Expert Systems with Applications 42 (2015) 94–100

Contents lists available at ScienceDirect

Expert Systems with Applications

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

Research on a frequent maximal induced subtrees mining method based on the compression tree sequence

Jing Wang, Zhaojun Liu, Wei Li, Xiongfei Li *

College of Computer Science and Technology, Jilin University, China

ARTICLE INFO

Article history:
Available online 9 August 2014

Keywords:
Data mining
Frequent subtree
Induced subtree
Maximal subtree
Compression
CFMIS

ABSTRACT

Most complex data structures can be represented by a tree or graph structure, but tree structure mining is easier than graph structure mining. With the extensive application of semi-structured data, frequent tree pattern mining has become a hot topic. This paper proposes a compression tree sequence (*CTS*) to construct a compression tree model; and save the information of the original tree in the compression tree. As any subsequence of the *CTS* corresponds to a subtree of the original tree, it is efficient for mining subtrees. Furthermore, this paper proposes a frequent maximal induced subtrees mining method based on the compression tree sequence, CFMIS (compressed frequent maximal induced subtrees). The algorithm is primarily performed via four stages: firstly, the original data set is constructed as a compression tree model; then, a cut-edge reprocess is run for the edges in which the edge frequent is less than the threshold; next, the tree is compressed after the cut-edge based on the different frequent edge degrees; and, last, frequent subtree sets maximal processing is run such that, we can obtain the frequent maximal induced subtree set of the original data set. For each iteration, compression can reduce the size of the data set, thus, the traversal speed is faster than that of other algorithms. Experiments demonstrate that our algorithm can mine more frequent maximal induced subtrees in less time.

1. Introduction

Structures of data are becoming increasingly complicated with the fast development of the Internet and storage technology. Most data with a complicated structure can be represented by a tree or graph structure. With the extensive application of semi-structured data, the research priority of frequent pattern mining has expanded from frequent item set mining (Liu, Lin, & Han, 2011; Wang & Chen, 2011; Yang & Huang, 2010) to frequent subtree mining (Balcázar, Bifet, & Lozano, 2010; Li, Li, & Zhao, 2010) and frequent subgraph mining (Hou, Ong, Nee, et al., 2011; Jiang, Coenen, & Zito, 2013). The complexity of tree mining is lower than that of graph mining, and tree mining algorithms can be applied to graph mining instances that contains a small amount of rings, so it is of great significance to be able to mine data represented by a tree structure. Frequent subtree mining has become an important field of data mining research.

Frequent subtree mining is the process of mining a subtree set from a given data set that satisfies user attention (support or frequent degree). Frequent subtree mining has a high value in computer vision, text acquisition, Web log analysis, XML document analysis, XML association rule mining, XML query pattern mining territory, semi-structured data analysis, analysis of biometric information and structural analysis of compounds. For example, through application of the frequent subtree mining method to web logs, users' degree of interest can be known by deep analysis of the information represented in a tree structure, and it is convenient to optimize the structure of the network. In analyzing the XML document, the frequent subtree mining method can find a frequent data structure that is implicit and represented by a tree structure.

When mining frequent subtrees on a given tree data set, the number of frequent subtrees increases exponentially with the decrease of the minimum degree, and the frequent subtrees in the result data set contains redundant information, so simplifying the result set is necessary. Finding the closed subtrees and the largest subtrees are two common methods to simplify the result set. Closed subtree T can express information that all other subtrees closed by T can express, so subtrees closed by T can be deleted from the result set. Recently, there has been great interest in mining closed subtrees, and many efficient closed subtree mining algorithms have been proposed. Maximal subtree T' can express information that all other subtrees maximized by T' can express, so the number of subtrees in the result set can be minimized. This is significant for the growth of large-scale data.

* Corresponding author.

http://dx.doi.org/10.1016/j.eswa.2014.07.053
0957-4174/

In this paper, we propose an efficient method, CFMIS, based on the compression tree sequence that focuses on mining frequent maximal induced subtrees. The algorithm is primarily performed via four stages: firstly, the original data set is constructed as a compression tree model; then, a cut-edge reprocess is run for the edges in which the edge frequent is less than the threshold; next, the tree is compressed after the cut-edge based on the different frequent edge degrees; and, last, frequent subtree sets maximal processing is run such that, we can obtain the frequent maximal induced subtree set of the original data set. We have demonstrated experiment that the proposed algorithm in this paper can mine frequent maximal induced subtrees in a rapid and efficient way.

2. Related work

Methods for mining frequent subtrees are classified into mining methods based on generation–test strategy and mining methods based on pattern growth strategy. The main idea of mining method based on generation–test strategy is to produce the corresponding candidate subtree first, and then traverse the corresponding data set to test whether the candidate subtree is frequent. This strategy is mainly used to expand or merge subtrees to generate a new candidate subtree, and then to test whether the new candidate subtree is frequent or not by traversing the database. Mining methods based on pattern growth strategy iterate through the database to find the frequent tree extension points through repeated search, until mining out all hidden frequent subtrees.

