Queensland University of Technology
Brisbane Australia

This is the author's version of a work that was submitted/accepted for publication in the following source:

Chowdhury, Israt J. & Nayak, Richi
(2015)
FreeS: A fast algorithm to discover frequent free subtrees using a novel canonical form. In
Wang, Jianyong & Cellary, Wojciech (Eds.)
Web Information Systems Engineering – WISE 2015, Springer, Miami, FL, pp. 123-137.

This file was downloaded from: http://eprints.qut.edu.au/93069/

© 2015 Springer International Publishing Switzerland

Notice: *Changes introduced as a result of publishing processes such as copy-editing and formatting may not be reflected in this document. For a definitive version of this work, please refer to the published source:*

http://doi.org/10.1007/978-3-319-26190-4_9

FreeS: A Fast Algorithm to Discover Frequent Free Subtrees Using a Novel Canonical Form

Israt J. Chowdhury and Richi Nayak

School of Electrical Engineering and Computer Science, Science and Engineering Faculty, Queensland University of Technology, Brisbane, Australia
{israt.chowdhury,r.nayak}@qut.edu.au

Abstract. Web data can often be represented in free tree form; however, free tree mining methods seldom exist. In this paper, a computationally fast algorithm *FreeS* is presented to discover all frequently occurring free subtrees in a database of labelled free trees. *FreeS* is designed using an optimal canonical form, BOCF that can uniquely represent free trees even during the presence of isomorphism. To avoid enumeration of false positive candidates, it utilises the enumeration approach based on a tree-structure guided scheme. This paper presents lemmas that introduce conditions to conform the generation of free tree candidates during enumeration. Empirical study using both real and synthetic datasets shows that *FreeS* is scalable and significantly outperforms (i.e. few orders of magnitude faster than) the state-of-the-art frequent free tree mining algorithms, *HybridTreeMiner* and *FreeTreeMiner*.

Keywords: Web data, free tree, canonical form, enumeration approach.

1 Introduction

In the Web domain, graphs and trees are commonly used data structures for modelling information with complex relations. Free trees - the connected, acyclic and undirected graphs - have become popular for presenting such data due to having unique properties [1-4]. For obtaining useful structural information, free tree mining provides a good compromise between the more expressive but computationally harder general graph mining and the less expressive but faster sequence mining. As a middle ground between these two extremes, free trees have been widely used for representing and mining data in diverse areas including web, bioinformatics, computer vision and networks. For example, in analysis of molecular evolution, an evolutionary free tree, called phylogeny, can describe the evolution history of certain species [5]. In bioinformatics various useful patterns can be treated as free trees during pattern mining [4]. In computer networking, multicast free trees have been mined and used for packet routing [6]. Web access logs represented as free trees give interesting insight about the user browsing behaviour without a specific point of entry [7].

The process of finding frequent subtrees incurs high cost due to the inclusion of expensive but unavoidable steps like frequency counting and candidate subtrees gen-

eration. Frequency counting step often requires subtree isomorphism checking which is computationally hard, even known as NP-complete problem in graph mining algorithms [4]. Exponential and redundant candidate generation is another problem. During candidate generation, determining a "good" growth strategy is critical as there can be many possible ways to extend a candidate subtree. These problems become worse in free trees, due to being less-constrained structurally, in comparison to other tree forms such as ordered and unordered. With these complexities involved, only a few free tree mining algorithms are available in the literature. Chi et al. developed an apriori-like algorithm *FreeTreeMiner* [8] as well as an enumeration tree based algorithm *HybridTreeMiner* [1] to discover frequent free subtrees in a database of free trees. Rückert et al. [4] and Zhao et al. [3] have proposed algorithms for mining frequent free trees from a graph database. These algorithms generate large number of false positives (i.e., invalid candidate subtrees) during enumeration that need to be pruned in the frequency counting step. This causes high processing time. Moreover, the necessity of performing isomorphism checking to avoid redundant candidate tree generation and false frequency counting causes additional computational complexity.

In this paper, we propose an algorithm, *FreeS* which is a fast and accurate method for mining frequent free induced subtrees in a database of labelled free trees. First, we propose a unique representation of free trees by introducing a new order-independent *balanced optimal canonical form* (BOCF) that can effectively handle the subtree isomorphism problem. We introduce conditions to conform free tree candidate generation in their BOCFs for which the necessary proofs are also provided. Second, we propose a *tree-structure guided scheme based enumeration* approach that only generates valid candidate subtrees. To the best of our knowledge, *FreeS* is the first algorithm that uses the underlying tree-structure information to avoid invalid subtree generation while mining frequent free subtrees. Because of using the optimal canonical form and tree-structure guided scheme based enumeration, *FreeS* does fast processing. Our experiments with both synthetic and real-life datasets confirm that *FreeS* is faster by few orders of magnitude than two leading free tree mining algorithms, *HybridTreeMiner* and *FreeTreeMiner* (abbreviated as HBT and FTM respectively).

2 Preliminaries

Let a graph constitute a set of nodes $V = \{v_1, v_2, \ldots, v_n\}$ and a set of edges $E = \{(v_i, v_j)| v_i, v_j \in V\} = \{e_1, e_2, \ldots, e_{n-1}\}$. A labelled graph has a set of labels Σ, where a function L: $V \cup E \rightarrow \Sigma$ maps nodes with unique labels. A graph is connected but acyclic when it has at least one node that is connected to the rest of the graph by only one edge, which is leaf. For our purposes, the class of connected acyclic labelled graphs is of special interest, which is also called free tree, an unrooted unordered tree-like structure. In this paper, we denote a free tree with n nodes as n-free tree.

