International Journal of Database Theory and Application
Vol. 2, No. 4, December, 2009

Algorithm for Enumerating All Maximal Frequent Tree Patterns among Words in Tree-Structured Documents and Its Application

Tomoyuki Uchida and Kayo Kawamoto

Faculty of Information Sciences, Hiroshima City University,
Hiroshima 731-3194, Japan
{uchida, kayo}@hiroshima-cu.ac.jp

Abstract

To extract structural features from tree-structured documents among nodes in which characteristic words appear, we described a text-mining algorithm for enumerating all frequent consecutive path patterns (CPP) on a list W of words in Uchida et al., PAKDD 2004 [14]. In this paper, we first extend a CPP to a tree pattern, which is called a tree association pattern (TAP), over a set W of words. A TAP is an ordered rooted tree t such that the root of t has no child or at least two children, all leaves of t are labeled with non-empty subsets of W and all internal nodes, if they exist, are labeled with strings. By modifying text-mining algorithms to find all frequent CPPs, next, we present text-mining algorithms for enumerating all maximal frequent TAPs in tree-structured documents, where a TAP t is maximal if there exists no frequent TAP, which has t as a proper subpattern. Then, we tested our algorithms using Reuters news wires. Finally, as one application of CPPs, we present an algorithm for a wrapper based on CPP using XSLT transformation language and demonstrate simply the use of the wrapper to translate one of the Reuters news wires into other XML document.

Keywords: *text-mining, tree-structured documents, tree-structured pattern, maximal frequent tree pattern, wrapper.*

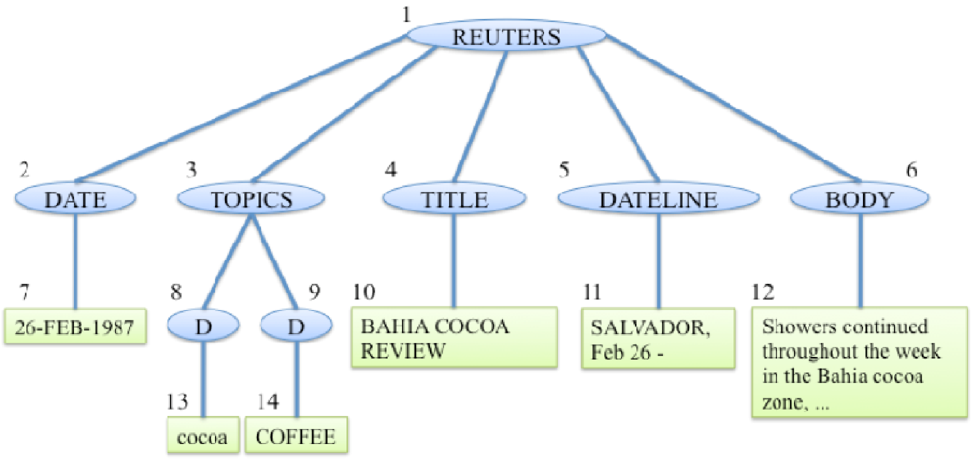
1. Introduction

Many electronic documents such as SGML/HTML/XML files and LaTeX files have tree-structures that do not have rigid structures. Such documents are called tree-structured documents. Since many tree-structured documents contain large amounts of plain texts, we focus on characteristics such as the usage of words and the structural relations among nodes, in which characteristic words appear, in tree-structured documents. The aim of this work is to present an efficient text-mining technique for extracting structural features among nodes, in which characteristic words appear in tree-structured documents. A tree-structured document is represented by a node-labeled ordered rooted tree, which is called an *OEM tree* (See [1]). As a value, each node of an OEM tree has a string such as a tag in HTML/XML files, or text such as text written in the PCDATA field in XML files. Moreover, an internal node has children, which are ordered. For example, we give an XML file xml_sample and its OEM tree T in Figure 1. In T, node 3 has ``TOPICS'' as a value and is the second child of node 1.

Many tree-structured documents have no absolute schema fixed in advance, and their structures may be irregular or incomplete. The formalization of representing knowledge is important for finding useful knowledge. For an integer $k \geq 2$, let $(w_1, w_2, \ldots, w_k)$ be a list of k words appearing in a given set of tree-structured documents such that the words are sorted in ASCII-based lexicographical order. A *consecutive path pattern* (*CPP*) on $(w_1, w_2, \ldots, w_k)$ has

```
<REUTERS>
  <DATE> 26-FEB-1987 </DATE>
  <TOPICS> <D> cocoa </D> <D> COFFEE </D> </TOPICS>
  <TITLE> BAHIA COCOA REVIEW </TITLE>
  <DATELINE> SALVADOR, Feb 26 - </DATELINE>
  <BODY>
                              Showers continued throughout the week
    in the Bahia cocoa zone, ...
  </BODY>
</REUTERS>
```

xml_sample



T

Figure 1. XML file xml_sample and OEM tree *T* of xml_sample

been introduced in [14] as a sequence $< t_1, t_2, \ldots, t_{k-1} >$ such that for i ($1 \leq i \leq k$-1), t_i is a path between nodes labeled with $\{w_i\}$ and $\{w_{i+1}\}$. For example, the sequence $\alpha = < t_1, t_2, t_3 >$

consisting of three paths t_1, t_2, t_3 given in Figure 2 is a CPP on (COFFEE, SALVADOR, cocoa, week). A CPP $\alpha = < t_1, t_2, \ldots, t_k >$ is said to *appear in* the OEM tree T_d of a given tree-structured document d, if there exists a sequence $(s_d^1, s_d^2, \ldots, s_d^k)$ of subtrees $s_d^1, s_d^2, \ldots, s_d^k$ of T_d satisfying the following two conditions.

(1) For each $1 \leq i \leq k$, subtree s_d^i is isomorphic to 2-tree t_i in α.

(2) For each $1 \leq j \leq k$-1, the right-hand leaf of s_d^i and the left-hand leaf of s_d^{i+1} are the same in T_d.

