equivalent classes $T_{\mathcal{I}}/\cong$ of $T_{\mathcal{I}}$, and each equivalent class contains closed itemsets [28].

Property 6 [28]: For any transaction database $\mathcal{D} = (\mathcal{T}, \mathcal{I})$ and $[I_j] \in T_{\mathcal{I}}/\cong$, if $I_j^{max} \in [I_j]$ is such that $\forall I_j' \in [I_j]$, $I_j' \subseteq I_j^{max}$, then I_j^{max} is a closed itemset.

Intuitively, every element $[I_j] \in T_{\mathcal{I}}/\cong$ is consisted by elements of the topology $T_{\mathcal{I}}$ which are contained in the same transactions, hence all elements of $[I_j]$ own the same support $|t(I_j)|/n$, the closed itemset I_j^{max} is the maximum element of $[I_j]$ according to inclusion relation. Naturally, if I_j is frequent, then I_j^{max} is a FCIs, *e.g.*, in the topology for $\mathcal{I}$ of Table 1, $T_{\mathcal{I}}/\cong = \{\emptyset, [\{i_1\}], [\{i_3\}], [\{i_5\}], [\{i_1, i_3\}], [\{i_1, i_5\}], [\{i_2, i_3\}], [\{i_3, i_5\}], [\{i_3, i_4, i_5\}], [\{i_1, i_2, i_3\}, \{i_1, i_3, i_5\}, \{i_2, i_3, i_5\}, \{i_1, i_3, i_4, i_5\}, \{i_1, i_2, i_3, i_5\}, \{i_2, i_3, i_4, i_5\}, \{i_1, i_2, i_3, i_4, i_5\}]\}$, in which, $\{i_1\}$, $\{i_3\}$, $\{i_5\}$, $\{i_1, i_3\}$, $\{i_1, i_5\}$, $\{i_2, i_3\}$, $\{i_3, i_5\}$, $\{i_3, i_4, i_5\}$ and $\{i_1, i_2, i_3, i_4, i_5\}$ are closed itemsets.

In summary, we confirm that the topology $T_{\mathcal{I}}$ for $\mathcal{I}$ contains all FCIs, the base $\mathcal{B}_{\mathcal{I}}$ of $T_{\mathcal{I}}$ can be used to generate all of them according to Properties 5-6.

IV. TT-MINER ALGORITHM DESIGN AND IMPLEMENTATION

In this section, we first introduce some basic notions of TT-tree in IV-A. And then, we give two basic algorithms for showing the processes of constructing TT-tree in IV-B and mining FCIs in IV-C. Then, we present an efficient algorithm TT-Miner for mining FCIs, which includes enumerating closed candidates in TT-tree and checking closure of them, in IV-D. Last, we give some techniques to reduce computation time and memory usage, in IV-E.

A. TT-TREE OF THE TOPOLOGY $T_{\mathcal{I}}$ FOR $\mathcal{I}$

Now, we give some basic notion of TT-tree. According to Theorem 1, the topology $T_{\mathcal{I}}$ for $\mathcal{I}$ of $\mathcal{D} = (\mathcal{T}, \mathcal{I})$ can be generated by revisiting elements of the base $\mathcal{B}_{\mathcal{I}}$ of $T_{\mathcal{I}}$. Inspired by hash-tree in Apriori, FP-tree in FP-growth and IT-tree in CHARM method, here we provide topology-transaction tree (TT-tree) to fast construct the topology $T_{\mathcal{I}}$ for $\mathcal{I}$, formally, a topology-transaction pair (TT-pair) is $I \times t(I)$, where $I \subseteq \mathcal{I}$, especially, for every element of the base $\mathcal{B}_{\mathcal{I}}$, it's TT-pair is called as base-transaction pair and has the form $\mathcal{B}_{\mathcal{I}}(i) \times t(\mathcal{B}_{\mathcal{I}}(i))$, for convenience, we write $I = \{i_1, i_2, \cdots, i_p\}$ as $i_1 i_2 \cdots i_p$, $t(I) = \{t_1, t_2, \cdots, t_q\}$ as $t_1 t_2 \cdots t_q$ and $I \times t(I)$ as $i_1 i_2 \cdots i_p \times t_1 t_2 \cdots t_q$. Similar to hash-tree, FP-tree and IT-tree, we order all base-transaction pairs by a decreasing order, *i.e.*, for any two base-transaction pairs

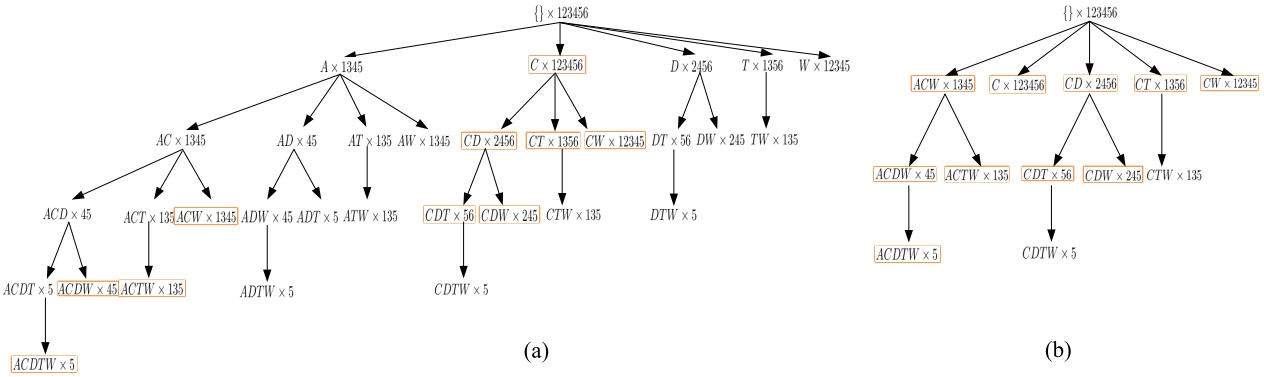


FIGURE 2. Comparison of search space between IT-tree (a) and TT-tree (b) with a same database in CHARM [25], FCIs are labeled by orange boxes.

$\mathcal{B}_{\mathcal{I}}(i) \times t(\mathcal{B}_{\mathcal{I}}(i))$ and $\mathcal{B}_{\mathcal{I}}(i) \times t(\mathcal{B}_{\mathcal{I}}(j))$, $\mathcal{B}_{\mathcal{I}}(i) \times t(\mathcal{B}_{\mathcal{I}}(i)) \leq \mathcal{B}_{\mathcal{I}}(j) \times t(\mathcal{B}_{\mathcal{I}}(j))$if and only if $|\mathcal{B}_{\mathcal{I}}(j)| \leq |\mathcal{B}_{\mathcal{I}}(i)|$, branches of TT-tree are generated by revisiting the ordered base-transaction pairs, *i.e.*,

1) The root of TT-tree is $\{\} \times \mathcal{T}$;
2) Children of the root are base-transaction pairs, from the left to right, base-transaction pairs are ordered by the decreasing order;
3) Children of any node $I \times t(I)(I \in \mathcal{I})$ have the form $(I \cup \mathcal{B}_{\mathcal{I}}(i)) \times (t(I) \cap t(\mathcal{B}_{\mathcal{I}}(i)))$, where according to the decreasing order, suppose $I = I' \cup \mathcal{B}_{\mathcal{I}}(i')$, then $\mathcal{B}_{\mathcal{I}}(i) \times t(\mathcal{B}_{\mathcal{I}}(i))$ is the backward element of $\mathcal{B}_{\mathcal{I}}(i') \times t(\mathcal{B}_{\mathcal{I}}(i'))$, $i \geq i'$ and such that $\mathcal{B}_{\mathcal{I}}(i) \nsubseteq I$.

