Enumeration Algorithms for All Characteristic Paths and Subtrees from Structurally Compressed Tree-Structured Data

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

***Abstract*—Enumerating structural features common to large tree-structured data is difficult with respect to time. Decreasing the input size by structurally compressing large tree-structured data without loss of information leads to a reduction in the running time needed to extract structural features. Since tree-structured data can be described by an edge-labelled ordered tree, we first introduce a compression tree, which is a hypertree, as a compressed representation of an edge-labelled ordered tree T obtained by structurally compressing T on the basis of a Lempel-Ziv compression scheme. Then, we define a compact coding $Code(T)$ of T as a sequence consisting of a coding of a list of edge-labels, a coding of dictionaries and a succinct representation of a compression tree for T. Second, given a compact coding $Code(T)$ of T as an input, we present an enumeration algorithm, called ENUFREQMAXPATH, for finding all frequent maximal paths as characteristic paths in T without decompressing $Code(T)$. For a set S of edge-labelled ordered trees, let $Code(S)$ be the set of compact codings of all edge-labelled ordered trees in S. Then, third, given $Code(S)$ as an input, we present an enumeration algorithm, called ENUFREQSUBTREE, for enumerating all frequent subtrees as characteristic subtrees appearing in S without decompressing $Code(S)$. Finally, we implement the proposed algorithms ENUFREQMAXPATH and ENUFREQSUBTREE on a computer, explain the experimental results obtained by applying ENUFREQMAXPATH to a compact coding of a synthetic edge-labelled ordered tree and ENUFREQSUBTREE to a compact coding of a set of synthetic edge-labelled ordered trees and provide discussion on evaluations of ENUFREQMAXPATH and ENUFREQSUBTREE.**

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

I. INTRODUCTION

Tree-structured data, such as Web documents, LaTeX sources, and parse trees of natural languages, can be described by edge-labelled ordered trees. An edge-labelled ordered tree is a rooted tree whose edges have labels and whose internal nodes have ordered children. Due to the rapid progress made on 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 [2], [3], [4], [5], [7], [10], [11], [13]. Specifically, a depth-first unary degree sequence (DFUDS) used as a succinct representation of an ordered tree was proposed [10], [12]. 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. [7] 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 [17] for strings, such as LZSS [15], previously seen text is used as a dictionary, and phrases in the input text are replaced with references to the dictionary to achieve compression. For an edge-labelled ordered tree T and its subgraph f having an edge-labelled 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 a reference to the entry of the dictionary to achieve compression. First, in this paper, we introduce a compression tree t for an edge-labelled ordered tree T such that t is an edge-labelled ordered tree obtained by structurally compressing T on the basis of an LZ compression scheme. We then define a succinct representation of a compression tree by extending the DFUDS of an ordered tree. In Fig. 1, we give a compression tree t for the edge-labelled ordered tree T and a DFUDS of t as a succinct representation of t as an example. The compression tree t uses the subtree f induced by the edge set $\{(8,b,10),(10,a,11),(8,a,14),(14,b,16),(16,a,17),(14,b,18),(8,b,19)\}$ as an internal dictionary. The edge-labelled ordered tree T in Fig. 1 is obtained by replacing 3 hyperedges represented by the squares 23, 31 and 40 and all incident edges with the subtree f. Moreover, we define a compact coding $Code(T)$ of an edge-labelled ordered tree T as a sequence consisting of a coding of a list of edge labels, a coding of a dictionary and a succinct representation of a compression tree for T. In Fig. 1, we give a compact coding of the edge-labelled ordered tree T as an example.

For an edge-labelled ordered tree T and an integer k ($k \geq 1$), a path p is k-frequent if p appears in T k times or more. A k-frequent path p is maximal if there exists no k-frequent path that has p as a subpath. Second, given a compact coding $Code(T)$ for an edge-labelled ordered tree T and an integer k ($k \geq 1$), we present an efficient algorithm, called ENUFREQMAXPATH, for enumerating all k-frequent maximal paths in T without decompressing $Code(T)$. ENUFREQMAXPATH finds k-frequent paths from each node toward the root on a level-wise strategy with respect to

T. Horibe, T. Uchida, Y. Suzuki and T. Miyahara are with Graduate School of Information Sciences, Hiroshima City University, Hiroshima, Japan, e-mail: {mb67018@e., uchida@, y-suzuki@, miyahara@}hiroshima-cu.ac.jp
Y. Itokawa is with Faculty of Psychological Science, Hiroshima International University, Hiroshima, Japan, e-mail: y-itoka@he.hirokoku-u.ac.jp

(Advance online publication: 10 February 2018)

