Efficient Tree Mining Using Reverse Search

Asai, Tatsuya
Kyushu University

Arimura, Hiroki
Kyushu University

Uno, Takeaki
National Institute of Informatics

Nakano, Shin-ichi
Gunma University

他

出版情報：DOI Technical Report. 218, 2003-06. Department of Informatics, Kyushu University
バージョン：
権利関係：

DOI-TR 218, Department of Informatics, Kyushu University, June, 2003
http://www.i.kyushu-u.ac.jp/doitr/trcs218.pdf (Submitting)

Efficient Tree Mining Using Reverse Search

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

[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,ksatoh}@nii.ac.jp
[3] Gunma University, Kiryu-shi, Gunma 376–8515, JAPAN
nakano@cs.gunma-u.ac.jp

Abstract. In this paper, we review our data mining algorithms for discovering frequent substructures in a large collection of semi-structured data, where both of the patterns and the data are modeled by *labeled trees*. These algorithms, namely FREQT for mining frequent ordered trees and UNOT for mining frequent unordered trees, efficiently enumerate all frequent tree patterns without duplicates using *reverse search*, which is a general scheme for designing efficient algorithms for hard enumeration problems, and incrementally compute of the occurrences of a pattern. We also discuss classes of trees to which reverse search is applicable, such as itemsets, sequential episodes, path trees, and graphs.

Correspondence:
Tatsuya Asai
Department of Informatics, Kyushu University,
6-10-1 Hakozaki Higashi-ku, Fukuoka 812-8581, JAPAN
E-mail: t-asai@i.kyushu-u.ac.jp
phone: +81-92-642-2697, fax: +81-92-642-2698

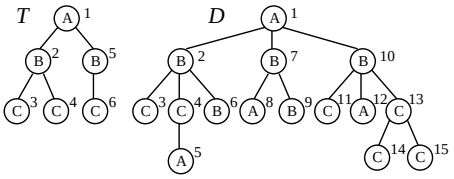


Fig. 1. A data tree D and a pattern tree T

1 Introduction

By rapid progress of network and storage technologies, huge amounts of electronic data has 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. Hence, there have been increasing demands for efficient methods for discovering patterns in large collections of graph and tree structures, namely *semi-structured data mining* and *graph mining* [1, 5–7, 10–12, 14, 17, 18, 20–23].

In this paper, we review our study on efficient tree mining from large semi-structured data [1, 5–7] particularly from the view of *reverse search*, a general scheme for designing efficient algorithms for hard enumeration problems. In our framework of semi-structured data mining, patterns and data are both modeled by trees whose nodes are labeled by symbols. We present two algorithms FREQT [1, 5] and UNOT [7] for discovering ordered trees and unordered trees, respectively, from large collections of tree structured data.

By using efficient enumeration of patterns using reverse search and incremental computation of their occurrences as key techniques, FREQT enumerates all frequent ordered trees without duplicates in $O(kbm)$ time per tree, and UNOT enumerates all frequent unordered tree in canonical form without duplicates in $O(kb^2m)$ time per tree, where k is the size of the current parent tree S, b is the maximum branching of $\mathcal{D}$, m is the number of occurrences of S. In addition to these algorithms, we also discuss application of reverse search to mining several tree and graph classes such as itemsets [2, 9], sequential episodes [13], path trees [21], and general graphs [11, 12].

1.1 Related Works Zaki [23], Nakano [15] and Asai *et al.* [5] independently developed efficient enumeration technique for ordered trees, called *rightmost expansion*. Recently, Asai *et al.* [7] devised a mining algorithm for unordered trees using an efficient enumeration of unordered trees by Nakano and Uno [16]. In the studies above, reverse search is effectively used for enumeration of trees and their occurrences.

There are a lot of other researches on tree mining and graph mining. Termier *et al.* [18] developed an unordered tree mining system. In their definition, a matching preserves the ancestor relation but the parent relation, as well as Zaki [23]. Wang and Liu [21] considered the frequent k-path tree discovery problem from semi-structured data. Miyahara *et al.* [14] studies mining of tag tree patterns. Inokuchi *et al.* [11], Kuramochi *et al.* [12], Vanetik *et al.* [20], and Yan *et al.* [22] study frequent subgraph discovery from a collection of directed/undirected graphs.

2 Preliminaries

In this section, we give basic definitions on ordered trees and unordered trees according to [3] and then 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 ordered trees and labeled unordered trees as a model of semi-structured data and patterns according to [3–5, 16]. 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 set of *nodes* V, a set of *edges* $E \subseteq V \times V$, the *root* $r \in V$ such that all but the root has exactly one parent. The mapping $label : V \to \mathcal{L}$ is called a *labeling*

function. If $(u, v) \in E$ then we say that u is a *parent* of v, or v is a *child* of u. A *leaf* is a node having no child. A *labeled ordered tree* (an *ordered tree*, for short) is an unordered tree $T = (V, E, B, r, label)$ augmented with the left-to-right ordering $B \subseteq V \times V$ among the children of each nodes. We denote by $\mathcal{U}$ and $\mathcal{T}$ the classes of unordered and ordered trees.

For a node $v \in V$, the *depth* of v is the length $dep(v) = d$ of the unique path $UP(v) = (v_0 = r, v_1, \ldots, v_d)$ $(d \geq 0)$ from the root r to v. If there is a path from node u to node 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 is defined as the number of its nodes $|T| = |V|$. We define the *empty tree* as a special tree $\perp$ of size 0.

For an ordered tree T, let $RMB(T) = (r_0, \ldots, r_c)$ $(c \geq 0)$ be the path from the root r to the rightmost leaf of T. The path $RMB(T)$ is called the *rightmost branch* of T. We denote by $rml(T)$ the rightmost leaf of T.

