Algorithm for Enumerating all Frequent Paths from Structurally Compressed Tree-Structured Data

Tomoya Horibe, Yuko Itokawa, Tomoyuki Uchida, Yusuke Suzuki and Tetsuhiro Miyahara

***Abstract*—Extracting structural features common to large tree-structured data is a difficult issue with respect to time. When the input size decreases by structurally compressing large tree-structured data without loss of information, it leads to reduction in running time needed to extract structural features. Tree-structured data can be described by an edge-labeled ordered tree. We first introduce a compression tree representing an edge-labeled ordered tree structurally compressed based on a Lempel-Ziv compression scheme. Then, given a compression tree t obtained by structurally compressing an edge-labeled ordered tree T, we propose a time- and memory-efficient algorithm for enumerating all frequent paths in T without decompressing t. Finally, we discuss the implementation of the proposed algorithm on a computer, explain the experimental results obtained by applying it to artificial edge-labeled ordered trees, and provide discussion on its evaluation.**

***Index Terms*—Enumeration algorithm, Structurally compressed edge-labeled ordered tree, Succinct representation**

I. Introduction

Tree-structured data, such as Web documents, LaTeX sources, and natural languages, can be described by edge-labeled ordered trees. An edge-labeled ordered tree is a rooted tree whose edges have labels and whose internal nodes have ordered children. Due to the rapid progress in networks and information technology, the amount of such tree-structured data increases daily. To find structural features common to large tree-structured data, time- and memory-efficient graph mining algorithms are needed.

To reduce the memory required to store an ordered tree, succinct data structures for ordered trees have been proposed [1], [2], [3], [4], [5], [7], [8], [10]. Specifically, a depth-first unary degree sequence (DFUDS) as a succinct representation of an ordered tree was proposed [7], [9]. For an ordered tree T, a DFUDS of T is a string of parentheses constructed in the depth-first traversal of T, in which the k-th $\boxed{(}$ and its subsequent $\boxed{)}$ are output if the index of a node is k. By taking $\boxed{(}$ to be '0' and $\boxed{)}$ to be '1', the DFUDS of an ordered tree can be handled as a bit string.

Itokawa et al. [5] proposed a structural compression algorithm for effectively compressing tree-structured data without loss of information that is based on a Lempel-Ziv (LZ) compression scheme. In an LZ compression scheme [14] for strings, such as LZSS [12], previously seen text is used as a dictionary, and phrases in the input text are replaced with references into the dictionary to achieve compression. For an edge-labeled ordered tree T and a its subgraph f having an ordered-tree structure, the first occurrence of f in the depth-first traversal of T is used as an entry of a dictionary, and the subgraphs in T that are isomorphic to f are replaced with the reference into the entry of the dictionary to achieve compression. In this paper, for an edge-labeled ordered tree T, we introduce a compression tree, which is an edge-labeled ordered tree obtained by structurally compressing T based on an LZ compression scheme. We then define a succinct representation of a compression tree by extending DFUDS of an ordered tree. Figure 1 shows a compression tree t and its succinct representation. The compression tree t uses the subgraph f induced by the edge set $\{(1, b, 2), (2, a, 3), (1, a, 4), (4, b, 5), (5, a, 6), (4, b, 7), (1, b, 8)\}$ as an internal dictionary. The edge-labeled ordered tree T in Fig. 1 is obtained by replacing nodes represented by the square and all incident edges with the subgraph f.

Next, given a compression tree t obtained by structurally compressing an edge-labeled ordered tree T, we propose an efficient algorithm, called EnuFreqPaths, for enumerating all frequent paths in T without decompressing t. EnuFreqPaths finds frequent paths from each node toward the root on a level-wise strategy w.r.t. the length of a path. Since a succinct representation of a given compression tree is provided using its DFUDS, EnuFreqPaths can naturally use the succinct data structures in implementation using succinct data structure library (SDSL) [11]. Hence, EnuFreqPaths is time- and memory-efficient for enumerating all frequent paths from a given compression tree of an edge-labeled ordered tree.

This paper is organized as follows. In Sec. II, we introduce a compression tree obtained by structurally compressing an edge-labeled ordered tree based on an LZ compression scheme. We also define the succinct representation of a compression tree. In Sec. III, given a compression tree t obtained by structurally compressing an edge-labeled ordered tree T, we present our time- and memory-efficient algorithm for enumerating all frequent paths in T without decompressing t. In Sec. IV, we explain the experimental results obtained by applying the proposed algorithm to artificial big data and discuss its efficiency. In Sec. V, we conclude this paper.

