社団法人　情報処理学会　研究報告
IPSJ SIG Technical Report

2004－AL－97　(10)
2004／10／15

節点包含制約を満たす頻出木マイニング

中村篤祥　　工藤峰一

† 北海道大学大学院情報科学研究科
〒 060-0814 札幌市北区北 14 条西 9 丁目
E-mail: †{atsu,mine}@main.ist.hokudai.ac.jp

あらまし　本論文では、与えられたラベル付き順序木の集合から、あらかじめ全ての木が含んでいることがわかっている特殊節点を全て含む部分木で、頻出するものを効率的に数えあげる方法を提案する。提案アルゴリズムは Zaki の TreeMiner アルゴリズム [9] を、制約を満たすものだけ候補として効率的に生成するように改造したものである。また、同じラベルが多く存在する場合に大量のメモリを使用するという問題に対処する方法についても提案する。検索エンジンで集められた Web ページに含まれる、レストランの名前と評判情報を含む頻出部分構造見つける問題に、提案アルゴリズムを適用することにより得られた結果につても報告する。

キーワード　頻出木, 最小サポート, 制約, Web マイニング

Mining Frequent Trees with Node-Inclusion Constraints

Atsuyoshi NAKAMURA and Mineichi KUDO

† Graduate School of Information Science and Technology, Hokkaido University
Kita 14, Nishi 9, Kita-ku, Sapporo, 060-0814
E-mail: †{atsu,mine}@main.ist.hokudai.ac.jp

Abstract　In this paper, we propose an efficient algorithm enumerating all frequent subtrees containing all special nodes that are guaranteed to be included in all trees belonging to a given data. Our algorithm is a modification of TreeMiner algorithm [9] so as to efficiently generate only candidate subtrees satisfying our constraints. We also propose a space saving method for a set of trees with a lot of nodes having the same label. We report mining results obtained by applying our algorithm to the problem of finding frequent structures containing the name and reputation of given restaurants in Web pages collected by a search engine.

Key words　freequent tree, minimum support, constraint, Web mining

1. Introduction

Frequent structure mining is one of the most popular way of data mining because of understandability of its analyzed results. The idea of finding frequent patterns is simple and easy, but for massive databases efficient algorithms to do this task are necessary, and it is not trivial to develop such algorithms. After Agrawal and Srikant developed efficient algorithm *Apriori*, various efficient algorithms have been developed for frequent itemsets [1], subsequences [2], subtrees [3], [9], subgraphs [6] and so on.

People sometimes have a certain point of view from which they want to analyze data. In such cases, they want to find frequent structures that satisfy certain constraints. This can be done by selecting structures satisfying the constraints after enumerating all frequent ones, but it is not efficient when a lot of more frequent other structures exist. In order to efficiently find the frequent structures satisfying constraints without enumerating unnecessary frequent ones, some algorithms have been also developed for itemsets [8] and subsequences [5].

In this paper, we consider a kind of constrained problem for frequent subtrees. Data we mine is a set of *labeled rooted ordered* trees, each of which contains just d special nodes labeled a different label belonging to a set of d special labels. Our mining problem is to find all frequent *embedded* subtrees [9] containing all d special nodes.

This research is motivated by *wrapper* induction [4], [7]. A *wrapper* is a program that extracts information necessary for some purpose from Web pages. Most wrappers extract necessary information by pattern matching around the information. Wrapper induction, automatic wrapper construction from training data, can be done by finding common patterns around the information in the data. Most Web pages are HTML documents of which tag structures can be represented by DOM-trees, so finding frequent subtrees of DOM-trees containing all nodes with necessary information can be used as a kind of wrapper induction methods, though some additional information like contents of the information and distance between nodes are necessary to construct high-precision wrappers.

In this paper, we propose an efficient algorithm enumerating all frequent subtrees containing all special nodes. Our algorithm is an extension of *TreeMiner* algorithm proposed by Zaki [9]. We modified TreeMiner algorithm so as to efficiently generate only candidate subtrees satisfying our constraints. We also propose a space saving method for a set of trees with a lot of nodes having the same label. We also report mining results obtained by using our algorithm for the problem of finding frequent structures containing the name and its reputation of given Ramen (lamian, Chinese noodles in Soup) shops in Web pages collected by a search engine.

2. Problem Statement

2.1 Notions and notations

In this paper, all trees we deal with are labeled ordered trees defined as follows. A *rooted* tree $T = (N, B)$ is a connected acyclic graph with a set N of vertices and a set B of directed edges $(u, v) \in N \times N$ that represent *parent-child* relation, which satisfies the condition that every vertex but just one vertex (*root*) has just one parent vertex. For a tree, a vertex and an edge are called a *node* and a *branch*, respectively. An *ordered* tree $T = (N, B, \preceq)$ is a rooted tree (N, B) with partial order $\preceq$ on N representing a sibling relation, where the order is defined just for all the pair of children having the same parent. Let L be the set of labels. A *labeled* ordered tree $T = (N, B, \preceq, l)$ is an ordered tree $(N, B, \preceq)$ of which nodes are labeled by the label function $l : N \rightarrow L$.

An *id* of a node of a tree $T = (N, B, \preceq, l)$ is its position in a depth-first traversal of the tree, where the passing order of children of the same parent follows order $\preceq$. We call this traversal *node-id traversal*. Note that, in our notation, any node with subscript i represents the node of id i.

The parent-child relation, which is defined by set B of branches, induces another partial order $\leq$, an *ancestor-descendant* relation, by extending the relation so as to satisfy reflexivity and transitivity. If t_i and t_j are not comparable in order $\leq$, t_j is said to be a *left-collateral* of t_i when $i > j$ and a *right-collateral* of t_i when $i < j$.

A *scope* of a node t is defined as $[l, r]$ using the minimum id l and the maximum id r among the ids of t's descendant nodes. Note that l is always t's id.

A *string encoding* of a tree T is a sequence of labels in $L \cup \{-1\}$ generated by starting with null string, appending the label of the node at its first visit, and appending -1 when we backtrack from a child in node-id traversal. For example, tree S_2 in Figure 1 is encoded as

a b 1 -1 -1 2 -1.