2.2 Tree Matching and the Occurrences of a Pattern In this paper, we give the semantics of labeled unordered trees by the tree matching, which is popular in computational learning theory, semi-structured data mining, and pattern matching [4, 5].

Let T and D be trees with node sets V_T and T_D and edge sets E_T and E_D, respectively. We call T the *pattern tree* (or *pattern* for short) and call D the *data tree*. Then, we say that the pattern tree T *occurs in* the data tree D if there is a mapping $\varphi : V_T \to V_D$ satisfying the following (1)–(4) 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., $label_T(x) = label_D(\varphi(x))$.
(4) If T and D are ordered trees, φ preserves the sibling relation, i.e., $(x, y) \in B_T$ iff $(\varphi(x), \varphi(y)) \in (B_D)^+$, where B_T and B_D are left-to-right orderings of T and D.

Then, the mapping φ is called a *matching from T into D*.

For a nonnegative integer k, a pattern T of size k is called *k-pattern*. We always assume that the node set of k-pattern T is $V = \{1, \ldots, k\}$ and that the elements in V are numbered consecutively in preorder if T is ordered. Obviously, the root of an ordered tree T is $r = 1$ and the rightmost leaf is $rml(T) = k$.

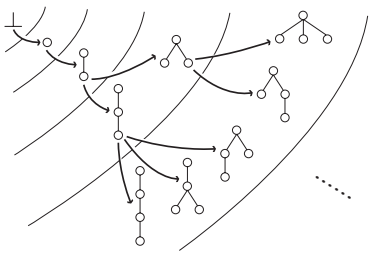
A *database* on $\mathcal{L}$ is a finite collection $\mathcal{D} = \{D_1, \ldots, D_n\}$ of data trees. We assume $V_{D_i} \cap V_{D_j} = \emptyset$ for every $1 \leq i, j \leq n$ $(i \neq j)$. Then, we denote by $V_{\mathcal{D}}$ the set of all nodes in the data trees in $\mathcal{D}$, and then we extend the notion of the matching from $\varphi : V_T \to V_D$ to $\varphi : V_T \to V_{\mathcal{D}}$, where $D \in \mathcal{D}$. We denote by $\mathcal{M}^{\mathcal{D}}(T)$ the set of all matchings from T into $\mathcal{D}$, and by $||\mathcal{D}|| = |V_{\mathcal{D}}|$ the number of the nodes in $\mathcal{D}$

Definition 1. Let $k \geq 1$ be any positive integer and T be a k-pattern. Assume that we have a matching $\varphi : V_T \to V_D \in \mathcal{M}^{\mathcal{D}}(T)$ from T into $\mathcal{D} = \{D_i\}_i$. Then, we define four types of the occurrences of T as follows:

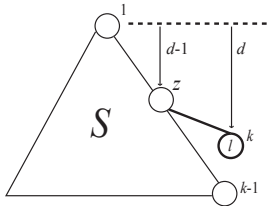
1. The *total occurrence (TO)* of T is the k-tuple $TO(\varphi) = \langle \varphi(1), \ldots, \varphi(k) \rangle \in (V_D)^k$.
2. The *embedding occurrence (EO)* of T is the set $EO(\varphi) = \{\varphi(1), \ldots, \varphi(k)\} \subseteq V_D$.
3. The *root occurrence (RO)* of T: $RO(\varphi) = \varphi(1)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

There is a trade off between the amount of information and the cost of computation for classes of the occurrences. The total occurrences are most specific, while the document occurrences are most abstract and most popular [11, 12, 14, 20–23]. The embedding and the root occurrences are used as the semantics of unordered trees and ordered trees, respectively, in [5] and [7].

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



(a) A search graph for ordered trees



(b) A rightmost-expansion of tree S

Fig. 2. The rightmost expansion for ordered trees

2.3 Frequent Pattern Discovery Problems Let $\tau \in \{TO, EO, RO, DO\}$ be types of the occurrences, U be a pattern and $\mathcal{D}$ be a database. Then, the τ-*count* of U in $\mathcal{D}$ is $|\tau^{\mathcal{D}}(U)|$. The τ-*frequency* of T in $\mathcal{D}$ is the ratio $freq^{\tau}(\mathcal{D}, T) = |\tau^{\mathcal{D}}(T)|/||\mathcal{D}||$ for $\tau \in \{Toc, Eoc, Roc\}$ and $freq^{\tau}(\mathcal{D}, T) = |\tau^{\mathcal{D}}(T)|/|\mathcal{D}|$ for $\tau = Doc$.

Let $0 < \sigma < 1$ be any positive number, called the *minimum frequency threshold*. A pattern T is τ-*frequent* with σ (or *frequent*) if T has the relative τ-frequency no smaller than σ. Let $\mathcal{C}$ be a class of patterns. Then, we state our data mining problems as follows.

Frequent Pattern Discovery in $\mathcal{C}$ with Occurrence Type τ
Given a database $\mathcal{D} \subseteq \mathcal{C}$ and a minimum frequency threshold $0 \leq \sigma \leq 1$,
find all the τ-*frequent patterns* $T \in \mathcal{C}$ satisfying $freq^{\tau}(\mathcal{D}, T) \geq ||\mathcal{D}||\sigma$.

We concentrate on the frequent pattern discovery problem throughout this paper since it is a prototypical data mining problem and well captures the characteristics in realworld applications.

3 Reverse Search

3.1 Basic Idea *Reverse search* is a general scheme for designing efficient algorithm for hard enumeration problems [8, 19]. Let $\mathcal{S}$ be the set of solusions. In reverse search, we define the parent-child relation $P \subseteq \mathcal{S} \times \mathcal{S}$ such that each $X \in \mathcal{S}$ has the unique parent $P(X) \in \mathcal{S}$. Using the search tree G over $\mathcal{S}$ defined by P, we can enumerate all solutions of $\mathcal{S}$ without duplicates by traversing the search tree G from the root to leaves (Fig. 2 (a)).