First, by extending a CPP to a tree pattern, we define a *tree association pattern* (TAP) on a set W of words as an ordered rooted tree t such that either of the following two conditions is satisfied.

(1) t consists of only one node labeled with W.

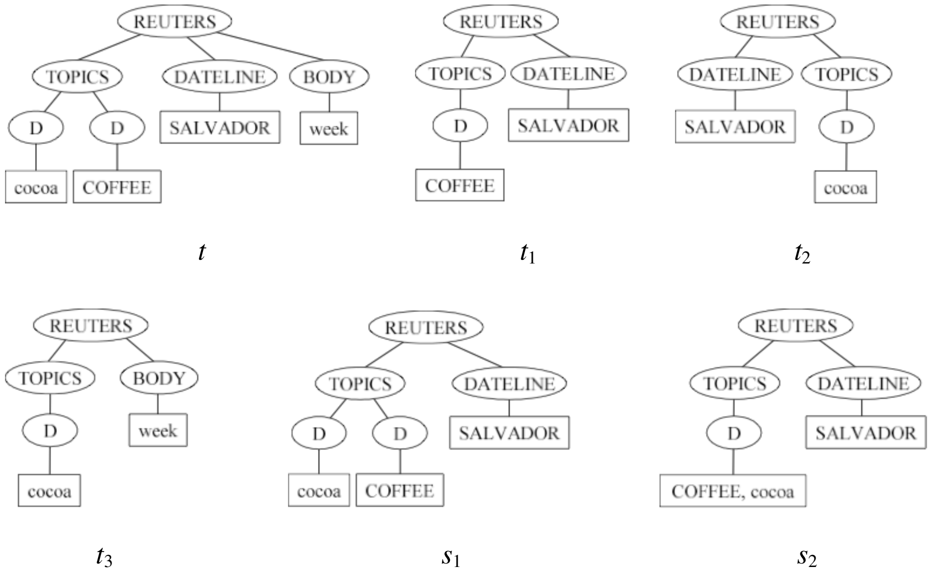


Figure 2. TAPs t over {COFFEE, SALVADOR, cocoa, week}, s_1 and s_2 over {COFFEE, SALVADOR, cocoa}, and 2-trees t_1, t_2 and t_3 over {COFFEE, SALVADOR}, {SALVADOR, cocoa}, and {cocoa, week}, respectively.

(2) The root of t has at least two children. Any leaf v and any internal node of t has a non-empty subset $W(v)$ of W and a string as labels, respectively, such that $W = \bigcup_{v \in V} W(v)$, where V is the set of all leaves of t.

If there are only two elements in W, a TAP on W is called a *2-tree*. For example, in Figure 2, t is a TAP on {COFFEE, SALVADOR, cocoa, week}, and t_1, t_2, and t_3

are 2-trees on {COFFEE, SALVADOR}, {SALVADOR, cocoa}, and {cocoa, week}, respectively.

A TAP t on $\{w_1, w_2, \ldots, w_k\}$ is said to *appear in* the OEM tree T_d of a given tree-structured document d if there exists a subtree s of T_d such that t and s are isomorphic under the following three conditions. Here, this isomorphism from the node set of t to the node set of s is denoted by π.

(1) v is a leaf of t if and only if $\pi(v)$ is a leaf of s.

(2) For any internal node v of t, the labels of v and $\pi(v)$ are the same.

(3) For any leaf v of t, each word in the label of v appears in the label of $\pi(v)$ as a word.

For example, TAP t and 2-trees t_1, t_2, t_3 in Figure 2 appear in T of Figure 1.

A TAP t and a CPP α are said to be *frequent* if t and α appear in a given set of tree-structured documents at a frequency more than a user-specified threshold, which is called a *minimum support, respectively*. A frequent TAP t and a frequent CPP α is *maximal* if there exists no frequent TAP and no frequent CPP, which has t and α as a proper subpattern, respectively. In [14], we described text-mining algorithms for extracting all frequent CPPs from a given set of tree-structured documents. Moreover, Ishida et al. [8] presented online text-mining algorithms for continuously extracting all frequent CPPs from an infinite sequence of tree-structured documents. These algorithms were based on a level-wise search strategy with respect to the length of frequent CPPs. Next, by modifying the text-mining algorithms in [14], we present an efficient text-mining algorithm for enumerating all maximal frequent TAPs in a given set of tree-structured documents that appear at a frequency more than a minimum support. The algorithm is based on a level-wise search strategy with respect to the length of a frequent pattern. We use compact data structures such as the FPTree presented by Han et al. (See [7]) for managing all frequent TAPs. Then, to evaluate the performance of our algorithms, we applied our algorithm to a database of Reuters news wires [9], which contains 21,578 SGML documents with a total size of 28MB. Experimental results on our algorithm showed that it has good performance for a set of a large number of tree-structured documents. Several data mining methods have been described for discovering frequent schema or maximal frequent schema from semi-structured data [2, 5, 10, 15]. For example, Miyahara et al. [10], presented an algorithm for extracting all frequent maximally structural relations between substructures from tree-structured documents. In the context of itemset and sequence mining, there are many studies [11, 12] for mining closed frequent itemsets and sequences. In [16], Yan and Han presented an algorithm for mining closed frequent graph patterns in large graph data sets.

Many studies, including those above, focused on tags and their structured relations appearing in semi-structured data. However, we are more interested in words and their structural relations rather than tags in tree-structured documents. We believe that our algorithms are useful for avoiding inappropriate documents and searching for documents of interest to users. Moreover, as one application of CPPs, we present an algorithm for a wrapper based on CPP using XSLT transformation language [4], which is an XML-based language used for transforming XML documents into other XML documents. A brief demonstration of the use of the wrapper to translate one Reuters news wires to other XML document shows the usefulness of our algorithms.