We abuse notation and use the same symbol to represent both a tree and its string encoding.

Size of a tree T is the number of nodes in T. A *size-k prefix* of a tree is the subtree composed of the nodes of which id is less than k.

An *embedded* subtree considered by Zaki [9] is defined as follows.

[Definition 1] Let $T = (N_T, B_T, \preceq_T, l_T)$ and $S = (N_S, B_S, \preceq_S, l_S)$ be labeled ordered trees. Assume that N_T and N_S are represented as $\{t_0, t_1, ..., t_{n-1}\}$ and $\{s_0, s_1, ..., s_{m-1}\}$, respectively. If there is an one-to-one mapping $i : j \mapsto i_j$ from $\{0, 1, ..., m-1\}$ to $\{0, 1, ..., n-1\}$ satisfying the following conditions, S is called an *embedded subtree* of T and i is called an *embedding mapping*.

(1) (Label preserving)

$$l_S(s_j) = l_T(t_{i_j}) \text{ for } j \in \{0, ..., m-1\}.$$

(2) (Ancestor-descendant relation preserving)

$$(s_j, s_k) \in B \Rightarrow t_{i_j} \leq t_{i_k} \text{ for } j, k \in \{0, 1, ..., m-1\}.$$

(3) (Sibling relation preserving)

$$j \leq k \Leftrightarrow i_j \leq i_k \text{ for } j, k \in \{0, 1, ..., m-1\}.$$

Let D denote a set of labeled ordered trees. For given D and a *minimum support* $0 < \sigma \leq 1$, a subtree S is *frequent* if the rate of trees in D that have S as an embedded subtree is at least σ.

Zaki [9] considered the problem of enumerating all frequent subtrees and developed an efficient algorithm for this problem.

2.2 Trees with special nodes

In this paper, we assume that there are d different special labels which are not members of L, and that every tree in D has just d special nodes labeled different special labels.

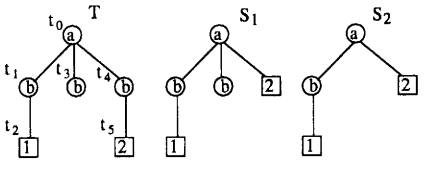

Figure 1 Example of trees with special nodes: squares denote special nodes. The size of tree T is 4, and trees S_1 and S_2 are its size-3 and size-2 prefixes, respectively.

The notions of size and size-k prefix defined above are extended as follows. *Size* of a tree is the number of *non*-special nodes. For example, the size of tree T in Figure 1 is 4. A *size-k prefix* of a tree is the subtree composed of k *non*-special nodes with k smallest ids and all special nodes. For example, trees S_1 and S_2 in Figure 1 are the size-3 and size-2 prefixes of tree T, respectively.

We consider the following problem.

[Problem 1] (Constrained tree minining problem) For given D and a minimum support σ, enumerate all frequent subtrees that have all special nodes.

3. Candidate subtree generation

Since all embedded subtrees of a frequent subtree are also frequent, enumerating frequent subtrees in size-increasing order reduces candidates, and as a result it is efficient. Zaki's efficient method [9] for enumerating frequent subtrees in size-increasing order is based on a notion of *prefix equivalence class*. We first describe his method in the next subsection, then propose modified version of the method for the constrained problem.

3.1 Method for the problem without constraints

A *prefix equivalence class* for a size-k tree P in a size-$(k+1)$ tree set G, that is denoted by $[P]_G$, is the set of size-$(k+1)$ trees in G of which size-k prefix is P. We let $[G]\text{pre}$ denote the set of all prefix equivalence classes. Every tree $T \in [P]_G$ is distinguished from other trees in $[P]_G$ by its last node, the node with maximum id. The last node of T can be uniquely represented in $[P]_G$ by a pair (x, i) of its label x and id i of its parent node in P, because the node must be the last child of its parent node in order to preserve tree's prefix.

Since two size-k subtrees of a size-$(k+1)$ frequent subtree T created by removing[注1] one of the last two nodes of T are also frequent, and the size-$(k-1)$ prefixes of the two nodes coincides with each other, all size-$(k+1)$ frequent subtrees can be enumerated by joining two size-k frequent subtrees that belong to the same prefix equivalence class. The relation between the last two nodes of size-$(k+1)$ tree is parent-child relation or not, thus join operator must generate trees of the both relations if possible.

When P_1 and P_2 are different size-$(k-1)$ trees, trees generated by join operator from $[P_1]_G$ and $[P_2]_G$ are trivially different because their size-$(k-1)$ prefixes are preserved. Thus, by applying join operator to each combination of two frequent size-k trees in each prefix equivalence class, we can enumerate all size-$(k+1)$ frequent tree candidates without duplication.

3.2 Method for the constrained problem

Our method for the constrained tree mining problem is based on the same idea as Zaki's method mentioned above. Only part we have to consider is how to deal with special nodes.

A *prefix equivalence class* $[P]_G$ in a size-$(k+1)$ tree set G for a size-k tree P, and the set $[G]\text{pre}$ of all prefix equivalence classes in G are defined similarly. The prefix equivalence class for tree S_2 in Figure 1 is shown in Figure 2, which indicates that there are 8 positions for the last node of a tree in the same class. The difference from the case without constraints is that the position of the last node of a tree in the same class cannot be uniquely determined by its parent node. For example, there are three positions for the last node with its parent node labeled a in the prefix equivalence class for tree S_2 in Figure 1.

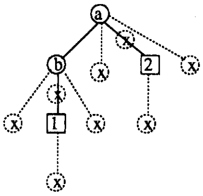

Figure 2 Prefix equivalence class for tree S_2 in Figure 1

To overcome this problem, we specify the position of the last node by its inserted position in the string encoding of its prefix. For example, in the above case, when the last node is between the nodes labeled a and 2, the tree is encoded as "a b 1 -1 -1 x 2 -1 -1" and the prefix of its last node is encoded as "a b 1 -1 -1 2 -1". In this case, its position of the last node is specified as $(4, 6)$, which means that the label x of the last node is inserted right after the 4th character and -1 is inserted right after the 6th character, where a string begins with the 0th character. Therefore, three positions for the last node with its parent node labeled a is specified as $(4, 4), (4, 6)$ and $(6, 6)$.