EvoMiner (Deepak, Fernández-Baca, Tirthapura, et al., 2011) is an Apriori-like level-wise method, which uses a novel phylogeny-specific constant-time candidate generation scheme, a fingerprinting based technique for downward closure, and a lowest-common-ancestor-based support counting step. Bui, Hadzic, and Tagarelli et al. introduce an associative classification method (Bui, Hadzic, Tagarelli, et al., 2014) based on a structure preserving flat representation of trees in which subtrees are constrained by the position in the original trees, leading to a drastic reduction in the number of rules generated, especially with data that has great structural variation among tree instances. Nguyen, Doi, and Yamamoto propose a new top-down method (Nguyen, Doi, & Yamamoto, 2012) for mining unordered closed tree patterns from a database of trees such that every mined pattern must contain a common piece of information in the form of a tree specified by the user. Lee and Lee introduce a new type of problem called the frequent common family subtree mining problem (Lee & Lee, 2013) for a collection of leaf-labeled trees in their paper and present some characteristics for the problem. It proposes an algorithm to find frequent common families in trees. Nguyen and Yamamoto propose a novel and efficient incremental mining algorithm (Nguyen & Yamamoto, 2010) for closed frequent labeled ordered trees. They adopt a divide-and-conquer strategy and apply different mining techniques in different parts of the mining process. The algorithm requires no additional scan of the entire database. PTG (Li & Yang, 2011) (partition tree growth) is put forward based on the partition principle. In the PTG algorithm, the database is divided into several partitions, the TG (tree growth) algorithm creates the local frequent subtrees of every partition, and then creates the global frequent subtrees according to the global support value for filtering. Deng, Lv proposed Nodeset (Deng & Lv, 2014), a novel structure where a node is encoded only by pre-order or post-order code to solve the memory-consumption problem. Xiao and Yao proposed the classic PathJoin (Xiao & Yao, 2003) algorithm based on the Apriori algorithm to effectively implement mining of maximal frequent subtrees. The algorithm uses a compact data structure called FST-Forest, which compresses the trees and retains the original tree structure. PathJoin generates candidate subtrees by joining the frequent paths in FST-Forest.

MFPTM (Wu & Li, 2011) constructs an MP[1] tree based on fusion compression and the FP[2] tree principle to mine maximal frequent subtrees. MFPTM is an advanced algorithm as it solves the problem of frequent pattern mining based on the Apriori algorithm which generates a large quantity of candidate patterns and improves the efficiency of mining frequent subtrees. MFPTM outperforms the classic algorithm PathJoin. The proposed algorithm, CFMIS, and the state-of-the-art MFPTM both focus on frequent maximal subtrees, not just frequent subtrees. Furthermore, the two algorithms both retain subtrees that only contain frequent nodes by compression, although the compression methods are different. Therefore, the two algorithms are compared by experiments on both synthetic and real datasets.

3. CFMIS algorithm

In this section, we provide the definitions for some general and specific concepts that will be used in the remainder of the paper. We also give the details of our algorithm.

3.1. Prepared knowledge

A tree is generally defined as an acyclic connected graph, and they can be classified according to their structural characteristics. If sibling nodes of tree T are ordered, the tree T is called an ordered tree; otherwise, it is known as an unordered tree. If the nodes in the tree contain labels, the tree T is called a label tree, otherwise, it is known as a non-label tree. If the sibling nodes of the same parent node have no repeats, the tree T is called an attribute tree, otherwise, it is known as a non-attribute tree. An unordered label attribute tree is denoted as *ULAT*,

Definition 1 (*ULAT*). *An* unordered tag attribute tree is an acyclic connected graph, which is denoted as $ULAT = (V, E, \Sigma, L, r)$, where V is the node set; E is the edge set in which $(x, y) \in E$ represents that node x is the parent of node y; Σ is the label set in which elements can be compared and sorted; L is the mapping from the node set to label set, $L : V \rightarrow \Sigma$, and sibling nodes of the same parent node without the same label; and r is the root node.

The CFMIS algorithm addresses unordered label attribute tree sets, and the 'tree' mentioned below is *ULAT*, assuming that there is no repeat label in a same tree.

Definition 2 (*Induced subtree*). A tree $T' = (V', E', \Sigma', L', r')$ is an induced tree of $T = (V, E, \Sigma, L, r)$, denoted as $T' \subset T$, if and only if $V' \subset V$; $E' \subset E$; $\Sigma' \subset \Sigma$; $L' \subset L$.

Fig. 1 shows an induced tree of a source tree.

Reserving parent–child relationships between nodes in the source tree and the absence of affection among sibling nodes are features of induced trees. The CFMIS algorithm addresses the induced trees of an original data set.

Definition 3 (*Frequent subtree*). Let the tree structure data set be $D = \{T_1, T_2, \ldots, T_n\}$. ε is the minimum frequency threshold, $T' \subset T_i$, where $i \in [1, n]$, $T_i \in D$. $Occ(T, T')$ represents whether T' occurs in T, if T' occurs in T, then $Occ(T, T') = 1$, and else $Occ(T, T') = 0$. The frequency of T' is denoted as $Frq(T')$, and $Frq(T') = \sum_{i=1}^{n} Occ(T, T')$. T' is a frequent tree if and only if $Frq(T') \geqslant \varepsilon$.

Definition 4 (*Maximal subtree*). Let the tree structure data set be

[1] MP (maximal path) tree is proposed in reference Wu and Li (2011), each path from the root node to a leaf node in an MP tree is frequent.

[2] FP (frequent pattern) tree, constructs different branches of an FP tree by traversing each item in the transaction dataset.

Fig. 1. Source tree and induced tree.

$D = \{T_1, T_2, \ldots, T_n\}$, ε is the minimum frequency threshold, F is the frequent subtree set of D. T' and T'' are two frequent subtrees, $T', T'' \subset F$, T' maximizes T'' (denoted as $T'' \rightarrow T'$) if and only if $T'' \subset T'$, $Frq(T') \leqslant Frq(T'')$, and there is no $T''' \overset{m}{\subset} F$ to make $T'' \underset{m}{\rightarrow} T'''$, and T' is called a maximal subtree.