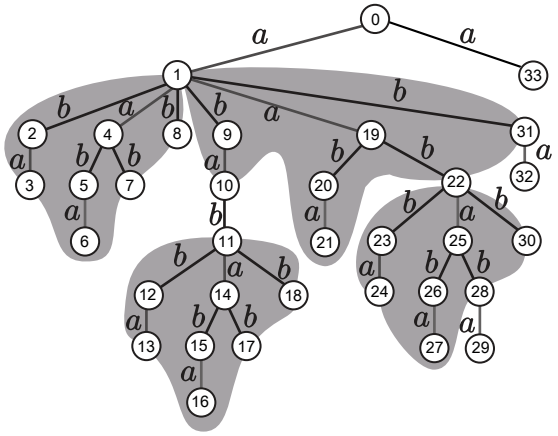
II. Preliminaries

In this section, we introduce a compression tree, which is a rooted edge-labeled ordered tree obtained from an edge-labeled ordered tree by replacing repeated occurrences of subgraphs having ordered-tree structures with references to a dictionary, which is a list of first occurrences of the repeated subgraphs in its depth-first traversal. Moreover, we define a succinct representation of a compression tree by extending the depth-first unary degree sequence (DFUDS)[7], [9] for an ordered tree.

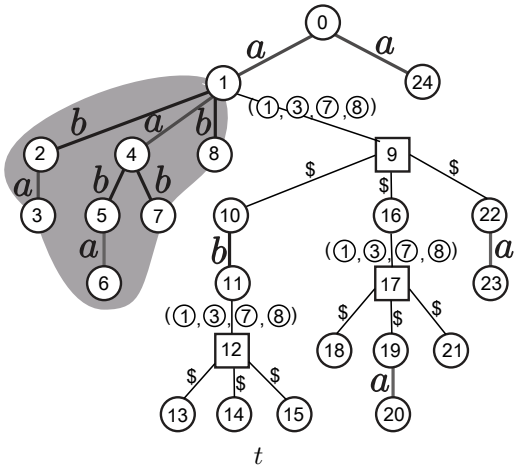
T. Horibe, T. Uchida, Y. Suzuki and T. Miyahara are with Department of Intelligent Systems, Hiroshima City University. Hiroshima, Japan, email: mb67018@e.,uchida@,y-suzuki@,miyahara@hiroshima-cu.ac.jp

Y. Itokawa is with Faculty of Psychological Science, Hiroshima International University. 555-36 Kurose-Gakuendai, Higashi-Hiroshima, Hiroshima Japan e-mail: y-itoka@he.hirokoku-u.ac.jp

ISBN: 978-988-14047-3-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)



T



t

DFUDS of t :

$$\underset{\textcircled{0}}{(}\ \underset{\textcircled{1}}{((\#}\ \underset{\textcircled{2}\textcircled{3}\textcircled{4}}{((((a}\ \underset{\textcircled{5}\textcircled{6}\textcircled{7}\textcircled{8}}{(b\ a}\ \underset{\boxed{9}}{((a}\ \underset{\textcircled{10}}{(b\ a\ b\ b}\ \underset{\textcircled{11}}{(((P}\ \underset{\boxed{12}\ \textcircled{13}\textcircled{14}\textcircled{15}\textcircled{16}}{(\$\ \$\ \$\ (\$}\ \underset{\boxed{17}\ \textcircled{18}\textcircled{19}\textcircled{20}\textcircled{21}\textcircled{22}\textcircled{23}\textcircled{24}}{(((P\ \$\ \$\ (\$}\ a\ \$\ \underset{}{(\$}\ a\ a$$

Succinct representation of t :

$\#, a, b!(1,3,7,8)!0,0,0,1,0,0,0,0,2,0,3,2,0,0,2,0,3,2,3,3,0,0,0,-2,0,-1,0,3,0,0,0,-2,-1,-1,-1,0,-1,0,0,0,-2,-1,0,-1,2,-1,0,-1,2,2$

Fig. 1. Edge-labeled ordered tree T, compression tree t obtained by structurally compressing T, DFUDS of t and succinct representation of t. Gray regions in T show repeated occurrences of subgraph having same ordered-tree structure. Gray region in t shows entry of its dictionary. In DFUDS of t, number in circle represents node ID of t and "P" denotes reference $(1,3,7,8)$.

A. Compression Tree

An edge-labeled ordered tree T is a rooted tree whose internal nodes have ordered children and whose edges have labels. The node and the edge sets of T are denoted as $V(T)$ and $E(T)$, respectively. We denote an edge $e \in E(T)$ as $e = (u, a, v)$ such that the two endpoints of e are nodes u and v and the label of e is a.

Let a list $(u, L, u_1, u_2, \ldots, u_k)$ consisting of an internal node u, consecutive children $u_1, u_2, \ldots, u_k$ of u and label L be called a *port list* of an edge-labeled ordered tree T. For a port list $h = (u, L_h, u_1, u_2, \ldots, u_k)$, u is called a *parent port* of h and each node u_i $(1 \leq i \leq k)$ is called a *child port* of h. Two port lists $h = (u, L_h, u_1, u_2, \ldots, u_k)$ and $h' = (v, L_{h'}, v_1, v_2, \ldots, v_\ell)$ of an edge-labeled ordered tree T are said to be *disjoint* if the following conditions are satisfied.