The rest of this paper is organized as follows. In Section 2, we introduced a consecutive path pattern (CPP) on a list of words described in [14]. In Section 3, we formally define a tree association pattern (TAP) on a set of words as an expansion of a CPP on a list of words. In Section 4, we present text-mining algorithms for enumerating all maximal frequent TAPs in tree-structured documents. In Section 5, as one application of CPPs, we present an algorithm for a wrapper based on CPP using XSLT transformation languages. Section 6 concludes this paper with future works.

2. Consecutive path pattern

Let Σ be an alphabet including the space symbol " ". A finite sequence $(a_1, a_2,\ldots, a_n)$ of symbols in Σ is called a *string* and is denoted by $a_1a_2\ldots a_n$ for short. A *word* is a substring $a_2a_3\ldots a_{n-1}$ of $a_1a_2\ldots a_n$ over Σ such that both a_1 and a_n are space symbols and each a_i (i=2,3, …, n-1) is a symbol in Σ, which is not the space symbol. The set of all words in Σ^+ is denoted by W. Since we deal with only tree-structured documents, hereafter, we simply call a tree-structured document a *document*. For a document d, $W(d)$ denotes the set of all words appearing in d. For a set $D=\{d_1, d_2, \ldots, d_m\}$ of documents, $W(D)=\bigcup_{1\le i\le m} W(d_i)$. For a document d and a word w, $AN_d(w)$ denotes the set of all nodes of T_d, whose value contains w as a word, where T_d is the OEM tree of a document d. For a set D of documents and a word w, $ANS_D(w)=\bigcup_{d\in D}AN_d(w)$ is called an *appearance node set* of w in D. For two words w and

w', if w is less than w' in ASCII-based lexicographical order, we denote $w \le w'$. For a set or a list S, the number of elements of S is denoted by $|S|$.

In [14], we introduced a consecutive path pattern, which is a list of ordered trees, as follows. Let k be an integer greater than 1 and $(w_1, w_2, \ldots, w_k)$ be a list of k words such that $w_i \le w_{i+1}$ for $1 \le i \le k$-1. Then, the list $< t_1, t_2, \ldots, t_{k-1} >$ is called a *consecutive path pattern* (CPP) on $(w_1, w_2, \ldots, w_k)$ if for each i $(1 \le i \le k\text{-}1)$, t_i is an ordered tree, which is called a 2-*tree*, such that t_i consists of only one node labeled with $\{w_i, w_{i+1}\}$ or t_i has only two leaves labeled with $\{w_i\}$ and $\{w_{i+1}\}$. We point out that, for each i $(1 \le i \le k\text{-}2)$, both t_i and t_{i+1} have leaves labeled with the label w_{i+1}. We give a CPP $< t_1, t_2, t_3 >$ on (COFEE, SALVADOR, cocoa, week) as an example, where t_1, t_2, t_3 are in Figure 2.

For a CPP $\alpha = < t_1, t_2, \ldots, t_k >$ and two integers i, j $(1 \le i < j \le k)$, a CPP $< t_i, t_{i+1}, \ldots, t_j >$ is said to be a *sub-CPP* of α. For a CPP $\alpha = < t_1, t_2, \ldots, t_k >$ and a document d, we can define a *matching function* π: $\bigcup_{1 \le i \le k} V_i \rightarrow V_{T_d}$ as follows, where T_d is the OEM tree of d, V_{Td} is the node set of T_d, and V_i is the node set of t_i for i $(1 \le i \le k)$.

(1) For any i $(1 \le i \le k)$, there exists a semimatching function $\pi_i: V_i \rightarrow V_{T_d}$ of t_i for T_d such that for a node v in t_i, $\pi(v) = \pi_i(v)$. The definition of a semimatching function is given in the next section.

(2) For any i $(1 \le i \le k\text{-}1)$, the right-hand leaf $rn(t_i)$ in t_i and the left-hand leaf $ln(t_{i+1})$ in t_{i+1}, $\pi(rn(t_i)) = \pi(ln(t_{i+1}))$.

For a CPP $\alpha = < t_1, t_2, \ldots, t_k >$ and a document d, let $JS_d(\alpha) = \{(\pi(ln(t_1)), \pi(rn(t_1)), \pi(rn(t_2)), \ldots, \pi(rn(t_k))) \mid \pi$ is a matching function of α for the OEM tree of d, and $ln(t_k)$ and $rn(t_k)$ are the left-hand leaf and the right-hand leaf of t_k, respectively$\}$. For a set D of documents, $JS_D(\alpha) = \bigcup_{d \in D} JS_d(\alpha)$. If $|JS_d(\alpha)| \ge 1$, we say that α *appears in d*. For example, the CPP $\alpha = < t_1, t_2, t_3 >$ appears in xml_sample in Figure 1, where 2-trees t_1, t_2, t_3 are in Figure 2.

3. Tree association pattern

We define a tree association pattern as an expansion of a consecutive path pattern as follows. For a non-empty subset W of W with $|\mathrm{W}| \ge 2$, we call a node-labeled ordered rooted tree t a *tree association pattern* (*TAP*) on W if t satisfies either of the following two conditions.

(1) t consists of only one node labeled with W.

(2) t has the root with at least two children, any internal node of t is labeled with a string in Σ^+, and any leaf v of t is labeled with a non-empty subset $W(v)$ of W such that W is equal to $\bigcup_{v \in V} W(v)$, where V is the set of all leaves of t.

We point out that a TAP on W is a 2-tree if $|W|=2$. As an example, we give a TAP t on {COFFEE, SALVADOR, cocoa, week} and 2-trees t_1, t_2 and t_3 over {COFFEE, SALVADOR}, {SALVADOR, cocoa} and {cocoa, week}, respectively.

For a TAP t and a document d, a *sem-imatching function of t for T_d* is any function $\pi: V_t \rightarrow V_{T_d}$ that satisfies the following three conditions, where T_d is the OEM tree of d, and V_t and V_{T_d} are the node sets of t and T_d, respectively.