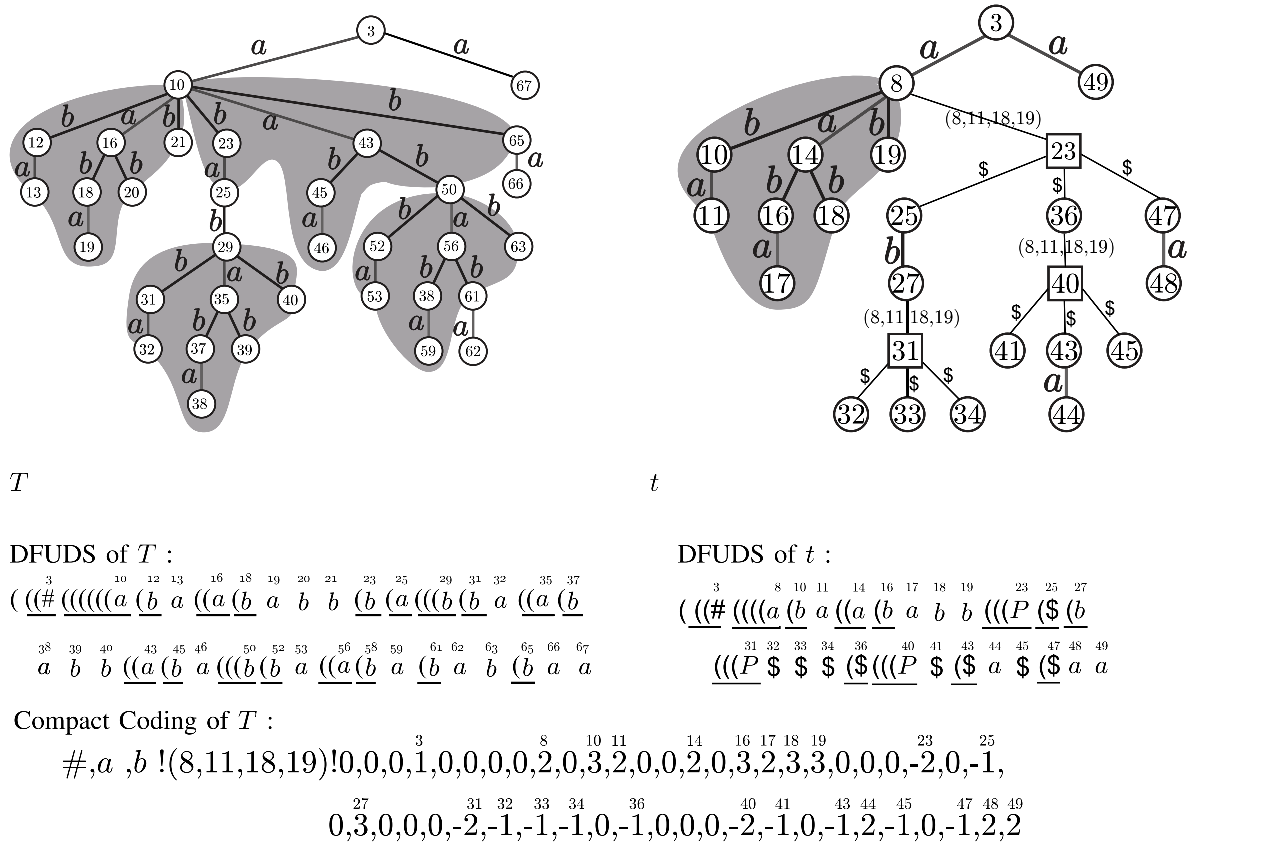

Fig. 1. Edge-labelled ordered tree T, compression tree t obtained by structurally compressing T, DFUDSs of T and t and compact coding of T. Grey regions in T show repeated occurrences of isomorphic subtrees. Grey region in t shows entry of its dictionary. In DFUDSs of T and t, numbers in circles in T and t represent node ID that is index on DFUDSs of T and t, numbers in squares in t represent port list ID that is an index on DFUDSs of T and t and "P" in t denotes reference $(8, 11, 18, 19)$.

the length of a path. Since a succinct representation of a given compression tree is provided by using its DFUDS, EnuFreqMaxPath can naturally use the succinct data structures in implementations that use the succinct data structure library (SDSL) [14]. Hence, EnuFreqMaxPath is time- and memory-efficient for enumerating all k-frequent paths from a given compression tree of an edge-labelled ordered tree.

