Mining Maximal Frequent Subtrees based on Fusion Compression and FP-tree

Xiaoke Wu, Xiongfei Li
Department of Computer Science and Technology, Jilin University,
Chang Chun, China, wuxk09@mails.jlu.edu.cn
*2,Corresponding Author
Department of Computer Science and Technology, Jilin University,
Chang Chun, China, xiongfei@jlu.edu.cn
doi:10.4156/jdcta.vol5.issue7.24

Abstract

It is commonly accepted that mining frequent subtrees play pivotal roles in areas like Web log analysis, XML document analysis, semi-structured data analysis, as well as biometric information analysis, chemical compound structure analysis, etc. An improved algorithm, i.e. MFPTM algorithm, which based on fusion compression and FP-tree principle, was proposed in this paper to determine a better way to mine maximal frequent subtrees. The algorithm firstly retains subtrees which only contain frequent nodes by fusion compression, then according to FP-tree principle mines frequent subtrees. In the process of mining frequent subtrees, MFPTM algorithm is the means by which we attempt to satisfy our appetite for saving searching space of mining candidate patterns, and our craving to solve problems of frequent pattern mining based on Apriori algorithm which is generating a large quantity of candidate patterns. MFPTM algorithm, which actively represents as many viewpoints as is both possible and feasible as an advanced algorithm, improves the efficiency of mining frequent subtrees.

Keywords: Fusion Compression, FP-tree, Frequent Subtrees, Frequent Nodes

1. Introduction

With the continuous development of its technology, data mining, take on a wider scope of analysis that inquires into association rules [1] and sequences [2], can be generalized to complicated patterns like tree pattern [3][4] and graph pattern [5][6][7]. Now, it is indeed that the proliferation of mining frequent subtrees has played dominant roles of academic fields from Web log analysis [8][9], XML document analysis [10][11], semi-structured data analysis [12][13][14], to relative research activities in biometric information analysis [15], chemical compound structure analysis [16][17], etc. One apt illustration of this point involves the application to web logs, frequent subtree mining can mine users' access mode. Also when applied to XML documents, it can discover frequent data structure, etc. Exploring substantial frequent patterns in chemical structure analysis from considerable quantities of chemical compound structures of the known, researchers could have gone a long way toward providing information to analyze new chemical compound structures.

By studying frequent subtree mining, its relative research activities have spawned great advances universally. Most of data mining algorithms are based on Apriori algorithm. Wang and Liu [18] introduced an algorithm based on Apriori to mine frequent paths in ordered trees. Yongqiao Xiao [19] presented an algorithm which is also based on Apriori to mine maximal frequent subtrees efficiently. On the basis of Apriori and depth-first strategy, Zaki [3] proposed TreeMiner algorithm to mine frequent subtrees by making use of the right extended techniques. And Asai [20] also put forward FREQT algorithm based on the right extended techniques to mine induced subtrees in ordered trees. Then by taking advantage of the scope and sequence, Zaki presented TreeMinerH algorithm and TreeMinerV algorithm. These algorithms almost have the same procedure: firstly need to generate candidate patterns, secondly measure whether

1 This work was supported by Natural Foundation of Jilin Province under Grant No. 20090704 and No. 201115020.
candidate patterns are frequent or not, finally mine frequent subtrees. Referring to Apriori character, the algorithms can compress the searching space to achieve significant effects, but when mining frequent subtrees, there are two major shortages. First, connecting step has certain blindness, because it will generate a lot of candidate itemsets and candidate subtrees which don't exist in real tree database, leading to waste much time in calculating the frequencies of candidate subtrees which don't exist actually. Second, the need of repeating scanning database is reducing the efficiency of mining algorithm. Jiawei Han [21][22] introduced a new efficient data mining algorithm, which is different from Apriori algorithm, to mine frequent patterns. Because FP-trees contain all frequent itemsets, it not only can avoid generating a lot of candidate patterns which don't exist actually, but also needn't scan database repeatedly. Then it can only mine frequent itemsets in FP-trees, so as to improve the efficiency of data mining greatly.

