Discovering Frequent Substructures in Large Unordered Trees

Tatsuya Asai[1], Hiroki Arimura[1], Takeaki Uno[2], and Shin-ichi Nakano[3]

[1] Kyushu University, Fukuoka 812–8581, JAPAN
{t-asai,arim}@i.kyushu-u.ac.jp
[2] National Institute of Informatics, Tokyo 101–8430, JAPAN
uno@nii.jp
[3] Gunma University, Kiryu-shi, Gunma 376–8515, JAPAN
nakano@cs.gunma-u.ac.jp

Abstract. In this paper, we study a frequent substructure discovery problem in semi-structured data. We present an efficient algorithm Unot that computes all frequent labeled unordered trees appearing in a large collection of data trees with frequency above a user-specified threshold. The keys of the algorithm are efficient enumeration of all unordered trees in canonical form and incremental computation of their occurrences. We then show that Unot discovers each frequent pattern T in $O(kb^2m)$ per pattern, where k is the size of T, b is the branching factor of the data trees, and m is the total number of occurrences of T in the data trees.

1 Introduction

By rapid progress of network and storage technologies, huge amounts of electronic data have been available in various enterprises and organizations. These weakly-structured data are well modeled by graph or trees, where a data object is represented by a nodes and a connection or relationships between objects are encoded by an edge between them. There have been increasing demands for efficient methods for *graph mining*, the task of discovering patterns in large collections of graph and tree structures [1,3,4,7,8,9,10,13,15,17,18,19,20].

In this paper, we present an efficient algorithm for discovering frequent substructures in a large graph structured data, where both of the patterns and the data are modeled by labeled unordered trees. A *labeled unordered tree* is a rooted directed acyclic graph, where all but the root node have exactly one parent and each node is labeled by a symbol drawn from an alphabet (Fig. 1). Such unordered trees can be seen as either a generalization of *labeled ordered trees* extensively studied in semi-structured data mining [1,3,4,10,13,18,20], or as an efficient specialization of *attributed graphs* in graph mining researches [7,8, 9,17,19]. They are also useful in modeling various types of unstructured or semi-structured data such as chemical compounds, dependency structure in discourse analysis and the hyperlink structure of Web sites.

On the other hand, difficulties arise in discovery of trees and graphs such as the combinatorial explosion of the number of possible patterns, the isomorphism problem for many semantically equivalent patterns. Also, there are other

G. Grieser et al. (Eds.): DS 2003, LNAI 2843, pp. 47–61, 2003.

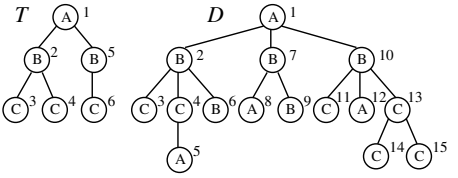

Fig. 1. A data tree D and a pattern tree T

difficulties such as the computational complexity of detecting the embeddings or occurrences in trees. We tackle these problems by introducing a novel definitions of the support and the canonical form for unordered trees, and by developing techniques for efficient enumeration of all unordered trees in canonical form without duplicates and for incremental computation of the embeddings of each patterns in data trees. Interestingly, these techniques can be seen as instances of the *reverse search* technique, known as a powerful design tool for combinatorial enumeration problems [6,16].

Combining these techniques, we present an efficient algorithm Unot that computes all labeled unordered trees appearing in a collection of data trees with frequency above a user-specified threshold. The algorithm Unot has a provable performance in terms of the output size unlike other graph mining algorithm presented so far. It enumerates each frequent pattern T in $O(kb^2m)$ per pattern, where k is the size of T, b is the branching factor of the data tree, and m is the total number of occurrences of T in the data trees.

Termier *et al.* [15] developed the algorithm TreeFinder for discovering frequent unordered trees. The major difference with Unot is that TreeFinder is not complete, i.e., it finds a subset of the actually frequent patterns. On the other hand, Unot computes all the frequent unordered trees. Another difference is that matching functions preserve the parent relationship in Unot, whereas ones preserve the ancestor relationship in TreeFinder.

Very recently, Nijssen *et al.* [14] independently proposed an algorithm for the frequent unordered tree discovery problem with an efficient enumeration technique essentially same to ours.

This paper is organized as follows. In Section 2, we prepare basic definitions on unordered trees and introduce our data mining problems. In Section 3, we define the canonical form for the unordered trees. In Section 4, we show an efficient algorithm Unot for finding all the frequent unordered trees in a collection of semi-structured data. In Section 5, we conclude the results.

2 Preliminaries

In this section, we give basic definitions on unordered trees according to [2] and introduce our data mining problems. For a set A, $|A|$ denotes the size of A. For a binary relation $R \subseteq A^2$ on A, R^* denotes the reflexive transitive closure of R.

2.1 The Model of Semi-structured Data

We introduce the class of labeled unordered trees as a model of semi-structured data and patterns according to [2,3,12]. Let $\mathcal{L} = \{\ell, \ell_1, \ell_2, \ldots\}$ be a countable set of *labels* with a total order $\leq_{\mathcal{L}}$ on $\mathcal{L}$.

A *labeled unordered tree* (an *unordered tree*, for short) is a directed acyclic graph $T = (V, E, r, label)$, with a distinguished node r called the *root*, satisfying the followings: V is a set of *nodes*, $E \subseteq V \times V$ is a set of *edges*, and $label : V \to \mathcal{L}$ is the *labeling function* for the nodes in V. If $(u, v) \in E$ then we say that u is a *parent* of v, or v is a *child* of u. Each node $v \in V$ except r has exactly one parent and the *depth* of v is defined by $dep(v) = d$, where $(v_0 = r, v_1, \ldots, v_d)$ is the unique path from the root r to v.

A *labeled ordered tree* (an *ordered tree*, for short) $T = (V, E, B, r, label)$ is defined in a similar manner as a labeled unordered tree except that for each internal node $v \in V$, its children are ordered from left to right by the *sibling relation* $B \subseteq V \times V$ [3]. We denote by $\mathcal{U}$ and $\mathcal{T}$ the classes of unordered and ordered trees over $\mathcal{L}$, respectively. For a labeled tree $T = (V, E, r, label)$, we write V_T, V_E, r_T and $label_T$ for V, E, r and $label$ if it is clear from the context.