Especially, the most important property of TT-tree is that the search space in TT-tree is compressed, when exist some $|\mathcal{B}_{\mathcal{I}}(i)| > 1$, $i \leq |\mathcal{B}_{\mathcal{I}}|$. The more items whose cardinal number of *topologicalbasis* are bigger than one, the more compressed of the search space in TT-tree. A father node $I \times t(I)(I \in \mathcal{I})$ and a *topologicalbasis* $\mathcal{B}_{\mathcal{I}}(i)$ combine a child node $(I \cup \mathcal{B}_{\mathcal{I}}(i)) \times (t(I) \cap t(\mathcal{B}_{\mathcal{I}}(i)))$, if $|\mathcal{B}_{\mathcal{I}}(i)| = N > 1$, this child node is roughly equivalent to the father node combining with at most N items.

In Fig. 2, we compare TT-tree and IT-tree of CHARM with the same database in [25], see Table 4. Figure a is the IT-tree structure of the database, and figure b is the TT-tree structure of the same database. All FCIs are labeled by orange boxes. It is obviously that, the IT-tree has 6 levels and 32 nodes, accordingly, TT-tree has only 4 levels and 13 nodes, and all the nodes of TT-tree are included in IT-tree. Not only the search space of TT-tree is highly compressed, but also the iteration times of constructing each node is fewer, *i.e.*, the left bottom node in TT-tree is $ACDTW \times 5$, generated by 3 iterations, but the same node in IT-tree needs 5 iterations. For constructing the TT-tree and IT-tree, the total iterations are 21 and 80, respectively, except iterations of no nodes being generated. For FCIs mining in this example, TT-tree comparing with IT-tree, the nodes and iterations are reduced by 59% and 73% respectively.

TABLE 4. A transaction database in [25].

tid	*items*
1	$ACTW$
2	CDW
3	$ACTW$
4	$ACDW$
5	$ACDTW$
6	CDT

B. CONSTRUCTION OF TT-TREE

We now present a basic algorithm *Construct-TT-tree* to show the process of construction of TT-tree for the topology $T_{\mathcal{I}}$ on the transaction database $\mathcal{D} = (\mathcal{T}, \mathcal{I})$. TT-tree structure combines both the *topologicalbasis* space and transaction space, the form is similar to IT-tree in CHARM [25]. But TT-tree is more condensed and convergence than IT-tree, for each child node is generated by its father node and a *topologicalbasis* rather than a singleton item.

The pseudocode for constructing TT-tree appears in Algorithm 2. The algorithm starts by calling *calc-topologicalbasis*, for scanning the transaction database, examining the frequent items, generating each *topologicalbasis* $\mathcal{B}_{\mathcal{I}}(i)$, and sorting all of them with descending order of $|\mathcal{B}_{\mathcal{I}}(i)|$. The corresponding transactions of each *topologicalbasis* be represented by an array list or a bitmap. And then add each *topologicalbasis* and its corresponding transactions as a child of root, in Line 4. TT-tree-Extended is responsible for considering each node whether to grow new nodes, in Line 5. For a existing node $I \times t(I)$, constructing by its father node and *topologicalbasis* $\mathcal{B}_{\mathcal{I}}(j)$, which is not the last *topologicalbasis* ($j < |\mathcal{B}_{\mathcal{I}}|$), may generate a new child node, in Line 8. If a candidate node meet the conditions that, its component *topologicalbasis* $\mathcal{B}_{\mathcal{I}}(k)$ is not included in its father node and its support is bigger than *minSup*, then it will be constructed, in Line 9-10. Then add the new node to TT-tree, and call TT-tree-Extend function, in Line 11-12. After return, a TT-tree has already been generated, then we can mine the FCIs from it.

Algorithm 2 *Construct-TT-Tree*($\mathcal{D}$, *minSup*)

Input: $\mathcal{D} = (\mathcal{T}, \mathcal{I})$, *minSup*.
Output: $\mathcal{N}$ // all TT-tree nodes
Method:
1) $\mathcal{N} = \emptyset$, $\mathcal{B}_{\mathcal{I}}$ = *calc-topologicalbasis*($\mathcal{D}$, *minSup*)
2) add $\{\} \times \mathcal{T}$ to $\mathcal{N}$ // root node
3) **for** $j := 1$ to $|\mathcal{B}_{\mathcal{I}}|$ //each *topologicalbasis*
4) add $\mathcal{B}_{\mathcal{I}}(j) \times t(\mathcal{B}_{\mathcal{I}}(j))$ to $\mathcal{N}$ // as a child of root
5) **TT-tree-Extend**($\mathcal{B}_{\mathcal{I}}(j), t(\mathcal{B}_{\mathcal{I}}(j)), j$)
6) **end for**
7) return $\mathcal{N}$

TT-tree-Extend($I, t(I), j$):
8) **for each** k, with $j < k \leq |\mathcal{B}_{\mathcal{I}}|$
9) **if** $\mathcal{B}_{\mathcal{I}}(k) \nsubseteq I$ and $\sigma(t(I) \cap t(\mathcal{B}_{\mathcal{I}}(k))) \geq minSup$ **do**
10) $I := I \cup \mathcal{B}_{\mathcal{I}}(k)$ and $t(I) := t(I) \cap t(\mathcal{B}_{\mathcal{I}}(k))$
11) add $I \times t(I)$ to $\mathcal{N}$ // a child node of $I \times t(I)$
12) **TT-tree-Extended**($I, t(I), k$)
13) **else** return
14) **end for**

C. OBTAINING FCIs IN TT-TREE

In this section, we present a basic algorithm to obtain FCIs in TT-tree. For convenience, we call node $I \times t(I)$ of TT-tree as FI (FCI) node if I is FI (FCI), according, FIs TT-tree means that every node is FI node. According to the downward closed property, if base-transaction pair $\mathcal{B}_{\mathcal{I}}(i) \times t(\mathcal{B}_{\mathcal{I}}(i))$ is not FI node, *i.e.*, $\frac{|t(\mathcal{B}_{\mathcal{I}}(i))|}{n} < minSup$ or $\mathcal{B}_{\mathcal{I}}(i)$ is infrequent, then for any base-transaction pair $\mathcal{B}_{\mathcal{I}}(i') \times t(\mathcal{B}_{\mathcal{I}}(i'))$, TT-pair $(\mathcal{B}_{\mathcal{I}}(i) \cup \mathcal{B}_{\mathcal{I}}(i')) \times t(\mathcal{B}_{\mathcal{I}}(i) \cup \mathcal{B}_{\mathcal{I}}(i'))$ is not FI node, *i.e.*, $\frac{|t(\mathcal{B}_{\mathcal{I}}(i) \cup \mathcal{B}_{\mathcal{I}}(i'))|}{n} \leq \frac{|t(\mathcal{B}_{\mathcal{I}}(i))|}{n} < minSup$ or $\mathcal{B}_{\mathcal{I}}(i) \cup \mathcal{B}_{\mathcal{I}}(i')$ is not FI. In other words, FIs TT-tree is generated by base-transaction pairs which are FIs nodes, *i.e.*,

1) Select base-transaction pair $\mathcal{B}_{\mathcal{I}}(i) \times t(\mathcal{B}_{\mathcal{I}}(i))$ such that $\frac{|t(\mathcal{B}_{\mathcal{I}}(i))|}{n} \geq minSup$;
2) Any TT-pair $I \times t(I)$ such that $\frac{|t(I)|}{n} < minSup$ will not be a node in TT-tree;
3) Any node $I \times t(I)$ is such that $\frac{|t(I)|}{n} \geq minSup$.

