Tree Pattern Matching Algorithm Using a Succinct Data Structure

Yuko Itokawa,Masanobu Wada,Toshimitsu Ishii and Tomoyuki Uchida

***Abstract*—Two things are important in developing a fast, memory-efficient graph mining method that extracts characteristic graph structures from Web pages and other tree-structured data. One is tree patterns that can express the features of a graph structure and the other is data structures for tree patterns and for representing tree-structured data. In this paper, we first apply a depth-first unary degree sequence (DFUDS), which is one succinct data structure for an ordered tree, as a succinct data structure for tree patterns that express the features of a graph structure. We then propose a pattern matching algorithm that uses the DFUDS succinct data structure, to determine whether or not a given tree-structured data has features of tree pattern. We also implement the proposed algorithm on a computer and evaluate the algorithm by experiment. The results are reported and discussed.**

***Index Terms*—pattern matching algorithm, succinct representation, tree-structured data, edge-labeled ordered term tree.**

I. Introduction

In recent years, with the rapid progress in networks and information technology, Web documents and other such material that does not have a clear structure and is referred to as semi-structured data have become innumerous. Semi-structured data that has a tree structure is called tree-structured data, and can be represented by an ordered tree. To extract useful information from tree-structured data, it is necessary to extract tree patterns that are common to tree-structured data (i.e., ordered trees). Moreover, to present efficient tree mining tools, pattern matching algorithms for determining whether or not given tree-structured data has features represented by given tree pattern are necessary to be efficient. Suzuki et alia [13] have proposed a matching algorithm for determining whether or not an edge-labeled ordered tree is generated by substituting all structural variables in given edge-labeled ordered tree pattern with arbitrary edge-labeled ordered trees. The aim of this paper is to present more efficient pattern matching algorithm for edge-labeled ordered tree patterns than Suzuki's.

To reduce the memory required to store an ordered tree, succinct data structures for ordered trees have been proposed [2], [3], [4], [5], [7], [8], [9], [11]. As one succinct data structure for an edge-labeled ordered tree, Ferragina et alia proposed the **xbw** transform and a path search algorithm for an xbw transformed edge-labeled tree [4]. Using the xbw transform enables both compact storage of tree-structured data and fast path search. As a succinct data structure for an ordered tree, Benoit et alia proposed a depth-first unary degree sequence (**DFUDS**) representation [2]. The DFUDS representation uses a string of parentheses constructed by a depth-first traversal of all vertexes in which, if the index of a vertex is k, the k-th $\boxed{(}$ and its subsequent $\boxed{)}$ are output. By taking $\boxed{(}$ to be '0' and $\boxed{)}$ to be '1', the ordered tree representation can be handled as a bit string. Also proposed is supplementary data for an ordered tree represented by DFUDS that enables use of the $rank$ and $select$ operations to tour the tree in constant time. Almost of results about succinct data structures are from theoretical viewpoints. However, most recently, a few practical results are known [1].

In this paper, in order to represent structural features of tree-structured data, firstly we propose a DFUDS representation for an edge-labeled ordered trees based on a DFUDS representation of an ordered tree presented by Benoit et alia [2]. Then, we also propose an efficient matching algorithm for solving the membership problem for tree patterns having structural variables using the DFUDS representation of an edge-labeled tree as the data structure. The matching algorithm we describe here performs top down matching, but the method applied by Suzuki et alia performs bottom up. The results in this paper leads us to design fast and memory-efficient tree mining tools for tree-structured data.

This paper is organized as follows. In section II, we describe the tree structure that we deal with in this paper and the tree pattern which represents the structural features of edge-labeled ordered trees. In section III, we briefly describe the DFUDS representation for ordered trees proposed by Benoit et alia. In section IV, we formulate the membership problem for edge-labeled trees and propose a matching algorithm for solving it. In section V, the algorithm for solving the membership problem is implemented on a computer. The test results for the implementation are reported and discussed. Section VI concludes the paper.

II. Tree Patterns

Let Σ and χ denote finite alphabets with $\Sigma \cap \chi = \emptyset$. Let V_t be the set of vertexes and $E_t \subseteq V_t \times (\Sigma \cup \chi) \times V_t$ be the set of edges. For edge $e = (u, a, v) \in E_t$, let character $a \in (\Sigma \cup \chi)$ be the **edge label** of e. In particular, when $a \in \chi$, a is a **variable label** and e is a **variable**. For variable $e = (u, x, v)$, u is the **parent port** of e and, v is a **child port** of e. $t = (V_t, E_t)$ is called a **edge-labeled term tree** if t has only one vertex u whose coming degree is 0 and $(V_t, \{\{u, v\} \mid (u, a, v) \in E_t\})$ is a rooted tree having u as its root. A edge-labeled term tree whose all children of all internal vertexes are ordered is called an **edge-labeled ordered term tree**. The ordered term trees of concern in this paper are assumed to have all mutually different variables. Trees that have no variables are simply **edge-labeled ordered trees**. In Fig. 1, edge-labeled ordered term tree p and edge-labeled ordered tree t, g_0, g_1, g_2 are shown.