In this paper, for an edge-labelled ordered tree T, a subgraph of T having a tree structure is called a *subtree* of T. For a set S of edge-labelled ordered trees, we define a compact coding $Code(S)$ of S as a sequence consisting of a coding of a list of edge labels in S, a coding of a dictionary and a coding of a list of DFUDSs of all edge-labelled ordered trees in S. For a set S of n edge-labelled ordered trees and a real number σ $(0 < \sigma \leq 1.0)$, a subtree t is said to be σ-frequent if t appears in $\lceil n \times \sigma \rceil$ edge-labelled ordered trees or more in S. Third, when a compact coding $Code(S)$ of a set S of edge-labelled ordered trees and a real number σ $(0 < \sigma \leq 1)$ are given as input, using a rightmost expansion strategy [1], [18], we present an efficient algorithm, called EnuFreqSubtree, for enumerating all σ-frequent subtrees in S without decompressing $Code(S)$. Finally, using experimental results obtained by applying the implemented EnuFreqMaxPath and EnuFreqSubtree on a computer to a large synthetic structurally compressed edge-labelled ordered tree and a large synthetic structurally compressed set of edge-labelled ordered trees, respectively, we discuss the efficiencies of EnuFreqMaxPath and EnuFreqSubtree and show the advantage of enumerating all frequent paths and all frequent subtrees without decompression in structurally compressed edge-labelled ordered trees.

This paper is organized as follows. In Sec. II, we introduce a compression tree obtained by structurally compressing an edge-labelled ordered tree on the basis of an LZ compression scheme. We also define a succinct representation of a compression tree. In Sec. III, we give a formal definition of FreqMaxPathEnu-Problem and present an algorithm EnuFreqMaxPath for solving FreqMaxPathEnu-Problem. This section is a complete version of the paper in reference [6]. In Sec. IV, we give a formal definition of FreqSubtreeEnu-Problem and present an algorithm EnuFreqSubtree for solving FreqSubtreeEnu-Problem. In Sec. V, we explain the experimental results obtained by applying EnuFreqMaxPath and EnuFreqSubtree to synthetic large data and discuss the efficiencies of EnuFreqMaxPath and EnuFreqSubtree. In Sec. VI, we conclude this paper. This paper is an extended and complete version of our paper in reference [6].

II. Preliminaries

In this section, we introduce a compression tree that is a hypertree obtained from an edge-labelled ordered tree by replacing repeated occurrences of subtrees with hyperedges labelled with references to entries in a dictionary. A dictionary is a list of first occurrences of repeated subtrees in the tree's depth-first traversal. Moreover, we define a succinct representation of a compression tree by extending the depth-first unary degree sequence (DFUDS)[10], [12] for an edge-labelled ordered tree.

A. Compression Tree

Let Λ be a finite alphabet. An edge-labelled 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 $a \in \Lambda$. Hereafter, a *tree* means an edge-labelled ordered tree since we deal with only edge-labelled ordered trees in this paper. For a set S, we denote the number of elements in S as $|S|$. Let $w = w_1, w_2, ..., w_n$ be a sequence, $L = (\ell_1, \ell_2, \dots, \ell_k)$ a list, and $|w|$ and $|L|$ denote the numbers of elements, in w and L, respectively, i.e., $|w| = n$ and $|L| = k$. Moreover, for i, j $(1 \le i \le n,\ 1 \le j \le k)$, $w[i]$ and $L[j]$ denote the elements at i and j on w and L, respectively, i.e., $w[i] = w_i$ and $\ell[j] = \ell_j$.

For a tree T and its internal node u, we denote a 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 a tree T, a *reference* of T is a list $(v, v_1, v_2, \dots, v_n)$ of nodes satisfying the following conditions.

(1) For each i $(1 \le i \le n)$, v_i is a descendant of an internal node v of T.
(2) For any i, j $(1 \le i, j \le n)$, v_i is not a descendant of v_j and vice versa.
(3) For any i, j $(1 \le i < j \le n)$, 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 $L = (v, v_1, v_2, \dots, v_n)$ of T, we denote a subgraph induced by the node set $\bigcup_{w \in W} V(P_{v,w})$ as $T\langle L\rangle$, called 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 is not included in $V(T[v_i]) - \{v_i\}$ for any $1 \le i \le n$. In Fig. 1, for reference $(10, 13, 20, 21)$, we give the reference tree $t\langle(10, 13, 20, 21)\rangle = \big(\{10, 12, \dots, 21\}, \{(10, b, 12), (12, a, 13), \dots, (10, b, 21)\}\big)$. A subset $\mathcal{D}_T$ of $\mathcal{R}_T$ is called a *dictionary* of T if for any two distinct references $L_1 = (u, u_1, u_2, \dots, u_\ell)$ and $L_2 = (v, v_1, v_2, \dots, v_r)$ of $\mathcal{D}_T$, $(V(T\langle L_1\rangle) - \{u\}) \cap (V(T\langle L_2\rangle) - \{v\}) = \emptyset$ holds. For example, in Fig. 1, for references $(10, 13, 20, 21), (10, 25, 50, 65), (29, 32, 39, 40)$, and $(50, 53, 61, 63)$ in T, the reference trees $T\langle(10, 13, 20, 21)\rangle, T\langle(10, 25, 50, 65)\rangle, T\langle(29, 32, 39, 40)\rangle$, and $T\langle(50, 53, 61, 63)\rangle$ are isomorphic.