3.2. Four stages in CFMIS

3.2.1. Stage 1: construct the compression tree model

The CFMIS algorithm constructs a compression tree set from the original data set.

Definition 5 (*Compression tree sequence (CTS)*). *CTS* is an ordered sequence composed by $label^{pd}$, $CTS = \left(label_0^{pd_0}, label_1^{pd_1}, \ldots, label_n^{pd_n}\right)$, where $label^{pd}$ is an element of the compression tree sequence, *label* is the element tag, *pd* is the index pointing to the parent node, and each element except the root element, has an associated element that is called the parent element. Let the parent element of $label_i^{pd_i}$ be $label_j^{pd_j}$, and then $pd_i = |i - j|$, where $label_0^{pd_0}$ is the root element.

For example, for a compression tree sequence $CTS = 1^0 2^1 5^1 11^2 8^2 9^3$, let η be a function mapping from a compression tree sequence to a tree; then, the tree mapping from $\eta(cts)$ is shown in Fig. 2.

Theorem 1 (Proof). *If there is only one node in ULAT, then* $CTS = \left(label_0^{pd_0}\right)$. *To normalize a ULAT that consists of one root node and n subtrees connected to the root, sort them as* $T_1, T_2, \ldots, T_n$ *according to the root label. Suppose that these subtrees can be normalized to* $CTS_1, CTS_2, \ldots, CTS_n$.

According to Definition 5, $CTS = \left(label_0^{pd_0}, CTS_1[0], \ldots, CTS_n[0], CTS_1[1 \ldots m_1), \ldots, CTS_n[1 \ldots m_n))\right)$, where $CTS_i[0]$, $1 \leqslant i \leqslant n$, which is the first element of CTS_i, and CTS_i $[1 \ldots m_i)$ is the sequence of remaining elements in CTS_i. m_i is the number of elements in CTS_i. The format of elements in *CTS* is $label_{i,0}^0$ or $label_{i,j}^{pd_j+i}$, $0 < j < m_j$, where $label_{i,j}^{pd_j}$ is the jth element in CTS_i.

For any i, there is $label_{i,0}^0$, so each subtree will be connected to the original root node. For the node $label_{i,j}^{pd_j}$ in any subtree, the root node in *CTS* is setback one parent node and $i - 1$ sibling nodes, so it will become $label_{i,j}^{pd_j+i}$. Above all, any *ULAT* can be normalized to *CTS*.

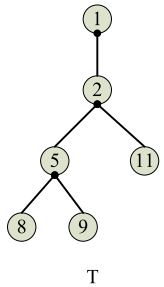

Fig. 2. The tree mapping from $\eta(cts)$.

Definition 6 (*Frequent edge degree*). Let the tree structure data set be $D = \{T_1, T_2, \ldots, T_n\}$, $T_i = (V, E, \sum, L, r) \in D$. The frequent edge degree of $(x, y) \in E$ is the number of times the edge with the same parent label and child label, (x, y), appears in the data set, denoted as $EFrq(x, y)$.

Definition 7 (*Compression tree*). A compression tree is an acyclic connected graph $CT = (V, E, CTSS, \Gamma, F, r)$, where V is the nodes set, E is the edge set, *CTSS* is the compression tree sequence set, L is the mapping from the node set to the compression tree sequence set, F is the frequent edge degree set, and r is the root node.

The compression tree set from Fig. 3 is shown in Fig. 4. Each node label format is $(f, label^{pd})$, where the first part, $f \in F$, represents the edge frequent degree, the second part represents the element of the compression tree sequence, and, in the initial state, the second part of the node is $label^0$.

3.2.2. Stage 2: cutting edge

This stage is divided into two subprocesses, trim edges and clean-up edges. First, trim the edges for which the edge frequent degree is less than ε, and delete the single node. Then, if node y becomes a single node, delete it; the total number of nodes in the data set will reduce. If node y is the root number of another subtree, add the subtree rooted on node y to the data set; the total number of trees in data set will increase. After the trim edges process, the retained compression tree data set is denoted by CCD.

Suppose that $\varepsilon = 2$; for the above compression tree set, the edge frequent degrees of the edges in T1, T2, T4, T5 are greater than 2, so trim edges is unnecessary. The edge frequent degrees of edges (1,7) and (7,2) are less than 2, so trim them. After that, node 7 becomes single, so delete it. Node 1is the root node of another subtree, so add the subtree rooted on node 1 to the data set. T3 after trimming is shown in Fig. 5. Now, CCD includes T1, T2, T4, T5 and T3a, T3b.

3.2.3. Stage 3: find frequent subtrees

Definition 8 (*Length of CTS*). *The* number of elements in *CTS* is the length of *CTS*.