3.2 Reverse Search of Patterns We formalize the frequent pattern discovery problems by reverse search as follows. Let $\mathit{\Pi}$ denote a frequent pattern minining problem and $(\mathcal{P}, \mathcal{D}B, \mathcal{O}, O, |\cdot|)$ be the tuple consisting of the undelying classes of *patterns*, *occurrences*, and *databases*, the occurrence function $O : \mathcal{D}B \times \mathcal{P} \to 2^{\mathcal{O}}$ and the size function $|\cdot| : \mathcal{P} \to \mathbf{N}$ as defined in Section 2.

Definition 2. A *parent function* for $\mathcal{P}$ w.r.t. $\mathit{\Pi}$ is a pair $(P, \perp_{\mathcal{P}})$ of a partial function $P : \mathcal{P} \to \mathcal{P}$ and a pattern $\perp_{\mathcal{P}} \in \mathcal{P}$, called the *empty pattern*, satisfying the followings:

1. $P(T)$ is defined for every $T \in \mathcal{P} \backslash \{\perp_{\mathcal{P}}\}$. $P(\perp_{\mathcal{P}})$ is undefined.
2. $|P(T)| < |T|$ for every pattern $T \in \mathcal{P}$.
3. $freq^{\mathcal{O}}(P(T), D) \geq freq^{\mathcal{O}}(T, D)$ for every $D \in \mathcal{D}$ and $T \in \mathcal{P} \backslash \{\perp_{\mathcal{P}}\}$ (Anti-monotone property).

For $S, T \in \mathcal{P}$, if $S = P(T)$ then we say that S is a *parent pattern* of T or T is a *child pattern* of S. Then, we define the set of children of S, or the *children set*, by $CH^P(S) = \{T \in \mathcal{P} \mid P(T) = S\}$. Note that $CH^P = P^{-1}$ is the inverse relation of P. For parent function P, the *search graph* for P is the graph $G = (V, E)$ with node set $V = \mathcal{P}$ and edge set E such that $(x, y) \in E$ iff $x = P(y)$.

Lemma 1. *If $(P, \perp_{\mathcal{P}})$ is a parent function for $\mathcal{P}$ w.r.t. $\mathit{\Pi}$ then the following properties (i)–(iv) hold:*

(i) The pattern $\perp_{\mathcal{P}}$ is the pattern of the smallest size.

Algorithm FindFreqBasic(database $\mathcal{D}$, min-frequency σ)
1. $\mathcal{F} := \emptyset$; $\alpha := \lceil |\mathcal{D}|\sigma \rceil$; $\mathcal{O} := O^{\mathcal{D}}(\perp_{\mathcal{P}})$; //Initialization
2. ExpandBasic($\perp_{\mathcal{P}}, \mathcal{O}, \alpha, \mathcal{F}$);
3. Return $\mathcal{F}$;

Procedure ExpandBasic(pattern S, occurrence set $\mathcal{O}$, min-count α, $\mathcal{F} \subseteq \mathcal{P}$)
1. If ($|\mathcal{O}| < \alpha$) then return;
   Else $\mathcal{F} := \mathcal{F} \cup \{S\}$;
2. For each child $T \in CH^{\mathcal{P}}(S)$, do:
   - $\mathcal{Q} :=$ UpdateOcc($T, \mathcal{O}$);
   - ExpandBasic($T, \mathcal{Q}, \alpha, \mathcal{F}$);

Fig. 3. An abstract algorithm FindFreqBasic for discovering all frequent patterns and its subprocedure ExpandBasic based on reverse search

(ii) For every pattern $T \in \mathcal{P}$, there exists some $k \geq 0$, $P^k(T) = \perp_{\mathcal{P}}$.
(iii) For every pattern $T \in \mathcal{P}$, every $k \geq 0$ and $0 \leq \sigma \leq 1$, if $S = P^k(T)$ is not σ-frequent in $\mathcal{D}$ then neither is T.
(iv) The search graph G for P is a rooted tree with the root $\perp_{\mathcal{P}}$.

Proof. (i) Assume that there is a pattern $T \in \mathcal{P}\backslash\{\perp_{\mathcal{P}}\}$ of the smallest size $|T| = \min|\mathcal{P}| < |\perp_{\mathcal{P}}|$. However, $|P(T)| < |T|$ holds by the second condition of Def. 2 and thus the contradicton occurs. (ii) For a pattern $T \in \mathcal{P}$, if there does not exist any integer $k \geq 0$ that satisfies $P^k(T) = \perp_{\mathcal{P}}$, T has an infinite size by the second condition of Def. 2. This contradicts the definition of the size function $|\cdot| : \mathcal{P} \to \mathbf{N}$. (iii) It is directly derived from the anti-monotone property of $freq^{\mathcal{O}}$ (3. of Def. 2). (iv) Any pattern T except $\perp_{\mathcal{P}}$ has the unique parent $P(T)$. Furthermore, G is a connected graph since (ii) holds. Thus G forms a tree with the root $\perp_{\mathcal{P}}$. □

In Fig. 3, we give an algorithm scheme FindFreqBasic for solving the frequent pattern problem for $\mathcal{P}$ and the class of occurrences $\mathcal{O}$ based on reverse search with a given parent function P, where UpdateOccBasic(T, O) is a procedure that computes the occurrence set $Q = O^{\mathcal{D}}(T)$ of the child tree T from that of the parent tree $P(T)$.

A parent function for $\mathcal{P}$ under Π is said to *have $O(f(n))$-time children function* if there is an algorithm that, given pattern $S \in \mathcal{P}$, enumerates the members of the children set $CH^P(S)$ in $O(f(n))$ time per pattern in terms of $n = |S|$. We have the following theorem.