Let a tree be represented by a pair $(P, (x, (i_1, j_1)))$ of its prefix P and last node $(x, (i_1, j_1))$, where x is its label and (i_1, j_1) is its inserted position in the string encoding of P. We define join operators $\otimes_{\text{in}}$ and $\otimes_{\text{out}}$ on two trees in the same prefix equivalence class as follows.

[Definition 2] Let $(P, (x, (i_1, j_1)))$ and $(P, (y, (i_2, j_2)))$ be two trees in the same prefix equivalence class.

$$(P, (x, (i_1, j_1))) \otimes_{\text{in}} (P, (y, (i_2, j_2)))$$
$$\overset{def}{=} ((P, (x, (i_1, j_1))), (y, (i_2 + 1, j_2 + 1)))$$
$$(P, (x, (i_1, j_1))) \otimes_{\text{out}} (P, (y, (i_2, j_2)))$$
$$\overset{def}{=} ((P, (x, (i_1, j_1))), (y, (i_2 + 2, j_2 + 2)))$$

[注1] : Note that, when the removed node has child nodes, the new parent of those child nodes is the parent of the removed node.

[Proposition 1] Let T be a size-$(k+1)$ tree ($k \geqq 2$) and let $(P,(x,(i_1,j_1)))$ and $(P,(y,(i_2,j_2)))$ be the trees generated by removing the second last and the last non-special nodes, respectively. Then just one of the following cases holds.

Case I $(P,(x,(i_1,j_1))) \otimes_{\text{in}} (P,(y,(i_2,j_2))) = T$ and $i_1 \leqq i_2, j_2 \leqq j_1$

Case II $(P,(x,(i_1,j_1))) \otimes_{\text{out}} (P,(y,(i_2,j_2))) = T$ and $j_1 \leqq i_2$

(Proof) Let t_{h_1} and t_{h_2} be the second last and the last non-special nodes in T, respectively. Since $h_1 < h_2$, t_{h_1} is an ancestor or left-collateral of t_{h_2}. It is easy to see that Case I holds when t_{h_1} is an ancestor of t_{h_2} and Case II holds when t_{h_1} is a left-collateral of t_{h_2}. □

Examples of size-4 trees of case I and case II are shown in Figure 3.

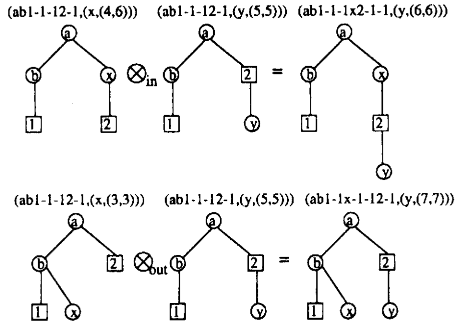

Figure 3 Operation examples for two join operators

The next proposition says that all size-$(k+1)$ candidate subtrees can be enumerated by generating $T_1 \otimes_{\text{in}} T_2$ for all $(T_1,T_2) \in \mathcal{P}_{\text{in}}$ and $T_1 \otimes_{\text{out}} T_2$ for all $(T_1,T_2) \in \mathcal{P}_{\text{out}}$ without duplication, where each of $\mathcal{P}_{\text{in}}$ and $\mathcal{P}_{\text{out}}$ consists of a pair of size-k frequent subtrees belonging to the same prefix equivalence class.

[Proposition 2] Let F_k denote the set of all size-k frequent subtrees for $k \geqq 2$. Let

$\mathcal{P}_{\text{in}} = \{((P,(x,(i_1,j_1))),(Q,(y,(i_2,j_2)))) \in F_k \times F_k :$ $P = Q, i_1 \leqq i_2, j_2 \leqq j_1\}$ and

$\mathcal{P}_{\text{out}} = \{((P,(x,(i_1,j_1))),(Q,(y,(i_2,j_2)))) \in F_k \times F_k :$ $P = Q, j_1 \leqq i_2\}$. Define

$$C_{k+1} = \bigcup_{(T_1,T_2)\in\mathcal{P}_{\text{in}}} \{T_1 \otimes_{\text{in}} T_2\} \cup \bigcup_{(T_1,T_2)\in\mathcal{P}_{\text{out}}} \{T_1 \otimes_{\text{out}} T_2\}$$

Then, the followings hold.

(1) $F_{k+1} \subseteqq C_{k+1}$

(2) $|C_{k+1}| = |\mathcal{P}_{\text{in}}| + |\mathcal{P}_{\text{out}}|$

(Proof)

(1) Let $T \in F_{k+1}$. Let $(P,(x,(i_1,j_1)))$ and $(P,(y,(i_2,j_2)))$ be the trees generated from T by removing the second last and the last non-special nodes, respectively. Then, $(P,(x,(i_1,j_1))),(P,(y,(i_2,j_2))) \in F_k$. Therefore, by Proposition 1, $T \in C_{k+1}$.

(2) Assume that $T_1 \otimes_{\text{in}} T_2 = T_1' \otimes_{\text{in}} T_2'$ for $(T_1,T_2),(T_1',T_2') \in \mathcal{P}_{\text{in}}$. Let $T_2 = (Q,(y,(i_2,j_2)))$ and $T_2' = (Q',(y',(i_2',j_2')))$. Then, $(T_1,(y,(i_2+1,j_2+1))) = (T_1',(y',(i_2'+1,j_2'+1)))$. Thus, $T_1 = T_1', y = y', i_2 = i_2'$ and $j_2 = j_2'$. Therefore, $(T_1,T_2) = (T_1',T_2')$. This means $|\bigcup_{(T_1,T_2)\in\mathcal{P}_{\text{in}}}\{T_1 \otimes_{\text{in}} T_2\}| = |\mathcal{P}_{\text{in}}|$. We can prove $|\bigcup_{(T_1,T_2)\in\mathcal{P}_{\text{out}}}\{T_1 \otimes_{\text{in}} T_2\}| = |\mathcal{P}_{\text{out}}|$ similarly.