Also, Kutty [23] proposed PCITMiner algorithm to mine induced subtrees by building mapping which is established on the first sequence of unordered trees. According to Kutty in [23]'s ideas, the technique of searching for frequent subtrees includes two aspects: generate-and-test technique and pattern growth technique. The main idea of the generate-and-test technique is to generate candidate trees, then to test whether these subtrees are frequent or not by traveling data sets. By extending a frequent tree or combine two frequent subtrees, generating method for candidate subtrees can generate a new candidate subtree, and testing method is to measure whether the candidate subtrees are frequent or not by matching with tree database. Through repeating to scan the tree database, pattern growth technique is to find certain expansion location of frequent tree, until all the frequent subtrees are found. PCITMiner is based on pattern growth technique.

This paper aims to provide a ground on which such introduction as well as analysis could be built a novel algorithm, MFPTM, which is based on pattern growth technique and FP-tree principle, to systematically research on fast mining maximal frequent subtrees from tree database which consists of unordered trees. From what has been introduced above, MFPTM algorithm: Firstly, based on fusion compression, MFPTM algorithm retains the subtrees which only have frequent nodes. Secondly, according to FP-tree principle, MFPTM algorithm mines maximal frequent subtrees. In the process of mining frequent subtrees, searching space could have been saved toward proposing MFPTM algorithm, which could avoid generating candidate subtrees meanwhile improving the efficiency of the mining algorithm.

2. Problem statement

A tree is an acyclic directed graph. Generally, we can denote a tree as $T = \langle N, B, R \rangle$, where $N$ is the set of nodes, $B$ is the set of directed edges, and $R \in N$ is the root of the tree. For each directed edge $b = \langle n_1, n_2 \rangle \in B$, where $n_1, n_2 \in N$, we call $n_1$ the parent of $n_2$, and $n_2$ a child of $n_1$. If all the children of each node are ordered, the tree is called an ordered tree, otherwise, it is an unordered tree. This paper researches on the unordered trees.

Path and Root Path [19] A path is composed of a series of directed edges. A rooted, unordered tree $T = \langle N, B, R \rangle$, $p = \langle \langle n_1, n_2 \rangle, \langle n_2, n_3 \rangle, \ldots, \langle n_{k-1}, n_k \rangle \rangle$, where $n_i \in N (1 \leq i \leq k)$, and $k$ is serial number of nodes on the path. Then, we can represent the path just by ordering the nodes on the path $\langle n_1, n_2, \ldots, n_k \rangle$. If there exists a path which starts from $N_0$, we call $N_0$ an ancestor of another node $N_i$, and $N_i$ a descendent of $N_0$. If a path starts from the root node, we call the path a root path. Since there is only one root path to other node, each root path in a tree can be uniquely identified by the last node on the path. In Figure 1, node E represents one root path $\langle 1, 4, 5 \rangle$.

Root Subtree [19] A tree $T' = \langle N', B', R' \rangle$ is a subtree of another tree $T = \langle N, B, R \rangle$, if and only if there exists a mapping $\theta$: $N' \to N$ such that (1) for each node $x \in N'$, $l(x) = l(\theta(x))$, where $l$ is the labeling function, (2) for each directed edge $b = \langle x, y \rangle \in B'$, $\theta(x), \theta(y) \in B$. If a subtree has the same root as the tree, i.e., $R' = R$, we call such subtree a root subtree.

Induced Subtree [3] A tree $T' = \langle N', B', R' \rangle$ is a subtree of another tree $T = \langle N, B, R \rangle$, if and only if there exists a mapping $\theta$: $N' \to N$ such that (1) $\theta$ preserves the parent-child
relationships, as well as vertex labels, (2) for each node \( x \in N' \), \( l(x) = l(\theta(x)) \), where \( l \) is the labeling function, and (3) for each directed edge \( b = < x, y > \in B' \), \( \theta(x), \theta(y) \in B \). Such subtrees are called induced subtrees. In this paper, induced subtree is considered as a data mining object.

**Hidden Subtree** A tree \( T_1' = < N_1', B_1', R_1' > \) is a subtree of another tree \( T_1 = < N_1, B_1, R_1 > \), and the other tree \( T_2 = < N_2, B_2, R_2 > \), if and only if there exists a mapping \( \theta: N_1' \rightarrow N_1 \) such that (1) for each node \( x \in N_1' \), \( l(x) = l(\theta(x)) \), where \( l \) is the labeling function, (2) for each directed edge \( b = < x, y > \in B_1' \), \( \theta(x), \theta(y) \in B_1 \), and (3) the subtree \( T_1' \) has the same root as the tree, i.e., \( R_1' = R_2 \). Such subtrees are called hidden subtrees.