Let two free trees be t and T. t is a subtree of T if t can be obtained from T by repeatedly removing one degree nodes from its structure. Free trees t and T are *isomorphic* to each other if a bijective mapping exists between their set of nodes that preserves node labels, edge labels and also reflects the tree structures.

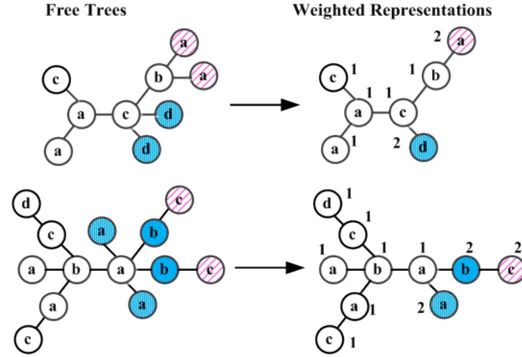

Fig. 1. Equivalent nodes and the condensed weighted representations of free trees[1].

Let T_{db} be a database where each transaction is a labelled free tree. The problem of frequent free tree mining is to discover the complete set of frequent free subtrees. If tree $T \in T_{db}$ has a subtree isomorphic to subtree t, that indicates T has an *occurrence* of t in its structure. Formally we define the support of subtree t in T_{db} using the concept of occurrence as follows,

$$Occurrence\ (t, T) = \begin{cases} 1 & if\ t\ exists\ in\ T \\ 0 & otherwise \end{cases} \tag{1}$$

$$Support\ (t, T_{db}) = \sum_{T \in T_{db}} Occurence(t, T) \tag{2}$$

The subtree t is called frequent if $Support\ (t, T_{db}) \geq minsup$ where $minsup$ is user-defined minimum support threshold.

In this paper, in a free tree, two adjacent nodes v_i and v_j with same label are defined as *equivalent nodes*, denoted by $v_i \cong v_j$. The *weight* of a node v_i is defined as the total number of its equivalent nodes and denoted by w_i (as shown in fig 1). Using weights, we represent free trees of a database in a concise manner for further processing. Fig 1 shows an example of two free trees and their corresponding weighted representations by combining equivalent nodes (highlighted using different color patterns).

3 Canonical Form for Labelled Free Trees

A *Canonical Form* (CF) of a tree is a representative form that can consistently represent many equivalent variations of that tree into one standard form [8, 9]. Several CFs have been proposed for rooted tree representations using traversing algorithms such as *depth-first-search* (DFS) or *breadth-first-search* (BFS) [8]. However, defining CF for free trees is non-trivial as it requires handling the vast variants that a free tree can have, i.e., the isomorphism problem. Due to the inherent structural flexibility (e.g., undefined root node and no direction among sibling nodes), there are more ways to

[1] Tree nodes are represented using labels. The edge labels are ignored in this paper.

represent a free tree than that of a rooted tree. A canonical form is critical for appropriate representation and efficient processing of free trees, because it ensures finding a common pattern amongst free trees. Before we define CF of free trees, we explain the process for unordered rooted trees and extend it to free trees.

3.1 Why Canonical Form is Needed for Free Trees?

A *rooted tree* has a distinguished root node. A rooted tree that preserves order among the sibling nodes is called *rooted ordered*. This type of trees can easily be represented uniquely by using either the depth-first or the breadth-first string representations [8]. They do not face isomorphism. Two ordered trees will be similar iff all of its properties are identical; no variation is possible in similar rooted ordered trees [2]. Whereas, two similar unordered trees can have different orders among sibling nodes and these trees are called isomorphic trees. A free tree is also an unordered tree. The chance of having isomorphic trees in a database of free tree is very high due to the flexible property of being unrooted and unordered. Representing free trees using a systematic approach is non-trivial but critical to ensure its proper indexing for further processing and knowledge discovery.

Optimal Order: we will now briefly describe the concept of *optimal order* that is the basis of the proposed canonical form. An optimal order of a tree is an order obtained by the *balance optimal tree search (BOS) algorithm* [10-12] that traverses a rooted labelled tree uniquely, without the presence of sibling order information. Unlike existing traversal strategies [9], this algorithm works based on optimization instead of enforcing a left-to-right order among siblings. Three heuristics are applied recursively in this traversing algorithm to find out the optimum traversing path of a tree. Heuristic 1 identifies a potential node during the traversal process. Heuristics 2 and 3 select the best node if multiple nodes are identified as candidates for traversal.

Heuristic 1 *After the root node traversal, the children of the root node, i.e.,* $\{v_i, v_j, \ldots, v_k\}$ *with weights* $\{w_i, w_j, \ldots, w_k\}$ *become eligible for traversing. The traversal order of these eligible nodes will be prioritized according to their ascending weights. The node with the highest weight is chosen first.*

Heuristic 2 *If two or more nodes* $\{v_i, v_j, \ldots, v_k\}$ *have the same maximum weight* (i.e. *maximum weight =* $MAX\{w_i, w_j, \ldots, w_k\}$), *the next node in the traversal order is selected based on the maximum number of their children* (*i.e., fan-out*).