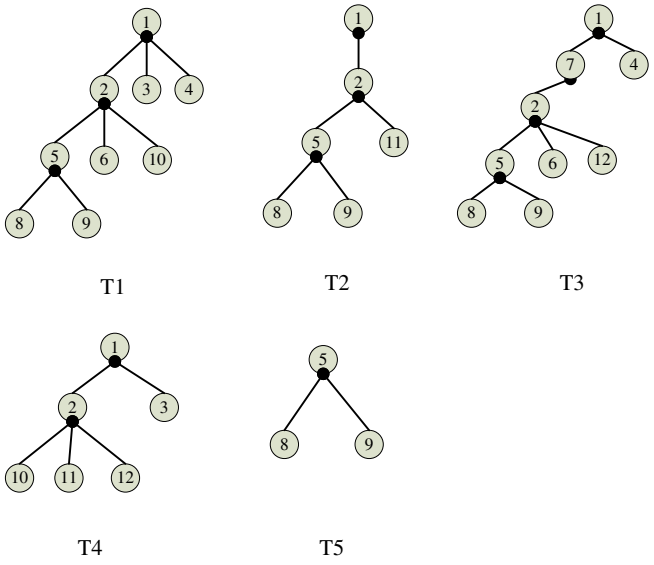

Fig. 3. Original data set.

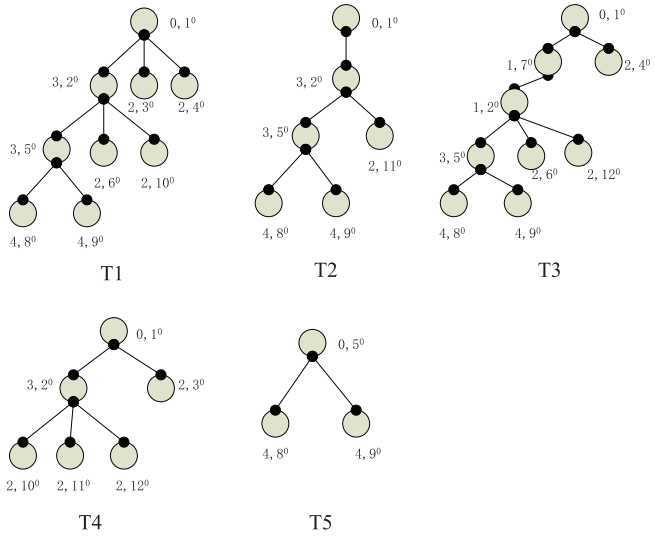

Fig. 4. Compression tree set.

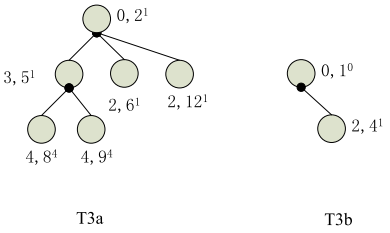

Fig. 5. T3 after cutting edge.

We can sort *CTSs* according to the length of each *CTS*. Compress CCD according to the descending edge frequent degree to obtain *CTSs*, and sort *CTSs* according to the length of each *CTS* from shortest to longest. Match the CTS_i with the *CTSs* following it; if matched (the CTS_i is obtained in another CTS_j), then the frequent degree of the T' represented by CTS_i is incremented by 1. When $Frq(T') = \varepsilon$, further comparison is unnecessary; add CTS_i to compression tree sequence set Ψ. If the total matched times m is less than ε, the CTS l needs subsequent processing: select $(\varepsilon - m - 1)$ *CTSs* from the *CTSs* not matched with CTS l to find the maximal public part l', and then the subtree corresponding to l' is a frequent subtree. It also should be added to Ψ.

For example, for the CCD in stage 2, suppose that $\varepsilon = 2$.

When $EFrq(x, y) = 4$, the edges (5, 8), (5, 9) are compressed; for T1, T2, T3a, T5, CFS is the same, i.e. $5^0 8^1 9^2$. This CFS appears 4 times, which is greater than 2, so its corresponding subtree is frequent. When $EFrq(x, y) = 3$, the edges (1, 2), (2, 5) are compressed, and the *CTSs* are $1^0 2^1$, $2^0 8^1 9^2$, $1^0 2^1 5^1 8^1 9^2$, $1^0 2^1 5^1 8^1 9^2$ after sorting. When $EFrq(x, y) = 2$, the edges (1, 3), (1, 4), (2, 6), (2, 10), (2, 11), (2, 12) are compressed, and the *CTSs* are $1^0 4^1$, $1^0 2^1 5^1 11^2 8^2 9^3$, $2^0 5^1 6^2 12^3 8^3 9^4$, $1^0 2^1 3^2 10^2 11^3 12^4$, $1^0 2^1 3^2 4^3 5^3 6^4 10^5 8^3 9^4$. For the *CTS* $1^0 2^1 5^1 11^2 8^2 9^3$, the matched time is 0. Now, select $(2 - 0 - 1) = 1$ *CTS* from the *CTSs* that are not matched with $1^0 2^1 5^1 11^2 8^2 9^3$ to find their maximal public part $l' = 2^0 5^1 8^1 9^2$, $1^0 2^1 11^1$, $1^0 2^1 5^1 8^1 9^2$.

Do this for the rest of the *CTSs* as well. Finally, the *CTSs* that represent subtrees are $\{5^0 8^1 9^2$, $1^0 2^1$, $2^0 8^1 9^2$, $1^0 2^1 5^1 8^1 9^2$, $1^0 4^1$, $2^0 5^1 8^1 9^2$, $1^0 2^1 11^1$, $2^0 12^1$, $2^0 5^1 6^2 8^2 9^3$, $1^0 2^1 3^2 10^2\}$.

Fig. 6 shows the main steps of frequent subtree mining algorithm proposed in this paper.