(1) $\{u_1, u_2, \ldots, u_k\} \cap \{v_1, v_2, \ldots, v_\ell\} = \emptyset$.
(2) If u and v are the same node, u_k is older than v_1 or u_1 is younger than v_ℓ.

For an edge-labeled ordered tree T and its internal node u, we denote the subgraph consisting of all descendants of u as $T[u]$, that is, $T[u]$ is the subtree of T having u as its root. For a subset $U \subseteq V(T)$, we denote the subgraph induced by U as $T[U]$, that is, $T[U] = (U, \{e \mid \text{both endpoints of } e \in E(T) \text{ are in } U\})$. For an internal node u and a descendant v of u, we denote the path between u and v as $P_{u,v}$. Note that $P_{u,v}$ is only one node if u and v are the same node. For an edge-labeled ordered tree T, a *reference* of T is a list $(v, v_1, v_2, \ldots, v_n)$ of nodes satisfying the following conditions.

(1) For each i $(1 \leq i \leq n)$, v_i is a descendant of an internal node v of T.
(2) For any i, j $(1 \leq i, j \leq n)$, v_i is not a descendant of v_j and vice visa.
(3) For any i, j $(1 \leq i < j \leq n)$, v_j appears after v_i in the depth-first traversal of T.

We denote the set of all references of T as $\mathcal{R}_T$. For a reference $L = (v, v_1, v_2, \ldots, v_n)$ of T, we denote the subgraph induced by the node set $\bigcup_{w \in W} V(P_{v,w})$ as $T\langle L \rangle$ called by a *reference tree*, where W is the set of leaves such that any leaf $w \in W$ appears from v_1 up to v_n in the depth-first traversal of T but not included in $V(T[v_i]) - \{v_i\}$ for any $1 \leq i \leq n$. In Fig. 1, for reference $(1,3,7,8)$, we give the reference tree $t\langle(1,3,7,8)\rangle = \big(\{1,2,\ldots,8\}, \{(1,b,2),(2,a,3),\ldots,(1,b,8)\}\big)$. A subset $\mathcal{D}_\mathcal{T}$ of $\mathcal{R}_\mathcal{T}$ is called a *dictionary* of T if for any two distinct references $L_1 = (u, u_1, u_2, \ldots, u_\ell)$ and $L_2 = (v, v_1, v_2, \ldots, v_r)$ of $\mathcal{D}_\mathcal{T}$, $V(T\langle L_1 \rangle) \cap V(T\langle L_2 \rangle) = \emptyset$ except u and v. For example, in Fig. 1, for references $(1,3,7,8), (1,10,22,31), (11,13,17,18)$, and $(22,24,28,30)$ in T, the reference trees $T\langle(1,3,7,8)\rangle, T\langle(1,10,22,31)\rangle, T\langle(11,13,17,18)\rangle$, and $T\langle(22,24,28,30)\rangle$ are isomorphic.

For a list or set S, we denote the number of elements in S as $|S|$.

Definition 1: (Hyper Tree) Let Λ be an alphabet, $T = (V_T, E_T)$ an edge-labeled ordered tree, H_T a set of disjoint port lists of T, and D_T a dictionary of T such that $\Lambda \cap D_T = \emptyset$. Then, a triplet $t = (V_t, E_t, H_t)$ is called a *hyper tree* obtained from T, H_T, and D_T, where V_t, E_t and H_t are defined as follows.

(1) $V_t = V_T$,
(2) $E_t = E_T - \bigcup_{(v_0,v_1,\ldots,v_r) \in H_T} \{(v_0, a_1, v_1), \ldots, (v_0, a_r, v_r)\}$, and the label of each edge $e \in E_T$ is preserved in E_t.
(3) $H_t = H_T$ and each port list in H_t is labeled with a reference $(w, w_1, w_2, \ldots, w_k) \in D_T$ such that w_k appears before w_1 on the depth-first traversal of T.

By modifying the LZ77 compression scheme for strings to a hyper tree, a compression tree is a hyper tree defined as follows.

Definition 2: (Compression Tree) A *compression tree* is a hyper tree obtained from T by replacing repeated oc-

ISBN: 978-988-14047-3-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

currences of subgraphs having ordered-tree structures with references to the first occurrence of the subgraphs.

Figure 1 shows the compression tree $t = (\{0, 1, \ldots, 24\}, \{(0, a, 1), (1, b, 2), \ldots, (0, a, 24)\}, \{(1, P, 10, 16, 22), (11, P, 13, 14, 15), (16, P, 18, 19, 21)\})$ of the edge-labeled ordered tree T, where P is reference $(1, 3, 7, 8)$.

B. Succinct Representation of Compression Trees

In this subsection, we explain a succinct representation of a compression tree.