Y. Itokawa is with the Faculty of Psychological Science, Hiroshima International University, 555-36 Kurose-Gakuendai, Higashi-Hiroshima, Hiroshima Japan e-mail: y-itoka@he.hirokoku-u.ac.jp.
M. Wada, T. Ishii and T. Uchida are with Hiroshima City University.

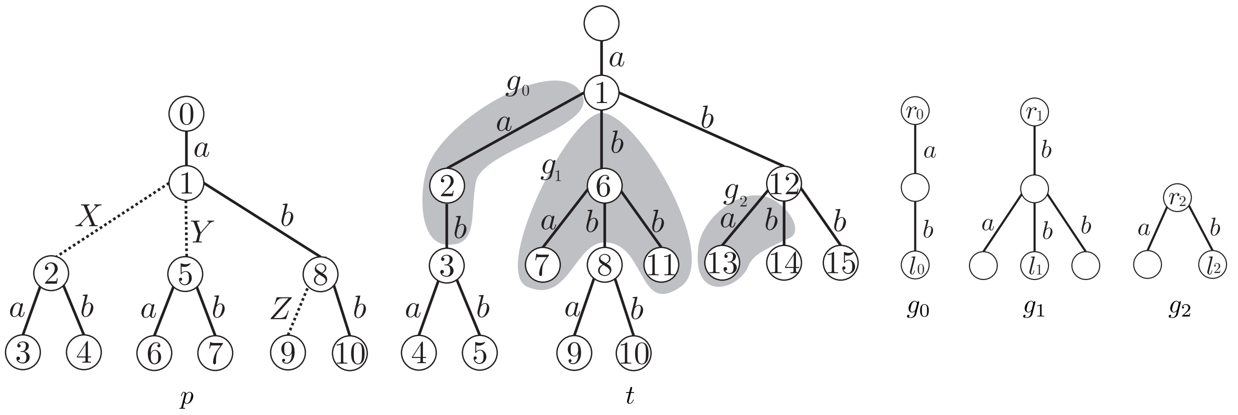

Fig. 1. Edge-labeled ordered term tree p and edge-labeled ordered trees $g_0,\ g_1,\ g_2,\ t \cong p\theta$, where $\theta = \{X := [g_0, (r_0, l_0)], Y := [g_1, (r_1, l_1)], Z := [g_2, (r_2, l_2)]\}$

For term tree t and its vertex u, the tree consisting of u and all of its descendants is called a **subtree** of t and is denoted as $t[u]$. In the same way, for edge $e = (u, a, v)$ of t, $t[e]$ denotes the tree consisting of e and $t[v]$. For edge-labeled ordered term trees $t = (V_t, E_t)$ and $f = (V_f, E_f)$, if bijection $\pi : V_t \to V_f$ that satisfies the following conditions (1)-(3) exists, then t and f are **isomorphic**, which is denoted as $t \cong f$. For two children u' and u'' of vertex u of edge-labeled ordered term tree h, $u' <^h_u u''$ denotes that in the ordering of the children of u, u' is lower than u''.

(1) For any $a \in \Sigma$, if and only if $(u, a, v) \in E_t$, $(\pi(u), a, \pi(v)) \in E_f$.
(2) If and only if there is $x \in \chi$ for which $(u, x, v) \in E_t$, there is $y \in \chi$ for which $(\pi(u), y, \pi(v)) \in E_f$.
(3) If and only if for vertexes u of t and the two children u' and u'' of u, $u' <^t_u u''$, $\pi(u') <^f_{\pi(u)} \pi(u'')$.

Let variable label $x \in \chi$ and let r be the root of an edge-labeled ordered term tree g, and l the leaf of g. Then, $x := [g, (r, l)]$ is the **binding** of x and a finite set of variable label bindings is a **substitution**. A new edge-labeled ordered term tree f can be obtained by applying substitution $\theta = \{x_0 := [g_0, (r_0, l_0)], \ldots, x_n := [g_n, (r_n, l_n)]\}$ to an edge-labeled ordered term tree $g = (V_g, E_g)$ in the following way. A new edge-labeled ordered term tree f can be obtained by regarding w_0 to be the same as r_i and w_1 to be the same as l_i in edge $e = (w_0, x_i, w_1)$ that has variable label x_i, and exchanging them for each $0 \le i \le n$. The resultant edge-labeled ordered term tree f is denoted by $g\theta$. Edge-labeled ordered tree t in Fig. 1 can be obtained by applying substitution $\theta = \{X := [g_0, (r_0, l_0)], Y := [g_1, (r_1, l_1)], Z := [g_2, (r_2, l_2)]\}$ to edge-labeled ordered term tree p.