According to Property 6, FCIs nodes are contained in FIs TT-tree. Formally, we can find all FCIs by searching the maximum element from the equivalence class $[I_j] \in T_{\mathcal{I}}/\cong$, *i.e.*, $I \in [I_j]$, if $\forall I' \in [I_j]$, such that $I' \subseteq I$, then I is a FCI.

Now, we provide a basic Algorithm 3 to obtain FCIs from TT-tree, which uses hash table for storing FIs and closure checking like CHARM [25]. This algorithm is responsible for visiting all nodes of TT-tree, and obtaining all the closed itemsets. Input variable $\mathcal{N}$ is a set containing all TT-pairs. Output variable C is the set containing all FCIs in $\mathcal{N}$. We use a hash function $h(I) = \Sigma_{T \in t(I)} T$, which is the cardinality of $t(I)$. Firstly, all nodes (TT-pairs) of TT-tree are insert in *HashTable*, in Line 2-5. Then all nodes of TT-tree are checked whether or not they are closed, in Line 6-11. *HashTable*[$h(I)$] contains TT-pairs having the same support. For a TT-pair

Algorithm 3 *Obtain-FCIs*($\mathcal{N}$)

Input: $\mathcal{N}$ // all TT-tree nodes
Output: $\mathcal{C}$ // set of FCIs
Method:
1) $\mathcal{C} = \emptyset$, *HashTable* $= \emptyset$
2) **for each** node $I \times t(I)$ of TT-tree **do**
3) $h(I) = \Sigma_{T \in t(I)} T$ //hash function
4) insert $I \times t(I)$ in *HashTable*[$h(I)$]
5) **end for**
6) **for each** node $I \times t(I)$ of TT-tree **do**
7) $h(I) = \Sigma_{T \in t(I)} T$
8) **for all** $I' \in$ *HashTable*[$h(I)$] **do**
9) **if** $I' \subseteq I$ and $t(I') = t(I)$
10) add I to $\mathcal{C}$ //I is closed
11) **end for**
12) return $\mathcal{C}$

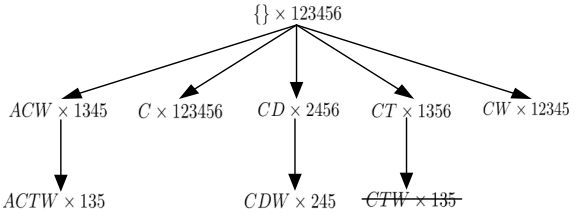


FIGURE 3. TT-tree for mining FCIs with a database in CHARM [25].

$I \times t(I)$, if there is no superset including I in *HashTable*[$h(I)$], then I is closed.

Example 1: Table 4 is a transaction database [11]. We fix $minSup = 0.5$, then a frequent itemset should at least occurrence in 3 transactions. Based Eqs.(1)-(3) and Algorithm 1, we obtain $C(A) = ACW$, $C(C) = C$, $C(D) = CD$, $C(T) = CT$, $C(W) = CW$. For $|t(ACW)| = 4$, $|t(C)| = 6$, $|t(CD)| = 4$, $|t(CT)| = 4$, $|t(CW)| = 5$, all of them are frequent and added in *topologicalbasis* $\mathcal{B}_{\mathcal{I}}$. After sorting them with the descending order of $|C(i)|$, we get $\mathcal{B}_{\mathcal{I}}(1) = ACW$, $\mathcal{B}_{\mathcal{I}}(2) = C$, $\mathcal{B}_{\mathcal{I}}(3) = CD$, $\mathcal{B}_{\mathcal{I}}(4) = CT$, $\mathcal{B}_{\mathcal{I}}(5) = CW$. By using Algorithm 2, we can construct TT-tree (shown in Fig.3). In TT-tree, we can obtain FIs, which are in the following TT-pairs

$$\begin{aligned}\mathcal{N} = \{&ACW \times 1345, C \times 123456, CD \times 2456,\\ &CT \times 1356, CW \times 12345, ACTW \times 135,\\ &CDW \times 245, CTW \times 135\}.\end{aligned}$$

We can obtain all FCIs nodes from Fig.3, by using Algorithm 3, *HashTable*[13] $= \{ACW \times 1345\}$, *HashTable*[21] $= \{C \times 123456\}$, *HashTable*[17] $= \{CD \times 2456\}$, *HashTable*[15] $= \{CT \times 1356, CW \times 12345\}$, *HashTable*[9] $= \{ACTW \times 135, CTW \times 135\}$, *HashTable*[11] $= \{CDW \times 245\}$. Only *HashTable*[9] has two nodes, and $CTW \subseteq ACTW$, and $t(CTW) = t(ACTW)$ so CTW is not closed. We obtain all closed itemsets $\mathcal{C} = \{ACW, C, CD, CT, CW, ACTW, CDW\}$.

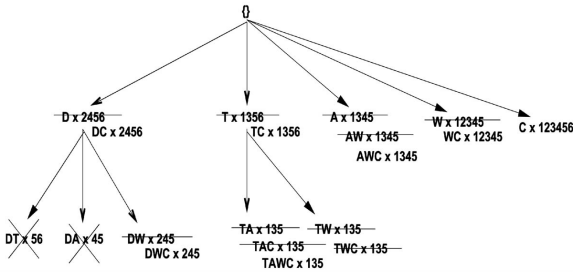


FIGURE 4. IT-tree for mining FCIs in CHARM [25].

In Fig. 3, two nodes $ACTW \times 135$ and $CTW \times 135$ have the same transactions 135, and $CTW \subseteq ACTW$, so $ACTW$ is closed, CTW is not closed. According to Property 6, we can obtain the same $\mathcal{C}$. Compared with IT-tree enumerating 17 candidate nodes, in Fig. 4, TT-tree enumerates only 8 nodes (except root node), in Fig. 3. It is obviously, that the search space of TT-tree is compressed. Particular, if for each i, $|\mathcal{B}_{\mathcal{I}}(i)| = 1$, then TT-tree and IT-tree have a same search space.