Heuristic 3 *If two or more nodes hold the maximum weight with equal number of children, the traversal order will be prioritized using the minimum lexicographical order.*

The *optimal order* is unique even for trees that are isomorphic. This property is advantageous for mining frequent labelled free trees. For a free tree, several rooted ordered tree variations are possible only by changing the position of root node and the order among sibling nodes. An example can be seen in Fig 2, where a free tree is

treated as rooted unordered tree with root node "v_a" (Fig 2a). Considering v_a as root node, several ordered variations of this free tree are shown in Fig 2(b, c, d, e).

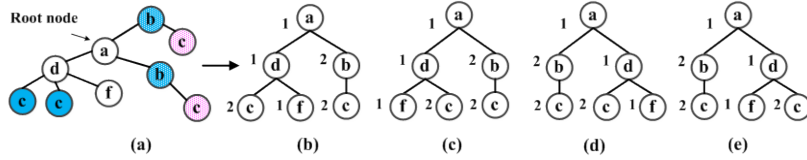

Fig. 2. Four rooted ordered trees obtained from the same rooted unordered tree.

According to the BOS algorithm [10] the unique optimal traversal order of all these equivalent ordered trees will be "v_a, v_b, v_c, v_d, v_c, v_f". In contrast, the BFS or DFS traversal [8] will provide different traversing order for each equivalent ordered tree because of its structure dependent strategy. It is desirable to obtain a unique canonical form of an ordered tree representation; however, it is absolutely critical to obtain a single canonical form for all equivalent variations of a free tree to allow efficient indexing for further processing. The proposed optimal traversal strategy is based on optimization and is not sensitive to the structural changes. It gives the same optimal traversing order for all equivalent ordered trees that originate from a same free tree.

3.2 Balanced Optimal Canonical Form of Free Labelled Trees

If we can uniquely define root node of a free tree, then the optimal order can be used to define its canonical form. In this paper, we propose a two-step process for defining the canonical form of free trees. First, we normalize a free tree into the rooted unordered tree by fixing a root node and then we define the canonical form as well as canonical string.

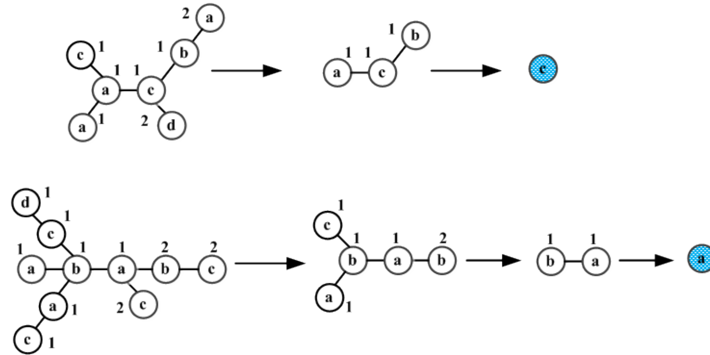

Fig. 3. Process of finding a root node in free trees

Normalization: This step includes a systematic approach to define a root node in a free tree. Following the commonly used technique [1-3], all the leaf nodes along with their incident edges in the free tree are removed at each step until a single node or two adjacent nodes are left. The tree with a single remained node is called a *central tree* and, the tree with a pair of remaining nodes is called a *bicentral tree* [1]. With the remaining single node, this node becomes the root of the free tree. With the remaining two nodes, we apply *heuristic* 3 to obtain the root; therefore the node with minimum lexicographically ordered label becomes the root node.

The overall normalization takes $O(|T|)$ time, where $|T|$ is the number of nodes in the free tree. Fig 3 shows the process of obtaining the root node from the free trees.

Canonical Form and String: After the free tree is normalized to a rooted unordered tree, the balanced optimal canonical form can be defined as follows:

Definition 1 (*Balanced Optimal Canonical Form*): For a rooted labelled unordered tree, the balanced optimal canonical form is its optimal order of node labels along with corresponding weights.

A *canonical string representation* for labelled trees is equivalent to, but simpler than, canonical forms which facilitates frequency counting of trees in a database. For a balanced optimal canonical string encoding, we introduce four unique symbols +1, -1, +2 and -2 to specify directions on depth and breadth. More specifically, +1 and -1 are used to represent forward and backward travel towards depth between child and parent nodes; +2 and -2 are used to represent forward and backward travel towards breadth between sibling nodes respectively. We assume that none of these symbols are included in the alphabet of node labels. The canonical string representation of the rooted unordered tree is achieved by a guided record of sibling nodes,–"*under a parent node, a new node will always be recorded in a breadthwise direction from the existing rightmost sibling node.*"

Example: For all the equivalent trees in Fig 2 with the unique optimal order "v_a, v_b, v_c, v_d, v_c, v_f , the balanced optimal string representation of these trees will be "1v_a, +1, 2v_b, +1, 2v_c, -1, +2, 1v_d, +1, 2v_c, -2, 1v_f". Similarly, the optimal canonical string of the free tree in Fig 4(a) will be "1v_c, +1, 2v_d, +2, 1v_a, +2, 1v_b, +1, 2v_a, -1, -2, +1, 1v_a, +2, 1v_c" and for the tree in Fig 4(b) will be "1v_a, +1, 2v_b, +2, 2v_a, -2, +1, 2v_c, -1, +2, +2, 1v_b, +1, 1v_a, +2, 1v_c, +2, 1v_a, -2, -2, +1, 1v_c, -1, +2, +1, 1v_d".