III. Succinct Data Structures for Tree Patterns

We explain the basic data structure for dealing with ordered term trees. In this paper, a word RAM with a word length of $\Theta(\log n)$ bits is used as the computation model. For sequence S of length n on alphabet $\mathcal{A}$, denote the i-th character $(0 \le i \le n-1)$ in S as $S[i]$. For each i, j $(0 \le i < j \le n-1)$, denote the sub-sequence from the i-th character to the j-th character of S as $S[i \ldots j]$. For sequence S of length n on alphabet $\mathcal{A}$, character c and natural number i $(0 \le i \le n-1)$, define a $rank$ function and a $select$ function as follows.

(1) $rank_c(S, i)$ returns the number of occurrences of character c in sub-sequence $S[0 \ldots i]$.
(2) $select_c(S, i)$ returns the position of the i-th character c from the beginning of S.

When the context makes it clear, S is omitted.

There are many succinct data structures for fast computation of the $rank$ function and the $select$ function [5], [12]. One is the fully indexable dictionary **(FID)** [10]. The FID is an $n + o(n)$ bit data structure that allows the $rank$ and $select$ functions to be calculated in constant time for a bit sequence on $\{0, 1\}$ of length n with a word RAM model. The FID has a bit sequence of length n and a supplementary data structure. The supplementary data structure is a table that stores responses for all inputs in advance, considering the fine division of the character sequence as the problem of small size. The responses for any bit sequence of length $\frac{1}{2}\log n$ in a bit array of length n are at most $2^{\frac{1}{2}\log n} = \sqrt{n}$, so all responses can be stored in a $(\sqrt{n}\cdot\mathrm{polylog}(n))$bit table, which can be searched in constant time. That area can be reduced when there are few '1' in S. For a character sequence that contains m '1', there is a $\log\binom{n}{m} + O(n \log\log n/\log n) = m \log\frac{n}{m} + \Theta(m) + O(n \log\log n/\log n)$ bit data structure for which the $rank$ and $select$ functions can be calculated in constant time. If $m = O(n/\log n)$ bits, the size of that data structure becomes $O(n \log\log n/\log n)$.

The $rank$ and $select$ functions can be expanded to a function for obtaining the number of occurrences and positions of a character sequence pattern. Consider the representation of an ordered tree of n vertexes that allows execution of the $rank$ and $select$ functions in constant time. There exist $\binom{2n+1}{n}/(2n+1)$ ordered trees of n vertexes, so the information theoretical lower bound of the data structure size is $2n - \Theta(\log n)$ bits. Many data structures that are asymptotically consistent with that lower bound have been proposed [5], [8].

One such data structure proposed by Benoit et alia [2] is the depth-first unary degree sequence (**DFUDS**) representation, which is a succinct data structure for ordered trees. The DFUDS representation for an ordered tree t of m edges is defined inductively as follows. The DFUDS representation of the tree consisting only one vertex is $\boxed{(}\,\boxed{)}$. The DFUDS representation of a t that has k subtrees $t_1, \ldots, t_k$ is a sequence of parentheses constructed by concatenating $k+1$ $\boxed{(}$, one $\boxed{)}$, k DFUDS representations of $t_1, \ldots,$ and t_k in

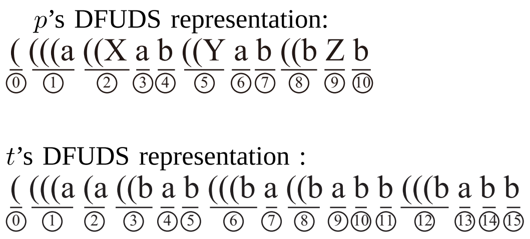
p's DFUDS representation:
((((a ((X a b ((Y a b ((b Z b
⓪ ① ② ③④ ⑤ ⑥⑦ ⑧ ⑨ ⑩
t's DFUDS representation :
((((a (a ((b a b (((b a ((b a b b (((b a b b
⓪ ① ② ③ ④⑤ ⑥ ⑦ ⑧ ⑨⑩⑪ ⑫ ⑬⑭⑮

Fig. 2. Edge-labeled DFUDS representations for p and t

TABLE I
MISMATCH CASES OF $T[j]$ AND $P[i]$

	$Case1$	$Case2$	$Case3$	$Case4$	$Case5$
$P[i]$	EL	VL	OP	VL	EL
$T[j]$	EL	EL	EL	OP	OP

EL, VL and OP denote edge label in Σ, variable label in χ and open parenthesis $\boxed{(}$, respectively.

this order (here, the initial $\boxed{(}$ of the DFUDS representation of each subtree has been removed). The DFUDS representation is a sequence of balanced parentheses of length $2m$.

A DFUDS representation can be regarded as a bit sequence by replacing $\boxed{(}$ with '0' and $\boxed{)}$ with '1'. Representing an ordered tree with a bit sequence DFUDS representation makes it possible to execute the $rank$ and $select$ functions in constant time. Furthermore, a supplementary data structure that allows execution of the following operations in constant time by using $rank$ and $select$ functions on DFUDS representation P has been proposed[9], [2].