D. TT-MINER:A FAST FCIs MINING ALGORITHM

We now present TT-Miner, a very efficient algorithm for mining all the FCIs. TT-Miner uses a novel search method, based on properties of *topologicalbasis* and TT-tree structure, that constructs less levels and generates fewer candidates and checks the closure quickly. The pseudocode for TT-Miner appears in Algorithm 4. It starts by calculating the set of *topologicalbasis* $\mathcal{B}_{\mathcal{I}}$ of dataset $\mathcal{D}$. The *topologicalbasis* disclosing the co-occurrence relations is a basic element for constructing TT-tree and not always a singleton item. The main procedure is performed in *TT-miner-Extend*, which returns all FCIs.

Algorithm 4 *TT-Miner*($\mathcal{D}$, *minSup*)

Input: $\mathcal{D} = (\mathcal{T}, \mathcal{I})$, *minSup*.
Output: $\mathcal{C}$ // all FCIs
Method:
1) $\mathcal{C} = \emptyset$, $\mathcal{B}_{\mathcal{I}}$ = *calc-topologicalbasis*($\mathcal{D}$, *minSup*)
2) call *TT-miner-Extend*($\mathcal{D}$, $\perp$, $\mathcal{T}$, $\mathcal{C}$, $\mathcal{B}_{\mathcal{I}}$);
3) return $\mathcal{C}$

Inspired by LCM [26], we take the framework of backtracking and pattern extending in *TT-miner-Extend*. The pseudocode provided in Algorithm 5, is responsible for closure checking of combination, of the current FCI and its subsequent *topologicalbasis*. First add the current FCI $\mathcal{P}$ into $\mathcal{C}$ in Line 1, which is generated in last iteration. Then reduce $\mathcal{D}$ by a technique called anytime database reduction in LCM in Line 2, which will be described in next section. And then, enumerate all the extend patterns of $\mathcal{P}$, and check the closure of them in the loop in Line 3-7. *key*($\mathcal{P}$) means a ordinal number of *topologicalbasis* generated $\mathcal{P}$, *i.e.*, $\mathcal{P} = \mathcal{P}' \cup \mathcal{B}_{\mathcal{I}}(j)$, then *key*($\mathcal{P}$) is j. *Fast-closed-test* is used to check the closure

Algorithm 5 *TT-Miner-Extend*($\mathcal{D}$, $\mathcal{P}$, $t(\mathcal{P})$, $\mathcal{C}$, $\mathcal{B}_{\mathcal{I}}$)

Input: $\mathcal{D} = (\mathcal{T}, \mathcal{I})$: transaction database, $\mathcal{P}$: current frequent closed itemset, $t(\mathcal{P})$: transactions including $\mathcal{P}$, $\mathcal{C}$: all FCIs.
Method:
1) add $\mathcal{P}$ into $\mathcal{C}$
2) Reduce $\mathcal{D}$ as $\mathcal{D}'$ by Anytime Database Reduction
3) **for** i:= key($\mathcal{P}$)+1 to $|\mathcal{B}_{\mathcal{I}}|$
4) **if** *Fast-closed-test*($\mathcal{P}$, $\mathcal{B}_{\mathcal{I}}(i)$) is true and $\sigma(\mathcal{P} \cup \mathcal{B}_{\mathcal{I}}(i)) \geq$ *minSup* **then**
5) $\mathcal{P} := \mathcal{P} \cup \mathcal{B}_{\mathcal{I}}(i)$, $t(\mathcal{P}) := t(\mathcal{P}) \cap t(\mathcal{B}_{\mathcal{I}}(i))$
6) call *TT-Miner-Extend*($\mathcal{D}'$, $\mathcal{P}$, $t(\mathcal{P})$, $\mathcal{C}$, $\mathcal{B}_{\mathcal{I}}$));
7) **end for**
8) return

Algorithm 6 *Fast-Closed-Test*($\mathcal{P}$, $\mathcal{B}_{\mathcal{I}}(i)$)

Input: $\mathcal{D} = (\mathcal{T}, \mathcal{I})$: transaction database, $\mathcal{P}$: current frequent closed itemset, $t(\mathcal{P})$: transactions including $\mathcal{P}$, $\mathcal{C}$: all FCIs.
Method:
1) tid_{min}:= the tid of the shortest transaction of $t(\mathcal{P} \cup \mathcal{B}_{\mathcal{I}}(i))$
2) $X := i(tid_{min})/(\mathcal{P} \cup \mathcal{B}_{\mathcal{I}}(i))$
3) **for** each $x \in X$
4) **if** x is included in all transactions of $t(\mathcal{P}) \cap t(\mathcal{B}_{\mathcal{I}}(i))$
5) return false;
6) **end for**
7) return true

of $\mathcal{P} \cup \mathcal{B}_{\mathcal{I}}(i)$, the detail is in Algorithm 6. If $\mathcal{P} \cup \mathcal{B}_{\mathcal{I}}(i)$ is closed and frequent, then it will be a new FCI, named $\mathcal{P}$ also. Then call *TT-Miner-Extend* to test and generate subsequent FCIs, iteratively.

Inspired by fast prefix-preserving test in LCM [26], we give a method for fast closure checking, $\mathcal{P} \cup \mathcal{B}_{\mathcal{I}}(i)$ in Algorithm 6. In order to check the closure of $\mathcal{P} \cup \mathcal{B}_{\mathcal{I}}(i)$, we need to know whether $\mathcal{P} \cup \mathcal{B}_{\mathcal{I}}(i)$ is equal to $i(t(\mathcal{P} \cup \mathcal{B}_{\mathcal{I}}(i)))$ or not. It is obvious that the calculation of $i(t(\mathcal{P} \cup \mathcal{B}_{\mathcal{I}}(i)))$ is very time consuming. From Property 2, we know that $\mathcal{P} \cup \mathcal{B}_{\mathcal{I}}(i) \subseteq C(\mathcal{P} \cup \mathcal{B}_{\mathcal{I}}(i))$. There are two possibile conditions:

1) if $\exists$ *item* $j \in i(t(\mathcal{P}) \cap (\mathcal{B}_{\mathcal{I}}(i)))$ and $j \notin \mathcal{P} \cup \mathcal{B}_{\mathcal{I}}(i)$, then $\mathcal{P} \cup \mathcal{B}_{\mathcal{I}}(i) \neq C(\mathcal{P} \cup \mathcal{B}_{\mathcal{I}}(i))$, so $\mathcal{P} \cup \mathcal{B}_{\mathcal{I}}(i)$ is not a *FCI*
2) if $\forall$ *item* $j \in i(t(\mathcal{P} \cup \mathcal{B}_{\mathcal{I}}(i)))$ and $j \in \mathcal{P} \cup \mathcal{B}_{\mathcal{I}}(i)$, then $\mathcal{P} \cup \mathcal{B}_{\mathcal{I}}(i) = C(\mathcal{P} \cup \mathcal{B}_{\mathcal{I}}(i))$, so $\mathcal{P} \cup \mathcal{B}_{\mathcal{I}}(i)$ is a *FCI*

In order to reduce the test amount of items, we find the shortest transaction of $t(\mathcal{P} \cup \mathcal{B}_{\mathcal{I}}(i))$, its tid is tid_{min}. For all the transaction are stored in array lists, it is easy to find tid_{min}. Then we set X as $i(tid_{min})/(\mathcal{P} \cup \mathcal{B}_{\mathcal{I}}(i))$. If a member of X is included in all transactions of $t(\mathcal{P}) \cap t(\mathcal{B}_{\mathcal{I}}(i))$, then $\mathcal{P} \cup \mathcal{B}_{\mathcal{I}}(i)$ is not a FCI, meeting condition 1. If all the members of X are not included in all transactions of $t(\mathcal{P}) \cap t(\mathcal{B}_{\mathcal{I}}(i))$, then $\mathcal{P} \cup \mathcal{B}_{\mathcal{I}}(i)$ is a FCI, meeting condition 2. In practice, $|X| \ll \mathcal{P} \cup \mathcal{B}_{\mathcal{I}}(i)$, so the test amount of X is obviously reduced.