(1) π is a one-to-one mapping. That is, for any v_1, v_2 in V_t, if $v_1 \ne v_2$ then $\pi(v_1) \ne \pi(v_2)$.

(2) π preserves the parent-child relation. That is, for the edge sets E_t and E_{T_d} of t and T_d, respectively, (v_1, v_2) in E_t if and only if $(\pi(v_1), \pi(v_2))$ in E_{T_d}.

(3) If t consists of only one node v, $\pi(v)$ is a leaf of T_d and all words in the label of v appear individually in the value of $\pi(v)$ in V_{T_d}. Otherwise, for each leaf v of t, $\pi(v)$ is a leaf of T_d and all words in the label of v are contained individually in the value of $\pi(v)$ as words. For each internal node u in V_t, the label of u is equal to the value of the internal node $\pi(u)$ of T_d.

A semi-matching function π: $V_t \rightarrow V_{T_d}$ of t for T_d is said to be a *matching function of t for* T_d if π preserves the sibling relation, that is, for any v_1, v_2 in V_t, v_2 is the next sibling of v_1 if and only if $\pi(v_2)$ is the next sibling of $\pi(v_1)$ of T_d. Let t' be a TAP on $W' \subseteq W$. The TAP t' is said to be a *subtree pattern* of t if there exists a matching function of t' for t. In particular, a subtree pattern t' of t is said to be *proper* if $W' \subseteq W$ and $W - W' \neq \emptyset$. When both a matching function of t' for t and a matching function of t for t' exist, t' is said to be *isomorphic to t*. For a TAP t having k leaves $v_1, v_2, \ldots, v_k$ in this order and a document d, the set $\{(\pi(v_1), \pi(v_2), \ldots, \pi(v_k)) \mid \pi$ is a matching function of t for $T_d\}$, which is denoted by $CPS_d(t)$, is called the *cardinal point set* of t in d, where T_d is the OEM tree of d. For a set D of documents, let $CPS_D(t)= \bigcup_{d \in D} CPS_d(t)$. If $|CPS_d(t)| \geq 1$, we say that *t appears in d*. For example, TAP t and the three 2-trees t_1, t_2, t_3 in Figure 2 appear in the document xml_sample in Figure 1.

Consider the CPP β=<t_1, t_2> on (COFFEE, SALVADOR, cocoa), which is a sub-CPP of α. Since α appears in xml_sample, β appears in xml_sample. Moreover, we can see that β appears in both s_1 and s_2 in Figure 2. TAP s_1 appears in xml_sample, but s_2 does not.

4. Enumerating all maximal frequent TAPs from tree-structured documents

4.1 Text-mining algorithm for maximal frequent TAPs problem

Let D be a set of documents. For a word w, a CPP α, and a TAP t, $Occ_D(w)$, $Occ_D(\alpha)$, and $Occ_D(t)$ denote the numbers of documents in which w, α, and t appear, respectively. Namely, $Occ_D(w)=|\{ d \in D \mid AN_d(w) \neq \emptyset \}|$, $Occ_D(\alpha)=|\{ d \in D \mid JS_d(\alpha) \neq \emptyset \}|$, and $Occ_D(t)=|\{ d \in D \mid CPS_d(t) \neq \emptyset \}|$. For a set D of documents and a real number σ ($0< \sigma \leq 1$), a word w, a CPP α, and a TAP t are σ*-frequent* with respect to D if $Occ_D(w)/|D| \geq \sigma$, $Occ_D(\alpha)/|D| \geq \sigma$ and $Occ_D(t)/|D| \geq \sigma$, respectively. In general, a real number σ is given by a user and is called a *minimum support*. We can define a maximal σ-frequent CPP as follows. A σ-frequent CPP is said to be *maximal* if there exists no σ-frequent CPP, which has α as a sub-CPP. In the same way as a CPP, a σ-frequent TAP t is said to be *maximal* if there exists no σ-frequent TAP, which has t as a proper subtree pattern. From the definitions of a CPP and a TAP, we can show the following facts.

Fact 1. Let W be a list of words sorted in ASCII-based lexicographical order. For a TAP t on W, there exists only one CPP on W that appears in t. On the other hand, for a CPP α on W, α can appear in some TAPs on W.

For the CPP α=<t_1, t_2>, two distinct TAPs s_1 and s_2 in Figure 2, we can see that α is the unique CPP on (COFFEE, SALVADOR, cocoa), which appears in s_1 and s_2.

Fact 2. Let α and β be CPPs such that β is a sub-CPP of α. For a set D of documents, $Occ_D(\alpha) \leq Occ_D(\beta)$ holds.

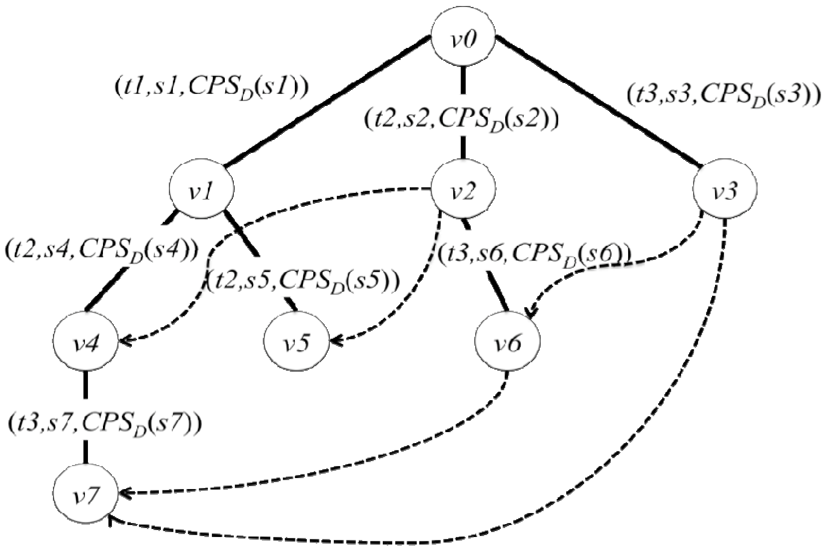


Figure 3. TAPTree T

Fact 3. Let s and t be TAPs such that t is a subtree pattern of s. For a set D of documents, $Occ_D(s) \leq Occ_D(t)$ holds.