Frequent Subtree Mining Algorithm
Input: CCL, ε, $i = 0$,
Output: Ψ
Step 1: for each tree in CCD
while $EFrq(x, y) = Frq[i]$ /* $Frq[n]$ reserves all the edge frequent degree values after sorting in descending order*/
compress the edge (x, y)
put the produced CTSs into the set Ω
Step 2: sort the CTSs in Ω according to the length in ascending order
Step 3: select the CTSs with single edge into Ψ
$q = 0$, $m = 0$
for the rest CTSs in Ω
$p = q + 1$
while CTS_q is matched with CTS_p
$m = m + 1$ /*m is the matched times*/
$p = p + 1$
if $m \geq \varepsilon$, add CTS_q into Ψ
else CTS_q needs subsequent processing;
Step 4: $i = i + 1$, GOTO Step 1.

Fig. 6. Frequent subtree mining algorithm.

3.2.4. Stage 4: maximal stage

Sort the sequence in Ψ in descending order. Here, we can also perform the maximal process during each compression. The sequences in Ψ after sorting are $1^0 2^1 5^1 8^1 9^2$, $2^0 5^1 6^2 8^2 9^3$, $2^0 5^1 8^1 9^2$, $1^0 2^1 3^2 10^2$, $5^0 8^1 9^2$, $2^0 8^1 9^2$, $1^0 2^1 11^1$, $1^0 2^1$, $1^0 4^1$, $2^0 12^1$.

Theorem 2. *Let* $\Psi_m = \{T_{m1}, T_{m2}, \ldots, T_{mn}\}$ $(mm \leqslant n)$ *be the set composed by the trees corresponding with the longest sequences from* Ψ. *For any* $T_{mi} \in \Psi_m$, T_{mi} *must be the maximal subtree.*

Proof (*Reductio ad absurdum*). Suppose that T_{mi} is not the maximal subtree; then, there must be a subtree T' that maximizes T_{mi}, $T_{mi} \subset T'$; thus, the number of nodes in T' must not be less than that of T_{mi}.

If the number of nodes in T' is equal to that of T_{mi}, it means that $T' = T_{mi}$, so T' is also a maximal subtree. If the number of nodes in T' is larger than that of T_{mi}, it is a contradiction with the prerequisites. □

Theorem 3. *Known that* $T_2 \subset T_1$; *if* $T_3 \subset T_2$, *then* $T_3 \subset T_1$.

Proof. As $T_2 \subset T_1$, it is also true that $V_2 \subset V_1$, $E_2 \subset E_1$, $\sum_2 \subset \sum_1$, and $L_2 \subset L_1$. And as $T_3 \subset T_2$, so too is it true that $V_3 \subset V_2$, $E_3 \subset E_2$, $\sum_3 \subset \sum_2$, and $L_3 \subset L_2$. Then, $V_3 \subset V_1$, $E_3 \subset E_1$, $\sum_3 \subset \sum_1$, and $L_3 \subset L_1$, and thus $T_3 \subset T_1$. □

Lemma 1. *Known that* $T_2 \subset T_1$; *if* $T_3 \not\subset T_1$, *then* $T_3 \not\subset T_2$.

Proof. Directly implied by the converse negative proposition of Theorem 3.

Let *MF* be the frequent maximal induced subtree set and $\Psi' = \{T_1, T_2, \ldots, T_n\}$ be the tree set corresponding to the sequences in Ψ after sorting in descending order. According to Theorem 2, T_1 must be included in *MF*. Add T_1 into *MF* and delete it from Ψ'. For all of the rest of the trees T_i in Ψ', compare them sequentially with the trees T_j in *MF*. If a matched is found, it means that $T_i \subset T_j$; delete it from Ψ'. According to Lemma 1, for any of the rest trees T_k in Ψ',

as $T_i \subset T_j$, if $T_k \not\subset T_j$, then $T_k \not\subset T_i$. Thus, deleting T_i directly from Ψ' does not affect the mining. If no match is found, it means that $T_i \not\subset T_j$; add it into *MF* and delete it from Ψ'. □

Sort the sequences in Ψ obtained from Stage3 in descending order. We can obtain $\{1^0 2^1 5^1 8^1 9^2,\ 2^0 5^1 6^2 8^2 9^3,\ 2^0 5^1 8^1 9^2,\ 1^0 2^1 3^2 10^2,\ 5^0 8^1 9^2,\ 2^0 8^1 9^2,\ 1^0 2^1 11^2 1^0 2^1,\ 1^0 4^1,\ 2^0 12^1\}$. After the maximal stage, the corresponding frequent maximal induced subtrees are shown in Fig. 7.

4. Experiments and results

4.1. Experimental environment

All of the experiments were conducted on a 2.70 GHz Intel Core2 PC with 1 GB main memory, running the Windows XP operating system. All algorithms were implemented in C++ and compiled using the Visual Studio 2010 complier.

The synthetic data set used in this paper was generated by tree generator using the method in Zaki (2005). Parameters of the synthetic dataset are set as follows: $f = 10$ (f represents fan-out), $d = 10$ (d represents the depth of the tree), $n = 100$ (n represents the number of labels), $m = 100$ (m represents the number of tree nodes), and $t = 100{,}000$ (t represents the number of trees). The real data set was obtained from CSLOGS DATA SET, which is from a month-log of the data of the Rensselaer Polytechnic Institute's web site.