(1) $findclose(x)$ return the position of the closing parentheses for an opening parenthesis in $P[x]$.
(2) $findopen(x)$ return the position of the opening parentheses for a closing parenthesis in $P[x]$.
(3) $enclose(x)$ return the position of the opening parentheses of the pair that most tightly encloses $P[x]$.

The three operations above can be used to express the following operations for touring the ordered tree t.

(4) $degree(x)$ return the number of children of vertex x.
(5) $child(x, i)$ return the position of the i-th child from the left of vertex x.
(6) $subtree(x)$ return the pair of the start and the end positions of the interval representing the subtree for which vertex x is the root.
(7) $id(x)$ return the visiting order of vertex x.
(8) $label(x)$ return the position of the closing parenthesis of the $id(x)$-th vertex.

$child$ takes $O(i)$ time, but the other operations can be executed in constant time using an $o(n)$ bit supplementary data structure [3], [11], [2]

The DFUDS representation proposed by Munro et alia [2] is a data structure for an ordered tree with no edge labels. Therefore, we consider a DFUDS representation for an edge-labeled ordered term tree. In the DFUDS representation of Munro et alia, $\boxed{)}$ must occupy the rightmost position for each vertex. Therefore, a hash function that returns the edge label that corresponds to that vertex for each $\boxed{)}$ makes a DFUDS representation of an edge-labeled ordered tree possible. DFUDS representations for edge-labeled ordered term tree p and edge-labeled ordered tree t are shown in Fig. 2. For convenience in this example, a hash function that returns the edge labels that correspond to all of the $\boxed{)}$ has been executed.

The sequence of parentheses that is a DFUDS representation can be interpreted as the result of visiting all vertexes in pre-order and outputting k $\boxed{(}$ for each vertex whose index is k the following one $\boxed{)}$ [2]. Hence, the following theorem obviously holds.

theorem 1: Given edge-labeled tree t of m edges, the edge-labeled DFUDS representation for t can be computed with $O(m)$.

IV. MEMBERSHIP PROBLEM FOR EDGE-LABELED TREES

$\mathcal{OT}_\Sigma$ denotes a set of edge-labeled ordered trees on Σ and $\mathcal{OTT}_{\Sigma\cup\chi}$ denotes a set of edge-labeled ordered term trees on $\Sigma \cup \chi$. In this section, we formulate the membership problem for $\mathcal{OTT}_{\Sigma\cup\chi}$ and propose a matching algorithm that uses a DFUDS representation as a data structure for solving that problem. Given edge-labeled ordered term tree $p \in \mathcal{OTT}_{\Sigma\cup\chi}$ and edge-labeled ordered tree $t \in \mathcal{OT}_\Sigma$, the problem of determining whether or not there exists a substitution θ for which $t \cong p\theta$ is referred to as the membership problem for $\mathcal{OTT}_{\Sigma\cup\chi}$.

Membership problem for $\mathcal{OTT}_{\Sigma\cup\chi}$

Instance:	Edge-labeled ordered term tree $p \in \mathcal{OTT}_{\Sigma\cup\chi}$ and edge-labeled ordered tree $t \in \mathcal{OT}_\Sigma$
Problem:	Determine if there exists a θ for which $t \cong p\theta$.

A matching algorithm Pattern_Matching that determines whether or not there exists a substitution θ for which $t \cong p\theta$ given edge-labeled ordered term tree $p \in \mathcal{OTT}_{\Sigma\cup\chi}$ and edge-labeled ordered tree $t \in \mathcal{OT}_\Sigma$, $t \cong p\theta$ is listed in Algorithm 1.

By treating the DFUDS representation of an edge-labeled tree as a data structure, we can consider pattern matching for the character sequence that consists of $\boxed{(}$ and edge labels. We therefore execute the pattern matching from the beginning of the DFUDS representation in the touring order from the root of the tree. Given the DFUDS representation $P[0 \ldots m-1]$ of edge-labeled ordered term tree p as the pattern and the DFUDS representation $T[0 \ldots n-1]$ of edge-labeled ordered tree t as the text, matching begins with the first characters of p and t. If characters $P[i]$ and $T[j]$ match, matching is performed for the next characters. If characters $P[i]$ and $T[j]$ do not match, the following procedure is performed (line 4 - line 19). The five pattern cases for which $P[i]$ and $T[j]$ do not match are shown in Table I. Mismatch for which the condition of line 4 of the algorithm applies relates to Case 1 in Table I, so edge-labeled ordered term tree p and edge-labeled ordered tree t are not the same. Accordingly, Pattern_Matching algorithm returns $false$. The first half of the condition of line 6 relates to Case 5 in Table I; the second half of the condition is the case in which parent and child have sequential variables in edge-labeled ordered term tree p. In that case, it is determined whether on not subtrees