We formally define the data mining problem of enumerating all maximal frequent TAPs as follows.

Maximal Frequent TAPs Problem
Instance : A set D of documents and a minimum support σ ($0 < \sigma \leq 1$).
Question : Enumerate all maximal σ-frequent TAPs with respect to D.

In [14], we presented a level-wise search strategy for the problem of enumerating all σ-frequent CPPs in documents. In the same way as [14], in reading given documents, we construct a *trie* (see [6]), which is a compact data structure, for efficiently storing and searching words and all appearance node sets of words. As heuristics of our algorithms, we assume that we exclude high-frequency words such as ``a" or ``the." Such a word is called a *stop word*. For a 2-tree s over $\{w_s^1, w_s^2\}$ with $w_s^1 \leq w_s^2$, the *coding* of s is the list of labels on the path from the leaf labeled with w_s^1 to the leaf labeled with w_s^2. We define a total order over the set of all 2-trees in an ordinary way with respect to codings. Let $(w_t^1, w_t^2, \ldots, w_t^n)$ and $(w_s^1, w_s^2, \ldots, w_s^m)$ be codes of 2-trees t and s, respectively. Then, t is less than s, which is denoted by $t < s$, if (1) when $m \leq n$, there exists i ($1 \leq i \leq m$) such that for each $j < i$, w_s^j and w_t^j are the same label and $w_t^i \leq w_s^i$, or (2) when $m > n$, for each i ($1 \leq i \leq n$), w_s^i and w_t^i are the same label or there exists i ($1 \leq i \leq n$) such that for $j < i$, w_s^j and w_t^j are the same label and $w_t^i \leq w_s^i$.

To manage all frequent TAPs with respect to D, we use an edge-labeled ordered rooted tree, called a *TAPTree*, which stores the information about a frequent TAP in the path from the root to node, and which is a compact data structure similar to FPTree presented by Han et al. [7].

```
Algorithm FMF_TAPs
Input: Set D of tree-structured documents, minimum support σ (0 < σ ≤ 1)
       and set Q of stop words.
Output: The set F of all maximal σ-frequent TAPs with respect to D.

begin
    F_Word:={(w, ANS_D(w)) | w ∈ W(D)-Q, w is σ-frequent with respect to D};
    F:=Make_2Tree(F_Word, σ);
    Create the initial TAPTree T from F;
    Let H be the set of all leaves of T;
    while H ≠ ∅ do (T, H):= Expand_TAPTree(T, H, F, σ);
    Let F be the set of all maximal σ-frequent TAPs obtained from T;
    return F
end.
```

Figure 4. Algorithm FMF_TAPs

```
Procedure Make_2Tree
Input: Set F_Word of pairs consisting of words and lists of nodes,
       and minimum support σ (0 < σ ≤ 1).
Output: Set F of triplets of frequent 2-trees and a set of pairs of nodes.

begin
    for each (w, k_w) in F_Word do
      for each (p, k_p) in F_Word such that w < p do
        begin
          tmp:=∅;
          for each node x in k_w and each node y in k_p do
            begin
              Create a 2-tree t over (w, p) identified with path from x to y, if exists;
              if (t, t, list) in tmp with y not in {b | (x, b) in list} then list:=list∪ {(x, y)};
              else if t not in {s | (s, list) in tmp} then tmp:=tmp∪ {(t, t, {(x, y)})};
            end;
          for each (t, t, list) in tmp do if} t is σ-frequent then F:= F ∪ {(t, t, list)};
        end;
    return F
end;
```

Figure 5. Procedure Make_2Tree

A TAPTree *T* for *D* satisfies the following conditions. For edge *e* of *T*, let $((v_0, v_1), (v_1, v_2), \ldots, (v_k, v_{k+1}))$ be the path from the root v_0 to the child node v_{k+1} of *e*.

(1) Each edge *e* is labeled with a triplet of 2-tree t_{k+1}, tap *s*, and set $CPS_D(s)$ if and only if *s* is a frequent TAP with respect to *D*, and CPP $< t_1, t_2, \ldots, t_{k+1} >$ appears in *s* such that, for each i $(1 \le i \le k)$, t_i is a 2-tree in label of edge (v_i, v_{i+1}).

(2) There is a pointer from leaf v_e to internal node v_f if and only if α_e is a sub-CPP of α_f, and TAP in the label of the edge (u_e, v_e) is a subtree pattern of TAP in the label of the

edge (u_f, v_f), where α_e and α_f are CPPs on the paths from the root to v_e and to the node v_f, respectively.

```
Procedure Expand_TAPTree
Input: Tree T, set H of leaves in T, set F of triplets of 2-trees, TAPs and set of lists of
       nodes, and minimum support σ (0 < σ ≤ 1).
Output: Pair (T, H') of an expanded tree T and a set H' of leaves in T.

begin
H' = ∅;
for each node v in H do
  begin
    Let p_v be the parent of v;
    Let (t_v, s_v, CPS_D(s_v)) be the label of the edge (p_v, v);
    for each (x_1, x_2, ..., x_k) in CPS_D(s_v) do
      begin
        tmp_v := ∅;
        for each (t, s, list) in F such that x_k in {a | (a, b) in list} do
          for each TAP s' expanded from s_v do
            if (t, s', tmp_list) in tmp_v then
              tmp_list := tmp_list ∪ {(x_1, x_2, ..., x_k, b) | (x_k, b) in list};
           else tmp_v := tmp_v ∪ {(t, s', list)};
        for each (t, s, tmp_list) in tmp_v do
          if s is σ-frequent then
            begin
              append a new child w labeled with (t, s, list) to the leaf v of T;
              H' := H' ∪ {w}
            end
      end
  end
return (T, H')
end;
```