The DFUDS for an ordered tree T of m edges is defined inductively as follows [7], [9]. The DFUDS of the tree consisting of only one node is $\boxed{(}\ \boxed{)}$. The DFUDS of an ordered tree T that has k subtrees $T_1, \ldots, T_k$ is a sequence of parentheses constructed by concatenating $k+1$ $\boxed{(}$s, one $\boxed{)}$, k DFUDSs of $T_1, \ldots, T_k$ in this order (the initial $\boxed{(}$ of the DFUDS of each subtree has been removed). The resultant DFUDS is a sequence of balanced parentheses of length $2m$. The sequence of parentheses, that is a DFUDS, can be interpreted as the result of visiting all nodes in preorder and outputting k $\boxed{(}$s for each node, whose degree is k, following the one $\boxed{)}$. The DFUDS is a succinct representation of an ordered tree with no edge labels. But since $\boxed{)}$ occupies the rightmost position for each node in the DFUDS, we can modify the DFUDS of an ordered tree to the DFUDS of an edge-labeled ordered tree, using a hash function that returns the label of the edge incident to the node corresponding to each $\boxed{)}$. To provide a succinct representation of a compression tree, we consider an underlying ordered tree for a compression tree. For a compression tree $t = (V_t, E_t, H_t)$, we call by an *underlying tree of* t the edge-labeled ordered tree obtained from t by applying the following replacements to all port lists of t. A port list $h = (u, u_1, u_2, \ldots, u_k) \in H_t$ is replaced with an edge-labeled ordered tree as follows.

(1) Remove h from t.
(2) Construct an edge-labeled ordered tree $s = (\{u', v, u'_1, u'_2, \ldots, u'_k\}, E_s)$ defined as follows. (a) v has u' as the parent and $u'_1, u'_2, \ldots, u'_k$ as the children in that order. (b) The edge between u' and v is labeled with the reference of h and any edge between v and its child is labeled with the special symbol "$".
(3) Identify the parent port u and each child port u_i $(1 \leq i \leq k)$ with the root u' of s and each leaf $u'_i (1 \leq i \leq k)$, respectively.

Definition 3: (Succinct Representations of Compression Trees) The succinct representation of a compression tree t is the DFUDS of the underlying tree of t.
Figure 1 shows the succinct representations of the edge-labeled ordered tree T and the compression tree t.

Hence, by using a succinct representation of a compression tree, we can give a compact coding for a structured-compression of an edge-labeled ordered tree T as follows.

Definition 4: (Compact Coding for an Edge-Labeled Ordered Tree) Let T be an edge-labeled ordered tree. Let EL_T be the coding of the list of all edge labels in T which are sorted by occurrence order in the depth-first traversal of T. Let D_T be the coding of a list of references in T which are sorted by occurrence order in the depth-first traversal of T. Let C_T be the coding of a compression tree t of T. Then, a *compact coding* of T is given by a sequence of $EL_T \circ$"!"$\circ D_T \circ$"!"$\circ C_T$, denoted as $Code(T) = \langle EL_T, D_T, C_T \rangle$, where $\circ$ is an operator of two concatenating sequences.
Since we create EL_T, D_T and C_T based on the depth-first traversal of T, we can easily create hash functions that return the edge label or reference for each edge in the compression tree t of T. Figure 1 shows the succinct representation of the compression tree t.

III. Enumeration Algorithm for Extracting Frequent Paths in Compression Trees

For a compression tree t of an edge-labeled ordered tree T and an integer k $(k \geq 1)$, a path p in t is *k-frequent* if p appears in T k or more times. Such an integer k is called a *minimum occurrence*. We define an enumeration problem for extracting all k-frequent paths in a given compression tree as follows.

Frequent-Paths Enumeration Problem
Instance: Compression tree t and minimum occurrence k $(k \geq 1)$.
Problem: Enumerate all k-frequent paths in t without decompressing t.

Algorithm 1 presents our enumeration algorithm, called EnuFreqPaths, for solving the frequent-paths enumeration problem. For a sequence S of length n on alphabet Λ, character c and integer i $(0 \leq i \leq n-1)$, the functions $rank$ and $select$ used in EnuFreqPaths are defined as follows.

(1) $rank_c(S, i)$ returns the number of occurrences of c in subsequence $S[0:i]$.
(2) $select_c(S, i)$ returns the i-th position of c from the beginning of S.

By using the SDSL [11], we can compute $rank$ and $select$ in constant time. When a compression tree t and a minimum occurrence k are given, EnuFreqPaths enumerates all k-frequent paths in t from each node toward the root of t on a level-wise strategy w.r.t. the length of a k-frequent path. Figure 2 shows an example of the enumeration process of k-frequent paths by using EnuFreqPaths.

EnuFreqPaths (Algorithm 1) has two procedures genOccPoints (Procedure 2) and makeCandFreqPaths (Procedure 3). For an edge e, $parent(e)$ returns the parent edge of e, and function $childrank(e)$ returns the index i such that e is the i-th child of the parent edge of e. For reference $L = (v, v_1, v_2, ..., v_k) \in D_T$ and an integer n $(1 \leq n \leq k)$, a function $dic(L, n)$ returns the index v_n. By using the SDSL, these operations on a compression tree can be executed in constant time.