The isomorphic free trees can be successfully tracked because of having the same balanced optimal string representation. This ensures correct frequency counting for the processing of frequent subtrees. During the mining process, tree structural information such as level, weight, fan-out is stored that allows to differentiate the same alphabet appearing in different position. For sorting the optimal order it requires $O(|T| \log |T|)$ complexity, where $|T|$ is the number of nodes in a tree.

The balanced optimal canonical forms of free tree and rooted unordered tree embrace an interesting relationship which is described under Lemma 1. This relation is a fundamental step for growing the enumeration tree of free trees.

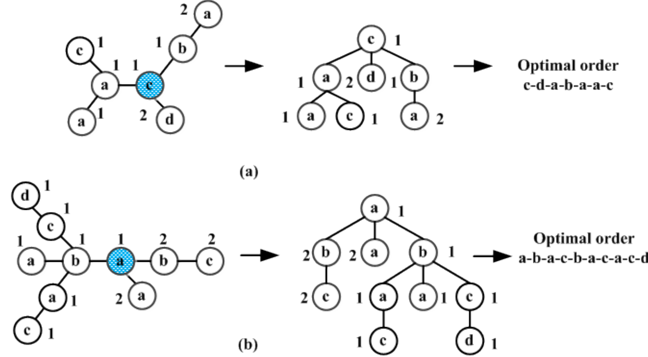

Fig. 4. Balanced optimal canonical form of free tree.

Lemma 1. *Balanced optimal canonical form of a free tree is always the balanced optimal canonical form of a rooted unordered tree; however, the reverse is not true.*

PROOF: Consider a free tree T, with v_1, v_2, …, v_n nodes, with its balanced optimal canonical form t_{v1} that has a normalized root v_1. The n-number of different rooted unordered trees can be derived in their balanced optimal canonical forms t_{v1}; t_{v2}; …; t_{vn} by changing the position of root in T. Only one of the BOCFs of these rooted unordered trees will have the same BOCF as the free tree, e.g. t_{v1}.

Prior to detailing our *FreeS* algorithm, we add following two lemmas that introduce important conditions which are essential to hold true during candidate free subtree enumeration through the BOCF representation of fee trees. Fist we give the definitions of tree dimensions including depth, height and level as [13].

Definition 2 (*Depth*, *Height*, *Level of Node*): For node v_i of a tree T, depth is the length of the unique path from that node towards the root node, denoted by $d(T, v_i)$. The height $h(v_i)$ of node v_i is the longest path from that node to a leaf. The height H of a tree is the height of root node, $h(v_0)$. The level of a node v_i in a tree T is defined as $Lv(T, v_i) = H - d(T, v_i)$.

Lemma 2. *Balanced optimal canonical form of a rooted unordered tree T with two nodes is the balanced optimal canonical form of a free tree iff the root node has lexicographically minimum label.*

PROOF: T is a rooted unordered tree with two nodes, where v_0 is root and v_1 is its child. The balanced optimal canonical form will be generated based on its optimal order, i.e., "v_0, v_1". Let us consider *case* 1, where root node v_0 has lexicographically minimum label. In this case treating T as free tree will end up having same canonical form as the rooted unordered tree, since a free tree considers the node with lexicographically minimum label as the center. Now consider *case* 2, where label of root node v_0 is higher than v_1. In this case the canonical form of free tree will be different than the rooted unordered tree, since v_1 will be the center instead of v_0.

Lemma 3. *Balanced optimal canonical form of a rooted unordered tree, T with* 3 *or more nodes and height H is the balanced optimal canonical form of a free tree iff the following conditions hold:*

1. *The root has at least 2 children;*
2. *The root node has lexicographically smaller label than the labels of its children; and*
3. *One branch or subtree induced by a child of the root has a leaf node,* v_i *positioned at level* $Lv(T, v_i) = 0$ (*bottom level of the tree*) *and at least another branch or one subtree induced by another child of the root has a leaf node,* v_j *positioned at level* $Lv(T, v_i) \leq 1$ (*at most one level up than the last level*).

***PROOF*:** For a rooted unordered tree T in its balanced optimal canonical form, we denote the root of T by v_0 and the children of v_0 by v_1; …; v_m. Let us consider *case* 1. Tree T has 3 or more nodes and v_0 has only one child. It indicates that the rest of the nodes are appeared in that tree as child nodes of the immediate child of the root node. The node v_0 will be removed in the first step of finding center/bicenter. Consequently, v_0 cannot be the center or one of the bicentres. Therefore condition 1 will be held in this case. Let us consider *case* 2 when the root node v_0 has more than one child. This indicates that the leaf node of a subtree induced by one of v_1; …; v_k is at the bottom level of tree T. Assume this child to be v_j. If none of the subtrees induced by other child node of v_0 has a leaf node at the bottom level or second last level of tree T, then v_0 cannot be the center or one of the bicentres. This is because the center (or the bicenter) must be a node (or nodes) of the subtree induced by v_j. Without the loss of generality, we assume the subtree t_{v1} induced by v_1 has a leaf node at the bottom level of tree for which the path from root is H. The subtree t_{v2} induced by v_2 has a leaf node either at the last level or second last level. Therefore the path of that leaf node from root is either H or H-1. Now 2H or 2H-1 will be the length of path considering from the bottom-level leaf of t_{v1} to the bottom-level leaf of t_{v2} which makes v_0 as the center or one of the bicenters of the free tree. Therefore, condition 3 holds. Besides in case 2, it is essential to hold the condition 2 true, when T turns out to a bicentral tree and v_0 will only become the center if it has lexicographically minimum label.