Theorem 1. *Suppose that the subprocedure* UpdateOccBasic *that correctly computes $O^{\mathcal{D}}(T)$ is given. If there exists $O(f(n))$-time parent function P for Π under Π, then the algorithm* FindFreqBasic *with subprocedures* ExpandBasic *in Fig. 3 computes the set $\mathcal{F}$ of all σ-frequent patterns within $\mathcal{P}$ in $\mathcal{D}$ in $O(f(n) + G(n))$ per pattern without duplicates, where S is the parent of the pattern T being enumerated, n is the size of S, and $G(n)$ is the time to compute the whole set $O(T)$ from the whole $O(S)$.*

Proof. By (iii) and (iv) of Lemma 1, the algorithm correctly enumerates all the frequent patterns without dupulicates. The analysis of the time complexity of the algorithm is straightforward. □

3.3 Reverse Search of Occurrences The idea of reverse search can be also applied to the computation of the occurrences of patterns. Let $(P, \perp_{\mathcal{P}})$ is a parent function for $\mathcal{P}$. A pair $(T, O) \in \mathcal{P} \times 2^{\mathcal{O}}$ of pattern and set occurrences is *admissible* if $O = O^{\mathcal{D}}(T)$. We denote the set of admissible pairs by $\mathcal{AP}$. Then, we extend the parent functions P from $P : \mathcal{P} \to \mathcal{P}$ to $P : \mathcal{AP} \to \mathcal{AP}$.

A *parent function* for $\mathcal{O}$ w.r.t. Π is a pair $(Q, \perp_{\mathcal{AP}})$ of a partial function $P : \mathcal{AP} \to \mathcal{AP}$ and a pair $\perp_{\mathcal{AP}} = (\perp_{\mathcal{P}}, \perp_{\mathcal{O}}) \in \mathcal{AP}$ satisfying the followings:

1. $\perp_{\mathcal{P}}$ is the empty pattern of $\mathcal{P}$ and $O(\perp_{\mathcal{P}}) = \{\perp_{\mathcal{O}}\}$.
2. For every $(T, o) \in \mathcal{AP}\backslash\{\perp_{\mathcal{AP}}\}$, its *parent* $Q(T, o) \in \mathcal{AP}$ is defined, and $Q(\perp_{\mathcal{P}}, \perp_{\mathcal{O}})$ is not defined.

If $\pi = (T, o)$ then we write $Q(\pi)$ for $Q(T, o)$. Similary to the parent function for $\mathcal{P}$, we define the *children set* $CH^Q(S, q) = \{ (T, o) \in \mathcal{AP} \mid Q(T, o) = (S, q) \}$ and the search graph for $\mathcal{AP}$.

Lemma 2. *If $(Q, \bot_{\mathcal{AP}})$ is a parent function for $\mathcal{O}$ w.r.t. Π then the following properties (i) and (ii) hold:*

(i) For every $(T, o) \in \mathcal{AP}$ $Q^k(T, o) = (\bot_{\mathcal{P}}, \bot_{\mathcal{O}})$ holds for some $k \geq 0$.
(ii) The search graph G for $\mathcal{AP}$ is a rooted tree with the root $(\bot_{\mathcal{P}}, \bot_{\mathcal{O}})$.

Proof. (i) By Lemma 2 (i), a sequence $T = T_k, \ldots, T_0 = \bot_{\mathcal{P}}$ exists for some $k \geq 0$, where $T_{i-1} = P(T_i)$ for every $i = k, \ldots, 1$. Since $Q(T_i, o_i)$ is defined for a pattern T_i and any occurrence $o_i \in O^{\mathcal{D}}(T_i)$ of T_i in $\mathcal{D}$ if $i \geq 1$, the claim holds. (ii) By the claim (i) abobe, the graph G is connected. Furthermore, any $(T, o) \in \mathcal{AP}$ except $(\bot_{\mathcal{P}}, \bot_{\mathcal{O}})$ has the unique parent $Q(T, o)$. These arguments imply that G is a tree with the root $(\bot_{\mathcal{P}}, \bot_{\mathcal{O}})$. □

In most cases, it is often easy to define the parent $P(T)$ of a given pattern $T \in \mathcal{P}$, while it is difficult to build an efficient enumeration procedure for the elements of the children set $CH^P(S)$ for a particular pattern $S \in \mathcal{P}$. In the following sections, we will see how to construct the children set for various pattern classes [15, 16, 5, 7]

4 Frequent Substructure Discovery from Ordered Trees

In this section, we present our mining algorithm FREQT [5], as well as [15, 23], for efficiently finding all frequent ordered trees by reverse search from a given database. Throughout this section, all pattern and data trees are ordered.

Let $T = (V, E, B, r, label)$ be a labeled ordered tree of size k, where $V = \{1, \ldots, k\}$ and the nodes are numbered consecutively in preorder. As a representation of T, we define the *depth-label sequence* of T as the sequence

$$code(T) = ((dep(1), label(1)), \ldots, (dep(k), label(k)) \in (\mathbf{N} \times \mathcal{L})^*$$

where $dep(v_i)$ and $label(v_i)$ are the depth and the label of the i-th node of T in pre-order [6, 15, 23]. In what follows, we often identify T and $code(T)$.

Then, we define a parent function $P : \mathcal{T} \to \mathcal{T}$ on $\mathcal{T}$ for patterns as follows.

Definition 3 (Parent function for ordered tree patterns). Let $T \in \mathcal{T} \backslash \{\bot\}$ be a labeled ordered tree. 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 children set $CH^P(S)$ of a pattern S is easily computed as follows. Let S be a pattern with depth-label sequence $code(S)$. We define the range of $RMB(S)$ by $range(S) = \{1, \ldots, |RMB(S)| + 1\}$ if $|S| > 0$, and $range(\bot_{\mathcal{P}}) = \{0\}$. For any $\boldsymbol{v} = (d, \ell) \in range(S)$, we denote by $S \cdot \boldsymbol{v}$ the tree $T \in \mathcal{T}$ such that $code(S) \cdot \boldsymbol{v} = code(T)$.