The following notions are common in both unordered and ordered trees. Let T be an unordered or ordered tree and $u, v \in T$ be its nodes. If there is a path from u to v, then we say that u is an *ancestor* of v, or v is a *descendant* of u. For a node v, we denote by $T(v)$ the *subtree* of T rooted at v, the subgraph of T induced in the set of all descendants of v. The *size* of T, denoted by $|T|$, is defined by $|V|$. We define the special tree $\perp$ of size 0, called the *empty tree*.

Example 1. In Fig. 1, we show examples of labeled unordered trees T and D on alphabet $\mathcal{L} = \{A, B, C\}$ with the total ordering $A > B > C$. In the tree T, the root is 1 labeled with A and the leaves are 3, 4, and 6. The subtree $T(2)$ at node 2 consists of nodes $2, 3, 4$. The size of T is $|T| = 6$.

Throughout this paper, we assume that for every labeled ordered tree $T = (V, E, B, r, label)$ of size $k \geq 1$, its nodes are exactly $\{1, \ldots, k\}$, which are numbered consecutively in preorder. Thus, the root and the rightmost leaf of T are $root(T) = 1$ and $rml(T) = k$, respectively. The *rightmost branch* of T is the path $RMB(T) = (r_0, \ldots, r_c)$ $(c \geq 0)$ from the root r to the rightmost leaf of T.

2.2 Patterns, Tree Matching, and Occurrences

For $k \geq 0$, a *k-unordered pattern* (k-pattern, for short) is a labeled unordered tree having exactly k nodes, that is, $V_T = \{1, \ldots, k\}$ such that $root(T) = 1$ holds. An *unordered database* (*database*, for short) is a finite collection $\mathcal{D} = \{D_1, \ldots, D_n\} \subseteq \mathcal{U}$ of (ordered) trees, where each $D_i \in \mathcal{D}$ is called a *data tree*. We denote by $V_{\mathcal{D}}$ the set of the nodes of $\mathcal{D}$ and $||\mathcal{D}|| = |V_{\mathcal{D}}| = \sum_{D \in \mathcal{D}} |V_D|$.

The semantics of unordered and ordered tree patterns are defined through *tree matching* [3]. Let T and $D \in \mathcal{U}$ be labeled unordered trees over $\mathcal{L}$, which are called the *pattern tree* and the *data tree*, respectively. Then, we say that T *occurs in* D as an unordered tree if there is a mapping $\varphi : V_T \to V_D$ satisfying the following (1)–(3) for every $x, y \in V_T$:

(1) φ is *one-to-one*, i.e., $x \neq y$ implies $\varphi(x) \neq \varphi(y)$.
(2) φ *preserves the parent relation*, i.e., $(x, y) \in E_T$ iff $(\varphi(x), \varphi(y)) \in E_D$.
(3) φ *preserves the labels*, i.e., $label_T(x) = label_D(\varphi(x))$.

The mapping φ is called a *matching from T into D*. Then we can extend the matching from matching $\varphi : V_T \to V_D$ into a data tree to a matching $\varphi : V_T \to 2^{V_D}$ into a database. $\mathcal{M}^{\mathcal{D}}(T)$ denotes the set of all matchings from T into $\mathcal{D}$. Then, we define four types of occurrences of U in $\mathcal{D}$ as follows:

Definition 1. Let $k \geq 1$, $T \in \mathcal{U}$ be a k-unordered pattern, $\mathcal{D}$ be a database. For any matching $\varphi : V_T \to V_D \in \mathcal{M}^{\mathcal{D}}(T)$ from T into $\mathcal{D}$, we define:

1. The *total occurrence* of T is the k-tuple $Toc(\varphi) = \langle \varphi(1), \ldots, \varphi(k) \rangle \in (V_D)^k$.
2. The *embedding occurrence* of T is the set $Eoc(\varphi) = \{\varphi(1), \ldots, \varphi(k)\} \subseteq V_D$.
3. The *root occurrence* of T: $Roc(\varphi) = \varphi(1) \in V_D$
4. The *document occurrence* of T is the index $Doc(\varphi) = i$ such that $Eoc(\varphi) \subseteq V_{D_i}$ for some $1 \leq i \leq |\mathcal{D}|$.

Example 2. In Fig. 1, we see that the pattern tree S has eight total occurrences $\varphi_1 = \langle 1, 2, 3, 4, 10, 11 \rangle$, $\varphi_2 = \langle 1, 2, 4, 3, 10, 11 \rangle$, $\varphi_3 = \langle 1, 2, 3, 4, 10, 13 \rangle$, $\varphi_4 = \langle 1, 2, 4, 3, 10, 13 \rangle$, $\varphi_5 = \langle 1, 10, 11, 13, 2, 3 \rangle$, $\varphi_6 = \langle 1, 10, 13, 11, 2, 3 \rangle$, $\varphi_7 = \langle 1, 10, 11, 13, 2, 4 \rangle$, and $\varphi_8 = \langle 1, 10, 13, 11, 2, 4 \rangle$ in the data tree D, where we identify the matching φ_i and $Toc(\varphi_i)$. On the other hand, there are four embedding occurrences $Eoc(\varphi_1) = Eoc(\varphi_2) = \{1, 2, 3, 4, 10, 11\}$, $Eoc(\varphi_3) = Eoc(\varphi_4) = \{1, 2, 3, 4, 10, 13\}$, $Eoc(\varphi_5) = Eoc(\varphi_6) = \{1, 2, 3, 10, 11, 13\}$, and $Eoc(\varphi_7) = Eoc(\varphi_8) = \{1, 2, 4, 10, 11, 13\}$, and there is one root occurrence $\varphi_1(1) = \varphi_2(1) = \cdots = \varphi_8(1) = 1$ of T in D.