4 Frequent Free Subtree Mining Algorithm: *FreeS*

FreeS consists of two main steps: (1) candidate subtree generation using the enumeration tree; and (2) frequency counting to determine frequent subtrees.

4.1 Candidate Subtree Generation using Enumeration Tree

Using the proposed balanced optimal canonical form of free trees and other tree structural information from a database, we define an enumeration tree that lists all subtrees in T_{db}, in their balanced optimal canonical forms. Since the underlying tree structure information is used for defining the enumeration tree, it is called tree-structure guided

scheme based enumeration. To the best of our knowledge, *FreeS* is the first algorithm where this enumeration approach is used to generate candidate free trees.

Tree-Structure Guided Scheme based Enumeration Tree: The task here is enumerating a complete and non-redundant list of candidate subtrees from a given database. A candidate enumeration technique can generate both valid and invalid candidates. A candidate subtree is called valid if it exists in the considered database [11]. It is desirable to enumerate only the valid subtrees in order to reduce the computational efforts, instead of generating all possible candidates and prune invalid subtrees later. The tree-structure guided scheme based enumeration allows invalid subtrees, which will never be significant in spite of being frequent, to be excluded from counting the number of candidate trees. It utilizes the tree structural information such as level, weight and fan-out of nodes, which are learned from a given database, in determining a valid subtree. This information is obtained after the free trees are normalized to rooted unordered trees. Instead of testing whether a tree actually exist in the database that is computationally expensive, a subtree is considered valid if it conforms to the tree structural information.

Extending the Enumeration Tree: The right-path extension and join operations have been used to grow the enumeration tree. Previous research has shown that the right-path extension produces a complete and non-redundant candidate generation [1, 8, 14]. However, the use of extension alone for growing enumeration tree can be inefficient because the number of potential growth may be very large, especially when the cardinality of alphabets for node labels is large [1, 8]. This shortcoming necessitates of using a join operation; however, it often generates invalid subtrees. *FreeS* controls it by using the tree-structure guided scheme based enumeration. The basis of growing the enumeration tree of free trees is as follows: *By removing the last leg (node along with edge), i.e., the rightmost leg at the bottom level, of a* $(n+1)$*-free tree BOCF will result in the BOCF for another n-free tree.* The definitions of two operations for extending the enumeration tree are as follows.

Definition 3 (*FreeS-extension*): For node v_i (fan-out $\neq 0$) of a n-free tree in its balanced optimal canonical form t_v, an extension is possible by applying every frequent node label v_j that has a level equal to $Lv(t_v, v_i)$-1. This extension operation will result in another balanced optimal canonical form t'_v of a new $(n+1)$-free tree, with v_j child of $v_{i,}$ in the enumeration tree iff conditions of Lemma 2 and 3 are held. Further extension is possible from this new right-most node v_j iff conditions are fulfilled again.

Before giving the definition of *FreeS-join* operation, we define *equivalent group.*

Definition 4 (*Equivalent group*): If two BOCFs t_v and t'_v of two n-free trees have equal height H and common first n-1 nodes (along with labels and weights), they are considered as equivalent group, denoted by $t_v \cong t'_v$. Only the nth node of each of these trees, that appear last in their canonical forms, are different.

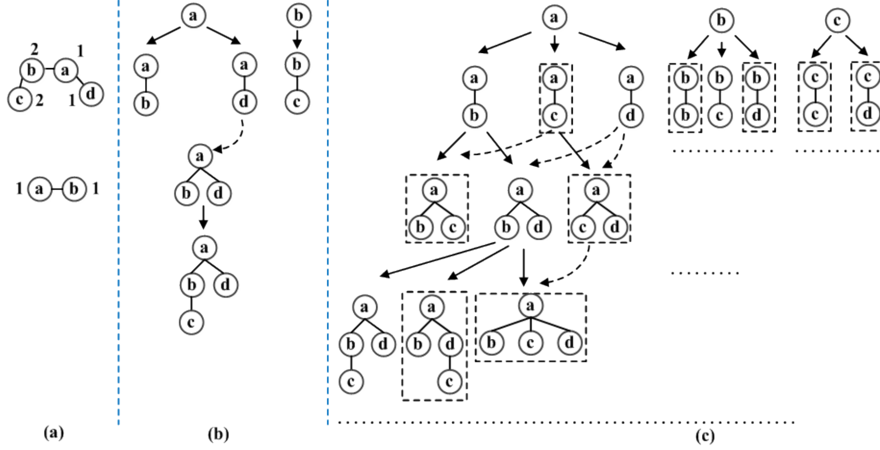

Fig. 5. Sample database of labelled free trees (a), enumeration tree for free trees using tree structure guided scheme in *FreeS* (b) enumeration tree using the approach from HBT algorithm (c) (the dotted line with arrow is showing the candidates that are generated using join operations in HBT, and the dotted rectangle is showing the invalid candidate tree).