Algorithm 1 Pattern_Matching

```
Require: DFUDS representations $P[0 \dots m-1]$ and $T[0 \dots n-1]$ of ordered term tree $p$ and ordered tree $t$, respectively.
Ensure: true if there exists a substitution $\theta$ with $t \cong p\theta$, otherwise false.
 1: $i \leftarrow 0$, $j \leftarrow 0$
 2: while $i < m$ and $j < n$ do
 3:   if $P[i] \neq T[j]$ then
 4:     if $P.label(i) \in \Sigma$ and $T.label(j) \in \Sigma$ {/* Case 1 */} then
 5:       return false
 6:     else if $P.label(i) \in \Sigma$ and $T[j]$ is an open parenthesis {/* Case 5 */} then
 7:       if (Pattern_Matching($P[P.subtree(i)]$, $T[T.subtree(j)]$) ==false) then
 8:         return false
 9:       else
10:         $i = P.subtree(i).second$
11:         $j = T.subtree(j).second$
12:       end if
13:     else
14:       if (for any $k \in [0 \dots T.degree(j)]$, Pattern_Matching($P[P.subtree(i)], T[T.subtree(T.child(j,k))]$) ==false)
          {/* Cases 2,3,4 */} then
15:         return false
16:       else
17:         $i = P.subtree(i).second$
18:         $j = T.subtree(j).second$
19:       end if
20:     end if
21:   end if
22:   $i++$, $j++$
23: end while
24: if $m - i > 0$ or $n - j > 0$ then
25:   return false
26: end if
27: return true
```

$p[P.id(i)]$ and $t[T.id(j)]$ match. If they do not match, then Pattern_Matching algorithm returns $false$. Line 13 relates to Cases 2, 3 and 4 in Table I, where, subtree $p[P.id(i)]$ is not found in subtree $t[T.id(j)]$ and Pattern_Matching algorithm returns $false$. If either character sequence p or t has not been examined to the end (line 24) when the while statement ends, then there exists no substitution θ for which $t \cong p\theta$. If none of the above cases hold, then there exists a θ such that $t \cong p\theta$.

When DFUDS representations of edge-labeled ordered term tree p and edge-labeled ordered tree t in Fig. 2 are given, we illustrate the process of Pattern_Matching algorithm. Since $P[0 \dots 5]$ and $T[0 \dots 5]$ are same, line 4 of Pattern_Matching algorithm is firstly executed at $i = 6$ and $j = 6$. Since $P[6] = \boxed{(}, T[6] = a \in \Sigma$, that is Case 3, line 13 is executed. Then, since there exists the subtree $t[3]$ which is isomorphic to the subtree $p[2](= p[P.id(6)])$, after lines 17 and 18 are executed, $i = 9$ and $j = 11$ are obtained. Next, $p[10 \dots 11]$ and $t[12 \dots 13]$ are same, and $P[12] = Y \in \chi$ and $T[6] = \boxed{(}$, that is Case 2. Hence, line 13 is executed again. Then, since there exists the subtree $t[8]$ which is isomorphic to the subtree $p[5](= p[P.id(12)])$, after lines 17 and 18 are executed, $i = 14$ and $j = 22$ are obtained. Next, since $P[17] = b \in \Sigma$ and $T[25] = \boxed{(}$, that is Case 5, line 6 is executed. when the subtree $p[(P.id(P.parent(17)), b, P.id(17))](= p[(1, b, 8)])$ of p and the subtree $t[(T.id(T.parent(25)), b, T.id(25))](= t[(1, b, 12)])$ of t are given, Pattern_Matching returns *true*. Hence, after lines 10 and 11 are executed, $i = 19$ and $j = 29$ are obtained. Finally, Pattern_Matching algorithm exits **while** loop, returns $true$, and terminates.

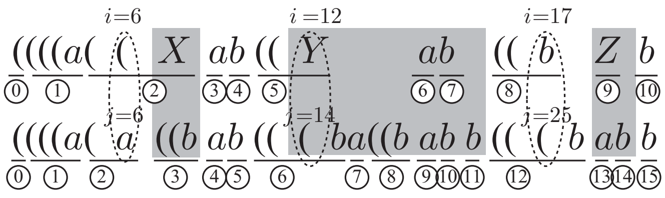

Fig. 3. Illustration of process of Pattern_Matching algorithm when DFUDS represents of p and t are given.

The following theorem is hold.

theorem 2: The membership problem for $\mathcal{OTT}_{\Sigma \cup \chi}$ can be computed in $O(mn)$ time.