A hidden subtree is a subtree of a rooted, unordered tree, and contains relative information of the other rooted, unordered tree which has the same root label. Meanwhile, in the process of mining maximal frequent subtrees, hidden subtrees can affect the width of maximal frequent subtrees which have the same root node. In other words, maximal frequent subtrees can contain more root subtrees. Thus, it not only affect the accuracy of final results of data mining, but also can make the mining results retain more complete information. And hidden subtrees play an important part in the process of mining maximal frequent subtrees. So, in this paper hidden subtree is also considered as a data mining object. This is because it is as important as induced subtree. All in all, in this paper, a subtree is referred to as an induced subtree or an hidden subtree, unless otherwise indicated explicitly.

**Support** Given a tree database \( D \), and a subtree \( S \), the frequency of \( S \) in \( D \), \( freq_0(S) = \sum_{T \in D} freq(T, S) \), where if \( S \) occurs in \( T \), \( freq(T, S) = 1 \), otherwise 0. The number of \( freq_0(S) \) is the total number of occurrences of \( S \) in \( D \). The support of \( S \) in \( D \), \( sup_0(S) = \frac{freq_0(S)}{|D|} \), where \( |D| \) is the total number of rooted, unordered trees in \( D \).

3. MFPTM algorithm

MFPTM, an improved algorithm which based on fusion compression and FP-tree principle, was proposed in this paper to determine a better way to mine maximal frequent subtrees as following aspects—Firstly, MFPTM needs to pretreat each rooted, unordered tree in tree database \( D \), remove infrequent nodes, and retain tree data set which only contains frequent nodes. Secondly, in order to save data storage space, MFPTM requires to deal with the remaining tree data set by fusion compression. Finally, based on FP-tree principle, MFPTM mine maximal frequent subtrees. Section 3.1 mainly describes the general process of MFPTM algorithm, and the related concepts are defined in Section 3.2 and Section 3.3.

3.1. MFPTM algorithm description

**Input:** rooted, unordered tree database: \( D \), support threshold: \( S_{\text{min}} \)

**Output:** all maximal frequent subtrees: \( \text{maxFST} \)
Method:
(1) Freone = GetFreSingleNode ( D, Smin ) ;
    /* Get all the frequent nodes and the updated tree data set: D' */
(2) FCTrees = constructFCTree ( D', Freone ) ;
    /* Construct each fusion compression tree and get the fusion compression tree set */
(3) FST = NULL ;
(4) for each tree in FCTrees do begin
    (5) FrePaths = trimFCTrees ( tree ) ;
        /* Trim each infrequent directed edge of a fusion compression tree and get frequent paths */
    (6) MPTrees = FPTrees ( FrePaths ) ;
        /* Construct MP tree based on FP-tree principle and get the MP tree set ( MPTrees ) which contain all the information of the frequent paths */
    (7) FST = generateFST ( MPTrees, FST ) ;
        /* Get all the frequent subtrees ( FST ) */
    end for
(8) maxFST = dealMax ( FST ) ;
    /* Remove the frequent subtree which is included by another frequent subtree and get the maximal frequent subtrees */

3.2. Construct fusion compression tree

Frequent Node
Given a tree database D, and support threshold Smin, if the support of node n_i in D, sup(n_i) ≥ Smin, we call n_i frequent node.

On condition that maintaining integrated tree structure information, fusion compression process can save data storage space. Therefore, each node in a tree of tree database D must record the complete information. Then we denote a node as N = < label, parent, children, positions >, where label is the label on current node, parent is the parent of current node, children is a set containing all the children of current node, and positions is the position mark sequence which records all the position information of current node discovered in tree database.

Each element of positions has two attributes: treeID and nodeID, where treeID is a serial number of the current node, corresponding to a tree of tree database D, and nodeID is a node label which represents current node, corresponding to a rooted, unordered tree.

Fusion compression is mainly devoted to the presentation as it stands: Traverse each rooted, unordered tree T_i = < N_i, B_i, R_i > which contains only frequent nodes from rooted, unordered tree set, and find another rooted, unordered tree T_j = < N_j, B_j, R_j > ( including induced subtrees and hidden subtrees ) which has the same root label as T_i. If all the child node sets of root node R_i don’t contains certain child node N_cj of root node R_j, then we need to add node N_cj to the child node sets of root node R_i, otherwise update the attribute positions of node N_cj which has the same node label as node N_cj in the child node sets of root node R_j. Then we need to similarly deal with each child node of node R_i until all the nodes of tree T_j are traversed.