Now, we analyze the relationship among the above definitions of the occurrences by introducing an ordering $\geq_{\text{occ}}$ on the definitions. For any types of occurrences $\tau, \pi \in \{Toc, Eoc, Roc, Doc\}$, we say π is *stronger than or equal to* τ, denoted by $\pi \geq_{\text{occ}} \tau$, iff for every matchings $\varphi_1, \varphi_2 \in \mathcal{M}^{\mathcal{D}}(T)$ from T to $\mathcal{D}$, $\pi(\varphi_1) = \pi(\varphi_2)$ implies $\tau(\varphi_1) = \tau(\varphi_2)$. For an unordered pattern $T \in \mathcal{U}$, we denote by $TO^{\mathcal{D}}(T)$, $EO^{\mathcal{D}}(T)$, $RO^{\mathcal{D}}(T)$, and $DO^{\mathcal{D}}(T)$ the set of the total, the embedding, the root and the document occurrences of T in $\mathcal{D}$, respectively. The first lemma describes a linear ordering on classes of occurrences and the second lemma gives the relation between the relative size of the occurrences.

Lemma 1. $Toc \geq_{\text{occ}} Eoc \geq_{\text{occ}} Roc \geq_{\text{occ}} Doc$.

Lemma 2. *Let $\mathcal{D}$ be a database and T be a pattern. Then,*

$$\frac{|TO^{\mathcal{D}}(T)|}{|EO^{\mathcal{D}}(T)|} = k^{\Theta(k)} \quad \textit{and} \quad \frac{|EO^{\mathcal{D}}(T)|}{|RO^{\mathcal{D}}(T)|} = n^{\Theta(k)}$$

over all pattern $T \in \mathcal{U}$ and all databases $\mathcal{D} \in 2^{\mathcal{U}}$ satisfying $k \leq cn$ for some $0 < c < 1$, where k is the size of T and n is the size of a database.

Proof. Omitted. For the proof, please consult the technical report [5]. □

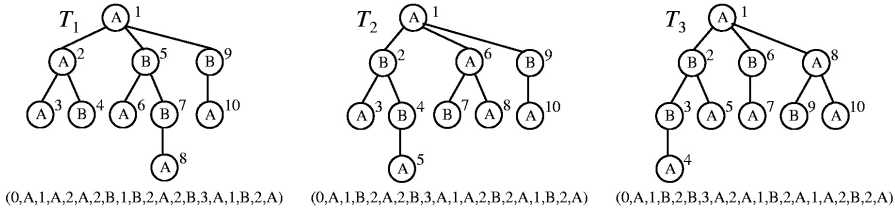

Fig. 2. The depth-label sequences of labeled ordered trees

For any of four types of the occurrences $\tau \in \{Toc, Eoc, Roc, Doc\}$, the *$\tau$-count* of an unordered pattern U in a given database $\mathcal{D}$ is $|\tau^{\mathcal{D}}(U)|$. Then, the (relative) *τ-frequency* is the ratio $freq_{\mathcal{D}}(T) = |\tau^{\mathcal{D}}(T)|/||\mathcal{D}||$ for $\tau \in \{Toc, Eoc, Roc\}$ and $freq_{\mathcal{D}}(T) = |Doc^{\mathcal{D}}(T)|/|\mathcal{D}|$ for $\tau = Doc$. A *minimum frequency threshold* is any number $0 \leq \sigma \leq 1$. Then, we state our data mining problems as follows.

Frequent Unordered Tree Discovery with Occurrence Type τ
Given a database $\mathcal{D} \subseteq \mathcal{U}$ and a positive number $0 \leq \sigma \leq 1$, find all unordered patterns $U \in \mathcal{U}$ appearing in $\mathcal{D}$ with relative τ-frequency at least σ, i.e., $freq_{\mathcal{D}}(T)$.

In what follows, we concentrate on the frequent unordered tree discovery problem with embedding occurrences with Eoc although Toc is more natural choice from the view of data mining. However, we note that it is easy to extend the method and the results in this paper for coarser occurrences Roc and Doc by simple preprocessing. The following substructure enumeration problem, is a special case of the frequent unordered tree discovery problem with embedding occurrences where $\sigma = 1/||\mathcal{D}||$.

Substructure Discovery Problem for Unordered Trees
Given a data tree $D \in \mathcal{U}$, enumerate all the labeled unordered trees $T \in \mathcal{U}$ embedded in D, that is, T occurs in D at least once.

Throughout this paper, we adopt the *first-child next-sibling representation* [2] as the representation of unordered and ordered trees in implementation. For the detail of the representation, see some textbook, e.g., [2].

3 Canonical Representation for Unordered Trees

In this section, we give the canonical representation for unordered tree patterns according to Nakano and Uno [12].

3.1 Depth Sequence of a Labeled Unordered Tree

First, we introduce some technical definitions on ordered trees. We use labeled ordered trees in $\mathcal{T}$ as the representation of labeled unordered trees in $\mathcal{U}$, where $U \in \mathcal{U}$ can be represented by any $T \in \mathcal{T}$ such that U is obtained from T by ignoring its sibling relation B_T. Two ordered trees T_1 and $T_2 \in \mathcal{T}$ are *equivalent*

each other as unordered trees, denoted by $T_1 \equiv T_2$, if they represent the same unordered tree.

We encode a labeled ordered tree of size k as follows [4,11,20]. Let T be a labeled ordered tree of size k. Then, the *depth-label sequence* of T is the sequence

$$C(T) = ((dep(v_1), label(v_1)), \ldots, (dep(v_k), label(v_k)) \in (\mathbf{N} \times \mathcal{L})^*,$$

where $v_1, \ldots, v_k$ is the list of the nodes of T ordered by the preorder traversal of T and each $(dep(v_i), label(v_i)) \in \mathbf{N} \times \mathcal{L}$ is called a *depth-label pair*. Since $\mathcal{T}$ and $\mathbf{N} \times \mathcal{L}$ have one-to-one correspondence, we identify them in what follows. See Fig. 2 for examples of depth-label sequences.