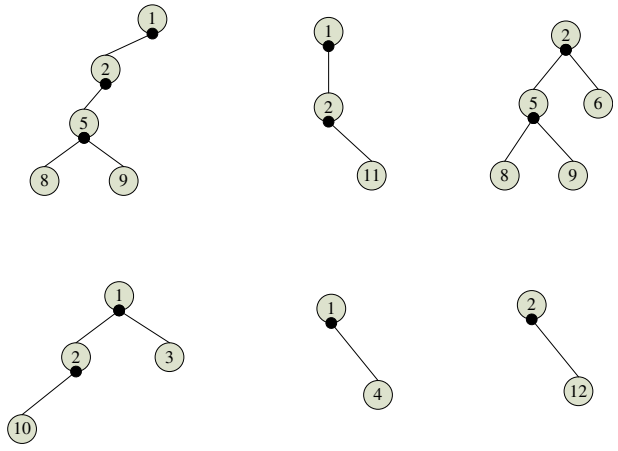

Fig. 7. Frequent maximal induced subtrees.

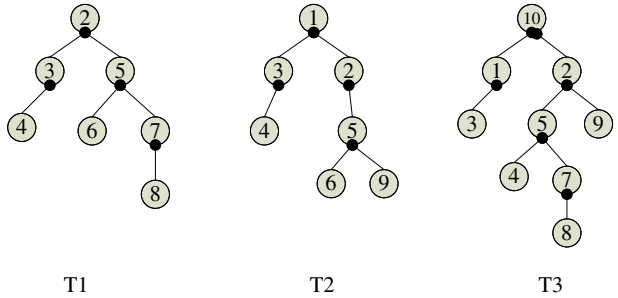

Fig. 8. Comparison data set.

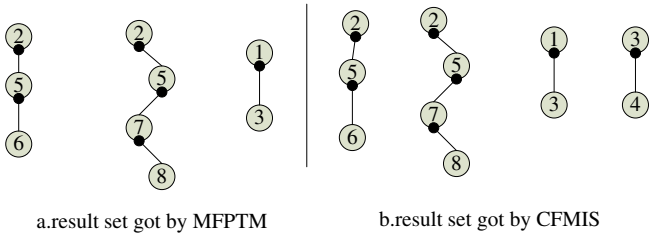

Fig. 9. Result set by two algorithms.

4.2. Experiments and analysis

For the comparison data set shown in Fig. 8, the frequent induced subtrees obtained by the MFPTM algorithm and the CFMIS algorithm proposed in this paper are shown in Fig. 9a, b. We can see that the CFMIS algorithm finds one more subtree than does the MFPTM algorithm. Next is the experiment on the artificial data set and the real data set, respectively.

The support threshold $S_{\min} = 1\%$ is given, where $S_{\min} = \frac{\varepsilon}{datascale}$. The synthetic data set's scale grows from 5–60 K, and the real data set's scale grows from 6–18 K. Figs. 10 and 11 illustrate that CFMIS

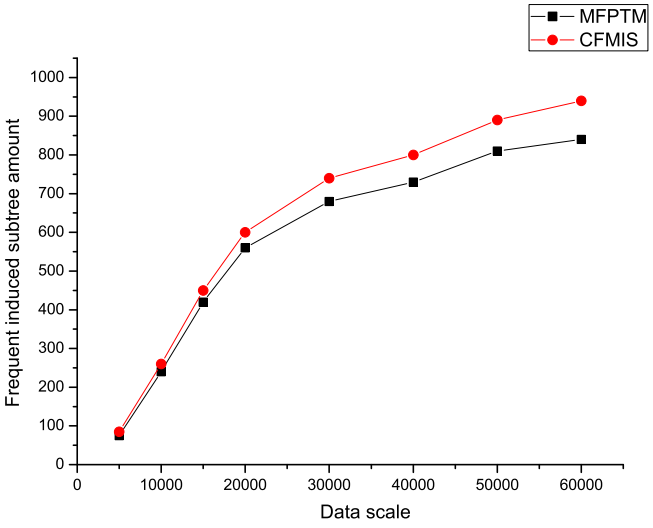

Fig. 10. Frequent induced subtree amount mined by two algorithm on different synthetic data scale.

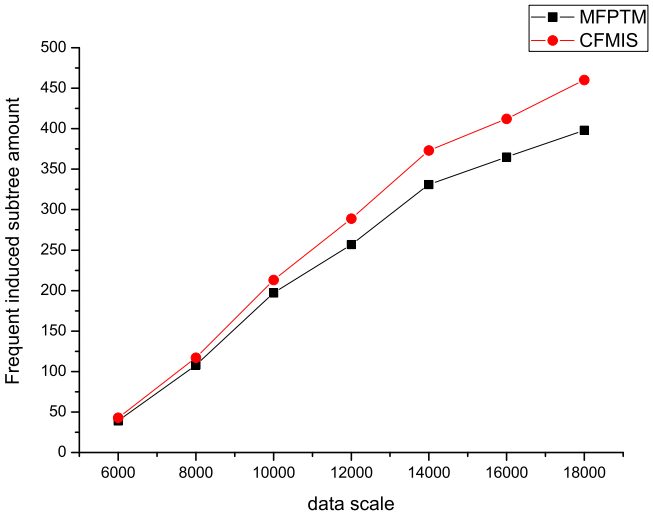

Fig. 11. Frequent induced subtree amount mined by two algorithm on different real data scale.