Fusion Compression Tree
Given a tree database D, and a tree T_i' = < N_i', B_i', R_i' > is a fusion compression tree, if and only if according with the following conditions such as (1) there exists two different rooted, unordered trees such as T_i = < N_i, B_i, R_i >, T_j = < N_j, B_j, R_j >, 0 < i < j ≤ |D|, where |D| is the total number of rooted, unordered trees in tree database D, (2) R_i = R_j (3) |T_i| > 1 and |T_j| > 1 ( This is because it is not significant to compress two rooted, unordered trees which have only single node ), (4) Starting from root node, each node and each child node of current node must be dealt with fusion compression. Eventually, a new rooted, ordered tree is formed, which is called fusion compression tree.

For example, given a tree database D, and support threshold Smin = 60%, by calculating the support, we can calculate that sup(J) < Smin, so we need to remove node J from the database. Because node J is an infrequent node. And we can obtain the trimmed database. Figure 2 shows the original trees ( upper part ) and the trimmed trees ( lower part ). Meanwhile we can see one hidden subtree in red dotted area. In Figure 2, we can discover that two of the trimmed trees are the same as original trees: T1 and T2. T3(2) and T3(4) are modified from the original tree T3.
The treeID of new trees: T3(2) and T3(4) are the same as the original tree T3, but they have the different nodeID which is the number in bracket. So we can distinguish the new trees and know that they are different branches in the original tree by different nodeID of each node in the original tree. Figure 3 shows the fusion compression tree which is formed after fusion compression.

3.3. Mining maximal frequent subtrees

Frequent Path A Tree $T = < N, B, R >$, $p = < < n_1, n_2 >, < n_2, n_3 >, \ldots, < n_{k-1}, n_k > >$, where $n_i \in N$ ( $1 < i \leq k$ ), $< n_{i-1}, n_i >$ is a directed edge, and $k$ is serial number of nodes on the path. Path $p$ is frequent path, if and only if each directed edge ($< n_{i-1}, n_i >$) of path $p$ is frequent.

Established on the basis of frequent paths, the process of mining frequent subtrees is further treatment of fusion compression tree set, so we must make sure that every path, which starts from root node of each fusion compression tree to any other leaf node, is frequent. But after the process of fusion compression, we cannot ensure that each directed edge of a tree from fusion compression tree set is frequent. Thus, it is necessary to trim each fusion compression tree to remove all the infrequent directed edges. Figure 4 shows the trimmed fusion compression trees.
Fig. 4. Trimmed fusion compression trees

**MP Tree** (maximal path tree) Given a trimmed fusion compression tree \( T = \langle N, B, R \rangle \), and root path \( p = \langle < n_1, n_2 >, < n_2, n_3 >, \ldots, < n_{k-1}, n_k > \rangle \), where \( n_i \in N \ (1 < i \leq k) \), \( < n_{k-1}, n_k > \) is a directed edge, \( k \) is serial number of nodes on the path, and \( n_k \) is a leaf node of tree \( T \). Let node \( n_{j-1} \) and \( n_j \) (1 < j ≤ M) represent leaf nodes of tree \( T \), where \( M \) is the total number of leaf nodes of tree \( T \), and the attribute positions represents the sequence of the same location information between node \( n_{j-1} \) and node \( n_j \). If \( \text{sup}(\text{positions}) \geq S_{\text{min}} \), there are two kinds of circumstances as following: (1) There is not a rooted, ordered tree which takes node \( n_{j-1} \) as a root node, we need to create a new tree whose root node is node \( n_{j-1} \), and set node \( n_j \) as a child node of node \( n_{j-1} \). (2) There exists a rooted, ordered tree which takes node \( n_{j-1} \) as a root node, we only need to set node \( n_j \) as a child node of node \( n_{j-1} \), such tree is called MP tree.

MP tree contains all the frequent paths starting from root node \( R \) of a trimmed fusion compression tree \( T = \langle N, B, R \rangle \). Meanwhile, based on FP-tree principle, MP tree is constructed. So we can mine maximal frequent paths from MP tree set. Containing no leaf node, each root path of MP tree is similar to the conditional pattern base of FP-tree. In consideration of it, we can mine unique root path by each unique leaf node of MP tree, and each root path is frequent. Also, each node of root path from MP tree represents a root path of the rooted, ordered tree database where infrequent nodes have been removed.