Definition 5 (*FreeS-join*): Join operation is a guided extension between two free trees in BOCFs t_v and t'_v, that are members of an equivalent group, $t_v \cong t'_v$. Assume, v_i and v_j are the corresponding right-most node of t_v and t'_v, where $w_i > w_j$ or, $w_i = w_j$ with v_i lexicographically sorts lower than v_j. By joining v_j in t_v at the position of $Lv(t_v, v_i)$-1 will result in a new (n+1) node balanced optimal canonical form of free tree, denoted by $t_v \odot t'_v$, of the same height as tree t_v.

The join operation does not change the height or the level position of leaf nodes of a newly generated candidate tree, therefore Lemma 2 and 3 are not considered. As in the tree-structure guided approach, the enumeration tree growth is guided by the prior learned tree structure information. Therefore only valid subtrees are expected to be generated as candidate trees.

Consider an example database in Fig 5, where for minimum support 1, we compare the enumeration tree (Fig 5b) used by *FreeS* with the enumeration tree (Fig 5c) used by the *HybridTreeMiner* (HBT) method [1]. HBT also uses the right-path extension and join operations for growing the enumeration tree, but, these are defined using a different canonical form (Breadth First Canonical Form) [8], whereas we use BOCF and the tree-structure guided scheme for growing the enumeration tree. The dotted rectangles in (Fig 5c) show the generation of invalid subtrees in HBT. We only show a small part of the enumeration tree for HBT. If it is continued, it will grow in a much bigger size and will result in much higher numbers of invalid subtrees. In contrast, Fig 5b is the complete enumeration tree of the considered database for *FreeS*.

It can be clearly seen that the *FreeS* enumeration tree generates much less candidates in comparison to HBT enumeration tree because of producing only valid subtrees. Generation of invalid subtrees causes extra memory space and then, pruning of these subtrees causes additional computational cost for existing methods.

FreeS Algorithm

Input: Balanced optimal canonical form strings of labelled free trees present in a database T_{db}; level, weight and fan-out information of each node, minimum support (*minsup*) threshold.
Output: All frequent free subtrees.

```
1.  Result ←∅;
2.  Frq1 ← the set of all frequent subtrees of size 1;
3.  Frq2 ←∅;
4.  while  Frq1 ≠ ∅ do
5.      for all c ∈ Frq1 do
6.          if fan-out(c) != 0
7.            Candidate ← Enumeration (c, Frq1, level, weight, fan-out );
8.          end if
9.          for all Ɛ´∈ Candidate do
10.             if support (Ɛ´) ≥ minsup then
11.                    Frq2 ← Frq2 ∪ Ɛ´;
12.             end if
13.         end for
14.     end for
15.     Frq1← Frq2;
16.     Result ← Result ∪ Frq1;
17.     Frq2 ← ∅;
18. end while
19. return Result
```

Fig. 6. High level pseudo code of *FreeS* algorithm.

Enumeration (l_k, *Frq*1, level, weight, fan-out)

```
1.  Output ←∅;
2.  for all Ɛ ∈ Frq1do
3.      Enumerate candidate l_{k+1} by adding Ɛ;     /* Using FreeS-extension */
4.      Output ← Output ∪ l_{k+1};
5.      end if
6.  end for
7.  for all equivalent groups in Output do
8.      l_{k+2} ← l_{k+1} ⊙ l´_{k+1};             /* Using FreeS-join and l_{k+1} ≅ l´_{k+1} */
9.      Output ← Output ∪ l_{k+2};
10. end for
11. return (Output)
```

Fig. 7. High level pseudo code of candidate generation.

4.2 Frequency Counting

For counting frequency we modified the method described in [1, 8], which is basically an apriori like frequency counting that gives the exact support measure of each candidate subtree by maintaining an occurrence list. We used a catching technique to make the process of keeping occurrence list more efficient, which is "*stopped counting tree*

when the ID counter reaches the min support", therefore the occurrence list becomes smaller than usual.

Figs 6 & 7 list the overall enumeration approach and the *FreeS* algorithm. The process of frequent subtree mining is initiated by scanning the database T_{db}, where free trees are stored as BOCF strings along with weight, level and fan-out information of each node. The set of frequent subtrees of size 1 is generated and the *Enumeration* method (in Fig 7) is called recursively for generating the candidates of larger sized subtrees. The frequency of every resultant candidate tree is computed. The full pruning is also performed to ensure downward-closure lemma [15]. But full pruning is expensive; therefore to accelerate this process we cease the frequency checking for a subtree belong to (K-1) set as soon as the K subtree is found frequent.

5 Empirical Analysis

The efficacy of *FreeS* is shown by conducting systematic experiments using both real-life and synthetic datasets. *FreeS* is benchmarked with the most relevant and leading algorithms *FreeTreeMiner* (FTM) [8] and *HybridTreeMiner* (HBT) [1] which are designed to mine frequent free subtrees from a database of labelled free trees. All experiments have been done on a 2.8GHz Intel Core i7 PC with 8GB main memory and running the UNIX operating system.