Let T be a tree. A list $(u, L, u_1, u_2, \dots, u_k)$ is called a *port list* of T if u is an internal node of T, $u_1, u_2, \dots, u_k$ are consecutive children of u and L is a reference of T such that $|L| = k+1$ holds. For a port list $h = (u, L_h, u_1, u_2, \dots, u_k)$, u is called a *parent port* of h, and each node u_i $(1 \le i \le k)$ is called a *child port* of h. Two port lists $h = (u, L_h, u_1, u_2, \dots, u_k)$ and $h' = (v, L_{h'}, v_1, v_2, \dots, v_\ell)$ of a tree T are said to be *disjoint* if the following conditions are satisfied.

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

Definition 1. (Hypertree) Let $T = (V_T, E_T)$ be a 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 *hypertree* 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) $E_t = E_T - \bigcup_{(v_0, L, v_1, \dots, v_r) \in H_T} \{(v_0, a_1, v_1), ..., (v_0, a_r, v_r)\}$, and the label of each edge $e \in E_T$ is preserved in E_t.
(3) $H_t = H_T$.

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

Definition 2. (Compression Tree) A *compression tree* of a tree T is a hypertree obtained from T by replacing repeated occurrences of subgraphs having ordered-tree structures with hyperedges labelled with the reference to the first occurrence of repeated occurred subgraphs.

Fig. 1 shows the compression tree $t = (\{3, 8, \dots, 49\}, \{(3, a, 8), (8, b, 10), \dots, (3, a, 49)\}, \{(8, P, 25, 36, 47), (27, P, 32, 33, 34), (36, P, 41, 43, 45)\})$ of the tree T, where P is the reference $(8, 11, 18, 19)$.

B. Compact Codings of Tree and Set of Trees

The DFUDS for an ordered tree T of n nodes is defined recursively as follows [10], [12]. The DFUDS of an ordered tree consisting of only one node is $\boxed{(}\,\boxed{)}$. The DFUDS of an ordered tree T that has k subtrees $T_1, \dots, T_k$ is a sequence of parentheses constructed by concatenating $k+1$ $\boxed{(}$s, one $\boxed{)}$ and k DFUDSs of $T_1, \dots, T_k$ 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 $2n$. It is known that the information-theoretic lower bound to represent an arbitrary tree of n nodes is $2n - o(n)$ bits. Hence, we can see that a DFUDS encoding of a tree is asymptotically close to the lower bound. 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.

However, 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-labelled ordered tree by 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 tree for a compression tree. For a compression tree $t = (V_t, E_t, H_t)$, an *underlying tree of* t is a tree obtained from t by applying the following replacements to all port lists of t. A port list $h = (u, L_h, u_1, u_2, \ldots, u_k) \in H_t$ is replaced with a tree in the following way.

(1) Remove h from t.
(2) Construct a tree $s = (\{u', v, u'_1, u'_2, \ldots, u'_k\}, E_s)$ defined as follows. (a) The node u' is the parent of v, and $u'_1, u'_2, \ldots, u'_k$ are the children of v that are in this order. (b) The edge between u' and v is labelled with the reference L_h of h, and any edge between v and its child is labelled 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 Representation) The succinct representation of a compression tree t is the DFUDS of the underlying tree of t, which is denoted as $DFUDS(t)$.

In Fig. 1, we give the DFUDS of the tree T and a succinct representation of the compression tree t, i.e. the DFUDS of t, as examples.

By using a succinct representation of a compression tree of a tree T, we can give a compact coding for a structured-compression of T as follows.

Definition 4. (Compact Coding) Let T be a tree. Let EL_T and D_T be the codings of the lists of all edge labels and all references in T, which are sorted by occurrence order in the depth-first traversal of T, respectively. For a compression tree t of T, we define a coding of t, denoted as C_T, as follows. For each index i $(0 \leq i < n_t)$,

$$C_T[i] = \begin{cases} 0 & \text{if } D_t[i] = \text{“(”}, \\ -1 & \text{if } D_t[i] = \text{“\$”}, \\ -|Ind(D_T, D_t[i]) + 2| & \text{if } D_t[i] \text{ is a reference}, \\ |Ind(EL_T, D_t[i]) + 1| & \text{if } D_t[i] \text{ is in } \Lambda, \end{cases}$$

where n_t is the number of nodes in the compression tree of T, $D_t = DFUDS(t)$ and, for a list Seq of sequences and a sequence α, $Ind(Seq, \alpha)$ is a function that returns the first index k with $Seq[k] = \alpha$ if it exists, otherwise "$\perp$". 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 that concatenates two sequences.