Proof: The Pattern_Matching algorithm for solving the membership problem for $\mathcal{OTT}_{\Sigma \cup \chi}$ is presented in Algorithm 1. Let the respective DFUDS representations for edge-labeled ordered term tree p and edge-labeled ordered tree t be $P[0 \dots m-1]$ and $T[0 \dots n-1]$, and let i and j be natural numbers that satisfy $0 \leq i < m$ and $0 \leq j < n$. Consider an $m \times n$ table that holds information on whether or not the subtree whose root is a child vertex of the edge that corresponds to the position $P[i]$ matches the subtree whose root is a child vertex of the edge that corresponds to the position $T[j]$. The Pattern_Matching algorithm can be understood as a repeated process of referring to the certain parts of the table to determine the uncertain parts of the table. In other words, the table is filled in by repeating the part of the Pattern_Matching algorithm from line 2 to line 23, incrementing i and j. Thus, whether or not the subtree whose root is a child vertex of the edge that corresponds to the position of $P[0]$ (namely, p) matches the subtree whose root is a child vertex of the edge that corresponds to the position of $T[0]$ (namely, t) can be calculated in $O(nm)$ time. ■

V. Experiment and Discussion

We implemented the algorithm for generating a DFUDS representation from an edge-labeled ordered term tree described in section III and the algorithm described in section IV on a computer. In this section, we describe the experimental setup, present the results, and discuss what was learned.

The algorithms listed below were implemented in C++ on a computer equipped with a 3.16 GHz Intel XEON X5460 processor, main memory of 4.00 GB and running the Microsoft Windows Vista SP1 operating system.

1) Algorithm for generating the DFUDS representation of an edge-labeled tree.
2) Pattern_Matching algorithm, described in section IV, for determining whether or not there exists a substitution θ for which $t \cong p\theta$, given DFUDS representations of edge-labeled ordered term tree p and edge-labeled ordered tree t .

To avoid effect of the edge label count, we set the edge label count to 1. Let a be an edge label in Σ. We artificially created a **pattern collection** $D^r_{\Sigma\cup\mathcal{X}}(m)$ of one hundred each of edge-labeled term trees in $\mathcal{OTT}_{\Sigma\cup\mathcal{X}}$ that has the edge label a, m edges including rm variables, and maximum degree 5, and a **data collection** $D_\Sigma(n)$ of one hundred each of edge-labeled trees in $\mathcal{OT}_\Sigma$ that has the edge label a, n edges and maximum degree 5. We remark that r denotes a **ratio** of the variable count for the edge count in an edge-labeled term tree in $D^r(m)$. From the design of Pattern_Matching algorithm, since Pattern_Matching algorithm may return $false$ as soon as finds mismatch positions of given edge-labeled term tree p and edge-labeled tree t, we can see that, in general, the execution time of Pattern_Matching algorithm may not depend on the edge count of p and t even if quit huge data collections are used. Hence, we artificially created data collection $D_\Sigma(n)$ from $D^r_{\Sigma\cup\mathcal{X}}(m)$ in the following way. For each edge-labeled term tree p in $D^r_{\Sigma\cup\mathcal{X}}(m)$, we add an edge-labeled tree t in $\mathcal{OT}_{\{a\}}$ so that t has edge counts of n and there exists a substitution θ with $t \cong p\theta$ to $D_\Sigma(n)$.

First of all, for each $n \in \{1000, 2000, 3000, 4000, 5000, 6000, 7000, 8000, 9000, 10000\}$, DFUDS representations for 100 trees in data collections $D_\Sigma(n)$ were generated and the average generation time was obtained. Those results are shown in Fig. 4. An equation that approximates the average execution time y of the DFUDS representation generating algorithm for an edge-labeled ordered tree is $y = 7.83m \times 10^{-1}$. That is, we can see that the DFUDS representation generation time varies linearly with the edge count. By the above experiments, we can demonstrate that theorem 1 holds.

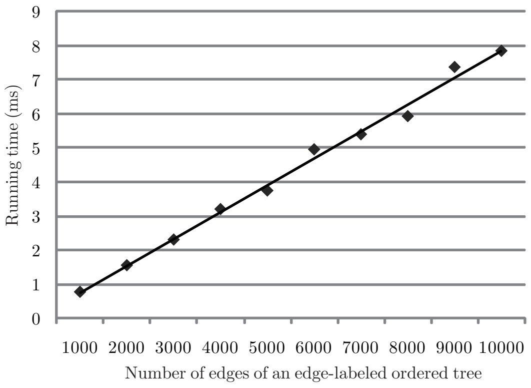

Fig. 4. Running times of the DFUDS representation generating algorithm

In order to show efficiency of Pattern_Matching algorithm, we evaluated Pattern_Matching algorithm by experiments using data collections and pattern collections. Let t be an edge-labeled tree selected in $D_\Sigma(10000)$. Secondly, for each $m \in \{100, 200, 300, 400, 500, 600, 700, 800, 900, 1000\}$, when t and an edge-labeled term tree p in $D^{0.1}_{\Sigma\cup\mathcal{X}}(m)$ are given, the average execution time of Pattern_Matching algorithm was shown in Fig.5. When the edge count of given edge-labeled tree is fixed, we can demonstrate that theorem 2 holds, because the matching time varies linearly with the edge count of given edge-labeled term tree.