Lemma 3 (Rightmost expansion [5, 15, 23]). *Let $P : \mathcal{T} \to \mathcal{T}$ be the parent function and $S \in \mathcal{T}$ be a pattern on $\mathcal{L}$. Then, the children set $CH^P(S) = P^{-1}(S)$ of S is computed as*

$$CH^P(S) = \{ T \in \mathcal{T} \mid T = S \cdot \boldsymbol{v},\ \forall \boldsymbol{v} \in range(S) \times \mathcal{L} \}.$$

Each element T in $CH^P(S)$ is called a *rightmost expansion* of S (Fig. 2 (b)). Then, we have the following lemma.

Lemma 4 ([15]). *The parent function $P : \mathcal{T} \to \mathcal{T}$ is a $O(1)$-time children function if we assume that the algorithm does not return each of entire tree but the difference from the parent tree.*

Next, we consider enumeration of the occurrences of a given pattern in a database $\mathcal{D}$. Note that for ordered trees, the total occurrences coincide the embedding occurrences. First, we consider the case for the total occurrences.

Algorithm Update-RMO(RMO, p, ℓ)

1. Set RMO_{new} to be the empty list ε and $check := null$.
2. For each element $x \in RMO$, do:
   (a) If $p = 0$, let y be the leftmost child of x.
   (b) Otherwise, $p \geq 1$. Then, do:
      – If $check = {\pi_D}^p(x)$ then skip x and go to the beginning of Step 2 (Duplicate-Detection).
      – Else, let y be the next sibling of ${\pi_D}^{p-1}(x)$ (the $(p-1)$st parent of x in D) and set $check := {\pi_D}^p(x)$.
   (c) While $y \neq null$, do the following:
      – If $L_D(y) = \ell$, then $RMO_{\text{new}} := RMO_{\text{new}} \cdot (y)$; /* Append */
      – $y := next(y)$; /* the next sibling */
3. Return RMO_{new}.

Fig. 4. The procedure UpdateRMO

Lemma 5. The following function Q is a parent function for total occurrences.

$$Q(T, \langle v_1, \ldots, v_{k-1}, v_k \rangle) = (P(T), \langle v_1, \ldots, v_{k-1} \rangle)$$

Reverse search is not directly applicable to the computation of the root and the document occurrences. Instead, we give the parent function for the rightmost occurrences. Let T be a k-pattern and $\varphi \in \mathcal{M}^{\mathcal{D}}(T)$ be a matching. Then, the rightmost leaf occurrence of T relative to φ is the node $RMO(\varphi) = \varphi(k)$. Let $RMO(T) = \{\, \varphi(k) \,|\, \varphi \in \mathcal{M}^{\mathcal{D}}(T) \,\}$ be the list of rightmost leaf occurrences of T.

Lemma 6 (Parent function for rightmost leaf occurrences). Let $T \in \mathcal{T}$ be any pattern and $o \in RMO(T)$ be any rightmost leaf occurrence of T. Then, the parent is $Q(T, o) = (P(T), q)$ where $q = \min\{\, \varphi(k-1) \in V_D \,|\, \varphi \in \mathcal{M}^{\mathcal{D}}(T)$ such that $\varphi(k) = o \,\}$.

It is difficult to efficiently compute the children set $CH^Q(S, q)$ by reverse search. Fortunately, we can directly give the procedure UpdateOcc in Fig. 4 that computes $RMO(T)$ from $RMO(P(T))$ using reverse search and the duplication check with previously generated occurrences.

Lemma 7. *Let $S \in \mathcal{T}$ be a pattern and $RMO = RMO^{\mathcal{D}}(S)$ be its rightmost occurrences. Let $T = S \cdot \boldsymbol{v}$ be any rightmost expansion of S. Then, the procedure* UpdateRMO *computes the set $RMO^{\mathcal{D}}(T)$ in $O(kbm)$ time per pattern, where $k = |S|$, b is the maximum branching of $\mathcal{D}$ and $m = |RMO^{\mathcal{D}}(S)|$.*

Theorem 2 ([5, 15, 23]). *All frequent patterns for the class of labeled ordered trees w.r.t. rightmost occurrences can be enumerated in $O(kbm)$ time per pattern, where k is the size of the current parent tree, b is the maximum branching of $\mathcal{D}$ and $m = |RMO^{\mathcal{D}}(S)|$.*

5 Frequent Substructure Discovery from Unordered Trees

Nakano and Uno [16] developed an efficient algorithm that enumerates all unlabeled unordered trees in $O(1)$ time per tree by extending the enumeration technique seen in Section 4 to that for unordered trees. Using this technique, we developed an efficient frequent tree miner UNOT for unordered trees from a given database [7].

5.1 Enumeration of Unordered Trees in Canonical Form In UNOT, we use as a representation of a labeled unordered tree $U \in \mathcal{U}$ a labeled ordered tree T which is isomorphic to U ignoring the left-to-right ordering B on siblings. However, there may be $O(k!)$ isomorphic ordered trees representing a given unordered tree of size k. In Fig. 5, for instance, three ordered trees T_1, T_2, and T_3 are equivalent representations of an unordered tree.