Since we create EL_T, D_T and C_T on the basis of the depth-first traversal of T, we can easily create hash functions that return the edge label or reference for each edge or hyperedge in the compression tree t of T. Let $Code(T) = \langle EL_T, D_T, C_T \rangle$ be a compact coding of T. A *length* of $Code(T)$, denoted as $|Code(T)|$, is defined as the length of the sequence $EL_T \circ$"!"$\circ D_T \circ$"!"$\circ C_T$. Fig. 1 shows the compact coding of the tree T.

III. Enumeration Algorithm for Finding All Frequent Maximal Paths

For a 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*. A k-frequent path p in T is *maximal* if there is no k-frequent pathp' in T such that p' has p as a subpath. We define an enumeration problem, denoted as FreqMaxPathEnu-Problem, for extracting all k-frequent maximal paths in a given compact coding of a tree as follows.

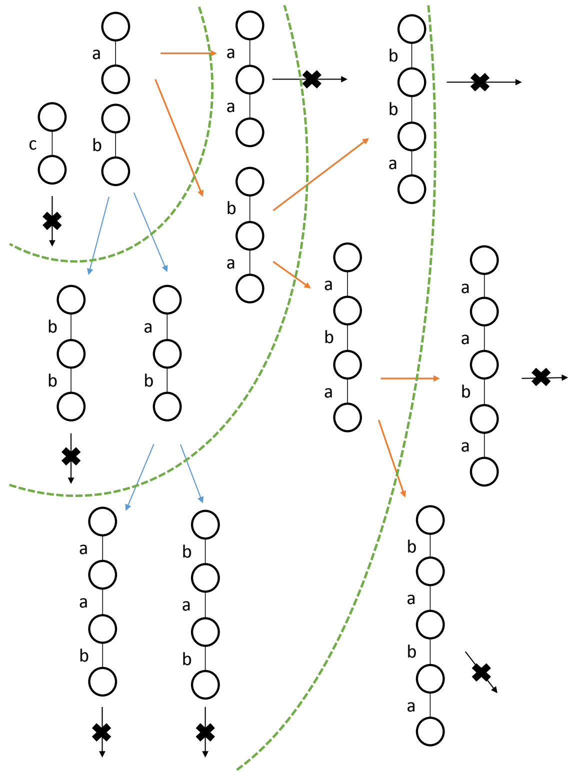

Fig. 2. Tree expressing numeration process of k-frequent paths

FreqMaxPathEnu-Problem

Instance: Compact coding $Code(T)$ of a tree T and minimum occurrence k $(k \geq 1)$.

Problem: Enumerate all k-frequent maximal paths in T without decompressing $Code(T)$.

In Algorithm 1, we present our enumeration algorithm, denoted as EnuFreqMaxPath, for solving FreqMaxPathEnu-Problem. For a sequence S of length n on alphabet Λ, a character c in Λ and an integer i $(0 \leq i \leq n-1)$, the two functions $rank$ and $select$ used in EnuFreqMaxPath are defined as follows.

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

By using the Succinct Data Structure Library (SDSL) [14], we can compute $rank$ and $select$ in constant time. When a compact coding $Code(T)$ of a tree T and a minimum occurrence k are given, EnuFreqMaxPath enumerates all k-frequent maximal paths in T from each node toward the root of T on a level-wise strategy with respect to the length of a k-frequent maximal path without decompressing $Code(T)$. EnuFreqMaxPath uses a trie structure to manage enumerated k-frequent paths from T. In Fig. 2, we give a tree expressing the enumeration process of k-frequent paths using EnuFreqMaxPath as an example.

EnuFreqMaxPath (Algorithm 1) has three procedures: GenOccPoint (Procedure 1), MakeCandFreqPath (Procedure 2) and MaximalCheck described later. 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 edge of the parent edge of e. By using the SDSL,

Algorithm 1 ENUFREQMAXPATH

Require: A compact coding $\langle EL_T, D_T, C_T\rangle$ of a tree T and a minimum occurrence k ($k \geq 1$).
Ensure: The set FM of all k-frequent maximal paths in T.
$Z_1 = $ GENOCCPOINT($\langle EL_T, D_T, C_T\rangle, k$)
$P_1 = \{a \mid (a, i, OP_i) \in Z_1\}$
$CFM = P_1$ **and** $ln = 1$
while $P_{ln} \neq \emptyset$ **do**
  $P_{ln+1} = \emptyset$
  $W_{ln+1} = $ MAKECANDFREQPATH(Z_{ln}, P_{ln}, P_1)
  for all $p \in P_{ln}$ **and** $a \in P_1$ **do**
    if $\sum_{(p\circ a, i, OP_i)\in W_{ln+1}} |OP_i| \geq k$ **then**
      $P_{ln+1} = P_{ln+1} \cup \{p \circ a\}$
      $Z_{ln+1} = Z_{ln+1} \cup \{(p \circ a, i, OP_i)\}$
    end if
  end for
  $CFM = CFM \cup P_{ln+1}$
  ln++
end while
$FM = $ MAXIMALCHECK(CFM)
return FM