E. REDUCING COMPUTATION TIME

In this section, we explain some popular techniques used for reducing computation time and memory usage.

1) REPRESENTATION OF THE DATABASE

We use a vertical bitmap to represent for the database like MAFIA [40]. It is efficient for dense databases. In a vertical bitmap, there is one bit stand for each transaction in the database. If item i appears in transaction j, then bit j of the bitmap for item i is set to one; otherwise, the bit is set to zero. Similarly, if a node in TT-tree is $I \times t(I) \equiv i_1 i_2 \cdots i_p \times t_1 t_2 \cdots t_q$, then itemset $t_1 t_2 \cdots t_q$ can be represented by a bitmap $X \equiv [t'_1, t'_2, \cdots, t'_n]^T$, if $t_1 \in t(I)$, then $t'_1 = 1$, else $t'_1 = 0$. For example, in Fig. 3, the top left node $ACW \times 135$, tidset 135 can be represented by $[1, 0, 1, 0, 1, 0]^T$. Let $onecount(X)$ be the number of ones in the vertical bitmap for X, then $onecount(X)$ is exactly the support of X, which is also the support of the node in TT-tree. For X and Y are two itemsets, $X \bigcap Y$ is simply computed as the bitwise-AND of bitmap X and bitmap Y.

2) FREQUENT COUNTING

In FCIs mining, the process of frequent counting is compute-intensive. In this paper , we take an efficient methods *Occurrence Deliver* [26] in this process.

Occurrence Deliver is particularly efficient for sparse database in reducing the computation of $t(\mathcal{P} \cup \mathcal{B}_{\mathcal{I}}(i))$. It computes the tids of $\mathcal{P} \cup \mathcal{B}_{\mathcal{I}}(i)$ for $i = key(\mathcal{P}) + 1$ to $|\mathcal{B}_{\mathcal{I}}|$ at once by tracing the tids in $t(\mathcal{P})$. It use a bucket for each i to be added, $i \in \mathcal{I}, i > key(\mathcal{P})$, and set them empty set at the beginning. Then, by visiting each transaction $t \in t(\mathcal{P})$, *Occurrence Deliver* inserts t to the bucked of i for each $i \in i(t), i > key(\mathcal{P})$. After visiting all the transaction in $t(\mathcal{P})$, and inserting each *tid* to its corresponding bucket, the bucket of i is equal to $t(\mathcal{P} \cup \mathcal{B}_{\mathcal{I}}(i))$.

3) ANYTIME DATABASE REDUCTION

In order to facilitate the computation of FCIs, the size of the input database is reduced by removing infrequent items and the same transactions are merged into one. We use anytime database reduction method [26] to reduce the database in *TT-Miner-Extend*. For input FCI $\mathcal{P}$ in each iteration of *TT-Miner-Extend*, those unnecessary items and transactions are deleted:

1) transactions not including $\mathcal{P}$;
2) transactions including no item bigger than $key(\mathcal{P})$,
3) items i, satisfies $i \leq key(\mathcal{P})$, and
4) items i, satisfies that $\sigma(\mathcal{P} \cup \mathcal{B}_{\mathcal{I}}(i)) < minSup$.

This process is efficient in practice, especially for large support.

V. EXPERIMENTS ANALYSIS

We chose 6 real databases and 1 synthetic database as benchmarks, publicly available from FIMI'03 [41], for the performance tests. The accidents database [42] contains traffic accidents data of a region of Belgium for period 1991-2000. The chess and connect databases are derived from their respective game steps. The mushroom data set contains characteristics of 23 species of mushrooms. The pumsb database contain census data and pumsb* is the same as pumsb without items with 80 percent or more support. The chess, connect and mushroom databases were originally taken from UCI Machine Learning Repository [43]. T40I10D100K is a synthetic dataset created by the IBM Almaden synthetic data generator. Table 5 shows the characteristics of the databases used in this paper, which shows the number of items, the average transaction length, the number of records and the value range of *minSup* in each data set .

All our experiments were performed on a ThinkPad laptop with 2.3 GHz Intel i5-6200U CPU, 20 GB of memory, running 64-bit Windows 10. TT-Miner was coded in C, using Microsoft Visual 2015. All times reported are wall-clock time and include all preprocessing costs.

TABLE 5. Database characteristics.

Dataset	# Items	Avg.Length	# Transactions	Minsup(%)
accidents	468	33.8	340,183	10-90
chess	75	37	3,196	20-90
Connect	129	43	67,557	15-95
pumsb	2113	74	49,046	45-95
pumsb*	2088	50.5	49,046	10-50
mushroom	119	23	8,124	0.1-20
T40I10D100K	1000	39.6	100,000	0.1-2

A. MINING CLOSED FREQUENT ITEMSETS

We compared the performance of TT-Miner against FPclosed [11], Apriori [1], Eclat [44] and LCM [26]. The first IEEE ICDM Workshop on frequent itemset mining (FIMI'03) [45] demonstrated FPclose and LCM had very competitive and robust performance in mining closed itemsets than other algorithms. So, they are the fastest benchmark programs. The programs of them were coded by their authors in C++ and ANSI C respectively, and obtained from [46]. The Windows console executable programs of Apriori and Eclat were coded by Christian Borgelt and obtained form [47]. The program of Eclat was a hybrid version including certain variants of LCM, so it was marked as Eclat/LCM. Unfortunately, we did not compile CHARM successfully in Microsoft Visual 2015, so we did not compare the performance of it.

All the performance on real database were shown in Fig. 5. We first compare how the methods perform on accidents database. We observe that TT-Miner, LCM, Ecalt/LCM and Apriori have a similar and better performance than fpclose for high values of support. For very low values of support, TT-Miner and LCM are 100 times faster than Eclat/LCM. The running time of FPclose rise very slowly when support is bigger than 30%, but it can not run on very low support values as Apriori. Among those methods, TT-Miner is the fastest and LCM is the second faster, the gap between the top two is very small.