Conversely, let p be an edge-labeled term tree in $D^{0.1}_{\Sigma\cup\mathcal{X}}(100)$. Thirdly, by varying edge counts n from 1000 to 10000, when p and edge-labeled tree in $D_\Sigma(n)$ are given, the average execution time of Pattern_Matching algorithm was shown in Fig.6. From Fig.6, we can also demonstrate that theorem 2 holds.

For each $d \in \{0.10, 0.15, 0.20, 0.25, 0.30, 0.35, 0.40, 0.45, 0.50\}$, we artificially created a pattern collection $D^{0.5}_{\Sigma\cup\mathcal{X}}(100, d)$ from $D^{0.5}_{\Sigma\cup\mathcal{X}}(100)$ as follows. For each edge-labeled term tree p in $D^{0.5}_{\Sigma\cup\mathcal{X}}(100)$, by replacing $(0.5 - d)m$ variables with edges whose labeled with a we added the resultant edge-labeled term tree having dm variables to $D^{0.5}_{\Sigma\cup\mathcal{X}}(100, d)$. Let t be an edge-labeled tree selected in $D_\Sigma(10000)$. Fourthly, by varying ratio r from 0.10 to 0.50, when p in $D^{0.5}_{\Sigma\cup\mathcal{X}}(100, d)$ and t are given, the average execution times of Pattern_Matching algorithm was shown in Fig.7. We can see that the variable counts and execution time of Pattern_Matching algorithm is proportional. The reason may be that the more variable count in edge-labeled term tree increases, the more number of calls of the line 7 or the line 14 of Pattern_Matching algorithm increases.

Suzuki et alia [13] have proposed a matching algorithm for determining whether or not, given an edge-labeled term tree p and an edge-labeled tree t, there exists a substitution θ so that $t \cong p\theta$. The strategy of Suzuki's matching algorithm differs from ours. The matching algorithm we describe here performs top down matching, but the method applied by Suzuki et alia performs bottom up. Let p be an edge-labeled term tree in $D^{0.1}_{\Sigma\cup\mathcal{X}}(100)$ and t an edge-labeled tree in $D_\Sigma(n)$ obtained by varying edge counts n from 1000 to 10000. In order to show the advantage of Pattern_Matching algorithm for Suzuki's matching algorithm, we compared Pattern_Matching algorithm with Suzuki's matching algorithm for each data collec-

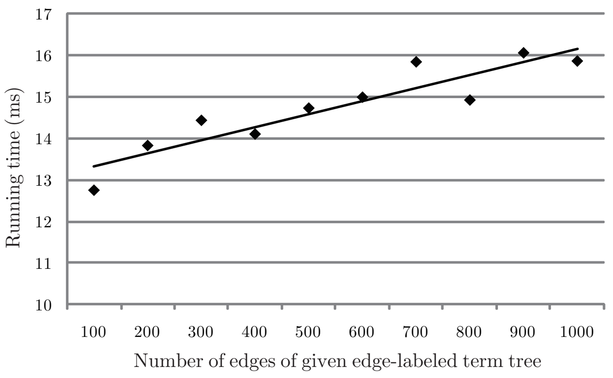

Fig. 5. Running times of Pattern_Matching algorithm for the edge count of given edge-labeled ordered term tree.

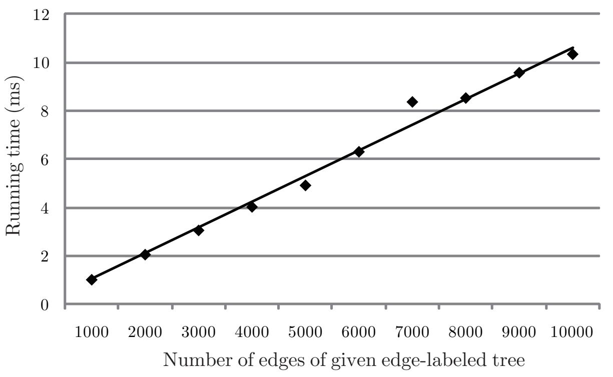

Fig. 6. Running times of Pattern_Matching algorithm for the edge count of given edge-labeled ordered tree.

tion. The results are shown in Fig.8. Fig.8 shows that Pattern_Matching algorithm is faster than Suzuki's matching algorithm. The reasons are follows. (1) If there exists no substitution θ with $t \cong p\theta$, Pattern_Matching algorithm returns $false$ faster than Suzuki's matching algorithm. (2) Pattern_Matching algorithm are implemented using DFUDS as data structure.

These experimental results leads us to give faster tree mining tools using Pattern_Matching algorithm proposed in this paper than using Suzuki's matching algorithm.

VI. Conclusion

We propose a DFUDS representation of edge-labeled ordered term trees that applies the DFUDS representation for ordered trees of Benoit et alia [2]. We also used the DFUDS representation as a data structure to formulate the membership problem for edge-labeled trees and propose a matching algorithm that solves that problem in polynomial time. Evaluation experiments performed with computer implementation of the algorithm demonstrated its efficiency.