A subtree \( S \) of each rooted, ordered tree \( T \) is represented by the leaf nodes of tree \( T \). Each root path represents \( 1 \)-itemsets, and a root subtree having \( n \) leaf nodes represents \( n \)-itemset. On account of it, according to FP-tree principle, MFPTM Algorithm constructs MP tree which is similar to FP-tree from all the frequent \( 1 \)-itemsets. So, MP tree construction is extended on the basis of all the frequent \( 1 \)-itemsets, finally each root path consisting of \( n \) nodes is \( n \)-itemset. Thus, we can denote a node of MP tree as \( \text{MP}_N = \langle \text{nodeID}, \text{Pointer}, \text{intersecPositions} \rangle \), where nodeID is the serial number of the node from frequent \( 1 \)-itemset, Pointer is a pointer which is pointing to the MP tree which takes current \( 1 \)-itemset as root node, intersecPositions is the intersection of nodes (starting from root node to current node in MP tree)' attribute positions: the purpose of that is to extend frequent path by pattern growth technique. Also, it can construct \( n \)-itemset by joining \( n \) \( 1 \)-itemsets.

Mining frequent subtrees mainly has two steps: (1) Running FPTrees can get MP tree set which contains all the information of frequent paths, (2) Running generateFST can get the frequent subtrees from MP tree set. The Algorithms are described as following:

**FPTrees( FrePaths )**

**Input:** all the frequent path set: FrePaths

**Output:** MP tree set: MPTrees

**Method:**

1. (initMPTree ( ) ;
2. (for each ( int i = 1 ; i < n ; i++ ) /* n is the size of FrePaths */
3. (for each ( int j = 0 ; j < i ; j++)
4. (positions = Merged ( FrePaths.p{i} , FrePaths.p{j} ) ; /* get the same position set : positions between them */

- 196 -
Mining Maximal Frequent Subtrees based on Fusion Compression and FP-tree
Xiaoke Wu, Xiongfei Li

if ( sup ( totalnum ) ≥ Smin ) then /* totalnum is the size of positions */
if ( FrePaths.pj.treeID < 0 ) then /* There doesn’t exist the corresponding MP tree */
Add a new MP tree to MPTrees ;
else
Update the corresponding MP tree of MPTrees ;
}
if ( !flag ) then /* flag represents the existence of the path which can be merged with it*/
Add a new MP tree to MPTrees ;
}
generateFST ( MPTrees, FST )
Input: MP tree set: MPTrees and FST (initial value is null )
Output: The frequent subtrees: FST
Method:
init() ;
for each ( int i = 0 ; i < m ; i ++ ) { /* m is the size of MPTrees */
childrens = getRootChildren ( MPTrees.mp ) ;
/* Get root paths of each MP tree of MPTrees */
for each ( int j = 0 ; j < k ; j ++ ) { /* k is the size of childrens */
for each node in childrens.pj
Add findChildPath ( node ) to childPaths ;
/* Get all the paths ( childPaths ) of original tree based on the current node */
FST’ = constructFST ( childPaths ) ;
/* Get the current frequent subtree: FST’ based on each path of the childPaths */
Add FST’ to FST ;
}
}
Maximizing If a frequent subtree S of frequent subtrees cannot be included by another frequent subtree S’, such frequent subtree S is maximal frequent subtree, otherwise, frequent subtree S will be removed from frequent subtrees.

4. Experiments and results

Two variations of the MFPTM algorithm ( MFPTM algorithm, MFPTM_noHST algorithm: not includes hidden subtree ) were compared with [19]’s PathJoin algorithm to provide performance analysis of MFPTM algorithm. All the experiments were conducted on Intel Core2 2.93GHz with 1GB main memory and running Eclipse3.5.

The synthetic data sets were generated by using the method in [3]. And real cslogs data set is also got from [3]’s CSLOGS Data set. Experiment one: Given the support threshold Smin: 1%, and synthetic data sets from 5K~60K. Figure 5 illustrates that MFPTM algorithm, MFPTM_noHST algorithm and PathJoin algorithm mine the total number of frequent subtrees.
As shown in Figure 5, we can find out that MFPTM_noHST algorithm and PathJoin algorithm mine the same total number of frequent subtrees. This is because they both cannot take hidden subtree as a mining object. When the data set increases from 5K to 60K, the total number of frequent subtrees by three algorithms demonstrates linear growth. In the 5K ~ 30K range, due to the small data set, the probability of hidden subtrees which can be discovered is high. Thus, hidden subtrees make significant influence on the total number of frequent subtrees. As a result, linear gradient of MFPTM is obviously much bigger than PathJoin algorithm and MFPTM_noHST algorithm. However in the 30K ~ 60K range, owing to the larger data set, the probability of hidden subtrees discovered becomes lower. So the influence of hidden subtrees affecting total number of frequent subtrees has become weak. Then linear gradient of MFPTM algorithm is approximately the same as PathJoin algorithm and MFPTM_noHST algorithm. Overall, mining effect of MFPTM algorithm is be superior to PathJoin algorithm.