To solve the problem, we define the canonical form representation for unordered tree as follows. For $(d_i, \ell_i) \in \mathbf{N} \times \mathcal{L}$ $(i = 1, 2)$, we define $(d_1, \ell_1) > (d_2, \ell_2)$ iff either $d_1 > d_2$ or $d_1 = d_2$

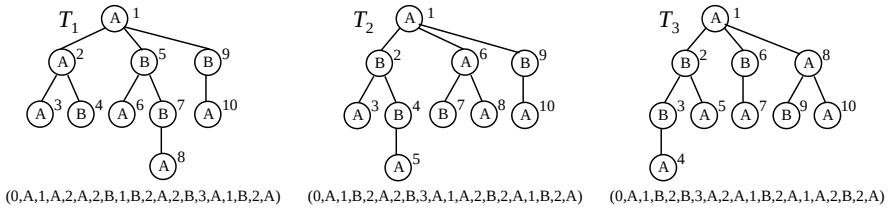


Fig. 5. The depth-label sequences of labeled ordered trees

and $\ell_1 > \ell_2$. Then for two labeled ordered trees T_i $(i = 1, 2)$, we define the ordering $>_{\text{lex}}$ by $code(T_1)$ is *heavier than* $code(T_2)$, denotes $T_1 >_{\text{lex}} T_2$, if $code(T_1) >_{\text{lex}} code(T_2)$ holds. The *canonical representation* of unordered tree T is the labeled ordered tree $CR(T) = S \in \mathcal{T}$ having the heaviest sequence among the isomorphic representations of T within $\mathcal{T}$. In the example of Fig. 5, only T_1 is a canonical form representation and T_2, and T_3 are not.

Nakano and Uno [16] gave the following characterization of the canonical representations.

Lemma 8 (Left-heavy condition [16]). *An 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 $code(T(v_1)) \geq_{\text{lex}} code(T(v_2))$.*

We denote by $\mathcal{C}$ the class of canonical representations for unordered trees. Now, we define the parent $P(T)$ of a canonical representation $T \in \mathcal{C}$ for unordered trees as in Section 4: $P(T)$ is the ordered tree obtained from T by removing the rightmost leaf. By Lemma 8, we can show the following lemma.

Lemma 9 (Parent function for canonical representations). *If ordered tree T is a canonical representation, then its parent $P(T)$ is a canonical representation.*

However, it is not easy to compute the children set $CH^P(S)$. A straightforward algorithm derived from Lemma 8 takes $O(n^2)$ time to test if a given pattern T is in canonical form. Nakano and Uno [16] gives efficient algorithm to enumerate all the unordered trees in canonical representations without duplicates in $O(1)$ time per pattern.

Combining the enumeration technique for unordered trees by [16] and the incremental computation technique for the embedding occurrences, we developed an efficient unordered tree miner UNOT [7].

The basic idea of the algorithm is to test the lexicographic order between the left and the right trees L_i and R_i on the rightmost branch $RMB(T) = (r_0, r_1, \ldots, r_g)$. For every $i = 0, 1, \ldots, g$, if the i-th node r_i on $RMB(T)$ has two or more children then we denote by R_i the rightmost subtree of r_i and L_i the subtree of r_i preceding R_i. Then, we can prove that $T \in \mathcal{T}$ is in canonical form iff $L_i \geq_{\text{lex}} R_i$ holds in T for every $i = 0, \ldots, g-1$. Furthermore, it is also sufficient to test if $code(L_c) >_{\text{lex}} code(R_c)$ for the left and the right trees L_c and R_c of the highest *copy node* r_c $(c \geq 0)$ that is highest node for which the code $code(L_c)$ is being copied into $code(R_c)$, that is, $code(L_c)$ a prefix of $code(R_c)$.

Combining the above idea together with the maintenance of the copy node and efficient enumeration technique for $\mathcal{T}$ in Section 4, we can compute the children set $CH^P(S)$ for a given canonical representation $S \in \mathcal{C}$. In Fig. 6, we show the procedure FindAllChildren that, given a canonical representation S, enumerates the elements the children set $CH^P(S)$ in canonical form.

Lemma 10 ([16]). *The parent function $P_{\mathcal{C}} : \mathcal{C} \to \mathcal{C}$ has a $O(1)$-time children function by using the procedure* FindAllChildren *of Fig. 6, when only the differences to T are output.*

5.2 Updating the Occurrence List Next, we describe the incremental computation of the embedding occurrences $EO^{\mathcal{D}}(T)$.

Let T be a canonical representation of an unordered tree. Recall that an embedding occurrence EO of T is given by $EO(\varphi)$ for some matching function, or equivalently a total occurrence, φ of T in D. Then, we say φ represents EO. If $EO(\varphi_1) = EO(\varphi_2)$ then we say

Procedure FindAllChildren(**pattern** S**, copy depth** c)
Case I : If $code(L_k) = code(R_k)$ for the copy depth k:
- The children set is $CH := \{S \cdot (1, \ell_1), \ldots, S \cdot (k+1, \ell_{k+1})\}$, where $label(r_i) \geq \ell_i$ for every $i = 1, \ldots, k+1$.
- The copy depth of $S \cdot (i, \ell_i)$ is $i-1$ if $label(r_i) = \ell_i$ and i otherwise for every $i = 1, \ldots, k+1$.

Case II : If $code(L_k) \neq code(R_k)$ for the copy depth k:
- Let $m = |code(R_k)|+1$ and $\boldsymbol{w} = (d, \ell)$ be the m-th component of $code(L_k)$ (the next position to be copied). The children set is $CH := \{T \cdot (1, \ell_1), \ldots, T \cdot (d, \ell_d)\}$, 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. 6. The procedure FindAllChildren

Algorithm UpdateOcc(canonical pattern $T \in \mathcal{C}$, list $\mathcal{O}$ of embeddings, depth d, label ℓ)
- $\mathcal{P} := \emptyset$;
- 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. 7. The procedure UpdateOcc for computing canonical total occurrences

φ_1 and φ_2 are isomorphic. The trouble is that there are $O(k!)$ total occurrences for a fixed embedding occurrence.

To avoid this difficulty, we define the canonical representation $C(EO)$ for the embedding occurrences EO: EO is the lexicographically-first total occurrence φ among isomorphic representation of EO such that $EO = EO(\varphi)$, where total occurrences are represented by a vector of nodes $\langle \varphi(1), \ldots, \varphi(k) \rangle$ that maps to.