Figure 6. Procedure Expand_TAPTree

Consider the TAPTree T for D in Figure 3. T indicates that $s1$, $s2$, $s3$, $s4$, $s6$, and $s7$ are frequent TAPs with respect to D, ti is equal to si for each i ($1 \leq i \leq 3$), <$t1$, $t2$> appears in $s4$ and $s5$, <$t2$, $t3$> appears in $s6$, and <$t1$, $t2$, $t3$> appears in $s7$. Since <$t2$, $t3$> is a sub-CPP of <$t1$, $t2$, $t3$>, <$t2$, $t3$> is not maximal. In the same way, if $s6$ is a subtree pattern of $s7$, $s6$ is not maximal. By modifying the algorithms in [14] to construct a TAPTree, we can present an algorithm for solving maximal frequent TAPs problem. In Figures 4-6, when a set D of documents, a minimum support $0 < \sigma \leq 1$ and a set Q of stop words are given as inputs, we give an algorithm ***FMF_TAPs*** which outputs the set of all maximal σ-frequent TAPs with respect to D. This algorithm is based on a level-wise search strategy with respect to the length of a CPP. At line 1, the set of all σ-frequent words with respect to D and the appearance node set of w in D for each σ-frequent word w are stored in the constructed data structure ``trie''. At line 3, an edge-labeled ordered rooted tree T, with a depth of 1 is obtained by appending a new child v labeled with (t, s, *list*) to the root of T for each element (t, s, *list*) in F constructed at line 2. At line 5, while there exist leaves in T, which can be expanded, T is revised by appending new nodes to the leaves. From Fact 1 and Fact 2, we can show that *FMF_TAPs* outputs correctly a TAPTree for D.

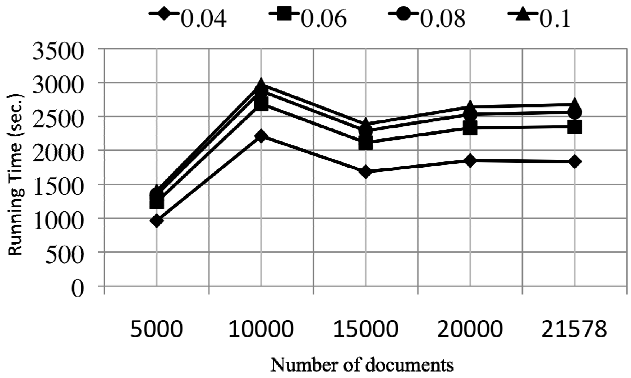


Figure 7. Running Times of FMF_TAPs

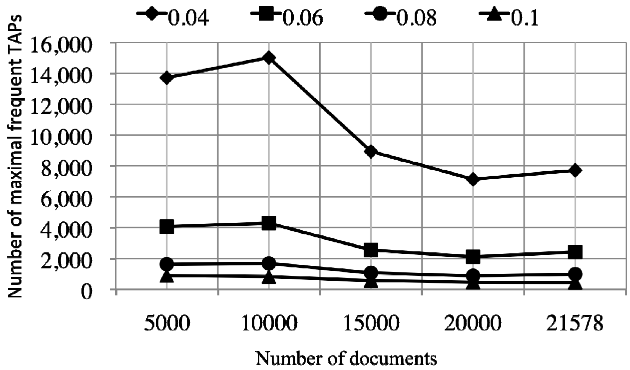


Figure 8. Numbers of frequent maximal TAPs enumerated by FMF_TAPs

Since the depth of the constructed TAPTree T is less than $|F(D)|$, we can easily show the termination of *FMF_TAPs*. Let M and N be the numbers of σ-frequent words in D and the extracted σ-frequent TAPs, respectively. Let n be the number of nodes of all OEM trees, each of which corresponds to a document in D. Let k be the maximum length of found CPPs. Then, *Make_2Tree* and *Expand_Pattern_Tree* are executable in $O(kn^3M^2 \log n \log N)$ and $O(kn^3MN \log n \log N)$, respectively. Hence, given a set D of documents, a minimum support $0 < \sigma \leq 1$ and a set of stop words, *FMF_TAPs* can construct the TAPTree for D in $O(\max(kn^3M^2 \log n \log N, k^2n^3MN \log n \log N))$ time.

4.2 Experimental results

We implemented the algorithms in C++ on a PC running CentOS 5.2 with a 3.0 GHz Xeon E5472 processor and 32 GB of main memory. In the following experiments, as stop words, we chose symbols such as "-, +,…", numbers such as "0, 1, 2, …", pronouns such as "it, this, …", articles "a, an, the", and auxiliary verbs "can, may, …" and so on. We applied these algorithms to the Reuters-21578 text categorization collection in [9], which has 21,578 SGML documents and is about 28.0 MB, in cases of each minimum support in {0.04, 0.06, 0.08, 0.10} and each number of documents in {5000, 10000, 15000, 20000, 21578}.

Figure 7 shows the running times of *FMF_TAPs* in the experiments. Each running time is the time to construct the TAPTree, which stores the information of all maximal frequent TAPs.

To precisely evaluate performance of our algorithms, running time contains no reading time of input documents. Even if minimum support is 0.04, the running time is less than 2,300 sec. In every minimum support, the running time is proportional regardless of the number of input documents. This shows the robustness of our algorithms.

Figure 8 shows the numbers of all maximal frequent TAPs in each experiment. When the minimum support was 0.04, our algorithm enumerated 15,018 maximal frequent TAPs within 2,211 sec, when 10,000 input documents were given. This shows the good performance of our algorithm. This is due to the data structure, TAPTree, which our algorithm uses.