Assume that $T_1 \otimes_{\text{in}} T_2 = T_1' \otimes_{\text{out}} T_2'$ for $(T_1,T_2) \in \mathcal{P}_{\text{in}}$ and $(T_1',T_2') \in \mathcal{P}_{\text{out}}$. Let $T_2 = (Q,(y,(i_2,j_2)))$ and $T_2' = (Q',(y',(i_2',j_2')))$. Then, $(T_1,(y,(i_2+1,j_2+1))) = (T_1',(y',(i_2'+2,j_2'+2)))$. Thus, $T_1 = T_1', y = y', i_2 = i_2'+1$ and $j_2 = j_2'+1$. Let $T_1 = (P,(x,(i_1,j_1)))$. Since $(T_1,T_2) \in \mathcal{P}_{\text{in}}$ and $(T_1',T_2') \in \mathcal{P}_{\text{out}}$, $i_1 \leqq i_2, j_2 \leqq j_1$ and $j_1 \leqq i_2'$ hold. Therefore, a contradiction $j_2 \leqq j_1 \leqq i_2' \leqq j_2' < j_2$ is derived. This means that $T_1 \otimes_{\text{in}} T_2 \neq T_1' \otimes_{\text{out}} T_2'$ for any $(T_1,T_2) \in \mathcal{P}_{\text{in}}$ and $(T_1',T_2') \in \mathcal{P}_{\text{out}}$. □

4. Scope-List representation for fast frequency counting

In order to efficiently count the number of trees in a given tree set D to which candidate subtree S can be embedded, Zaki introduced a *scope-list* of S, which is a list of information about S-embeddable positions in D. For our problem with constraints, we also use the scope-list representation with slight modification. An element (t,m,s) of scope-list $\mathcal{L}(S)$ of size-k tree S represents one S-embeddable position in some tree T belonging to D, where t is the id of T, m is a sequence of ids of the *non*-special nodes in T in which the size-$(k-1)$ prefix of S can be embedded, and s is the scope of the last *non*-special node in the S-embeddable position. Note that the size-$(k-1)$ prefix of S contains all special nodes, but m is a sequence of $k-1$ ids of non-special nodes in T, because the special nodes can be uniquely embedded in any tree T belonging to D. We call an element of scope-list of tree S an *S-instance-scope.*

For example, let tree id of tree T in Figure 1 be 0. Then $(0,0,[1,2])$ is an S_2-instance-scope for S_2 in Figure 1, and $(0,01,[3,3])$ is an S_1-instance-scope for S_1 in the same figure.

For two size-k trees S_1 and S_2 with a common size-$(k-1)$ prefix, we define join operators $\otimes$ on a pair of S_1-instance-scope and S_2-instance-scope as follows.

[Definition 3] Let S_1 and S_2 be size-k trees with a common size-$(k-1)$ prefix. Let $(t,m,[l_1,r_1])$ and $(t,m,[l_2,r_2])$ be S_1-instance-scope and S_2-instance-scope, respectively.

$$(t,m,[l_1,r_1]) \otimes (t,m,[l_2,r_2]) \overset{def}{=} (t,ml_1,[l_2,r_2])$$

Note that ml_1 is node id sequence m appended l_1.

[Proposition 3] Let T be a size-$(k+1)$ tree ($k \geqq 2$) and let $S_1 = (P, (x, (i_1, j_1)))$ and $S_2 = (P, (y, (i_2, j_2)))$ be the trees generated by removing the second last and the last non-special nodes, respectively. Let $(t, ml_1, [l_2, r_2])$ be a T-instance-scope and let $[l_1, r_1]$ be the scope of the node with id l_1. Then the followings hold.

(1) $(t, m, [l_1, r_1]) \in \mathcal{L}(S_1), (t, m, [l_2, r_2]) \in \mathcal{L}(S_2)$

(2) $\begin{cases} l_1 < l_2, r_2 \leqq r_1 & \text{if } S_1 \otimes_{\text{in}} S_2 = T \\ r_1 < l_2 & \text{if } S_1 \otimes_{\text{out}} S_2 = T \end{cases}$

(Proof) (1) This is trivial because any domain-restricted embedding mapping also satisfies the conditions of embedding mapping.

(2) Assume $S_1 \otimes_{\text{in}} S_2 = T$. Then, $i_1 \leqq i_2$ and $j_2 \leqq j_1$ hold by Proposition 1. This means that the last two nodes in T are ancestor-descendant relation. Thus, nodes labeled by l_1 and l_2 are also ancestor-descendant relation. Since two nodes are distinct nodes, $l_1 < l_2$ and $r_2 \leqq r_1$ hold. In the case with $S_1 \otimes_{\text{out}} S_2 = T$, $r_1 < l_2$ can be proved similarly. □

The next proposition says that scope-lists of $S_1 \otimes_{\text{in}} S_2$ and $S_1 \otimes_{\text{out}} S_2$ can be generated without duplication by joining all pairs (q_1, q_2) of elements satisfying a certain condition, where q_1 and q_2 belong to the scope-lists of S_1 and S_2, respectively.

[Proposition 4] Let $S_1 = (P, (x, (i_1, j_1)))$ and $S_2 = (P, (x, (i_2, j_2)))$ be size-k trees with a common size-$(k-1)$ prefix P. Let

$$\mathcal{I}_{\text{in}} = \{((t, m, [l, r]), (t', m', [l', r'])) \in \mathcal{L}(S_1) \times \mathcal{L}(S_2) : t = t', m = m', l < l', r' \leqq r\} \text{ and}$$

$$\mathcal{I}_{\text{out}} = \{((t, m, [l, r]), (t', m', [l', r'])) \in \mathcal{L}(S_1) \times \mathcal{L}(S_2) : t = t', m = m', r < l'\}.$$

Then, the followings hold.