Next, we introduce the total ordering $\geq$ over depth-label sequences as follows. For depth-label pairs $(d_i, \ell_i) \in \mathbf{N} \times \mathcal{L}$ $(i = 1, 2)$, we define $(d_1, \ell_1) > (d_2, \ell_2)$ iff either (i) $d_1 > d_2$ or (ii) $d_1 = d_2$ and $\ell_1 > \ell_2$. Then, $C(T_1) = (x_1, \ldots, x_m)$ is *heavier than* $C(T_2) = (y_1, \ldots, y_m)$, denoted by $C(T_1) \geq_{\text{lex}} C(T_2)$, iff $C(T_1)$ is lexicographically larger than or equal to $C(T_2)$ as sequences over alphabet $\mathbf{N} \times \mathcal{L}$. That is, $C(T_1) \geq_{\text{lex}} C(T_2)$ iff there exists some k such that (i) $x_i = y_i$ for each $i = 1, \ldots, k-1$ and (ii) either $x_k > y_k$ or $m > k-1 = n$. By identifying ordered trees and their depth-label sequences, we may often write $T_1 \geq_{\text{lex}} T_2$ instead of $C(T_1) \geq_{\text{lex}} C(T_2)$.

Now, we give the canonical representation for labeled unordered trees as follows.

Definition 2 ([12]). A labeled ordered tree T is *in the canonical form* or a *canonical representation* if its depth-label sequence $C(T)$ is heaviest among all ordered trees over $\mathcal{L}$ equivalent to T, i.e., $C(T) = \max\{\, C(S) \mid S \in \mathcal{T}, S \equiv T \,\}$.

The *canonical ordered tree representation* (or canonical representation, for short) of an unordered tree $U \in \mathcal{U}$, denoted by $COT(U)$, is the labeled ordered tree $T \in \mathcal{T}$ in the canonical form that represents U as unordered tree. We denote by $\mathcal{C}$ the class of the canonical ordered tree representations of labeled unordered trees over $\mathcal{L}$.

The next lemma gives a characterization of the canonical representations for unordered trees [12].

Lemma 3 (Left-heavy condition [12]). *A labeled ordered tree T is the canonical representation of some unordered tree* iff *T is* left-heavy*, that is, for any node $v_1, v_2 \in V$, $(v_1, v_2) \in B$ implies $C(T(v_1)) \geq_{\text{lex}} C(T(v_2))$.*

Example 3. Three ordered trees T_1, T_2, and T_3 in Fig. 2 represents the same unordered tree, but not as ordered trees. Among them, T_1 is left-heavy and thus it is the canonical representation of a labeled unordered tree under the assumption that $A > B > C$. On the other hand, T_2 is not canonical since the depth-label sequence $C(T_2(2)) = (1B, 2A, 2B, 3A)$ is lexicographically smaller than $C(T_2(6)) = (1A, 2B, 2A)$ and this violates the left-heavy condition. T_3 is not canonical since $B < A$ implies $C(T_3(3)) = (2B, 3A) <_{\text{lex}} C(T_3(5)) = (2A)$.

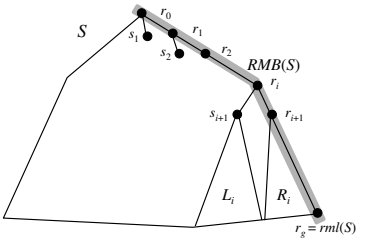

Fig. 3. Notions on a canonical representation

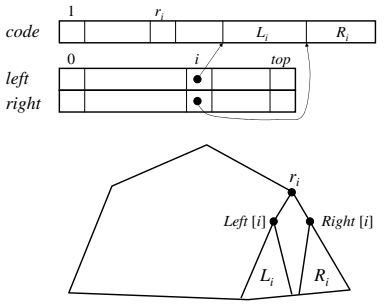

Fig. 4. Data structure for a pattern

3.2 The Reverse Search Principle and the Rightmost Expansions

The *reverse search* is a general scheme for designing efficient algorithm for hard enumeration problems [6,16]. In reverse search, we define the parent-child relation $P \subseteq \mathcal{S} \times \mathcal{S}$ on the solution space $\mathcal{S}$ of the problem so that each solution X has the unique parent $P(X)$. Since this relation forms a search tree over $\mathcal{S}$, we enumerate the solutions starting from the *root* solutions and by computing the children for the solutions. Iterating this process, we can generate all the solutions without duplicates.

Let T be a labeled ordered tree having at least two nodes. We denote by $P(T)$ the unique labeled ordered tree derived from T by removing the rightmost leaf $rml(T)$. We say $P(T)$ is the parent tree of T or T is a child tree of $P(T)$. The following lemma is crucial to our result.

Lemma 4 ([12]). For any labeled ordered tree $T \in \mathcal{T}$, if T is in canonical form then so is its parent $P(T)$, that is, $T \in \mathcal{C}$ implies $P(T) \in \mathcal{C}$.

Proof. For a left-heavy tree $T \in \mathcal{C}$, the operation to remove the rightmost leaf from T does not violate the left-heavy condition of T. It follows from Lemma 3 that the lemma holds. □

Definition 3 (Rightmost expansion [3,11,20]). Let $S \in \mathcal{T}$ be a labeled ordered tree on $\mathcal{L}$. Then a labeled ordered tree $T \in \mathcal{T}$ is the *rightmost expansion* of S if T is obtained from S by attaching the new node v as the rightmost child of a node on the rightmost branch $RMB(S)$ of S. If $(dep(v), label(v)) = (d, \ell)$ then we call T the (d, ℓ)-*expansion* of S. We define the $(0, \ell)$-expansion of $\perp$ to be the single node tree with label ℓ.

Since newly attached node v is the last node in preorder on T, we denote the (d, ℓ)-expansion of S by $S \cdot (d, \ell)$. We sometimes write $\boldsymbol{v} = (dep(v), label(v))$.

If we can compute the set of the child trees of a given labeled ordered tree $S \in \mathcal{T}$ then we can enumerate all the labeled ordered trees in $\mathcal{T}$. The method is called the *rightmost expansion* and has been independently studied in [3,11,20].