To count the number m of paths, which is isomorphic to a path p and terminates at an index i of C_T, EnuFreqPaths uses a triplet (u, p, OP_u), where OP_u is a multi-set of indexes in the interval $[0, |C_T|)$ and satisfies $|OP_u| = m$. Such a triplet is called a *path-count triplet* at index i. Given a compact coding $\langle EL_T, D_T, C_T \rangle$ of an edge-labeled ordered tree T, EnuFreqPaths first generates the set Z_1 of path-count triplets at all indexes between 0 and $|C_T|$. Then, it constructs the set P_1 of all k-frequent edges from Z_1. That

ISBN: 978-988-14047-3-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

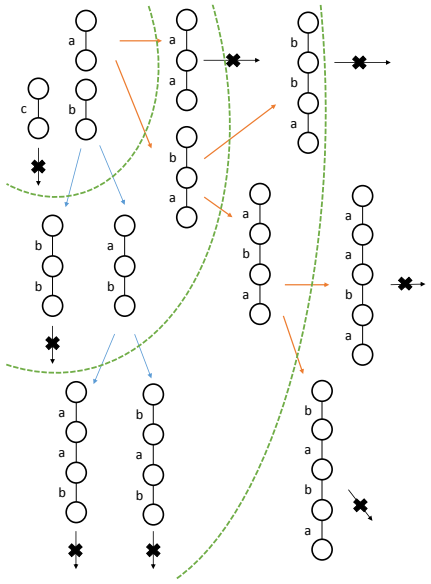


Fig. 2. Tree representing process of enumerating k-frequent paths.

Algorithm 1 ENUFREQPATHS

Require: A compact coding $\langle EL_T, D_T, C_T\rangle$ of an edge-labeled ordered tree T and a minimum occurrence k ($k \geq 1$)
Ensure: The set F of all k-frequent paths in T
1: $Z_1 = genOccPoints(\langle EL_T, D_T, C_T\rangle)$
2: **for all** a is in EL_T **do**
3: **if** $\sum_{(i,a,OP_i)\in Z_1} |OP_i| \geq k$ **then**
4: $P_1 = P_1 \cup \{a\}$
5: **end if**
6: **end for**
7: $F = P_1$ **and** $ln = 1$
8: **while** $P_{ln} \neq \emptyset$ **do**
9: $P_{ln+1} = \emptyset$
10: $C_{ln+1} = makeCandFreqPaths(Z_{ln}, P_{ln}, P_1)$
11: **for all** $p \in P_{ln}$ **and** $a \in P_1$ **do**
12: **if** $\sum_{(i,p\circ a,OP_i)\in C_{ln+1}} |OP_i| \geq k$ **then**
13: $P_{ln+1} = P_{ln+1} \cup \{p \circ a\}$
14: $Z_{ln+1} = Z_{ln+1} \cup \{(i, p \circ a, OP_i)\}$
15: **end if**
16: **end for**
17: $F = F \cup P_{ln+1}$
18: ln++
19: **end while**
20: **return** F

is, P_1 is the set of all k-frequent paths, the length of each is 1. Next, by using Procedure MAKECANDFREQPATHS, ENUFREQPATHS recursively generates the set C_{ln+1} of all path-count triplets having candidate paths, the length of each is $ln+1$ from set Z_1 and set Z_{ln} each of which has a path whose length is ln. From C_{ln+1}, ENUFREQPATHS constructs the set P_{ln+1} of k-frequent paths, the length of each is $ln+1$ and set $Z_{ln+1} = \{(i, p, OP_i) \in C_{ln+1} \mid p \in P_{ln+1}\}$. Finally, ENUFREQPATHS returns the set F of all k-frequent paths appearing in T from t.

In Procedure MAKECANDFREQPATHS, when we construct a candidate path whose length is $ln+1$ by expanding the k-frequent path whose length is ln, four cases are considered

Procedure 2 GENOCCPOINTS

Require: The compact coding $\langle EL_T, D_T, C_T\rangle$
Ensure: The set Z_1 of all path-count triplets, each of which has a path whose length is 1.
1: $Z_1 = \emptyset$ **and** $OP_i = \emptyset$ for i $(0 \leq i < |C_T|)$
2: **for all** i $(0 \leq i < |C_T|)$ **do**
3: **if** $C_T[i]$ is in EL_T **then**
4: $OP_i = OP_i \cup \{i\}$
5: **end if**
6: **if** $C_T[i]$ is in D_T **then**
7: **for all** an edge j of reference tree $T\langle C_T[i]\rangle$ **do**
8: $OP_j = OP_j \cup \{j\}$
9: **end for**
10: **end if**
11: **end for**
12: **for all** i $(0 \leq i < |C_T|)$ such that $C_T[i]$ is in EL_T **do**
13: $Z_1 = Z_1 \cup \{(i, C_T[i], OP_i)\}$
14: **end for**
15: **return** Z_1

(see Fig. 3). If we try to expand a path by appending an edge to the edge e incident to the root of a reference tree, we must consider two cases in which the label of $parent(e)$ is '$' (case 1) or is included in EL_T (case 2). Otherwise, if e is not incident to the root of any reference tree, we must also consider two cases in which the label of the parent edge $parent(e)$ is '$' (case 3) or is included in EL_T (case 4).