Experiment two: Given synthetic data set: 75K, and support threshold $S_{\text{min}}$: 0.1%~2.5%. Figure 6 shows the total number of frequent subtrees which are mined by MFPTM algorithm, MFPTM_noHST algorithm and PathJoin algorithm.

In Figure 6, with the support threshold $S_{\text{min}}$ increasing from 0.1% to 2.5%, the total number of frequent subtrees of the three algorithms decreased accordingly. In the cases of the same support threshold $S_{\text{min}}$, MFPTM algorithm is better than PathJoin algorithm and MFPTM_noHST algorithm. And the smaller support threshold $S_{\text{min}}$ is, the more obvious the advantage is. This is because in the cases of the same data set and a small support threshold (the frequency of frequent subtrees is small), the probability of hidden subtrees discovered is higher. Thus, hidden subtree can make more significant influence on the total number of frequent subtrees. With the support threshold $S_{\text{min}}$ increasing, the probability of hidden subtrees discovered goes down, and influence of hidden subtrees affecting total number of frequent subtrees has become weaker, finally the total number of frequent subtrees of MFPTM is approximately the same as PathJoin algorithm and MFPTM_noHST algorithm.

Experiment three: Given real cslogs data set: 6K~18K, and the support threshold $S_{\text{min}}$: 1%. Figure 7 illustrates that MFPTM compares the execution time of frequent subtrees mining with MFPTM_noHST algorithm and PathJoin algorithm.

---

**Figure 6.** Data sets: 75K

**Figure 7.** Support threshold: $S_{\text{min}} = 1\%$
From Figure 7, with real cslogs data set increasing from 6K to 18K, the execution time of three algorithms demonstrates linear growth. When data set increases gradually, the execution time of MFPTM algorithm, MFPTM_noHST algorithm becomes less than PathJoin algorithm, and the execution time of MFPTM_noHST algorithm is less than MFPTM algorithm. The reason is that MFPTM_noHST algorithm cannot include hidden subtrees, so it only need to deal with the less number of frequent subtrees, and save the execution time. All in all, dealing with large quantity data set, MFPTM algorithm reflects obviously that it is time-saving.

The above the experimental results show that MFPTM algorithm is superior to PathJoin algorithm under the same conditions of the same data set and the same support threshold. The reasons are presented below: (1) In process of mining frequent subtrees, PathJoin algorithm only considers the root subtrees which have the same root node label, and neglects hidden subtrees, leading to reduce the total number of frequent subtrees, (2) With constructing candidate subtrees, PathJoin algorithm based on Apriori principle has certain blindness in the connecting step, and it must waste much time in testing candidate subtrees which don’t exist actually. But in the process of mining frequent subtrees, MFPTM algorithm not only considers hidden subtrees, but also is based on FP-tree principle to generate MP tree to mine frequent subtrees. So, MFPTM algorithm needn’t generate candidate subtrees. Finally, MFPTM algorithm can mine more total number of frequent subtrees and save the execution time, so as to enhance the accuracy and efficiency of data mining.

5. Conclusion

We are seeing the increasingly emergence of mining frequent pattern, which relative research activities have spawned great advances in the field of data mining, along with huge quantities of tree structure data existed in network, therefore on a systemic scale emphasis on the proposition of an advanced algorithm for mining frequent subtrees would be of great significance. Unlike Apriori, due to analysis mentioned above, we may safely conclude that MFPTM algorithm is advisable. Next we will popularize MFPTM algorithm to other fields of data mining.

6. References


[21] Jiawei Han, Jian Pei, Yiwen Yin, “Mining Frequent Patterns without Candidate Generation”, In Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, pp.1–12, 2000.

[22] Jiawei Han, Jian Pei, Yiwen Yin, Runying Mao, “Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach”, Data Mining and Knowledge Discovery, vol. 8, no. 1, pp. 53–87, 2004.