Algorithm Unot$(\mathcal{D}, \mathcal{L}, \sigma)$
Input: the database $\mathcal{D} = \{D_1, \ldots, D_m\}$ $(m \geq 0)$ of labeled unordered trees, a set $\mathcal{L}$ of labels, and the minimum frequency threshold $0 \leq \sigma \leq 1$.
Output: the set $\mathcal{F} \subseteq \mathcal{C}$ of all frequent unordered trees of size at most $N_{\max}$.
Method:
1. $\mathcal{F} := \emptyset$; $\alpha := \lceil |\mathcal{D}|\sigma \rceil$; //Initialization
2. For any label $\ell \in \mathcal{L}$, do:
   $T_\ell := (0, \ell)$; /* 1-pattern with copy depth 0 */
   Expand$(T_\ell, \mathcal{O}, 0, \alpha, \mathcal{F})$;
3. Return $\mathcal{F}$; //The set of frequent patterns

Fig. 5. An algorithm for discovering all frequent unordered trees

4 Mining Frequent Unordered Tree Patterns

In this section, we present an efficient algorithm Unot for solving the frequent unordered tree pattern discovery problem w.r.t. the embedding occurrences.

4.1 Overview of the Algorithm

In Fig. 5, we show our algorithm Unot for finding all the canonical representations for frequent unordered trees in a given database $\mathcal{D}$. A key of the algorithm Unot is efficient enumeration of all the canonical representations, which is implemented by the subprocedures FindAllChildren in Fig. 5 to run in $O(1)$ time per pattern. Another key is incremental computation of their occurrences. This is implemented by the subprocedure UpdateOcc in Fig. 8 to run in $O(bk^2m)$ time per pattern, where b is the maximum branching factor in $\mathcal{D}$, k is the maximum pattern size, m is the number of embedding occurrences. We give the detailed descriptions of these procedures in the following subsections.

4.2 Enumerating Unordered Trees

First, we prepare some notations (Fig. 3.1). Let T be a labeled ordered tree with the rightmost branch $RMB(T) = (r_0, r_1, \ldots, r_g)$. For every $i = 0, 1, \ldots, g$, if r_i has two or more children then we denote by s_{i+1} the child of r_i preceding r_{i+1}, that is, s_{i+1} is the second rightmost child of r_i. Then, we call $L_i = T(s_{i+1})$ and $R_i = T(r_{i+1})$ the left and the right tree of r_i. If r_i has exactly one child r_{i+1} then we define $L_i = \top_\infty$, where $\top_\infty$ is a special tree such that $\top_\infty >_{\text{lex}} S$ for any $S \in \mathcal{T}$. For a pattern tree $T \in \mathcal{T}$, we sometimes write $L_i^{(T)}$ and $R_i^{(T)}$ for L_i and R_i by indicating the pattern tree T.

By Lemma 3, an ordered tree is in canonical form iff it is left-heavy. The next lemma claims that the algorithm only checks the left trees and the right trees to check if the tree is in canonical form.

Lemma 5 ([12]). *Let $S \in \mathcal{C}$ be a canonical representation and T be a child tree of S with the rightmost branch $(r_0, \ldots, r_g)$, where $g \geq 0$. Then, T is in canonical form iff $L_i \geq_{\text{lex}} R_i$ holds in T for every $i = 0, \ldots, g-1$.*

Procedure Expand$(S, \mathcal{O}, c, \alpha, \mathcal{F})$
Input: A canonical representation $S \in \mathcal{U}$, the embedding occurrences $\mathcal{O} = EO^{\mathcal{D}}(S)$, and the copy-depth c, nonnegative integer α, and the set $\mathcal{F}$ of the frequent patterns.
Method:
- If $(|\mathcal{O}| < \alpha)$ then return;
  Else $\mathcal{F} := \mathcal{F} \cup \{S\}$;
- For each $\langle S{\cdot}(i, \ell), c_{\rm new}\rangle \in$ FindAllChildren(S, c), do;
  - $T := S{\cdot}(i, \ell)$;
  - $\mathcal{P} :=$ UpdateOcc$(T, \mathcal{O}, (i, \ell))$;
  - Expand$(T, \mathcal{P}, c_{\rm new}, \alpha, \mathcal{F})$;

Fig. 6. A depth-first search procedure Expand

Let T be a labeled ordered tree with the rightmost branch $RMB(T) = (r_0, r_1, \ldots, r_g)$. We say $C(L_i)$ and $C(R_i)$ have a *disagreement* at the position j if $j \leq \min(|C(L_i)|, |C(R_i)|)$ and the j-th components of $C(L_i)$ and $C(R_i)$ are different pairs.

Suppose that T is in canonical form. During a sequence of rightmost expansions to T, the i-th right tree R_i grows as follows.

1. Firstly, when a new node v is attached to r_i as a rightmost child, the sequence is initialized to $C(R_i) = \boldsymbol{v} = (dep(v), label(v))$.
2. Whenever a new node v of depth $d = dep(v) > i$ comes to T, the right tree R_i grows. In this case, v is attached as the rightmost child of r_{d-1}. There are two cases below:
   (i) Suppose that there exists a disagreement in $C(L_i)$ and $C(R_i)$. If $\boldsymbol{r}_{dep(v)} \geq \boldsymbol{v}$ then the rightmost expansion with v does not violate the left-heavy condition of T, where $r_{dep(v)}$ is the node preceding v in the new tree.
   (ii) Otherwise, we know that $C(R_i)$ is a prefix of $C(L_i)$. In this case, we say R_i is copying L_i. Let $m = |C(R_i)| < |C(L_i)|$ and $\boldsymbol{w}$ be the m-th component of $C(L_i)$. For every new node $\boldsymbol{v}$, $T{\cdot}\boldsymbol{v}$ is a valid expansion if $\boldsymbol{w} \geq \boldsymbol{v}$ and $\boldsymbol{r}_{dep(v)} \geq \boldsymbol{v}$. Otherwise, it is invalid.
   (iii) In cases (i) and (ii) above, if $r_{dep(v)-1}$ is a leaf of the rightmost branch of T then $\boldsymbol{r}_{dep(v)}$ is undefined. In this case, we define $\boldsymbol{r}_{dep(v)} = \top_\infty$.
3. Finally, T reaches $C(L_i) = C(R_i)$. Then, the further rightmost expansion to R_i is not possible.

If we expand a given unordered pattern T so that all the right trees $R_0, \ldots, R_g$ satisfy the above conditions, then the resulting tree is in canonical form.