Miyoshi et alia analyzed a TTSP graph and proposed its representation as a series of edge-labeled ordered trees (**forest representation**) [7]. Miyoshi et alia further proposed use of the xbw transform for the forest representation of a TTSP graph, as well as an xbw transform path search algorithm for a TTSP graph. As applications of this research, we are considering adaptation of the DFUDS representation proposed here to succinct data structures of TTSP graphs based on forest representation proposed by Miyoshi et alia [7] and the compressed tree proposed by Katoh et alia [6].

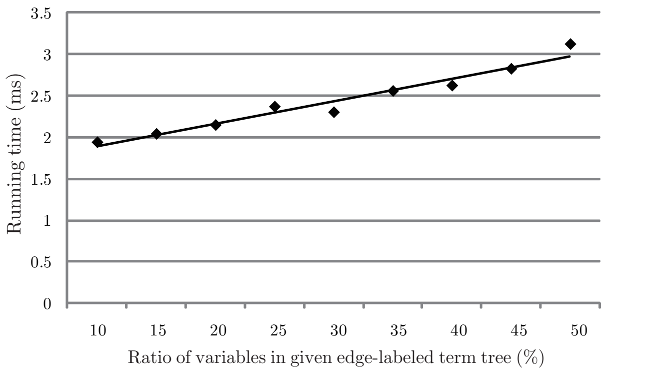

Fig. 7. Running times of Pattern_Matching algorithm for ratios of the variable count in given edge-labeled ordered term tree.

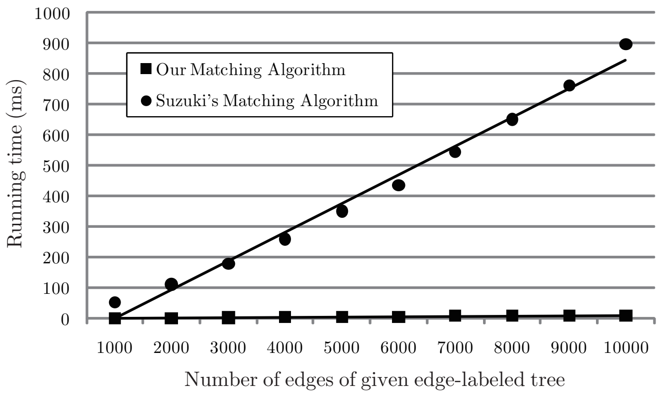

Fig. 8. Pattern_Matching algorithm vs Suzuki's matching algorithm.

Moreover, we are considering graph mining algorithms for semi-structured data using a succinct data structure.

References

[1] D. Arroyuelo, B. Cánovas, G. Navarro, and K. Sadakane. Succinct trees in practice. In *ALENEX*, pages 77–83, 2010.
[2] D. Benoit, E. D. Demaine, J. I. Munro, and V. Raman. Representing trees of higher degree. *Algorithmica*, 43(4):275–292, 2005.
[3] Y.-T. Chiang, C.-C. Lin, and H.-I. Lu. Orderly spanning trees with applications. *SIAM Journal on Computing*, 34(4):924–945, 2005.
[4] P. Ferragina, F. Luccio, G. Manzini, and S. Muthukrishnan. Structuring labeled trees for optimal succinctness, and beyond. In *IEEE FOCS 2005*, pages 184–196, 2005.
[5] R. F. Geary, N. Rahman, R. Raman, and V. Raman. A simple optimal representation for balanced parentheses. In *CPM*, pages 159–172, 2004.
[6] Y. Itokawa, K. Katoh, T. Uchida, and T. Shoudai. *Algorithm using Expanded LZ Compression Scheme for Compressing Tree Structured Data*, pages 333–346. Lecture Notes in Electrical Engineering. Springer, 2010.
[7] Y. Itokawa, J. Miyoshi, M. Wada, and T. Uchida. Succinct representation of ttsp graphs and its application to the path search problem. In *Sixth IASTED International Conference on Advances in Computer Science and Engineering*, pages 33–40, 2010.
[8] G. Jacobson. Space-efficient static trees and graphs. In *IEEE FOCS*, pages 549–554, 1989.
[9] J. Jansson, K. Sadakane, and W.-K. Sung. Ultra-succinct representation of ordered trees. In *ACM-SIAM SODA 2007*, pages 575–584, 2007.
[10] J. I. Munro. Table. In *FSTTCS*, LNCS 1180, pages 37–42. Springer, 1996.
[11] J. I. Munro and V. Raman. Succinct representation of balanced parentheses and static trees. *SIAM Journal on Computing*, 31(3):762–776, 2001.
[12] R. Pagh. Low redundancy in static dictionaries with constant query time. *SIAM Journal on Computing*, 31(2):353–363, 2001.
[13] Y. Suzuki, K. Inomae, T. Shoudai, T. Miyahara, and T. Uchida. A polynomial time matching algorithm of structured ordered tree patterns for data mining from semistructured data. In *ILP-2002*, LNAI 2583, pages 270–284. Springer, 2003.