(1) For $i_1 \leqq i_2, j_2 \leqq j_1$,

$$\mathcal{L}(S_1 \otimes_{\text{in}} S_2) = \bigcup_{(q_1, q_2) \in \mathcal{I}_{\text{in}}} \{q_1 \otimes q_2\},$$

$$|\mathcal{L}(S_1 \otimes_{\text{in}} S_2)| = |\mathcal{I}_{\text{in}}|.$$

(2) For $j_1 \leqq i_2$,

$$\mathcal{L}(S_1 \otimes_{\text{out}} S_2) = \bigcup_{(q_1, q_2) \in \mathcal{I}_{\text{out}}} \{q_1 \otimes q_2\},$$

$$|\mathcal{L}(S_1 \otimes_{\text{out}} S_2)| = |\mathcal{I}_{\text{out}}|.$$

(Proof) (1) Let $T = S_1 \otimes_{\text{in}} S_2$ and $q \in \mathcal{L}(T)$. Then S_1 and S_2 are subtrees generated by removing the second last and the last non-special nodes, respectively. By Proposition 3, there exists $(q_1, q_2) \in \mathcal{I}_{\text{in}}$ such that $q = q_1 \otimes q_2$. Thus, $\mathcal{L}(S_1 \otimes_{\text{in}} S_2) \subset \bigcup_{(q_1, q_2) \in \mathcal{I}_{\text{in}}} \{q_1 \otimes q_2\}$.

Let $(q_1, q_2) \in \mathcal{I}_{\text{in}}$. Then q_1 and q_2 can be represented by $(t, m, [l_1, r_1])$ and $(t, m, [l_2, r_2])$, respectively. The embedding mapping represented by $q_1 \otimes q_2$ is extension of two embedding mapping represented by q_1 and q_2. Therefore, the extended mapping satisfies all the conditions of embedding mapping except the relation between two nodes labeled l_1 and l_2. In $S_1 \otimes_{\text{in}} S_2$, the relation between the last two nodes is ancestor-descendant relation, and the relation between the two nodes labeled l_1 and l_2 is also ancestor-descendant relation because $l_1 < l_2, r_2 \leqq r_1$. Thus, the extended mapping also satisfies the condition between the two nodes labeled l_1 and l_2. Therefore $\mathcal{L}(S_1 \otimes_{\text{in}} S_2) \supset \bigcup_{(q_1, q_2) \in \mathcal{I}_{\text{in}}} \{q_1 \otimes q_2\}$.

For $(q_1, q_2), (q_1', q_2') \in \mathcal{I}_{\text{in}}$, $(q_1, q_2) \neq (q_1', q_2') \Rightarrow q_1 \otimes q_2 \neq q_1' \otimes q_2'$ holds trivially, so $|\mathcal{L}(S_1 \otimes_{\text{in}} S_2)| = |\mathcal{I}_{\text{in}}|$ holds.

(2) This can be proved similarly. □

5. ConstrainedTreeMiner algorithm

5.1 Algorithm

ConstrainedTreeMiner algorithm in Figure 4 enumerates all frequent embedded subtrees containing all special nodes efficiently. Its algorithm structure is the same as TreeMiner algorithm [9], but its data structure and join operations are different as mentioned in the previous sections.

The basic algorithm structure is as follows. First, by executing procedure Enumerate-F_2 described in the next subsection, the algorithm creates the set F_2 of size-2 frequent subtrees and divides it into the set $[F_2]_{\text{pre}}$ of its prefix equivalence classes while creating the scope-list $\mathcal{L}(S)$ of $S \in F_2$. For each $[P] \in [F_2]_{\text{pre}}$, all larger frequent trees having prefix P can be created from $[P]$ and $\{\mathcal{L}(S) : S \in [P]\}$ by recursively applying Enumerate-Frequent-Subtrees procedure. In Enumerate-Frequent-Subtrees procedure, one size larger candidate subtree is created by joining two subtrees S_1 and S_2 with the same prefix P using operators $\otimes_{\text{in}}$ and $\otimes_{\text{out}}$, and frequency counting for the candidate is done by joining elements in $\mathcal{L}(S_1)$ and $\mathcal{L}(S_2)$ using operator $\otimes$.

Here, we assume that the least common ancestor of all special nodes in any tree in D is a non-special node for the sake of simplicity. Note that slight modification is necessary to deal with the case that the least common ancestor is a special node.

5.2 Procedure Enumerate-F_2

Procedure Enumerate-F_2, which creates the set F_2 of size-2 frequent subtrees and scope-lists for its elements, is the following process for each tree T in D. (See Figure 5 for examples of each step.)

Step 0 Obtain the paths p_i from the root node to each special node labeled i in T. (In many cases, the paths are given.)