Then, a parent function Q for canonical representation for embedding is defined by $Q(T, \langle \varphi(1), \ldots, \varphi(k-1), \varphi(k) \rangle) = (P_{\mathcal{C}}(T), \langle \varphi(1), \ldots, \varphi(k-1) \rangle)$, where T is a pattern and $\varphi = \langle \varphi(1), \ldots, \varphi(k-1), \varphi(k) \rangle$ is a canonical representation for an embedding of T. In Fig. 7, we show the procedure UpdateOcc that computes the set of embedding occurrences $EO(T)$ of the child tree T from that of the parent tree S. The algorithm uses the computation of the children set $CH^Q(S, \varphi)$. The algorithm runs in $O(kbm)$ time per pattern to compute $EO(T)$, where $k = |S|$, b is the maximum branching factor of $\mathcal{D}$, and $m = |EO(S)|$.

Now, we have the following theorem.

Theorem 3 ([7]). *Let $\mathcal{D}$ be a database and $\alpha \geq 0$ be any nonnegative integer. Then, the algorithm* UNOT *enumerates all frequent unordered trees w.r.t. embedding occurrences without duplicates in $O(kb^2m)$ time per pattern T, where b is the maximum branching in $\mathcal{D}$, k is the size of the parent pattern S of T, and m is the number of embeddings of S.*

6 Application to Other Classes of Tree Patterns

In this section, we review previous results in tree and graph mining and reformulate them as instances of reverse search. In what follows, $\mathcal{C}$, C and P denote the class of canonical representations, the canonical form function, and the parent function for the class. Let $\mathcal{L}$ be an alphabet of labels with total order $\leq_{\mathcal{L}}$.

6.1 Item sets An *itemset* is a subset $X \subseteq \mathcal{L}$ and a *transaction database* is a collection $\mathcal{D} = \{t_1, \ldots, t_m\} \subseteq 2^{\mathcal{L}}$ $(m \geq 0)$ of tuples [2]. $X \subseteq \mathcal{L}$ appears in $t \in \mathcal{D}$ if $X \subseteq t$ and the frequency is measured by the document count. We define $\mathcal{C}$ and P as follows.

For itemset $X \subseteq \mathcal{L}$, $C(X)$ is the lexicographically-first string $x_1 \cdots x_k \in \mathcal{L}^*$ of items such that $X = \{x_1, \ldots, x_k\}$. This is equivalent to that $x_1 < \cdots < x_k$ holds. For every $X = x_1 \cdots x_{k-1} x_k \in \mathcal{C}$, its parent is the string $P(X) = x_1 \cdots x_{k-1}$ obtained from X by removing the last element. A child $T \in CH^P(S)$ of a pattern $S = x_1 \cdots x_{k-1}$ is given by $T = x_1 \cdots x_{k-1} x$ for any $x \in \mathcal{L}$ such that $x_{k-1} < x$

Then, Bayardo [9] developed an efficient depth-first mining algorithm MaxMiner for frequent itemsets that traverses the search graph G for $\mathcal{P}_{it}$, where G is called the *set-enumeration tree*.

6.2 Sequential Episodes A *sequential pattern without window size* (sequential pattern, for short) is a string $x_1 \cdots x_k \in \mathcal{L}^*$ of items (Mannila et al. [13]). A database is a collection $\mathcal{D} = \{s_1, \ldots, s_m\}$, where each $s = y_1 \cdots y_n \in \mathcal{L}^*$ is a sequence of items. X appears in $s \in \mathcal{D}$ if X is a non-contiguous subsequence of s. We define the canonical form of X to be the $C(X) = X$ itself, and then we can use the same construction for P and $CH^P(S)$ as the case of itemsets above.

6.3 Wang and Liu's k-Path Trees Wang and Liu [21] developed an Apriori-based miner for the class of path trees. A database is a collection of labeled unordered trees $\mathcal{D} \subseteq \mathcal{U}$ and let Π be the set of all (partial) path in $\mathcal{D}$. A *path tree* is a labeled unordered trees T consisting of a set $path(T) = \{\pi_1, \ldots, \pi_k\}$ of paths appearing in $\mathcal{D}$ branching only at the root. We identify T and $path(T)$ in the followings. The occurrences of a path tree is given by the document occurrences $DO(\varphi)$ with matching functions φ as in Section 2.

Now, we will see that we can build a reverse search algorithm for enumerating frequent path trees as follows.

Representation $C(T) \in \mathcal{C}$ of a path tree $T \subseteq \Pi$ is the lexicographically-first vector $(\pi_1, \ldots, \pi_k) \in \Pi^*$ of paths such that $T = \{\pi_1, \ldots, \pi_k\}$, where $\pi_i <_{\mathcal{L}} \pi_j$ iff π_i is lexicographically smaller than π_j in terms of $\leq_{\mathcal{L}}$). This is equivalent to that $\pi_1 <_{\mathcal{L}} \cdots <_{\mathcal{L}} x_k$ holds. For every $T = (\pi_1, \ldots, \pi_{k-1}, \pi_k) \in \mathcal{C}$, its parent is the string $P(T) = (\pi_1, \ldots, \pi_{k-1})$ obtained from X by removing the last path π_k. A child $T \in CH^P(S)$ of a pattern $S = (\pi_1, \ldots, \pi_{k-1})$ is given by $T = (\pi_1, \ldots, \pi_{k-1}, \pi)$ for any $\pi \in \Pi$ such that $\pi_i < \pi$ for every $i = 1, \ldots, k-1$.

For every path tree T, let $sort(D)$ be the labeled ordered tree obtained by sorting at each internal node the paths in the lexicographic order over paths, and let $sort(\mathcal{D}) = \{\, sort(D) \,|\, D \in \mathcal{D} \,\}$. Then, we can see that a path tree T appears in an ordered tree D iff $sort(T)$ appears in $sort(D)$. By this property, we can build a MaxMiner-style miner for frequent path trees using the depth-first search of the search graph G and the rightmost occurrences used in our Freqt with the vertical layout. The details are omitted.