For example, in the compression tree t of Fig. 1, we consider the path from index 44 (the edge $(19, a, 20)$) to index 3 (the root of t). Since reference having index 44 as the first argument does not exist in D_T, we determine whether or not $C_T[parent(44)]$ is '$'. From $parent(44) = 43$, $C_T[parent(44)] = C_T[43] =$'$'. We can see that expanding from the edge at index 44 is case 3. Hence, we obtain $j = 43$, $u = dic(C_T[parent(43)], childrank(43)) = dic(P, 2) = 18$, and $OP'[18] = OP'[18] \cup (18, ab, \{40\})$. Because $parent(14)$ is the root of the reference tree referred from the port list at index 40 and $C_T[parent(40) = 36] =$'$', we can apply the expansion of case 2. Next, we obtain $q = dic(C_T[parent(36) = 23], childrank(36)) = dic(P, 2) = 18$ and $OP'[18] = OP'[18] \cup (18, ab, \{23\})$. Finally, we can obtain the path '$ababaa$' as the path from index 44 to the root.

We explain ENUFREQPATHS when a compression tree t in Fig. 1 and the minimum occurrence $k = 5$ are given. In line 1 of ENUFREQPATHS, Procedure GENOCCPOINTS generates the following set Z_1.

$Z_1 = \{(8, a, \{8\}), (10, b, \{10, 23, 31, 40\}),$
$(11, a, \{11, 23, 31, 40\}), (14, a, \{14, 23, 31, 40\}),$
$(16, b, \{16, 23, 31, 40\}), (17, a, \{17, 23, 31, 40\}),$
$(18, b, \{18, 23, 31, 40\}), (19, b, \{19, 23, 31, 40\}),$
$(27, b, \{27\}), (44, a, \{44\}), (48, a, \{48\}), (49, a, \{49\})\}$

Moreover, ENUFREQPATHS generates $P_1 = \{a, b\}$ as the set of 5-frequent paths whose length is 1. By recursively applying Procedure MAKECANDFREQPATHS, ENUFREQPATHS generates the path-count triplets $Z_4 = \{(8, baba, \{8\}), (11, baba, \{23, 23\}), (14, baba, \{31, 31\})\}$, and $P_4 = \{baba\}$. Since t has no5-frequent path whose length is 5, ENUFREQPATHS terminates after outputting the set $F = \{a, b, ab, ba, aba, bab, baba\}$ of all 5-frequent paths

ISBN: 978-988-14047-3-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

Procedure 3 MAKECANDFREQPATHS

Require: A set Z_{ln} of path-count triplets, each of which has a path with length ln, set P_{ln} of k-frequent paths, the length of each is ln, and set P_1 of k-frequent edges
Ensure: The set Z_{ln+1} of path-count triplets each of which has a path whose length is $ln+1$

$OP'_i = \emptyset$ for i $(0 \leq i < |C_T|)$
$U = \emptyset$
for all $p \in P_{ln}$ **do**
  for all $(i, p, OP_i) \in Z_{ln}$ **do**
    if $(rank(i)+1, v_1, ...v_k) \in D_T$ **then**
      for all $w \in OP_i$ **do**
        $u = parent(w)$
        if $C_T[u] =' \$'$ **then**
          /* case 1 */
          $q = dic\,(C_T[parent(u)], childrank(u))$
          $U = U \cup \{q\}$
          $OP'_q = OP'_q \cup \{parent(u)\}$
        else
          /* case 2 */
          $U = U \cup \{u\}$
          $OP'_u = OP'_u \cup \{u\}$
        end if
      end for
    else if $C_T[parent(i)] =' \$'$ **then**
      /* case 3 */
      for all $w \in OP_i$ **do**
        $j = parent(i)$
        $u = dic\,(C_T[parent(j)], childrank(j))$
        $U = U \cup \{select(u+1)\}$
        $OP'_u = OP'_u \cup \bigcup_{w \in OP_i} \{select(u+1)\}$
      end for
    else
      /* case 4 */
      $e = parent(i)$
      $U = U \cup \{e\}$
      $OP'_e = OP'_e \cup OP_i$
    end if
  end for
end for
for all $w \in U$ **do**
  $Z_{ln+1} = Z_{ln+1} \cup \{(w, p \circ C_T[w], OP'_w)\}$
end for
return Z_{ln+1}

in t.

IV. EXPERIMENTAL RESULTS AND DISCUSSION

To discuss the efficiency of ENUFREQPATHS, we explain the experimental results obtained from applying ENUFREQPATHS to artificial big data, which were randomly generated, and discuss its efficiency.