Step 1 Create a subtree U composed of all the paths

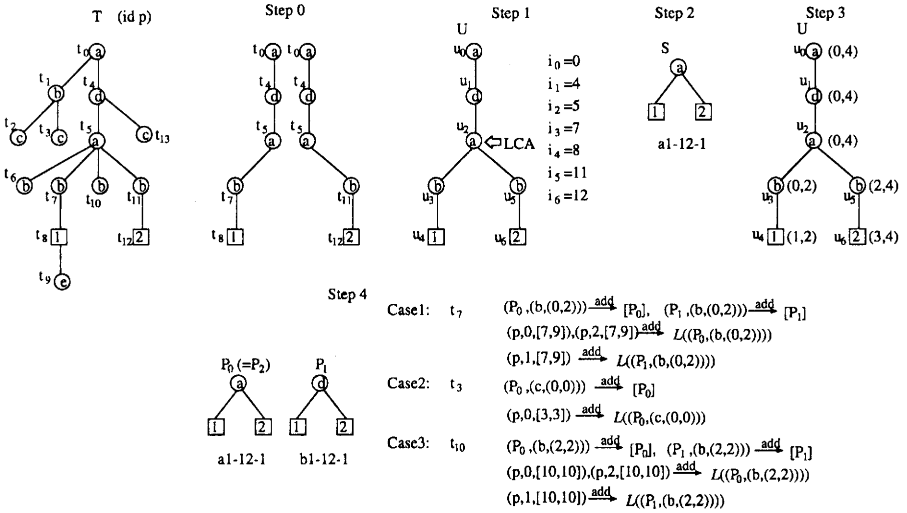
T (id p)
Step 0
Step 1
Step 2
Step 3
U
S
U
t_0 t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8 t_9 t_{10} t_{11} t_{12} t_{13}
LCA
i_0=0
i_1=4
i_2=5
i_3=7
i_4=8
i_5=11
i_6=12
a1-12-1
(0,4)
(0,4)
(0,4)
(0,2)
(2,4)
(1,2)
(3,4)
Step 4
P_0 (=P_2)
P_1
a1-12-1
b1-12-1
Case1: t_7
$(P_0,(b,(0,2))) \xrightarrow{add} [P_0]$, $(P_1,(b,(0,2))) \xrightarrow{add} [P_1]$
$(p,0,[7,9]),(p,2,[7,9]) \xrightarrow{add} \mathcal{L}((P_0,(b,(0,2))))$
$(p,1,[7,9]) \xrightarrow{add} \mathcal{L}((P_1,(b,(0,2))))$
Case2: t_3
$(P_0,(c,(0,0))) \xrightarrow{add} [P_0]$
$(p,0,[3,3]) \xrightarrow{add} \mathcal{L}((P_0,(c,(0,0)))$
Case3: t_{10}
$(P_0,(b,(2,2))) \xrightarrow{add} [P_0]$, $(P_1,(b,(2,2))) \xrightarrow{add} [P_1]$
$(p,0,[10,10]),(p,2,[10,10]) \xrightarrow{add} \mathcal{L}((P_0,(b,(2,2))))$
$(p,1,[10,10]) \xrightarrow{add} \mathcal{L}((P_1,(b,(2,2))))$

Figure 5 Example of each steps of the procedure enumerating all size-2 frequent subtrees and creating its scope-list

```
ConstrainedTreeMiner($D$,minsup)
begin
  Enumerate-$F_2$($D$,minsup)
  for all $[P] \in [F_2]_{\text{pre}}$ do
    Enumerate-Frequent-Subtrees($[P]$, $\{\mathcal{L}(S) : S \in [P]\}$,minsup)
  enddo
end

Enumerate-Frequent-Subtrees($[P]$, $\{\mathcal{L}(S) : S \in [P]\}$,minsup)
begin
  for all $(P, (x, (i_1, j_1))) \in [P]$ do
    $S_1 = (P, (x, (i_1, j_1)))$
    $[S_1] = \emptyset$
    for all $(P, (y, (i_2, j_2))) \in [P]$ do
      $S_2 = (P, (y, (i_2, j_2)))$
      if $i_1 \leqq i_2 \leqq j_2 \leqq j_1$ then
        Create $\mathcal{L}(S_1 \otimes_{\text{in}} S_2)$
        if $S_1 \otimes_{\text{in}} S_2$ is frequent then
          $[S_1] \leftarrow [S_1] \cup \{(S_1, (y, (i_2 + 1, i_2 + 1)))\}$
      if $j_1 \leqq i_2$ then
        Create $\mathcal{L}(S_1 \otimes_{\text{out}} S_2)$
        if $S_1 \otimes_{\text{out}} S_2$ is frequent then
          $[S_1] \leftarrow [S_1] \cup \{(S_1, (y, (i_2 + 2, i_2 + 2)))\}$
    enddo
    Enumerate-Frequent-Subtrees($[S_1]$, $\{\mathcal{L}(S) : S \in [S_1]\}$,minsup)
  enddo
end
```

Figure 4 ConstrainedTreeMiner algorithm

$p_1, p_2, ..., p_d$. Let a one-to-one mapping $i : j \mapsto i_j$ denote the embedding mapping from the set $\{0, 1, .., m-1\}$ of node ids of U to the set of node ids of T. Let u_l, the node in U with id l, denote the least common ancestor of all special nodes.

Step 2 Create an embedded subtree S of U composed of the root node and all special nodes.

Step 3 Attach *insertable position* (s_u, e_u) for S to each node u of U. *Insertable positions* are calculated as follows. Set v to 0 initially. Starting from the root node, traverse all nodes in the order of node ids. For each node u, set s_u to the value of v at the first visit, and set e_u to the value of v at the last visit. When visiting a special node at the first and last times, add 1 to v before setting s_u and e_u.

Step 4 For all node t_h in T, do the followings. Let t_{i_k} be the least ancestor of t_h among all nodes in $\{t_{i_0}, t_{i_1}, ..., t_{i_{m-1}}\}$. Let $C = \{j : i_j < h, u_j \text{ is a child of } u_k\}$. Let $l' = \min\{l, k\}$. Let P_i denote a tree that is created from S by replacing the root node label with u_i. Let b denote the tree id of T, and let x and s denote the label and the scope of node t_h, respectively.

Case 1 $i_k = h$
Add $(x, (s_{u_k}, e_{u_k}))$ to $[P_i]$ and add (b, i, s) to $\mathcal{L}((P_i, (x, (s_{u_k}, e_{u_k}))))$ for all $i \in \{0, 1, ..., l'\}$.

Case 2 $i_k \neq h$ and $C = \emptyset$
Add $(x, (s_{u_k}, s_{u_k}))$ to $[P_i]$ and add (b, i, s) to $\mathcal{L}((P, (x, (s_{u_k}, s_{u_k}))))$ for all $i \in \{0, 1, ..., l'\}$.

Case 3 $i_k \neq h$ and $C \neq \emptyset$
Let $j^* = \max C$. Add $(x, (e_{u_{j^*}}, e_{u_{j^*}}))$ to $[P_i]$ and add (b, i, s) to $\mathcal{L}((P_i, (x, (e_{u_{j^*}}, e_{u_{j^*}}))))$ for all $i \in \{0, 1, ..., l'\}$.