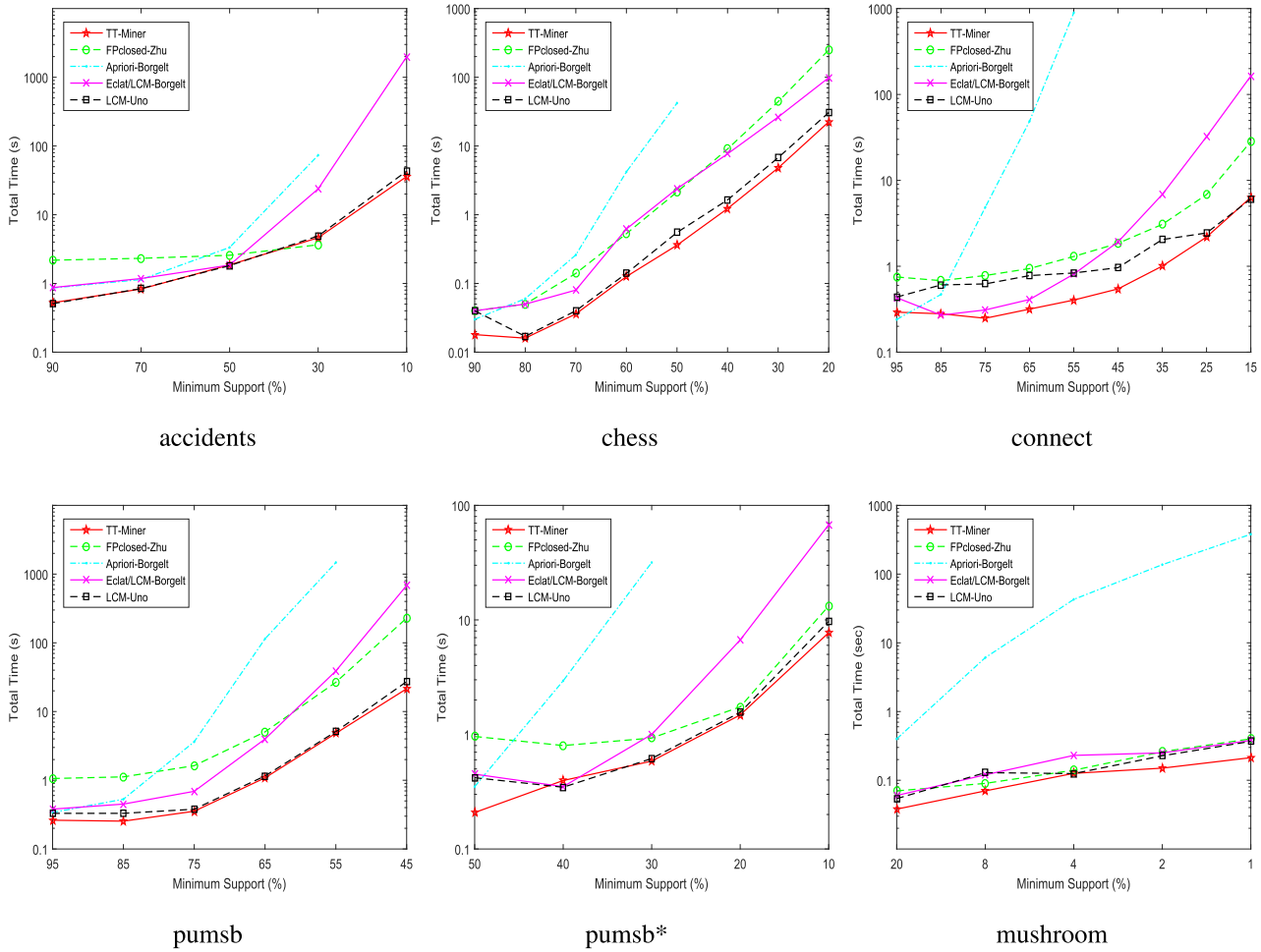


FIGURE 5. Performance of TT-Miner on 6 real databases.

On chess database, it is obvious that TT-Miner performs well against others. There is a counterintuitive phenomenon, that the running time of TT-Miner and LCM for the support value 80% is smaller than it for the support value 90%. We think the reason is the programs read dataset from the cache except for the first time reading from disk. Apriori can not run on low support, *i.e.*, support values are smaller than 40%. TT-Miner is about 10 times faster than FPclose and Eclat/LCM.

On connect database, Apriori runs fast at very low support, but runs slower and slower, as the support is reduced. Comparing with Eclat/LCM, we find that both TT-Miner and Eclat/LCM have a good performance for higher support values. However, the performance gap between TT-Miner and Eclat/LCM widens, as the support is reduced. TT-Miner is significantly better than others. For the support value is 15%, LCM has almost the same running time as TT-Miner.

On pumsb database, TT-Miner is only a little faster than LCM. For very high support values, Eclat/LCM and Apriori are faster than FPclose, but their performances are going into reverse for very low support values. TT-Miner is 10 times faster than FPclose, and more than 50 times faster than Eclat/LCM.

On pumsb* database, TT-Miner is faster than other except for the support value is 40%. FPclose is slower than others for very low support, but is fast for high support. Apriori is the slowest among all the program, and can not run on low support, *i.e.*, support values are smaller than 30%.

The performance on synthetic database was shown in Fig. 6. On T40I10D100K, which is a synthetic database, except for Apriori, others have a very similar trend: as supports decrease, the running time increases gradually. Apriori runs fastly for high support values, but runs slowly for low support values and can not run for very low support values. TT-Miner has the best performance for low support values.

Finally, we compare the perform on mushroom database. Apriori runs several order of magnitudes slower than others no matter for high or low support values. TT-Miner, FPclose, Eclat/LCM and LCM perform very well, and have very small speed gaps.

To sum up, TT-Miner runs faster than other algorithms in most instances, no matter the support values are high or low.

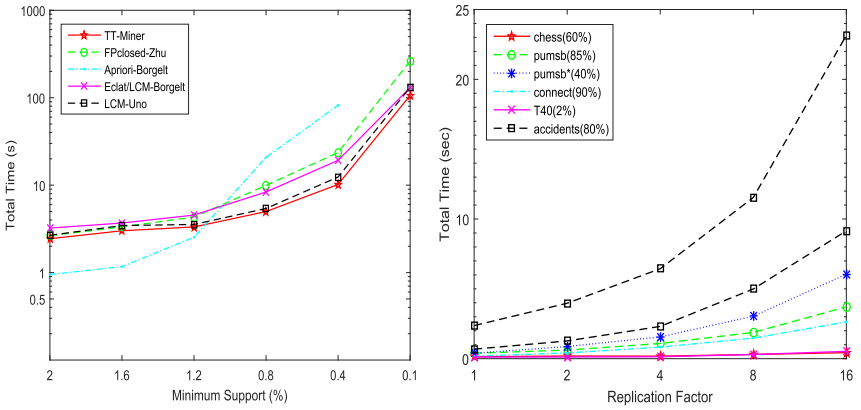


FIGURE 6. Scaleup test.

TABLE 6. Comparison item set space with *topologicalbasis* space.