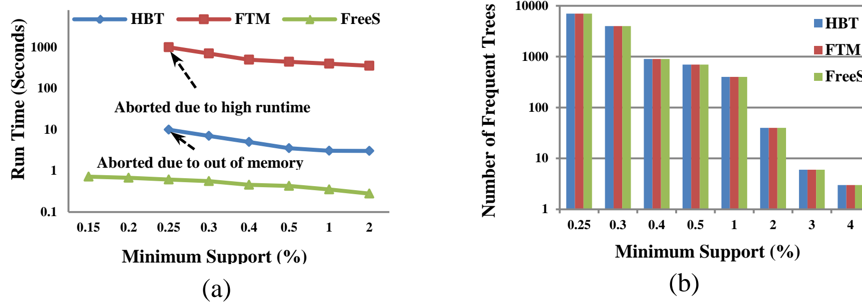

Fig. 8. Run time comparison (a) and completeness test (b) using CSLOGS data (a $\log_{10}$ scale is used in Y axis).

CSLOGS: This real-life dataset has been widely used in evaluating various tree mining algorithms. CSLOGS [14] contains web access trees of the CS department of Rensselaer Polytechnic Institute during one month. There are a total of 59,691 transactions and 13,209 unique node labels (corresponding to the URLs of the web pages).

Fig 8(a) shows that *FreeS* can find the same amount of subtrees in significant lesser time than its counterparts. Results show that below a certain support threshold (0.25%) the number of frequent trees explodes that causes huge memory consumption for HBT and consequently, the software automatically aborts the process. For calculating support of free trees, HBT uses occurrence list that makes the process faster, but, it is responsible for high memory usage too. *FreeS* performs this step within the memory size even for smaller minimum support threshold such as 0.15% because of

using modified occurrence list. FTM does not suffer from the memory exhaustion problem though; however the run time increases drastically for smaller supports due to the lack of efficient frequency counting and inclusion of the expensive apriori candidate generation.

The runtime performance of *FreeS* is few orders of magnitude better than HBT and FTM due to several reasons. (1) *FreeS* uses tree-structure guided based enumeration tree that allows enumerating only valid subtrees. (2) BOCF is defined to enumerate only one free tree for either of central or bicentral free trees, hence the occurrence list only keeps record of one tree. (3) A catching technique assists in keeping the occurrence list shorter. On the other hand, HBT can't avoid generating invalid candidate subtrees during enumeration, which results in extra memory consumption. HBT may also enumerate two free trees from a bicentral tree because of the supplementary canonical form concept [1]. Consequently, it will keep record of both trees which increases the size of the occurrence list.

Results in Fig 8(b) show that *FreeS* extracts the same amount of frequent patterns as the other state-of-the-art methods. The tree model guided enumeration employed in *FreeS* does not generate any invalid trees but does not miss on any valid trees. All three algorithms satisfy the completeness property and do not miss any frequent patterns since they all used full pruning (downward closure lemma), not an opportunistic pruning. This shows the accuracy of *FreeS* in finding subtrees.

Synthetic Data Sets: We conducted few more experiments using synthetic datasets with varied properties to support all of the above findings. The synthetic data sets were generated by a tree generator as described in [14]. The dataset called D1 is created using following parameters: the number of labels $L = 10$, the number of vertices in the master tree $M = 100$, the maximum depth $D = 10$, the maximum fan-out $F = 5$ and the total number of subtrees $T = 5000$. Such characteristics reflect the properties of web-browsing but not of very large databases. Result in Fig 9(a) shows that *FreeS* requires less runtime than HBT and FTM as expected. The memory consumption is also low for *FreeS*, whereas for being the small dataset the other two can also perform within the given memory size, Fig 9(b).

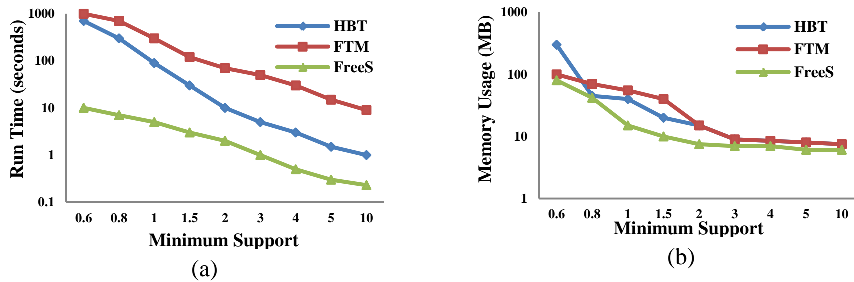

Fig. 9. Runtime (a) and memory usage (b) comparison using dataset D1 (a $\log_{10}$ scale is used in Y axis).

The dataset called D2 is generated using high fan-out, F = 20 with low number of labels L =10 and a moderate size dataset T = 10,000. The rest of the parameters are kept the same. This makes D2 having wider trees than the deep trees. The isomorphic problem is known to occur more commonly when trees have several siblings at same label. This facet of experiment will support the claim that *FreeS* can handle isomorphism more effectively than any other algorithms due to the use of BOCF.

As shown in Fig 10, *FreeS* consumes much less processing time in comparison to other methods. It happens as *FreeS* does not generate a candidate tree multiple times because of using BOCF that ensures same identity for all isomorphic trees. Therefore, no additional test is required for checking the presence of isomorphism during frequency counting. In contrast, the state-of-the-art algorithms perform a mandatory isomorphism checking which makes them more expensive (Fig 10a).