A. Experimental Environments and Setting

We implemented ENUFREQPATHS on a PC with macOS 10.12 Sierra and 32GB memory and 4-GHz Intel Core i7 by using C++. In implementing ENUFREQPATHS, we use the SDSL[11] as the data structure for compression trees. We randomly created a set D of 1000 edge-labeled ordered trees containing from 2000 to 14000 nodes and having two edge labels. For each edge-labeled ordered tree T in D, we created a compression tree t of T that had 5 entries in the dictionaries. We denote the set of these 1000 compression trees as $CT(D)$.

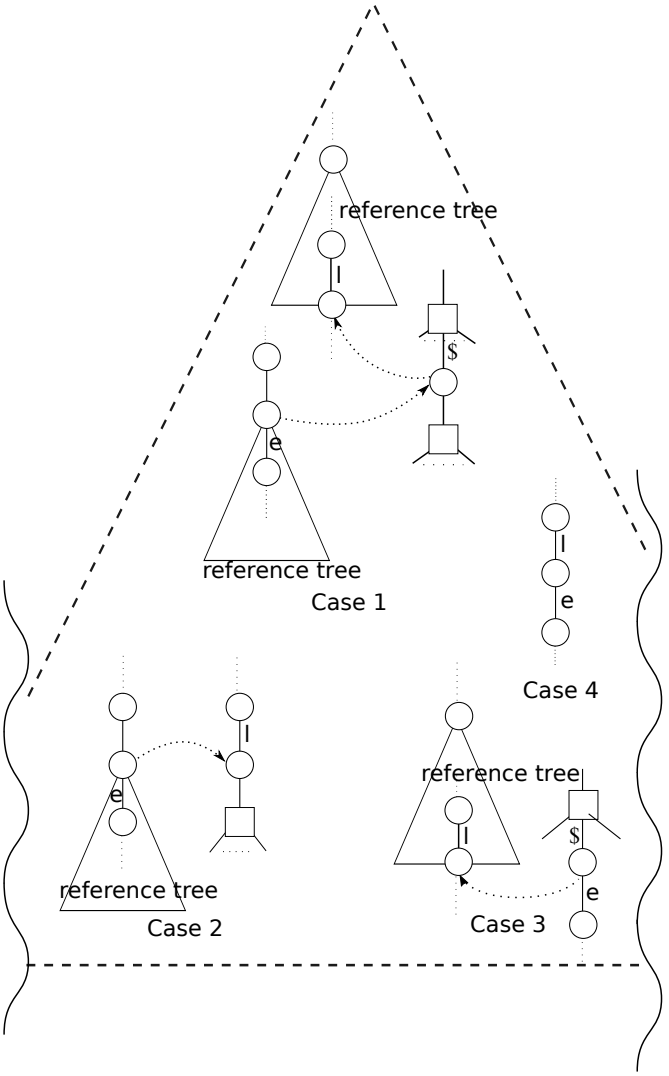


Fig. 3. All case of extending path

B. Experimental Results and Discussion

Figure 4 shows a comparison of the running time of ENUFREQPATHS when each $T \in D$ and $t \in CT(D)$, which is the compression tree of T, are given as inputs. We can see that ENUFREQPATHS given the compression tree $t \in CT(D)$ of T is always faster than that given an edge-labeled ordered tree $T \in D$. Figure 5 shows that the running time of ENUFREQPATHS is proportional to the number of occurrence points of frequent paths. Figure 6 shows a comparison of the running time needed to find all frequent paths. For $i \geq 1$, ENUFREQPATHS constructs candidate paths p^{i+1}, whose lengths are $i+1$, from all frequent paths p^i whose lengths are i by expanding at the occurrence points of p^i. From Fig. 5, we can see that ENUFREQPATHS is a sequential linear time algorithm w.r.t. the number of found occurrence points. From Fig. 6, for a compression tree t of an edge-labeled ordered tree T, we can see that the running time