5.3 Space problem

Keeping scope-lists of size-k frequent subtrees reduces computational time needed to calculate scope-list of size-$(k+1)$ candidate trees, but sometimes requires a lot of space, especially when data includes some trees with a lot of nodes having the same labels. For example, consider the case shown in Figure 6. In this case, $\mathcal{L}(S_k)$ contains $\binom{100}{k}$ S_k-instance-scopes for T, thus the number of S_k-instance-scopes for T increases exponentially with respect to k. This often happens for DOM-trees of HTML documents, for example, '<br>' and '<td>' possibly occurs many times in each of them. We should calculate scope-lists when we want to know the number of occurrences of a subtree in each tree, but we do not have to calculate them when we only want to know whether a subtree occurs in each tree or not as the case of Problem 1.

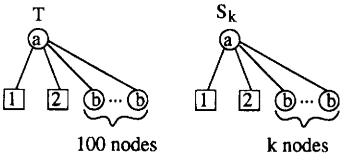

Figure 6 A tree that causes space problem

One of the countermeasures for this problem is as follows. Instead of keeping the scope-lists of frequent subtrees S, keep only the first S-instance-scope in lexicographical order for each tree, namely, S-instance-scope $(t, m, [l, r])$ for tree T of id t such that the sequence ml of t's node ids comes first in lexicographical order among all sequence $m'l'$ of t's node ids for S-instance-scope $(t, m', [l', r'])$ in $\mathcal{L}(S)$. Let S_1 and S_2 be frequent subtrees and let $(t, m_1, [l_1, r_1])$ and $(t, m_2, [l_2, r_2])$ be the first S_1- and S_2-instance-scopes, respectively. Then, calculation of $\mathcal{L}(S_1 \otimes_{\text{in}} S_2)$ in Enumerate-Frequent-Subtree procedure is replaced with the following procedure for each tree T of id t that finds the first $(S_1 \otimes_{\text{in}} S_2)$-instance-scope for T. Note that giving the first S_1- and S_2-instance-scopes to the procedure reduces the computational time of Step 1. Letting each node of T have the id of the next node with the same tag reduces computational time of all 'min' functions efficiently. Also note that the first 'min' functions in Step 1 and Step 4 can be implemented by recursive procedure that finds the next P-instance-scope for one size smaller prefix P. Calculation of $\mathcal{L}(S_1 \otimes_{\text{out}} S_2)$ in Enumerate-Frequent-Subtree procedure is also replaced with the corresponding procedure similarly.

Step 1 Calculate a pair of S_1-instance-scope $(t, m', [l'_1, r'_1])$ and S_2-instance-scope $(t, m', [l'_2, r'_2])$ such that $m'l'_1 = \min\{m^*l^*_1 : (t, m^*, [l^*_1, r^*_1]) \in \mathcal{L}(S_1), (t, m^*, [l^*_2, r^*_2]) \in \mathcal{L}(S_2)\}$ and $l'_2 = \min\{l^*_2 : (t, m', [l^*_2, r^*_2]) \in \mathcal{L}(S_2)\}$. If there is no such pair, no $(S_1 \otimes_{\text{in}} S_2)$-instance-scope exists in the tree of id t. Otherwise, go to Step 2.

Step 2 If $l'_1 < l'_2$ and $r'_2 \leqq r'_1$ hold, then $(t, m'l'_1, [l'_2, r'_2])$ is the first $(S_1 \otimes_{\text{in}} S_2)$-instance-scope. Otherwise, go to Step 3.

Step 3 Set l to the value of l'_2. Calculate the next S_2-instance-scope $(t, m', [l'_2, r'_2])$ such that $l'_2 = \min\{l^*_2 : (t, m', [l^*_2, r^*_2]) \in \mathcal{L}(S_2), l^*_2 > l\}$. If there are no such S_2-instance-scope, set l_1 to the value of l'_1, set m_1 to the value of m' and go to Step 4. Otherwise go to Step 2.

Step 4 Calculate a pair of S_1-instance-scope $(t, m', [l'_1, r'_1])$ and S_2-instance-scope $(t, m', [l'_2, r'_2])$ such that $m'l'_1 = \min\{m^*l^*_1 : (t, m^*, [l^*_1, r^*_1]) \in \mathcal{L}(S_1), (t, m^*, [l^*_2, r^*_2]) \in \mathcal{L}(S_2), m^*l^*_1 > m_1l_1\}$ and $l'_2 = \min\{l^*_2 : (t, m', [l^*_2, r^*_2]) \in \mathcal{L}(S_2)\}$. If there is no such pair, no $(S_1 \otimes_{\text{in}} S_2)$-instance-scope exists in the tree of id t. Otherwise, go to Step 2.

6. Application to DOM-tree analysis

We conducted an experiment of extracting common structures from DOM-trees of HTML documents. The HTML documents we used in our experiment are Web pages containing the information about a given Ramen (lamian, Chinese noodles in Soup) Shop. We selected the most popular 10 Ramen shops with more than 15 collected Web pages among the shops mentioned in the Ramen-shop's popularity ranking of a popular local town information magazine (Hokkaido Walker). From the collected Web pages about 10 Ramen shops, we further selected the pages that contain both the shop name and its reputation consisting of only one text node each. There are 99 pages satisfying these conditions. Each tree in our experimental data is created by transforming each Web page to a DOM-tree, relabeling its text nodes of the shop name and its reputation by special labels 1 and 2, respectively, and finally choosing the subtree consisting of all descendants of the least common ancestor (LCA) of the special nodes. The reason why we do not use the whole tree is that we do not want to extract trivial frequent structures like "html title -1 body ⋯ -1" that are common to all HTML documents. For the data constructed by the above procedure, we applied our algorithm to enumerate all frequent subtrees containing all special labels.