Procedure 1 GENOCCPOINT

Require: The compact coding $\langle EL_T, D_T, C_T\rangle$ and a minimum occurrence k ($k \geq 1$).
Ensure: The set Z_1 of all path-count triplets of k-frequent paths whose length is 1.
$Z_1 = \emptyset$ **and** $W_1 = \emptyset$ **and** $OP_i = \emptyset$ for i ($0 \leq i < |C_T|$)
for all i ($0 \leq i < |C_T|$) **do**
  if $C_T[i] \in EL_T$ **then**
    $OP_i = OP_i \cup \{i\}$
  end if
  if $C_T[i] \in D_T$ **then**
    for all an edge j of reference tree $T\langle C_T[i]\rangle$ **do**
      $OP_j = OP_j \cup \{i\}$
    end for
  end if
end for
for all i ($0 \leq i < |C_T|$) such that $C_T[i] \in EL_T$ **do**
  $W_1 = W_1 \cup \{(C_T[i], i, OP_i)\}$
end for
for all $a \in EL_T$ **do**
  if $\sum_{(a,i,OP_i)\in W_1} |OP_i| \geq k$ **then**
    $Z_1 = Z_1 \cup (a, i, OP_i)$
  end if
end for
return Z_1

Procedure 2 MAKECANDFREQPATH

Require: A set Z_{ln} of path-count triplets, each of which has a path with length ln, a set P_{ln} of k-frequent paths whose lengths are ln and a set P_1 of k-frequent edges.
Ensure: The set W_{ln+1} of path-count triplets, each of which has a path with length $ln + 1$.
$OP'_i = \emptyset$ for i ($0 \leq i < |C_T|$)
$U = \emptyset$ **and** $W_{ln+1} = \emptyset$
for all $p \in P_{ln}$ **do**
  for all $(p, i, OP_i) \in Z_{ln}$ **do**
    if $(rank(i) + 1, v_1, ...v_k) \in D_T$ **then**
      for all $w \in OP_i$ **do**
        $u = parent(w)$
        if $C_T[u] = -1$ **then**
          /* case 1 */
          $d_u = D_T[|C_T[parent(u)] + 2|]$
          $q = d_u[childrank(u)]$
          $U = U \cup \{q\}$
          $OP'_q = OP'_q \cup \{parent(u)\}$
        else
          /* case 2 */
          $U = U \cup \{u\}$
          $OP'_u = OP'_u \cup \{u\}$
        end if
      end for
    else if $C_T[parent(i)] = -1$ **then**
      /* case 3 */
      for all $w \in OP_i$ **do**
        $j = parent(i)$
        $d_j = D_T[|C_T[parent(j)] + 2|]$
        $u = d_j[childrank(j)]$
        $U = U \cup \{u\}$
        $OP'_u = OP'_u \cup \bigcup_{w\in OP_i} \{parent(j)\}$
      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**
  $W_{ln+1} = W_{ln+1} \cup \{(p \circ C_T[w], w, OP'_w)\}$
end for
return W_{ln+1}

these operations on a compression tree can be executed in constant time.

To count the number m of paths, which are isomorphic to a path p and are obtained by appending the edge at an index i of C_T, ENUFREQMAXPATH uses a triplet (p, i, OP_i), where OP_i is a multi-set of indexes in the interval $[0, |C_T|)$ and satisfies $|OP_i| = m$. 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 a tree T and a minimum occurrence k ($k \geq 1$), ENUFREQMAXPATH first generates the set Z_1 of path-count triplets of k-frequent edges from all indexes between 0 and $|C_T|$. Then, it constructs the set P_1 of all k-frequent edges from Z_1. That is, P_1 is the set of all k-frequent paths, and the length of each is 1. Second, by using Procedure MAKECANDFREQPATH, ENUFREQMAXPATH recursively generates the set W_{ln+1} of all path-count triplets having candidate paths, and the length of each is $ln + 1$ from the set P_1 and the set P_{ln}, each of which has a path whose length is ln. From W_{ln+1}, ENUFREQMAXPATH constructs the set P_{ln+1} of k-frequent paths, and the length of each is $ln + 1$ and set $Z_{ln+1} = \{(p, i, OP_i) \in W_{ln+1} \mid p \in P_{ln+1}\}$.