ISBN: 978-988-14047-3-2
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

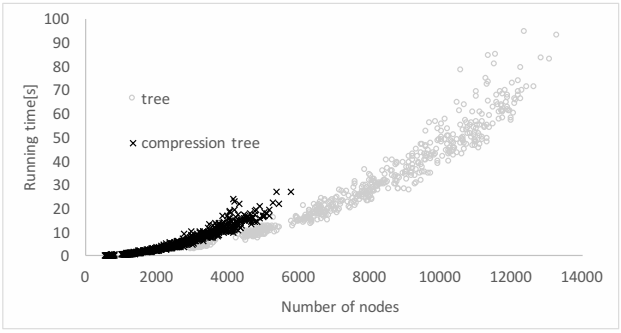


Fig. 4. Number of nodes vs running time

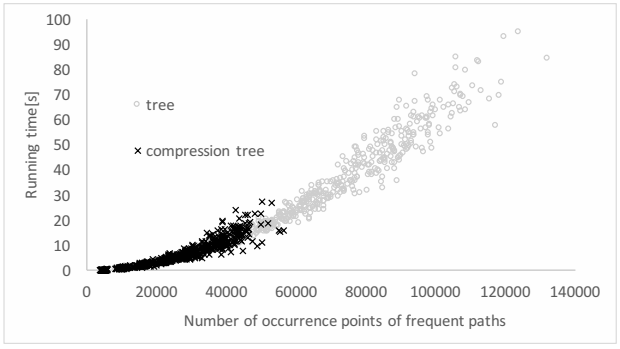


Fig. 5. Number of occurrence points of frequent paths vs running time

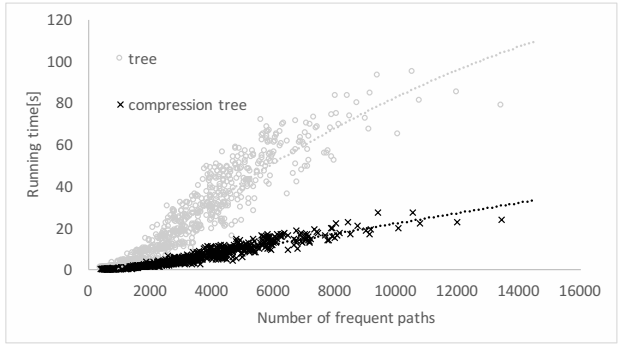


Fig. 6. Number of frequent paths vs running time

required for extracting all k-frequent paths from t is faster than that from T by about 4.11 times.

Paths occurring in an entry l of a dictionary always occur in all port lists having L as the edge label. Hence, if the compression ratio becomes higher, the number of occurrence points of frequent paths in a compression tree decreases. This leads to reduction in memory usage and increase in enumeration speed of frequent paths in compression trees.

V. Conclusion

We have introduced a compression tree, which is obtained by replacing repeated occurrences of subgraphs having ordered tree structures with references to the first occurrence point based on an LZ compression scheme, and presented a succinct representation of a compression tree. We then have proposed a time- and memory-efficient algorithm for enumerating all frequent paths in a given compression tree without decompression. To discuss the efficiency of the proposed algorithm, we explained the experimental results obtained from applying the algorithm to artificial big data, which were randomly generated.

For future work, we will apply the proposed algorithm to real-world big data. By extending the proposed algorithm, we will develop more time- and memory-efficient algorithms for enumerating all frequent subgraphs having ordered-tree structures for a large set of compression trees and for enumerating all characteristic tree patterns having structured variables common to a given large set of edge-labeled ordered trees by extending the pattern the matching algorithm proposed by Itokawa et al. [6] and Suzuki et al.[13] for edge-labeled ordered trees to a matching algorithm for compression trees.

Acknowledgments

This work was partially supported by Grant-in-Aid for Scientific Research (B) (Grant Number 26280087) and (C) (Grant Number 15K00313) from the Japan Society for the Promotion of Science (JSPS), Japan.

References

[1] J. Barbay, L.C. Aleardi, M. He, J.I. Munro. Succinct representation of labeled graphs *International Symposium on Algorithms and Computation*, pp.316-328, 2007.

[2] D. Benoit, E. D. Demaine, J. I. Munro, and R. Raman. Representing trees of higher degree. *Algorithmica*, 43(4):275–292, 2005.

[3] P. Ferragina, F. Luccio, G. Manzini, and S. Muthukrishnan. Structuring labeled trees for optimal succinctness, and beyond. In *Proceedings of IEEE Symposium on Foundations of Computer Scie nce (FOCS)*, pages 184–196, 2005.

[4] R. F. Geary, N. Rahman, R. Raman, and V. Raman. A simple optimal representation for balanced parentheses. In *CPM*, pages 159–172, 2004.

[5] 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.

[6] Y. Itokawa, M. Wada, T. Ishii, and T. Uchida. *Pattern Matching Algorithm Using a Succinct Data Structure for Tree-Structured Patterns*, pages 349–361. Lecture Notes in Electrical Engineering 110. Springer, 2012.

[7] J. Jansson, K. Sadakane, W.-K Sung. Ultra-succinct representation of ordered trees with applications *Journal of Computer and System Sciences* 78, pp.619631, 2012.

[8] H.-I. Lu, C.-C. Yeh. Balanced parentheses strike back. *ACM Transactions on Algorithms*: Volume 4 Issue 3, 2008.

[9] J. I. Munro and V. Raman. Succinct representation of balanced parentheses and static trees. *SIAM Journal on Computing*, 31(3):762–776, 2001.

[10] R. Raman, S.S. Rao. Succinct representations of ordinal trees. *Space-efficient data structures, streams, and algorithms*, Vol. 8066 of the series LNCS, pp.319-332, 2013

[11] SDSL: Succinct data structure library. http://simongog.github.io/sdsl/

[12] J. A. Storer and T. G. Szymanski. Data compression via textual substitution. *Journal of the ACM*, 29(4):928–951, 1982.

[13] Y. Suzuki, T. Shoudai, T. Uchida and T. Miyahara. An efficient pattern matching algorithm for ordered term tree patterns. *IEICE Trans. Fundamentals*, E98-A(6):270–284, 2015.

[14] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, IT-23(3):337–343, 1977.