Table 1 shows frequency distribution of HTML tags (non-special labels) attached to the root nodes, the LCA of the special nodes. Since we wanted to extract frequent subtrees even for the trees that have several other trees with a root node labeled the same tag, we calculated support rate for each set of trees with a root node labeled the same tag. By this support calculation, every subtree of a tree with a root

table	22	p	9	div	1
body	21	tbody	6	dl	1
td	18	li	4	span	1
tr	15	blockquote	1		

Tablex 1 LCA's tags of special nodes and its frequency in the data

node labeled 'blockquote', 'div', 'dl' or 'span' achieves support rate 1.0, so we did not enumerate such subtrees. In our experiment, all subtrees containing text nodes were not enumerated. All *maximal* frequent subtrees for minimum support $\sigma = 1.0, 0.9, 0.8, 0.7$ are shown in Figure 7. Note that a *maximal* frequent subtree is a subtree such that its any super-tree is not frequent. When a shop name and its reputation are embedded in a table, the positional relation between them is clear: upper-lower relation for "table tr 1 -1 -1 tr td 2 -1 -1 -1", "tbody tr td 1 -1 -1 -1 tr td 2 -1 -1 -1" and "td 1 -1 br -1 2 -1", left-right relation for "tr td 1 -1 -1 td 2 -1 -1". Especially, when both a shop name and its reputation are in one cell in a table, namely, the case with "td 1 -1 br -1 2 -1", the shop names are emphasized like "td strong 1 -1 -1 br -1 2 -1" in most such pages. When the tag of LCA is 'body', 'p' or 'li', the structures around the nodes of a shop name and its reputation is not so clear, but the positional relation between them is upper-lower one in most such trees like "body br -1 1 -1 br -1 2 -1 br -1 br -1 a -1", "p 1 -1 br -1 2 -1" and "li 1 -1 br -1 br -1 br -1 2 -1 br -1 br -1 br -1 br -1 br -1".

```
σ = 1.0
 22   table tr 1 -1 -1 tr td 2 -1 -1 -1
  6   tbody tr td 1 -1 -1 -1 tr td 2 -1 -1 -1
 15   tr td 1 -1 -1 td 2 -1 -1
σ = 0.9
 19   body 1 -1 2 -1 br -1
 20   table tr td 1 -1 -1 -1 tr td 2 -1 -1 -1
  6   tbody tr td 1 -1 -1 -1 tr td 2 -1 -1 -1
 17   td 1 -1 br -1 2 -1
 15   tr td 1 -1 -1 td 2 -1 -1
σ = 0.8
 17   body br -1 1 -1 2 -1 a -1 br -1
 17   body br -1 1 -1 2 -1 br -1 a -1
  8   p 1 -1 2 -1 br -1
  8   p 1 -1 br -1 2 -1
 20   table tr td 1 -1 -1 -1 tr td 2 -1 -1 -1
 18   table tr td 1 -1 -1 -1 td -1 2 -1
  5   tbody tr td 1 -1 -1 -1 tr td 2 -1 -1 -1 tr td -1 -1
  5   tbody td 1 -1 -1 td -1 td 2 -1 -1
 17   td 1 -1 br -1 2 -1
 15   tr td 1 -1 -1 td 2 -1 -1
σ = 0.7
 15   body br -1 1 -1 2 -1 br -1 br -1 br -1 br -1 br -1 br -1 br -1 br -1 a -1 br -1
 15   body br -1 1 -1 br -1 2 -1 br -1 br -1 a -1
 15   body br -1 1 -1 br -1 2 -1 br -1 a -1 br -1
 15   body 1 -1 2 -1 hr -1
  3   li 1 -1 br -1 br -1 br -1 2 -1 br -1 br -1 br -1 br -1 br -1
  3   li 1 -1 a -1 img -1 2 -1 a -1
  3   li 1 -1 a img -1 -1 2 -1
  7   p 1 -1 br -1 2 -1 br -1
 20   table tr td 1 -1 -1 -1 tr td 2 -1 -1 -1
 16   table tr td 1 -1 -1 -1 td -1 2 -1 td -1
 16   table tr td 1 -1 -1 -1 td -1 td 2 -1 -1
 17   table tr td 1 -1 -1 -1 td 2 -1 -1 td -1
  5   tbody tr td 1 -1 -1 -1 tr td 2 -1 -1 -1 tr td -1 -1
  5   tbody td 1 -1 -1 td -1 td 2 -1 -1
 13   td 1 -1 br -1 br -1 2 -1
 13   td strong 1 -1 -1 br -1 2 -1
 13   td 1 -1 2 -1 br -1
 11   tr td 1 -1 -1 td -1 td 2 -1 -1
```

Figure 7 Maximal subtrees with at least support σ and their frequencies

6.1 Concluding Remarks

Structures found in our experiment appear to be useful, but they are too general to specify the place of necessary information. To construct a wrapper, other features of HTML documents like contents of the information and distance between nodes should be additionally used. We are now trying to develop such combined method, which is potentially able to extract necessary information from arbitrary Web pages retrieved by a search engine, while most conventional wrappers can do only from the pages in the same site as the training pages.

References

[1] R. Agrawal and R. Srikant. First algorithms for mining association rules. In *Proc. 20th Int'l Conf. on VLDB*, pages 487–499, 1994.

[2] R. Agrawal and R. Srikant. Mining sequencial patterns. In *Proc. 11th Int'l Conf. on Data Eng.*, pages 3–14, 1995.

[3] T. Asai, K. Abe, S. Kawasoe, H. Arimura, H. Sakamoto, and S. Arikawa. Efficient substructure discovery from large semi-structured data. In *Proc. 2nd SIAM Int'l Conf. on Data Mining*, pages 158–174, 2002.

[4] W. W. Cohen, M. Hurst, and L. S. Jensen. A flexible learning system for wrapping tables and lists in html documents. In *Proc. 11th Int'l World Wide Web Conf.*, pages 232–241, 2002.

[5] M. Garofalakis, R. Rastogi, and K. Shim. Mining sequential patterns with regular expression constraints. *IEEE Transactions on Knowledge and Data Engineering*, 14(3):530–552, 2002.

[6] A. Inokuchi, T. Washio, and H. Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In *Proc. PKDD 2000*, pages 13–23, 2000.

[7] N. Kushmerick. Wrapper induction:efficiency and expressiveness. *Artificial Intelligence*, (118):15–68, 2000.

[8] R. Srikant, Q. Vu, and R. Agrawal. Mining association rules with item constraints. In *Proc. 3rd Int'l Conf. on Knowledge Discovery and Data Mining*, pages 67–73, 1997.

[9] M. J. Zaki. Efficiently mining frequent trees in a forest. In *Proc. SIGKDD'02*, pages 71–80, 2002.