6.4 Graph Mining Finally, we will discuss application of reverse search to frequent pattern mining for general graphs and the difficulties in such an approach. Inokuchi, Washio and Motoda [11] and Kuramochi and Karypis [12] presented an efficient frequent pattern mining algorithms AGM and FSG, respectively, for the general class of graphs. A pattern G appears in a graph $D \in \mathcal{D}$ if G is embedded into D, and thus has a document occurrence. The class $\mathcal{C}$ of canonical forms and the parent function P implicitly used in their algorithms.

The canonical representation $C(G) \in \mathcal{C}$ of a graph pattern with adjacency matrix $X = (x_{ij}) \in \{0,1\}^{k \times k}$ is the lexicographically-first vector $C(X) = (y_{11}, \ldots, y_{1k}, \ldots, y_{k1}, \ldots, y_{kk}) \in \{0,1\}^{k^2}$ such that $Y = (y_{ij}) \in \{0,1\}^{k \times k}$ is some matrix obtained from $X = (x_{ij})$ by permutation of the columns and rows of X.

For every $k \times k$ matrix $X = (x_{ij}) \in \{0,1\}^{k \times k}$, its parent is the $(k-1) \times (k-1)$ matrix $P(X) \in \{0,1\}^{(k-1) \times (k-1)}$ obtained from X by removing the rightmost column and the topmost row.

Unfortunately, there seems no uniform way to compute a child $T \in CH^P(S)$ in canonical form for general graph patterns. AGM and FSG employ sophisticated methods that incrementally compute the children set $CH^P(S)$ and efficiently run on realworld applications [11, 12].

7 Conclusions

In this paper, we review the recent developments of tree and graph mining algorithms mainly from the view of *reverse search*. We reformulate our frequent tree miners FREQT and UNOT for labeled ordered trees and labeled unordered trees in terms of reverse search. Also, we considered search mechanisms of several known algorithms can be considered as instances of reverse search.

We have to note that formulation of a tree/graph mining problem in reverse search does not necessarily give an efficient solution in uniform way. Most difficult part in reverse search is the design of child patterns and it strongly depends on the domain of the problem. However, as seen in Section 5 for unordered tree miner UNOT, reverse search can be a powerful tool for difficult substructure mining problem of special types of graph structures.

Acknowledgement

The authors would like to thank Hideaki Takeda, Tsuyoshi Murata, and Ryutaro Ichise for the fruitful discussions on Semantic Web mining. Tatsuya Asai and Hiroki Arimura would like to thank Takashi Washio, Akihiro Inokuchi, Michihiro Kuramochi, Alexandre Termier, 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. R. Agrawal, R. Srikant, Fast Algorithms for Mining Assoiciation Rules, In *Proc. the 20th VLDB*, 487–499, 1994.
3. Aho, A. V., Hopcroft, J. E., Ullman, J. D., *Data Structures and Algorithms*, Addison-Wesley, 1983.
4. T. R. Amoth, P. Cull, and P. Tadepalli, Exact learning of unordered tree patterns from queries, In *Proc. COLT'99*, ACM Press, 323–332, 1999.
5. 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.
6. 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.
7. 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` (submitting)
8. D. Avis, K. Fukuda, Reverse Search for Enumeration, Discrete Applied Mathematics, 65(1–3), 21–46, 1996.
9. R. J. Bayardo Jr., Efficiently Mining Long Patterns from Databases, In *Proc. SIGMOD98*, 85–93, 1998.
10. L. B. Holder, D. J. Cook, S. Djoko, Substructure Discovery in the SUBDUE System, In *Proc. KDD'94*, 169–180, 1994.
11. A. Inokuchi, T. Washio, H. Motoda, An Apriori-Based Algorithm for Mining Frequent Substructures from Graph Data, In *Proc. PKDD 2000*, 13–23, LNAI, Springer, 2000.
12. M. Kuramochi, G. Karypis, Frequent Subgraph Discovery, In *Proc. IEEE ICDM'01*, 2001.
13. H. Mannila, H. Toivonen, A. I. Verkamo, Discovering Frequent Episodes in Sequences, In *Proc. KDD'95*, 210–215, 1995.
14. 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-2002*, 341–355, 2002.
15. S. Nakano, Efficient generation of plane trees, Information Processing Letters, 84, 167–172, 2002.
16. S. Nakano, T. Uno, Another Simple Algorithm to Generate All Rooted Trees, 2003. (Submitting)
17. S. Nestrov, S. Abiteboul, R. Motwani, Extracting Schema from Semistructured Data, In *Proc. SIGKDD'98*, 295–306, 1998.
18. A. Termier, M. Rousset, M. Sebug, TreeFinder: a First Step towards XML Data Mining, In *Proc. IEEE ICDM'02*, 450–457, 2002.
19. T. Uno, A Fast Algorithm for Enumerating Bipartite Perfect Matchings, In *Proc. ISAAC 2001*, LNCS, Springer-Verlag, 367–379, 2001.
20. N. Vanetik, E. Gudes, E. Shimony, Computing Frequent Graph Patterns from Semistructured Data, In *Proc. IEEE ICDM'02*, 458–465, 2002.
21. K. Wang, H. Liu, Schema Discovery from Semistructured Data, In *Proc. KDD'97*, 271–274, 1997.
22. X. Yan, J. Han, gSpan: Graph-Based Substructure Pattern Mining, In *Proc. IEEE ICDM'02*, 721–724, 2002.
23. M. J. Zaki. Efficiently Mining Frequent Trees in a Forest, In *Proc. SIGKDD 2002*, ACM, 2002.