Third, ENUFREQMAXPATH constructs the set CFM of

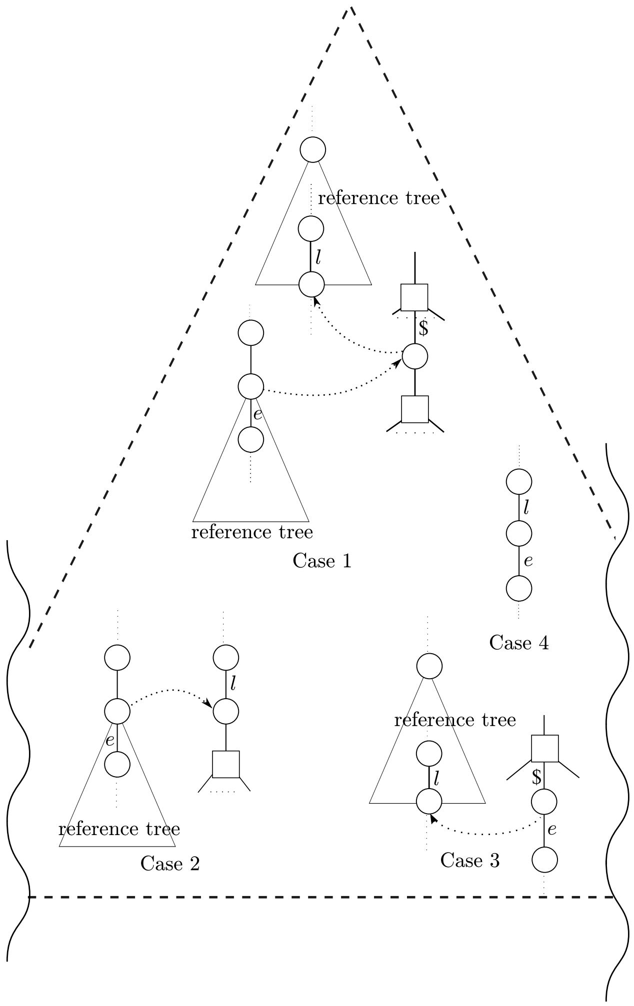

Fig. 3. Illustration of four cases of extending paths

all k-frequent paths appearing in T from t. Next, ENUFREQMAXPATH obtains the set FM of all k-frequent maximal paths from CFM by applying MAXIMALCHECK described later to CFM. Finally, ENUFREQMAXPATH returns FM and terminates.

In Procedure MAKECANDFREQPATH, when we construct a candidate path whose length is $ln+1$ by extending the k-frequent path whose length is ln, four cases are considered (see Fig. 3). If we try to extend 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 $(43, a, 44)$) to index 3 (the root of t). Since a reference having index 44 as the first argument does not exist in D_T, we determine whether or not $DFUDS(t)[parent(44)]$ is "\$". From $parent(44) = 43$, $DFUDS(t)[parent(44)] = DFUDS(t)[43]$ ="\$". We can see that the extension of the path from the edge at index 44 is case 3. Hence, we obtain $j = 43$, $d_j = D_T[|C_T[parent(43)] + 2|] = D_T[|C_T[40] + 2|] = D_T[0] = (8, 11, 18, 19)$, $u = P[childrank(43)] = P[2] = 18$ and $OP'_{18} = OP'_{18} \cup \{40\}$, where P is the reference

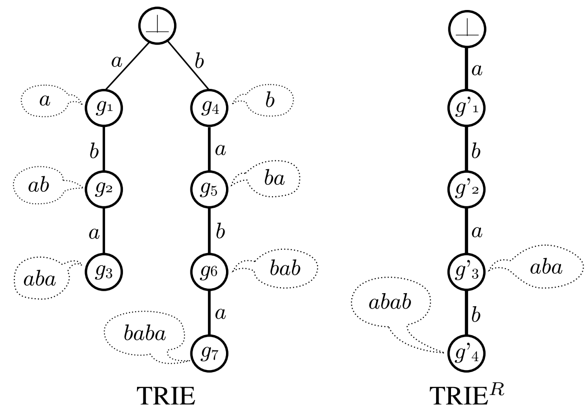

Fig. 4. TRIE and TRIE^R constructed by MAXIMALCHECK as data structures

$(8, 11, 18, 19)$. This is the extension of case 1, because $parent(14)$ is the root of the reference tree corresponding to $DFUDS(t)[40]$ = "P" and $DFUDS(t)[parent(40)] = DFUDS(t)[36]$ ="\$". By applying this extension to the path, we obtain $u = 36$, $d_u = D_T[|C_T[parent(36)] + 2|] = D_T[|C_T[23] + 2|] = D_T[0] = (8, 11, 18, 19)$, $q = P[childrank(36)] = P[2] = 18$ and $OP'_{18} = OP'_{18} \cup \{23\}$, where "$P$"$= DFUDS(t)[parent(36)] = DFUDS(t)[23]$. Finally, we can obtain the path "$ababaa$" from index 44 to the root.