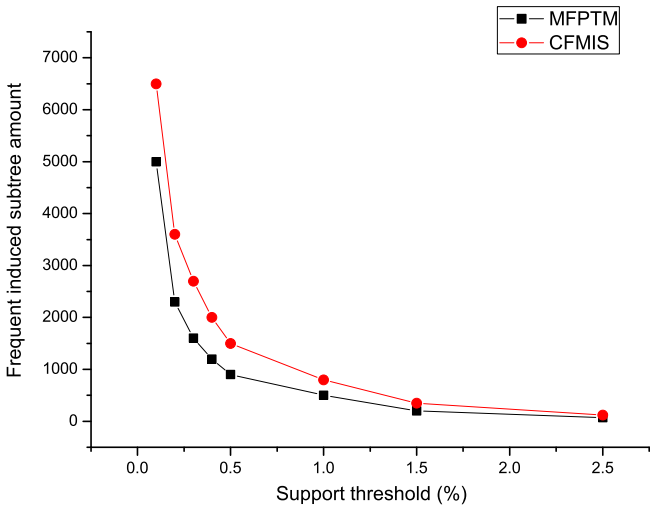

Fig. 12. Frequent induced subtree amount mined by two algorithm on different support threshold on synthetic dataset.

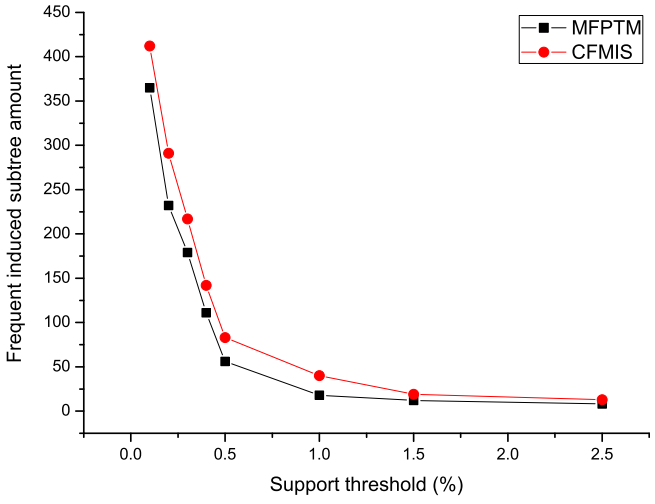

Fig. 13. Frequent induced subtree amount mined by two algorithm on different support threshold on real dataset.

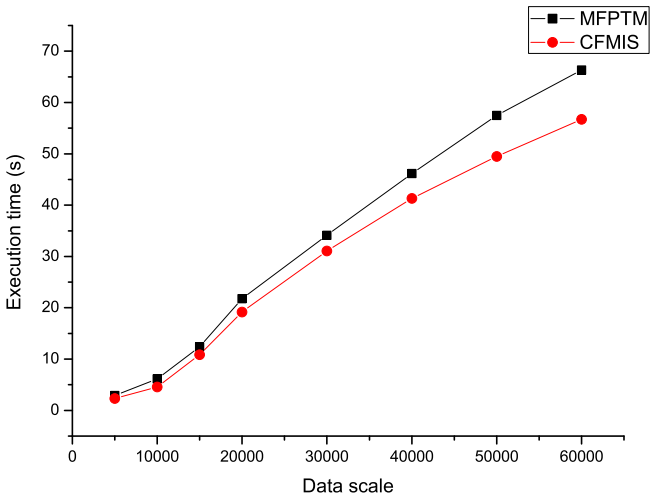

Fig. 14. The execution time of CFMIS and MFPTM on synthetic dataset dataset.

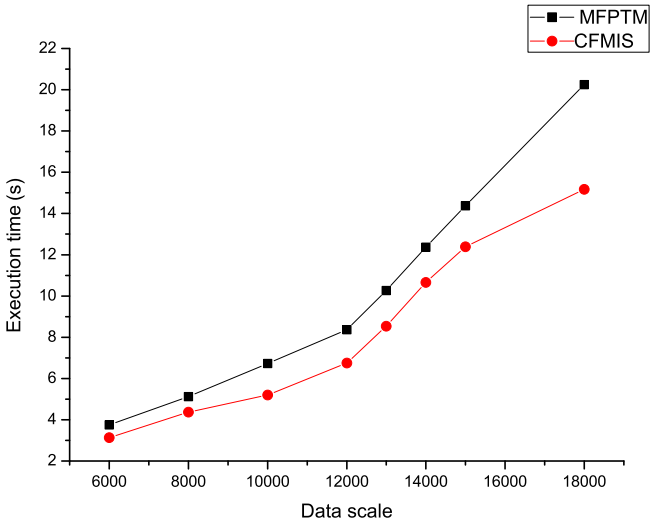

Fig. 15. The execution time of CFMIS and MFPTM on real dataset.

and MFPTM mine the total number of frequent induced subtrees on different data scales.

As shown in Figs. 10 and 11, we can find that, when the data scale grows from 5–60 K, 6–18 K, the total number of frequent subtrees mined by the two algorithms both show a growing trend. CFMIS can mine more frequent subtrees than MFPTM. As shown in Fig. 9, the CFMIS algorithm finds one more subtree than does the MFPTM algorithm, which is not the rooted subtree or hidden subtree mentioned in Wu and Li (2011). While data scale becomes larger, the appearance probability of such a subtree increases, so CFMIS has more advantages than MFPTM.

Given synthetic data set: 75 K; real data set: 16 K; support threshold S_{min} : 0.1—2.5%. Figs. 12 and 13 show that CFMIS and MFPTM mine the total number of frequent induced subtrees.

As shown in Figs. 12 and 13, we can find that, when the support threshold grows from 0.1–2.5%, the total number of frequent subtrees mined by the two algorithms both show a downward trend. CFMIS is better than MFPTM especially when the support threshold is small. This is because, on the same data scale, the appearance probability of a non-rooted subtree and non-hidden subtree increases as the support threshold reduces. Thus, CFMIS can mine more frequent subtrees than MFPTM.