5. Generating XSLT stylesheets based on CPP

XSLT (eXtensible Style sheet Language Transformation) [4] is the W3C recommendation for an XML style sheet language and is an XML-based language used for the transformation of XML documents into other XML documents such as HTML documents and RTF documents. Documents written with XSLT are called *XSLT stylesheets*. Software that transforms a given XML document into other XML document by following a given XML stylesheet is called an *XSLT processor*.

For a TAP t in which a CPP $\alpha=<t_1, t_2, \ldots, t_m>$ appears and has a matching function π of α to t, the *root* of α with respect to t is the root r_i of t_i such that $\pi(r_i)$ is the root of t and for any j ($1 \leq j < i$), $\pi(r_j)$ is not the root of t. Number i is called a *key number* of α. We note that the root and key number of α do not depend on TAP in which α appears.

Badica et al. [3] presented an approach for the efficient implementation of L-wrappers using XSLT transformation language, which is a standard language for processing XML documents. In a similar way as [3], we present an algorithm for a wrapper based on CPP using XSLT transformation language, shown in Figure 9. In algorithm *GEN_CPP_WRAPPER*, *GEN-FIRST-TEMPLATE*, *GEN-TEMPLATE-WITH-VAR*, *GEN-TEMPLATE-NO-VAR*, and *GEN-LAST-TEMPLATE* procedures, described precisely in [3], are as follows. The *GEN-FIRST-TEMPLATE* procedure appends the first template rule to the stylesheet ST_α. The *GEN-TEMPLATE-WITH-VAR* and the *GEN-TEMPLATE-NO-VAR* procedures append a template rule for paths, which are second and third arguments, respectively, to ST_α depending on whether or not the parent of the start node of the path has attributes. The *GEN-LAST-TEMPLATE* procedure appends the last template rule to ST_α. The constructing part of this rule fully instantiates the returned tree fragment, thus stopping the transformation process of the OEM tree of the input document.

As experiments, we produced XSLT stylesheets by applying our algorithm *GEN_CPP_WRAPPER* to CPPs. Figure 11 shows the XSLT stylesheet that was generated by applying *GEN_CPP_WRAPPER* to α in Figure 10. We used Xalan [13] as an XSLT processor. Xalan is a widely used open source software library from the Apache Software Foundation that implements the XSLT XML transformation language and the XPath language. The XML document in Figure 12 is an XML document that was produced by applying Xalan to the sixth document in the Reuters-21578 text categorization collection [9] and the XSLT stylesheet in Figure 11.

6. Conclusion and future works

Building on previous work, by extending a CPP to a tree pattern, we have defined a tree association pattern (TAP) over a set of words as a node-labeled ordered rooted tree that can represent structural features among nodes in which given words appear. Then, by modifying text-mining algorithms in [8, 14], we have presented text-mining algorithms for enumerating all maximal frequent TAPs from a set of tree-structured documents. Moreover, as one application of CPPs and TAPs, we have showed a wrapper based on CPP using XSLT transformation language, which translates XML documents into other XML documents.

To design more effective Internet search engines, we plan to develop our mining techniques based on CPPs and TAPs for mining multimedia data such as Web content with pictures, voice data, and movie data.

```
Algorithm GEN CPP WRAPPER
Input: CPP α=<t_1, t_2,..., t_m>
Output: Stylesheet ST_α

begin
 1. Let r be the root of α;
 2. Let k be the key number of α;
 3. GEN-FIRST-TEMPLATE(ST_α, [rn(t_k), r]);
      // [u, v] denotes the path from u to the parent of v;
 4. V=∅;
 5. for ( i=k; i<m; i++) do begin
 6.    if the label of the parent of rn(t_i) is a tag having attribute then
 7.      begin
 8.        var =GEN-VAR(rn(t_i));
 9.        GEN-TEMPLATE-WITH-VAR(ST_α, [ln(t_i+1), rt(t_i+1)], [rn(t_i+1), rt(t_i+1)], var, V);
             // rt(t_i+1) denotes the root of 2-tree t_i+1
10.        V=V ∪ {var};
11.      end
12.    else GEN-TEMPLATE-NO-VAR(ST_α, [ln(t_i+1), rt(t_i+1)], [rn(t_i+1), rt(t_i+1)], V);
13. end;
14. for ( i=k; i > 1; i--) do begin
15.    if the label of the parent of rn(t_i-1) is a tag having attribute then
16.      begin
17.        var =GEN-VAR(rn(t_i-1));
18.        GEN-TEMPLATE-WITH-VAR(ST_α, [rn(t_i-1), rt(t_i-1)], [ln(t_i-1), rt(t_i-1)], var, V);
19.        V=V ∪ {var};
20.      end
21.    else GEN-TEMPLATE-NO-VAR(ST_α, [rn(t_i-1), rt(t_i-1)], [ln(t_i-1), rt(t_i-1)], V);
22. end;
23. GEN-LAST-TEMPLATE(ST_α, V);
24. return ST_α
end.
```

Figure 9. Algorithm GEN_CPP_WRAPPER

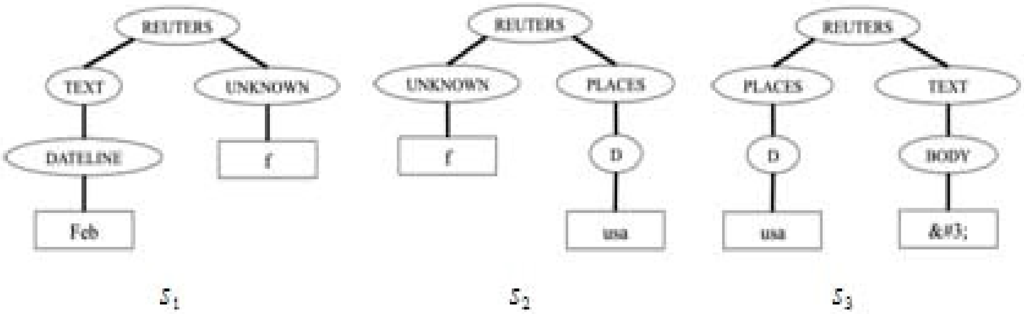


Figure 10. CPP α=<s_1, s_2, s_3>

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
  <result>
    <xsl:apply-templates select="/REUTERS/TEXT/DATALINE" mode="selw1"/>
  </result>
</xsl:template>