Database	pumsb						pumsb*				
$minSup$	95%	85%	75%	65%	55%	45%	50%	40%	30%	20%	10%
$\#item$	13	24	27	37	44	63	27	46	60	86	110
$\#topologicalbasis$	12	20	23	33	39	52	23	38	50	69	93
$\frac{\#item}{\#topologicalbasis}$	**1.08**	**1.2**	**1.17**	**1.12**	**1.21**	**1.21**	**1.17**	**1.21**	**1.2**	**1.24**	**1.18**
$\#item\_iters$	118	9051	107600	529584	2934943	19088290	267	3064	22687	213122	3270492
$\#topologicalbasis\_iters$	110	8510	101225	498212	2752389	17853778	261	2900	20599	170610	2206747
$\frac{\#item\_iters}{\#vector\_iters}$	**1.0631**	**1.0634**	**1.063**	**1.063**	**1.0663**	**1.0691**	**1.02**	**1.05**	**1.1**	**1.2492**	**1.4872**

B. SCALEUP EXPERIMENTS

Scaleup test in Fig. 6 shows how TT-Miner scales with an increasing number of transactions. In this research, we replicated the transactions from 2 to 16 times, and kept the parameters constant of 7 databases, *i.e.*, for chess having 3196 transactions, at a replication factor of 2, it will have 6392 transactions. We find the running time of TT-Miner increasing by linear, with increasing number of transactions, at a given *minSup*.

C. COMPARISON ITEM SPACE AND TOPOLOGICAL BASIS SPACE

In property 4, $|\mathcal{B}_{\mathcal{I}}| \leq |\mathcal{I}|$, which means the *topologicalbasis* space is not bigger than the item space. In the experiments above, we found the cardinal number of *topologicalbasis* is smaller than that of item, on pumsb and pumsb*, see in Table 6, *i.e.*, in pumsb* database, if $minSup = 10\%$, the volume of item set is 110, but the volume of *vecctorbase* is reduced by 18% to 93, the number of iterations is also reduced by 48.72%. The radio of #*item* and #*topologicalbasis* is between 1.08 and 1.24. In other words, the *topologicalbasis* space is at least 8% and at most 24% smaller then the item space. If we do not remove the redundant *topologicalbasis*, and mark the iterations of the TT-Miner program as #*iter_iters*. #*vector_iters* is the iterations of TT-Miner program, which has removed the redundant *topologicalbasis*. Accordingly, radio of #*item_iters* and #*topologicalbasis_iters* is between 1.02 and 1.4872. In other words, the iterations of the program using *topologicalbasis* is reduced by 2% to 48.72%.

VI. CONCLUSIONS AND FURTHER WORK

Frequent itemsets and closed itemsets are widely used to generate association rules from transaction databases, and many methods have been proposed to obtain frequent itemsets or closed itemsets. In this paper, the topology on itemset of a transaction database is constructed to represent more general associative relationship among items of transaction databases, it has been proved that closed itemsets are included in the topology on itemset, it is important that the basis for the topology deduced directly from the transaction database can be used to generate the topology on itemset, this can efficiently avoid a large number of unnecessary enumerations. The experimental and comparative results show that FCIs mining based on topology for itemset is an efficient method.

In the view of granular computing, the knowledge granular of the topology of item set is coarser than a singleton item. Theoretically the construct speed of TT-tree should be faster than other tree structure mining methods, and the searching space of FCIs is reduced naturally. According to analyze the

procedure of constructing of TT-tree, it may satisfy parallelism. In our next work, we will try to modify it to a parallel version of TT-tree.

Furthermore, TT-tree can be used to fast mine minimal non-redundant association rules from transaction databases.

REFERENCES

[1] R. Agrawal, T. Imielinski, and A. Swami, "Mining association rules between sets of items in large databases," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, vol. 22, no. 2, 1993, pp. 207–216.

[2] E. Baralis, L. Cagliero, T. Cerquitelli, V. Delia, and P. Garza, "Expressive generalized itemsets," *Inf. Sci.*, vol. 278, pp. 327–343, Sep. 2014.

[3] L. Cagliero, T. Cerquitelli, P. Garza, and L. Grimaudo, "Misleading generalized itemset discovery," *Expert Syst. Appl.*, vol. 41, no. 4, pp. 1400–1410, 2014.

[4] T. Calders and B. Goethals, "Non-derivable itemset mining," *Data Mining Knowl. Discovery*, vol. 14, no. 1, pp. 171–206, 2007.

[5] T. Calders, N. Dexters, J. J. M. Gillis, and B. Goethals, "Mining frequent itemsets in a stream," *Inf. Syst.*, vol. 39, pp. 233–255, Jan. 2014.

[6] J. Han, H. Cheng, D. Xin, and X. Yan, "Frequent pattern mining: Current status and future directions," *Data Mining Knowl. Discovery*, vol. 15, no. 1, pp. 55–86, Aug. 2007.

[7] H. Li and H. Chen, "Mining non-derivable frequent itemsets over data stream," *Data Knowl. Eng.*, vol. 68, no. 5, pp. 481–498, 2009.

[8] Ö. M. Soysal, "Association rule mining with mostly associated sequential patterns," *Expert Syst. Appl.*, vol. 42, no. 5, pp. 2582–2592, 2015.

[9] A. Cuzzocrea, C. K.-S. Leung, and R. K. MacKinnon, "Mining constrained frequent itemsets from distributed uncertain data," *Future Gener. Comput. Syst.*, vol. 37, pp. 117–126, Jul. 2014.

[10] T. F. Gharib, "An efficient algorithm for mining frequent maximal and closed itemsets," *Int. J. Hybrid Intell. Syst.*, vol. 6, no. 3, pp. 147–153, 2009.

[11] G. Grahne and J. Zhu, "Fast algorithms for frequent itemset mining using FP-trees," *IEEE Trans. Knowl. Data Eng.*, vol. 17, no. 10, pp. 1347–1362, Oct. 2005.

[12] F. Guil and R. Marín, "A theory of evidence-based method for assessing frequent patterns," *Expert Syst. Appl.*, vol. 40, no. 8, pp. 3121–3127, 2013.

[13] T. Guns, S. Nijssen, and L. De Raedt, "Itemset mining: A constraint programming perspective," *Artif. Intell.*, vol. 175, nos. 12–13, pp. 1951–1983, 2011.

[14] T. Hashem, C. F. Ahmed, M. Samiullah, S. Akther, B.-S. Jeong, and S. Jeon, "An efficient approach for mining cross-level closed itemsets and minimal association rules using closed itemset lattices," *Expert Syst. Appl.*, vol. 41, no. 6, pp. 2914–2938, 2014.

[15] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal, "Efficient mining of association rules using closed itemset lattices," *Inf. Syst.*, vol. 24, no. 1, pp. 25–46, 1999.

[16] J. Pei, J. Han, and R. Mao, "CLOSET: An efficient algorithm for mining frequent closed itemsets," in *Proc. ACM SIGMOD Workshop Res. Issues Data Mining Knowl. Discovery*, vol. 4, no. 2, 2000, pp. 21–30.

[17] M. J. Zaki, "Closed itemset mining and non-redundant association rule mining," in *Encyclopedia of Database Systems*, L. Liu and M. T. Ozsu, Eds. New York, NY, USA: Springer-Verlag, 2017.

[18] J. M. Luna, F. Padillo, M. Pechenizkiy, and S. Ventura, "Apriori versions based on mapreduce for mining frequent patterns on big data," *IEEE Trans. Cybern.*, vol. 48, no. 10, pp. 2851–2865, Oct. 2018.

[19] B. Huynh, B. Vo, and V. Snasel, "An efficient parallel method for mining frequent closed sequential patterns," *IEEE Access*, vol. 5, pp. 17392–17402, 2017.