Given synthetic data set: 5–60 K; real data set: 6–18 K; support threshold S_{min}: 1%. Figs. 14 and 15 show the execution times of CFMIS and MFPTM.

As shown in Figs. 14 and 15, CFMIS does better than MFPTM in regards on time performance. The CFMIS algorithm produces probable *CTSs* during each round of compression, and subtrees only need to be matched in these rounds of compression. This reduces the size of the calculation. However, MFPTM is based on the principle of FP trees, so it has to repeatedly traverse the data set when constructing MP trees. In addition, for the trimmed subtrees, if frequent nodes are relatively concentrated (i.e., trimmed subtrees are frequent), its matched times will reach the threshold after compression, so there is no need to look for the maximal public part. Moreover, CFMIS algorithm sorts the *CTSs* in Ψ in the maximal stage, so the *CTSs* in Ψ only need to be compared with those in *MF* instead of being compared with each sequence. This saves time.

5. Conclusions

This paper has proposed a compression tree sequence and a compression tree model, and, on this basis, a frequent maximal induced subtree mining method, CFMIS, has also been proposed.

Experiments have shown that the CFMIS algorithm has more advantages than the MFPTM algorithm in regards to the number of subtrees mined and the execution time on both synthetic and real data sets.

The CFMIS algorithm can be applied in various applications involving frequent subtree mining, especially for data in frequent feature sets. The CFMIS algorithm can extract maximal frequent patterns from relatively decentralized frequent patterns, which removes redundant frequent patterns. It has great value in web log analysis and XML document analysis. In addition, the compressing frequent edges strategy proposed in this paper can be applied in graph mining research as well.

As a future extensive of this work, we plan to apply the CFMIS algorithm to frequent closed subtree mining, frequent embedded subtree mining and clustering methods based on the character of frequent subtrees. Extending the CFMIS algorithm to non-property tree datasets is also included in our future work.

Acknowledgments

This work is partially supported by Project 201215045 from the Jilin Province Natural Science Foundation of China. We are also grateful to the anonymous reviewers for their helpful comments.

References

Balcázar, J., Bifet, A., & Lozano, A. (2010). Mining frequent closed rooted trees. *Machine Learning, 78*(1), 1–33.

Bui, D. B., Hadzic, F., Tagarelli, A., et al. (2014). Evaluation of an associative classifier based on position-constrained frequent/closed subtree mining. *Journal of Intelligent Information Systems*, 1–25.

Deepak, A., Fernández-Baca, D., Tirthapura, S., et al. (2011). EvoMiner: Frequent subtree mining in phylogenetic databases. *Knowledge and Information Systems*, 1–32.

Deng, Z. H., & Lv, S. L. (2014). Fast mining frequent itemsets using nodesets. *Expert Systems with Applications, 41*(10), 4505–4512.

Hou, X., Ong, S. K., Nee, A. Y. C., et al. (2011). GRAONTO: A graph-based approach for automatic construction of domain ontology. *Expert Systems with Applications, 38*(9), 11958–11975.

Jiang, C., Coenen, F., & Zito, M. (2013). A survey of frequent subgraph mining algorithms. *The Knowledge Engineering Review, 28*(01), 75–105.

Lee, K. M., & Lee, K. M. (2013). Efficient identification of frequent family subtrees in tree database. *Applied Mechanics and Materials, 241*, 3165–3170.

Li, W., Li, X-f, & Zhao, Y. (2010). *XML documents clustering research based on weighted cosine measure. Proceedings of the 5th international conference on frontier of computer science and technology (FCST '10)*. Washington: IEEE (pp. 95–100). Washington: IEEE.

Li, J., & Yang, J. (2011). Research of frequent subtree mining algorithm based on partition. *Computer Engineering and Design, 32*(6), 2054–2057.

Liu, H., Lin, Y., & Han, J. (2011). Methods for mining frequent items in data streams: An overview. *Knowledge and Information Systems, 26*(1), 1–30.

Nguyen, A., V., Doi, K., & Yamamoto, A. (2012). Efficient mining of closed tree patterns from large tree databases with subtree constraint. *International Journal on Artificial Intelligence Tools, 21*(06).

Nguyen, V. A., & Yamamoto, A. (2010). Incremental mining of closed frequent subtrees. In *Discovery science* (pp. 356–370). Berlin Heidelberg: Springer.

Wang, E. T., & Chen, A. L. P. (2011). Mining frequent itemsets over distributed data streams by continuously maintaining a global synopsis. *Data Mining and Knowledge Discovery, 23*(2), 252–299.

Wu, X.-K., & Li, X.-F. (2011). Mining maximal frequent subtrees based on fusion compression and FP-tree. *International Journal of Digital Content Technology and its Applications, 5*(7). 6 p.

Xiao, Y., & Yao, J. F. (2003). Efficient data mining for maximal frequent subtrees. In *Data Mining, 2003. Third IEEE International Conference on ICDM 2003*. IEEE (pp. 379–386).

Yang, B., & Huang, H. (2010). Mining top-K significant itemsets in landmark windows over data streams. *Journal of Computer Research and Development, 47*(3), 463–473.

Zaki, M. J. (2005). Efficiently mining frequent trees in a forest: Algorithms and applications. *IEEE Transactions on Knowledge and Data Engineering, 17*(8), 1021–1035.