Let $RMB(T) = (r_0, r_1, \ldots, r_g)$ be the rightmost branch of T. For every $i = 0, 1, \ldots, g-1$, the internal node r_i is said to be *active* at depth i if $C(R_i)$ is a prefix of $C(L_i)$. The *copy depth* of T is the depth of the highest active node in T. To deal with special cases, we introduce the following trick: We define the leaf r_g to be always active. Thus we have that if all nodes but r_g are not active

Procedure FindAllChildren(T, c) :
Method : Return the set *Succ* of all pairs $\langle S, c\rangle$, where S is the canonical child tree of T and c is its copy depth generated by the following cases:
Case I : If $C(L_k) = C(R_k)$ for the copy depth k:
- The canonical child trees of T are $T{\cdot}(1, \ell_1), \ldots, T{\cdot}(k+1, \ell_{k+1})$, where $label(r_i) \geq \ell_i$ for every $i = 1, \ldots, k+1$. The trees $T{\cdot}(k+2, \ell_{k+2}), \ldots, T{\cdot}(g+1, \ell_{g+1})$ are not canonical.
- The copy depth of $T{\cdot}(i, \ell_i)$ is $i-1$ if $label(r_i) = \ell_i$ and i otherwise for every $i = 1, \ldots, k+1$.

Case II : If $C(L_k) \neq C(R_k)$ for the copy depth k:
- Let $m = |C(R_k)|+1$ and $\boldsymbol{w} = (d, \ell)$ be the m-th component of $C(L_k)$ (the next position to be copied). The canonical child trees of T are $T{\cdot}(1, \ell_1), \ldots, T{\cdot}(d, \ell_d)$, where $label(r_i) \geq \ell_i$ for every $i = 1, \ldots, d-1$ and $\ell \geq \ell_d$ holds.
- The copy depth of $T{\cdot}(i, \ell_i)$ is $i-1$ if $label(r_i) = \ell_i$ and i otherwise for every $i = 1, \ldots, d-1$. The copy depth of $T{\cdot}(d, \ell_d)$ is k if $\boldsymbol{w} = \boldsymbol{v}$ and d otherwise.

Fig. 7. The procedure FindAllChildren

then its copy-depth is g. This trick greatly simplies the description of the update below.

Now, we explain how to generate all child trees of a given canonical representation $T \in \mathcal{C}$. In Fig. 7, we show the algorithm FindAllChildren that computes the set of all canonical child trees of a given canonical representation as well as their copy depths.

The algorithm is almost same as the algorithm for unlabeled unordered trees described in [12]. The update of the copy depth is slightly different from [12] by the existence of the labels. Let T be a labeled ordered tree with the rightmost branch $RMB(T) = (r_0, r_1, \ldots, r_g)$ and the copy depth $k \in \{-1, 0, 1, \ldots, g-1\}$. Note that in the procedure FindAllChildren, the case where all but r_g are inactive, including the case for chain trees, is implicitly treated in Case I.

To implement the algorithm FindAllChildren to enumerate the canonical child trees in $O(1)$ time per tree, we have to perform the following operation in $O(1)$ time: updating a tree, access to the sequence of left and right trees, maintenance of the position of the shorter prefix at the copy depth, retrieval of the depth-label pair at the position, and the decision of the equality $C(L_i) = C(R_i)$.

To do this, we represent a pattern T by the structure shown in Fig. 4.

- An array $code : [1..size] \to (\mathbf{N} \times \mathcal{L})$ of depth-label pairs that stores the depth-label sequence of T with length $size \geq 0$.
- A stack $RMB : [0..top] \to (\mathbf{N} \times \mathbf{N} \times \{=, \neq\})$ of the triples $(left, right, cmp)$. For each $(left, right, cmp) = RMB[i]$, $left$ and $right$ are the starting positions of the subsequences of *code* that represent the left tree L_i and the right tree R_i, and the flag $cmp \in \{=, \neq\}$ indicates whether $L_i = R_i$ holds. The length of the rightmost branch is $top \geq 0$.

It is not difficult to see that we can implement all the operation in FindAllChildren of Fig. 7 to work in $O(1)$ time, where an entire tree is not output

Algorithm UpdateOcc$(T, \mathcal{O}, d, \ell)$
Input: the rightmost expansion of a pattern S, the embedding occurrence list $\mathcal{O} = EO^{\mathcal{D}}(S)$, the depth $d \geq 1$ and a label $\ell \in \mathcal{L}$ of the rightmost leaf of T.
Output: the new list $\mathcal{P} = EO^{\mathcal{D}}(T)$.
Method:
- $\mathcal{P} := \emptyset$;
- For each $\varphi \in \mathcal{O}$, do:
  - $x := \varphi(r_{d-1})$; /* the image of the parent of the new node $r_d = (d, \ell)$ */
  - For each child y of x do:
    - If $label_D(y) = \ell$ and $y \notin E(\varphi)$ then
      $\xi := \varphi \cdot y$ and $flag := true$;
    - Else, skip the rest and continue the for-loop;
    - For each $i = 1, \ldots, d-1$, do:
      If $C(L_i) = C(R_i)$ but $\xi(left_i) \neq \xi(right_i)$ then
      $flag := false$, and then break the inner for-loop;
    - If $flag = true$ then $\mathcal{P} = \mathcal{P} \cup \{\xi\}$;
- Return $\mathcal{P}$;

Fig. 8. An algorithm for updating embedding occurrence lists of a pattern

but the difference from the previous tree. The proof of the next lemma is almost same to [12] except the handling of labels.

Lemma 6 ([12]). *For every canonical representation T and its copy depth $c \geq 0$,* FindAllChildren *of Fig. 7 computes the set of all the canonical child trees of T in $O(1)$ time per tree, when only the differences to T are output.*

The time complexity $O(1)$ time per tree of the above algorithm is significantly faster than the complexity $O(k^2)$ time per tree of the straightforward algorithm based on Lemma 3, where k is the size of the computed tree.