Fig 10b shows that HBT consumes larger memory space than FTM and *FreeS*, and it becomes worse for smaller support thresholds. As explained before, FTM does not use occurrence list for frequency counting but computes the occurrences of each free tree. Therefore, it saves memory but consumes additional computational time. The usage of occurrence list becomes a pressing concern in terms of memory for large data, especially when the support threshold is low, but allows fast and efficient frequency checking. The catching mechanism employed in *FreeS* makes it consume less memory as well as the enumeration strategy does not generate any invalid subtrees, therefore *FreeS* can offer a good trade-off between memory usage and runtime.

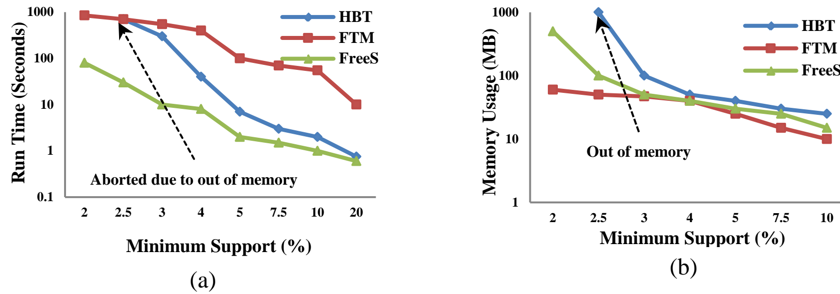

Fig. 10. Runtime (a) and memory (b) comparison using dataset D2 (a $\log_{10}$ scale is used in Y axis).

6 Conclusion

In this paper, we consider an important problem of mining frequent free subtrees from a collection of free trees. We proposed a computationally efficient algorithm *FreeS* to discover all frequent subtrees in a database of free trees. A novel balanced optimal canonical form is introduced that ensures unique identity of frequent free trees even in presence of isomorphism. Because of this canonical form the isomorphism problem can be handled, that is responsible for computational complexity in this process. Moreover, the proposed tree-structure guided scheme based enumeration enables *FreeS* to reduce the cost for candidate generation by enumerating only valid subtrees.

We modified the efficient apriori like occurrence list based frequency counting method that ensures less memory consumption.

Our empirical analyses show *FreeS* is scalable to mine frequent free trees in a large database of free trees with low support thresholds. In future we are planning to extend our algorithm for mining free trees in graph database.

Reference

1. Chi, Y., Yang, Y., Muntz, R.R.: HybridTreeMiner: An Efficient Algorithm for Mining Frequent Rooted Trees and Free Trees Using Canonical Forms. In: Proceedings of the 16th International Conference on Scientific and Statistical Database Management, pp. 11-20. IEEE, Santorini (2004)
2. Chi, Y., Yang, Y., Muntz, R.R.: Indexing and mining free trees. In: In Third IEEE International Conference on Data Mining, 2003, (ICDM'03) pp. 509-512. IEEE, (2003)
3. Zhao, P. Yu, J.: Fast Frequent Free Tree Mining in Graph Databases. World Wide Web. 11(1), 71-92 (2008)
4. Rückert, U. Kramer, S.: Frequent free tree discovery in graph data. In: Proceedings of the 2004 ACM symposium on Applied computing, pp. 564-570. ACM, (2004)
5. Hein, J., Jiang, T., Wang, L., Zhang, K.: On the complexity of comparing evolutionary trees. Discrete Applied Mathematics. 71(1), 153-169 (1996)
6. Cui, J.-H., Kim, J., Maggiorini, D., Boussetta, K., Gerla, M.: Aggregated multicast–a comparative study. Cluster Computing. 8(1), 15-26 (2005)
7. Chi, Y., Muntz, R.R., Nijssen, S., Kok, J.N.: Frequent Subtree Mining - An Overview. Fundamental Informatic. 66(1-2), 161-198 (2004)
8. Chi, Y., Yang, Y., Muntz, R.R.: Canonical Forms for Labelled Trees and Their Applications in Frequent Subtree Mining. Knowledge and Information System. 8(2), 203-234 (2005)
9. Valiente. Algorithms on Trees and Graphs. Springer, Berlin Heidelberg, New York (2002)
10. Chowdhury, I.J. Nayak, R.: A Novel Method for Finding Similarities between Unordered Trees Using Matrix Data Model. In: WISE 14th International Conference on Web Information Systems Engineering, pp. 421-430. Springer Berlin Heidelberg, (2013)
11. Chowdhury, I. Nayak, R.: BEST: An Efficient Algorithm for Mining Frequent Unordered Embedded Subtrees. In: PRICAI 2014: Trends in Artificial Intelligence. Lecture Notes in Computer Science, vol. 8862, pp. 459-471. Springer International Publishing (2014)
12. Chowdhury, I.J. Nayak, R.: BOSTER: An Efficient Algorithm for Mining Frequent Unordered Induced Subtrees. In: WISE 15th International Conference on Web Information Systems Engineering, pp. 146-155. Springer Berlin Heidelberg, Athens, Greece (2014)
13. Ullman, J.D., Aho, A.V., Hopcroft, J.E.: The design and analysis of computer algorithms. (1974)
14. Zaki, M.J.: Efficiently Mining Frequent Trees in A Forest: Algorithms and Applications. IEEE Transactions on Knowledge and Data Engineering. 17(8), 1021-1035 (2005)
15. Agrawal, R. Srikant, R.: Fast Algorithms for Mining Association Rules in Large Databases. In: Proceedings of the 20th International Conference on Very Large Data Bases, pp. 487-499. Morgan Kaufmann Publishers Inc., (1994)