[20] M. Ghorbani and M. Abessi, "A new methodology for mining frequent itemsets on temporal data," *IEEE Trans. Eng. Manag.*, vol. 64, no. 4, pp. 566–573, Nov. 2017.

[21] M. J. Zaki, "Scalable algorithms for association mining," *IEEE Trans. Knowl. Data Eng.*, vol. 12, no. 3, pp. 372–390, May 2000.

[22] C. C. Aggarwal and J. Han, Eds., *Frequent Pattern Mining*. New York, NY, USA: Springer, 2012.

[23] J. Han, J. Pei, Y. Yin, and R. Mao, "Mining frequent patterns without candidate generation: A frequent-pattern tree approach," *Data Mining Knowl. Discovery*, vol. 8, no. 1, pp. 53–87, 2004.

[24] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal, "Discovering frequent closed itemsets for association rules," in *Database Theory—ICDT*, C. Beeri and P. Buneman, Eds. Berlin, Germany: Springer, 1999, pp. 398–416.

[25] M. J. Zaki and C. J. Hsiao, "Efficient algorithms for mining closed itemsets and their lattice structure," *IEEE Trans. Knowl. Data Eng.*, vol. 17, no. 4, pp. 462–478, Apr. 2005.

[26] T. Uno, T. Asai, Y. Uchida, and H. Arimura, "LCM: An efficient algorithm for enumerating frequent closed item sets," in *Proc. Workshop Frequent Itemset Mining Implement. (FIMI)*, vol. 90, 2003, pp. 1–10.

[27] C. Borgelt, "Frequent item set mining," *Wiley Interdiscipl. Rev., Data Mining Knowl. Discovery*, vol. 2, no. 6, pp. 437–456, 2012.

[28] Z. Pei, D. Ruan, D. Meng, and Z. Liu, "Formal concept analysis based on the topology for attributes of a formal context," *Inf. Sci.*, vol. 236, pp. 66–82, Jul. 2013.

[29] B. Ganter and R. Wille, *Formal Concept Analysis: Mathematical Foundations*. New York, NY, USA: Springer-Verlag, 1999.

[30] Z. Ma, J. Li, and J. Mi, "Some minimal axiom sets of rough sets," *Inf. Sci.*, vol. 312, pp. 40–54, Aug. 2015.

[31] Z. Pawlak and A. Skowron, "Rough sets and Boolean reasoning," *Inf. Sci.*, vol. 177, no. 1, pp. 41–73, 2007.

[32] K. Qin, J. Yang, and Z. Pei, "Generalized rough sets based on reflexive and transitive relations," *Inf. Sci.*, vol. 178, no. 21, pp. 4138–4141, 2008.

[33] L. Qin and Z. Shi, "SFP-Max—A sorted FP-tree based algorithm for maximal frequent patterns mining," *Jisuanji Yanjiu Fazhan (Comput. Res. Develop.)*, vol. 42, no. 2, pp. 217–223, 2005.

[34] Y.-R. Syau and E.-B. Lin, "Neighborhood systems and covering approximation spaces," *Knowl.-Based Syst.*, vol. 66, pp. 61–67, Aug. 2014.

[35] H.-P. Zhang, Y. Ouyang, and Z. Wang, "Note on 'generalized rough sets based on reflexive and transitive relations,'" *Inf. Sci.*, vol. 179, no. 4, pp. 471–473, 2009.

[36] Z. Zhao, "On some types of covering rough sets from topological points of view," *Int. J. Approx. Reasoning*, vol. 68, pp. 1–14, Jan. 2016.

[37] W. Zhu and S. Wang, "Rough matroids based on relations," *Inf. Sci.*, vol. 232, pp. 241–252, May 2013.

[38] W. Zhu and F.-Y. Wang, "The fourth type of covering-based rough sets," *Inf. Sci.*, vol. 201, pp. 80–92, Oct. 2012.

[39] W. Zhu, "Relationship between generalized rough sets based on binary relation and covering," *Inf. Sci.*, vol. 179, no. 3, pp. 210–225, 2009.

[40] D. Burdick, M. Calimlim, J. Flannick, J. Gehrke, and T. Yiu, "MAFIA: A maximal frequent itemset algorithm," *IEEE Trans. Knowl. Data Eng.*, vol. 17, no. 11, pp. 1490–1504, Nov. 2005.

[41] (Jun. 2017). *Frequent Itemset Mining Dataset Repository*. [Online]. Available: http://fimi.ua.ac.be/

[42] K. Geurts, G.Wets, T. Brijs, and K. Vanhoof, "Proling of high-frequency accident locations by use of association rules," *Transp. Res. Rec., J. Transp. Res. Board*, vol. 1840, no. 1840, pp. 123-130, 2003.

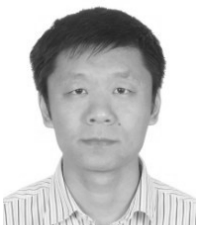
[43] UCI. (Jun. 2017). *Machine Learning Repository*. [Online]. Available: http://archive.ics.uci.edu/ml/index.php

[44] M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li, "New algorithms for fast discovery of association rules," in *Proc. Int. Conf. Knowl. Discovery Data Mining*, 1997, pp. 283–286.

[45] B. Goethals and M. J. Zaki, "FIMI'03: Workshop on frequent itemset mining implementations," in *Proc. 3rd IEEE Int. Conf. Data Mining Workshop Frequent Itemset Mining Implement.*, Nov. 2003, pp. 1–13.

[46] G. Bart. (Jun. 2017). *Frequent Itemset Mining Implementations Repository*. [Online]. Available: http://fimi.ua.ac.be/src/

[47] C. Borgelt. (Jun. 2017). *Christian Borgelt's Web Pages*. [Online]. Available: http://www.borgelt.net/software.html



BO LI received the B.S. degree in computer science from the Shaanxi University of Technology, in 2006, and the M.S. degree in computer science from Xihua University, in 2009. He is currently pursuing the Ph.D. degree with the School of Information Science and Technology, Southwest Jiaotong University, Chengdu, China. His research interests include data mining and machine learning.

ZHENG PEI received the M.S. and Ph.D. degrees from Southwest Jiaotong University, Chengdu, China, in 1999 and 2002, respectively. He is currently a Professor with the School of Computer and Software Engineering, Xihua University, Chengdu. He has published nearly 70 research papers in academic journals or conferences. His research interests include rough set theory, fuzzy set theory, logical reasoning, and linguistic information processing.

KEYUN QIN received the M.Sc. and Ph.D. degrees in applied mathematics from Southwest Jiaotong University, China, in 1994 and 1997, respectively. He is currently a Full Professor with the College of Mathematics, Southwest Jiaotong University. He has published nearly 80 research papers in academic journals or conferences. His current research interests include rough set theory, soft set theory, fuzzy logic-based systems, and formal concept analysis.

MINGMING KONG received the M.S. degree in applied mathematics from Xihua University, Chengdu, Sichuan, China, in 2010. He is currently a Lecturer with the School of Computer and Software Engineering, Xihua University. His current research interests include rough set theory, fuzzy set theory, linguistic information processing, and spectrum management.