4.3 Updating the Occurrence List

In this subsection, we give a method for incrementally computing the embedding occurrences $EO^{\mathcal{D}}(T)$ of the child tree T from the occurrences $EO^{\mathcal{D}}(S)$ of the canonical representation S. In Fig. 8, we show the procedure UpdateOcc that, given a canonical child tree T and the occurrences $EO^{\mathcal{D}}(S)$ of the parent tree S, computes its embedding occurrences $EO^{\mathcal{D}}(T)$.

Let T be a canonical representation for a labeled unordered tree over $\mathcal{L}$ with domain $\{1, \ldots, k\}$. Let $\varphi \in \mathcal{M}^{\mathcal{D}}(T)$ be a matching from T into $\mathcal{D}$. Recall that the total and the embedding occurrences of T associated with φ is $TO(\varphi) = \langle \varphi(1), \ldots, \varphi(k) \rangle$ and $EO(\varphi) = \{\varphi(1), \ldots, \varphi(k)\}$, respectively. For convenience, we identify φ to $TO(\varphi)$.

We encode an embedding occurrence EO in one of the total occurrences φ corresponding to $EO = EO(\varphi)$. Since there are many total occurrences corresponding to EO, we introduce the canonical representation for embedding occurrences similarly as in Section 3.

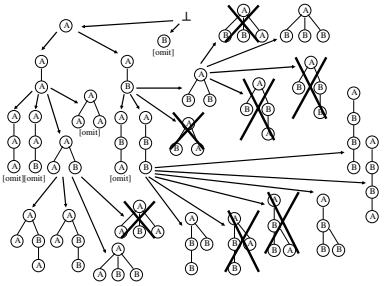

Fig. 9. A search tree for labeled unordered trees

Two total occurrences φ_1 and φ_2 are *equivalent each other* if $EO(\varphi_1) = EO(\varphi_2)$. The occurrences φ_1 is *heavier than* φ_2, denote by $\varphi_1 \geq_{\text{lex}} \varphi_2$, if φ_1 is lexicographically larger than φ_2 as the sequences in $\mathbf{N}^*$. We give the canonical representation for the embedding occurrences.

Definition 4. Let T be a canonical form of a labeled unordered tree and $EO \subseteq V_{\mathcal{D}}$ be its embedding occurrence in $\mathcal{D}$. The canonical representation of EO, denoted by $CR(EO)$, is the total occurrence $\varphi \in \mathcal{M}^{\mathcal{D}}(T)$ that is the heaviest tuple in the equivalence class $\{\, \varphi' \in \mathcal{M}^{\mathcal{D}}(T) \,|\, \varphi' \equiv \varphi \,\}$.

Let $\varphi = \langle \varphi(1), \ldots, \varphi(k) \rangle$ be an total occurrence of T over $\mathcal{D}$. We denote by $P(\varphi)$ the unique total occurrence of length $k-1$ derived from φ by removing the last component $\varphi(k)$. We say $P(\varphi)$ is a *parent occurrence* of φ. For a node $v \in V_T$ in T, we denote by $\varphi(T(v)) = \langle \varphi(i), \varphi(i+1), \ldots, \varphi(i+|T(v)|-1) \rangle$ the restriction of φ to the subtree $T(v)$, where $\langle i, i+1, \ldots, i+|T(v)|-1 \rangle$ is the nodes of $T(v)$ in preorder.

Now, we consider the incremental computation of the embedding occurrences.

Lemma 7. *Let $k \geq 1$ be any positive integer, S be a canonical tree, and φ be its canonical occurrence of S in $\mathcal{D}$. For a node $w \in V_{\mathcal{D}}$, let $T = S \cdot \boldsymbol{v}$ be a child tree of S with the rightmost branch $(r_0, \ldots, r_g)$. Then, the mapping $\phi = \varphi \cdot \boldsymbol{w}$ is a canonical total occurrence of T in D iff the following conditions (1)–(4) hold.*

(1) $label_D(w) = label_T(v)$.
(2) For every $i = 1, \ldots, k-1$, $w \neq \varphi(i)$.
(3) w is a child of $\varphi(r_{d-1})$, where $r_{d-1} \in RMB(S)$ is the node of depth $d-1$ on the rightmost branch $RMB(S)$ of S.
(4) $C(L_i) = C(R_i)$ implies $\phi(root(L_i)) = \phi(root(R_i))$ for every $i = 0, \ldots, g-1$.

Proof. For any total occurrence $\varphi \in \mathcal{M}^{\mathcal{D}}(T)$, if φ is in canonical form then so is its parent $P(\varphi)$. Moreover, φ is the canonical form iff φ is *partially left-heavy*, that is, for any nodes $v_1, v_2 \in V_T$, both of $(v_1, v_2) \in B$ and $T(v_1) = T(v_2)$ imply $\varphi(T(v_1)) \geq_{\text{lex}} \varphi(T(v_2))$. Thus the lemma holds. □

Lemma 7 ensures the correctness of the procedure UpdateOcc of Fig. 8. To show the running time, we can see that the decision $C(L_i) = C(R_i)$ can be decidable in $O(1)$ time using the structure shown in Fig. 4. Note that every canonical tree has at least one canonical child tree.

Thus, we obtain the main theorem of this paper as follows.

Theorem 1. *Let $\mathcal{D}$ be a database and $0 \leq \sigma \leq 1$ be a threshold. Then, the algorithm* Unot *of Fig. 5 computes all the canonical representations for the frequent unordered trees w.r.t. embedding occurrences in $O(kb^2m)$ time per pattern, where b is the maximum branching factor in $V_{\mathcal{D}}$, k is the maximum size of patterns enumerated, and m is the number of embeddings of the enumerated pattern.*

Proof. Let S be a canonical tree and T be a child tree of S. Then, the procedure UpdateOcc of Fig. 8 computes the list of all the canonical total occurrences of T in $O(k'bm')$ time, where $k' = |T|$ and $m' = |EO(S)|$. From Lemma 6 and the fact $|EO(T)| = O(b|EO(S)|)$, we have the result. □

Fig. 9 illustrates the computation of the algorithm Unot that is enumerating a subset of labeled unordered trees of size at most 4 over $\mathcal{L} = \{A, B\}$. The arrows indicates the parent-child relation and the crossed trees are non-canonical ones.