<xsl:template match="*" mode="selw1">
  <xsl:variable name="var0"><xsl:value-of select="."/> </xsl:variable>
  <xsl:apply-templates mode="selw2" select="parent::*/parent::*/UNKNOWN">
    <xsl:with-param name="var0" select="$var0"/>
  </xsl:apply-templates>
          ...
  <xsl:template match="*" mode="display">
   <xsl:param name="var0"/>
          ...
   <xsl:value-of select="."/>
   </xsl:variable>
   <tuple>
     <d0><xsl:value-of select="$var0"/></d0>
     <d1><xsl:value-of select="$var1"/></d1>
     <d2><xsl:value-of select="$var2"/></d2>
     <d3><xsl:value-of select="$var3"/></d3>
   </tuple>
  </xsl:template>
</xsl:stylesheet>
```

Figure 11. Generated XSLT stylesheet. We omit part of this stylesheet, as it is too long to describe in full

```
<result>
  <tuple>
    <d0>PORTLAND, Ore., Feb 26 -</d0>
    <d1>&#5; ... reute d f BC-RED-LION-INNS-FILES-P 02-26 0082</d1>
    <d2>usa</d2>
    <d3>Red Lion Inns Limited Partnership said
        ...
      dlr mortgage loan, will be used to finance its planned
      acquisition of 10 Red Lion hotels.  Reuter &#3;
    </d3>
  </tuple>
</result>
```


Figure 12. Produced XML document. We omit part of this document, as it is too long to describe in full.

Acknowledgements

We would like to express our deepest gratitude to Miss Yoko Fukuyama and Miss Yumi Ae, whose enormous support and insightful comments were invaluable for this work.

References

[1] S. Abiteboul, P. Buneman, and D. Suciu, Data on the Web: From Relations to Semistructured Data and XML, Morgan Kaufmann, 2000.

[2] T. Asai, K. Abe, S. Kawasoe, H. Arimura, H. Sakamoto, and S. Arikawa, “Efficient substructure discovery from large semi-structured data”, Proceedings of the Second SIAM International Conference on Data Mining (SDM-2002), pp. 158-174, 2002.

[3] A. Bădică, C. Bădică, and E. Popescu, “Implementing logic wrappers using xslt stylesheets”, Proceedings of the International Multi-Conference on Computing in the Global Information Technology (ICCGI-2006), pp. 31, 2006.

[4] J. Clark, ”XSL transformations (XSLT) version 1.0”, W3C recommendation. http://www.w3.org/TR/xslt, 1999.

[5] M. Fernandez and D. Suciu, “Optimizing regular path expressions using graph schemas”, Proceedings of the Fourteenth International Conference on Data Engineering (ICDE-98), pp. 14-23, 1998.

[6] G. Gonnet and R. Baeza-Yates, Handbook of Algorithms and Data Structures, Addison-Wesley, 1991.

[7] J. Han and M. Kamber, Data Mining: Concepts and Techniques, Morgan Kaufmann Publishers, 2001.

[8] K. Ishida, T. Uchida, and K. Kawamoto, “Extracting structural features among words from document data streams”, Proceeding of 19th Australian Joint Conference on Artificial Intelligence (AI-2006), pp. 332-341, 2006.

[9] D. Lewis, Reuters-21578 text categorization test collection, UCI KDD Archive, http://kdd.ics.uci.edu/databases/reuters21578/reuters21578.html, 1997.

[10] T. Miyahara, T. Shoudai, T. Uchida, K. Takahashi, and H. Ueda, “Discovery of frequent tree structured patterns in semistructured web documents”, Proceedings of 5th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD-2001), pp.47-52, 2001.

[11] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal, “Discovering frequent closed itemsets for association rules”, Proceedings of 7th International Conference on Database Theory (ICDT-1999), pp. 398-416, 1999.

[12] J. Pei, J. Han, and R. Mao, “Closet: An efficient algorithm for mining frequent closed itemsets”, 2000 ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD-2000), Springer-Verlag, pp. 21-30, 2000.

[13] Apache Xalan Project, Xalan, http://xalan.apache.org/

[14] T. Uchida, T. Mogawa, and Y. Nakamura, "Finding frequent structural features among words in tree-structured documents", Proceedings of the 8th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD-2004), Springer-Verlag, LNAI 3056, pp. 351--360, 2004.

[15] K. Wang and H. Liu, "Discovering structural association of semistructured data", IEEE Transactions on Knowledge and Data Engineering, 12:353--371, 2000.

[16] J. Yan, X. Han and A. Afshar, "Clospan: Mining closed sequential patterns in large datasets", Proceedings of the Third SIAM International Conference on Data Mining (SDM03), pp. 166-177, 2003.

Authors

Tomoyuki Uchida received the B.S. degree in Mathematics, the M.S. and Dr. Sci. degrees in Information Systems all from Kyushu University in JAPAN, in 1989, 1991 and 1994, respectively. Currently, he is an associate professor of Graduate School of Information Sciences, Hiroshima City University in JAPAN. His research interests include data mining from semi-structured data, algorithmic graph theory and algorithmic learning theory. He is a member of the Institute of Electronics, Information and Communication Engineers, and ACM.

Kayo Kawamoto received the B.S. and the M.S. degrees in Education from Tokyo Gakugei University in JAPAN, in 1989 and 1992, respectively. Currently, she is a research assistant of Graduate School of Information Sciences, Hiroshima City University in JAPAN. Her research interests include e-learning, educational evaluation, information education and text-mining. She is a member of Japan Society for Educational Technology, Japanese Society for Information and Systems in Education, and Japan Society for Science Education.