4.4 Comparison to a Straightforward Algorithm

We compare our algorithm Unot to the following straightforward algorithm Naive. Given a database $\mathcal{D}$ and a threshold σ, Naive enumerates all the labeled ordered trees over $\mathcal{L}$ using the rightmost expansion, and then for each tree it checks if it is in canonical form applying Lemma 3. Since the check takes $O(k^2)$ time per tree, this stage takes $O(|\mathcal{L}|k^3)$ time. It takes $O(n^k)$ time to compute all the embedding occurrences in $\mathcal{D}$ of size n. Thus, the overall time is $O(|\mathcal{L}|k^3+n^k)$.

On the other hand, Unot computes the canonical representations in $O(kb^2m)$ time, where the total number m of the embedding occurrences of T is $m = O(n^k)$ in the worst case. However, m will be much smaller than n^k as the pattern size of T grows. Thus, if it is the case that b is a small constant and m is much smaller than n, then our algorithm will be faster than the straightforward algorithm.

5 Conclusions

In this paper, we presented an efficient algorithm Unot that computes all frequent labeled unordered trees appearing in a collection of data trees. This algorithm has a provable performance in terms of the output size unlike previous graph mining algorithms. It enumerates each frequent pattern T in $O(kb^2m)$ per pattern, where k is the size of T, b is the branching factor of the data tree, and m is the total number of occurrences of T in the data trees.

We are implementing a prototype system of the algorithm and planning the computer experiments on synthesized and real-world data to give empirical evaluation of the algorithm. The results will be included in the full paper.

Some graph mining algorithms such as AGM [8], FSG [9], and gSpan [19] use various types of the canonical representation for general graphs similar to our canonical representation for unordered trees. AGM [8] and FSG [9] employ the adjacent matrix with the lexicographically smallest row vectors under the permutation of rows and columns. gSpan [19] uses as the canonical form the DFS code generated with the depth-first search over a graph. It is a future problem to study the relationship among these techniques based on canonical coding and to develop efficient coding scheme for restricted subclasses of graph patterns.

Acknowledgement. Tatsuya Asai and Hiroki Arimura would like to thank Ken Satoh, Hideaki Takeda, Tsuyoshi Murata, and Ryutaro Ichise for the fruitful discussions on Semantic Web mining, and to thank Takashi Washio, Akihiro Inokuchi, Michihiro Kuramochi, and Ehud Gudes for the valuable discussions and comments on graph mining. Tatsuya Asai is grateful to Setsuo Arikawa for his encouragement and support for this work.

References

1. K. Abe, S. Kawasoe, T. Asai, H. Arimura, and S. Arikawa. Optimized Substructure Discovery for Semi-structured Data, In *Proc. PKDD'02*, 1–14, LNAI 2431, 2002.
2. Aho, A. V., Hopcroft, J. E., Ullman, J. D., *Data Structures and Algorithms*, Addison-Wesley, 1983.
3. T. Asai, K. Abe, S. Kawasoe, H. Arimura, H. Sakamoto, S. Arikawa, Efficient Substructure Discovery from Large Semi-structured Data, In *Proc. SIAM SDM'02*, 158–174, 2002.
4. T. Asai, H. Arimura, K. Abe, S. Kawasoe, S. Arikawa, Online Algorithms for Mining Semi-structured Data Stream, In *Proc. IEEE ICDM'02*, 27–34, 2002.
5. T. Asai, H. Arimura, T. Uno, S. Nakano, Discovering Frequent Substructures in Large Unordered Trees, *DOI Technical Report DOI-TR 216*, Department of Informatics, Kyushu University, June 2003.
   `http://www.i.kyushu-u.ac.jp/doitr/trcs216.pdf`
6. D. Avis, K. Fukuda, Reverse Search for Enumeration, Discrete Applied Mathematics, 65(1–3), 21–46, 1996.
7. L. B. Holder, D. J. Cook, S. Djoko, Substructure Discovery in the SUBDUE System, In *Proc. KDD'94*, 169–180, 1994.
8. A. Inokuchi, T. Washio, H. Motoda, An Apriori-Based Algorithm for Mining Frequent Substructures from Graph Data, In *Proc. PKDD'00*, 13–23, LNAI, 2000.
9. M. Kuramochi, G. Karypis, Frequent Subgraph Discovery, In *Proc. IEEE ICDM'01*, 2001.
10. T. Miyahara, Y. Suzuki, T. Shoudai, T. Uchida, K. Takahashi, H. Ueda, Discovery of Frequent Tag Tree Patterns in Semistructured Web Documents, In *Proc. PAKDD'02*, 341–355, LNAI, 2002.
11. S. Nakano, Efficient generation of plane trees, Information Processing Letters, 84, 167–172, 2002.
12. S. Nakano, T. Uno, Efficient Generation of Rooted Trees, *NII Technical Report NII-2003-005E*, ISSN 1346-5597, Natinal Institute of Informatics, July 2003.
13. S. Nestrov, S. Abiteboul, R. Motwani, Extracting Schema from Semistructured Data, In *Proc. SIGKDD'98*, 295–306, ACM, 1998.

14. S. Nijssen, J. N. Kok, Effcient Discovery of Frequent Unordered Trees, In *Proc. MGTS'03*, September 2003.
15. A. Termier, M. Rousset, M. Sebag, TreeFinder: a First Step towards XML Data Mining, In *Proc. IEEE ICDM'02*, 450–457, 2002.
16. T. Uno, A Fast Algorithm for Enumerating Bipartite Perfect Matchings, In *Proc. ISAAC'01*, LNCS, 367–379, 2001.
17. N. Vanetik, E. Gudes, E. Shimony, Computing Frequent Graph Patterns from Semistructured Data, In *Proc. IEEE ICDM'02*, 458–465, 2002.
18. K. Wang, H. Liu, Schema Discovery from Semistructured Data, In *Proc. KDD'97*, 271–274, 1997.
19. X. Yan, J. Han, gSpan: Graph-Based Substructure Pattern Mining, In *Proc. IEEE ICDM'02*, 721–724, 2002.
20. M. J. Zaki. Efficiently Mining Frequent Trees in a Forest, In *Proc. SIGKDD 2002*, ACM, 2002.