For a sequence $w = w_1, w_2, \ldots, w_k$, the reverse sequence of w is denoted as $w^R = w_k, \ldots, w_2, w_1$. For a set $W = \{p_1, p_2, \ldots, p_n\}$ of sequences, let $W^R = \{p_1^R, p_2^R, \ldots, p_n^R\}$. To manage extracted frequent paths, Algorithm ENUFREQMAXPATH uses a data structure that is represented by a tree, called *TRIE*. If a path p whose length is less than k is a subpath of a path w whose length is k, there exists a subpath x and y of w such that $w = xpy$ and $|x| + |y| > 0$ hold. In a TRIE storing the set CFM, a path stored in a leaf of TRIE is not always maximal. If a path p is a subpath of a path $w = py$ ($y \in \Lambda^+$), the node storing p is on the path from the node storing w to the root; that is, the node storing p is not a leaf of TRIE. However, if a path p is a subpath of a path $w' = xp$ ($x \in \Lambda^+$), the node storing p may not be on the path from the node storing w' to the root of TRIE; that is, the node storing p and the node storing w' may be leaves in TRIE. Hence, Procedure MAXIMALCHECK constructs a set of all frequent maximal paths from CFM as follows. Procedure MAXIMALCHECK makes the set, denoted as $PATH$, of all paths stored in leaves of TRIE and constructs a trie, denoted as TRIE^R, that manages the set $PATH^R$. Then, Procedure MAXIMALCHECK selects all frequent maximal paths by gathering all paths stored in leaves of TRIE^R and outputs the set FM of all frequent maximal paths. For example, we consider TRIE managing a set $\{a, b, ab, ba, aba, bab, baba\}$ of paths in Fig. 4. Since $PATH = \{aba, baba\}$ is obtained from TRIE, we can see that TRIE^R in Fig. 4 can be constructed to manage $PATH^R = \{aba, abab\}$. Then, Procedure MAXIMALCHECK outputs the set $\{baba\}$ of the reverse sequence of the path "$abab$" stored in the leaf g'_4 of TRIE^R.

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

$$Z_1 = \left\{ \begin{array}{l} (a, 8, \{8\}), (a, 11, \{11, 23, 31, 40\}), \\ (a, 14, \{14, 23, 31, 40\}), (a, 17, \{17, 23, 31, 40\}), \\ (a, 44, \{44\}), (a, 48, \{48\}), (a, 49, \{49\}), \\ (b, 10, \{10, 23, 31, 40\}), (b, 16, \{16, 23, 31, 40\}), \\ (b, 18, \{18, 23, 31, 40\}), (b, 19, \{19, 23, 31, 40\}), \\ (b, 27, \{27\}) \end{array} \right\}.$$

Moreover, ENUFREQMAXPATH generates $P_1 = \{a, b\}$ as the set of 5-frequent paths whose length is 1. By recursively applying Procedure MAKECANDFREQPATH, ENUFREQMAXPATH generates the path-count triplets $Z_4 = \{(baba, 8, \{8\}), (baba, 11, \{23, 23\}), (baba, 14, \{23, 23\})\}$ and $P_4 = \{baba\}$. Since t has no 5-frequent path whose length is 5, by selecting 5-frequent maximal paths from the set $CFM = \{a, b, ab, ba, aba, bab, baba\}$, ENUFREQMAXPATH terminates after outputting the $FM = \{baba\}$ of all 5-frequent paths in t.

IV. ENUMERATION ALGORITHM FOR FINDING ALL FREQUENT SUBTREES

For a set S of trees and a real number σ $(0 < \sigma \leq 1)$, a tree T is σ-frequent with respect to S if the number of trees in S having T as subgraphs is greater than or equal to $\lceil |S| \times \sigma \rceil$. Such a real number σ is called a *minimum support rate*. For a set S of trees, a *corresponding tree* for S is a tree T_S whose root has the root of each tree in S as a child and whose edge from the root of T_S to the root of each tree in S is labelled with the special symbol ε not in Λ. Then, a *compact coding* of S is defined as a compact coding $Code(T_S) = \langle EL_{T_S}, D_{T_S}, C_{T_S} \rangle$ of T_S. In Fig. 5, we give a corresponding tree T_S for the set $S = \{T_1, T_2, T_3\}$ as an example of a corresponding tree.

We define an enumeration problem, denoted as FREQSUBTREEENU-PROBLEM, for extracting all σ-frequent trees in a compact coding for a given set of trees.

FREQSUBTREEENU-PROBLEM

Instance: Compact coding $Code(S)$ for a set S of trees and a real number σ $(0 < \sigma \leq 1)$.

Problem: Enumerate all σ-frequent trees with respect to S without decompressing $Code(S)$.

In Algorithm 2, we present an algorithm, denoted as ENUFREQSUBTREE, for solving FREQSUBTREEENU-PROBLEM. Using the rightmost expansion strategy [9], [1], [18] for trees, ENUFREQSUBTREE enumerates all σ-frequent trees appearing in an input compact coding $Code(S)$ without decompressing $Code(S)$. The rightmost expansion strategy is a strategy that makes a candidate tree whose length is $k+1$ from a σ-frequent tree T by expanding a new edge from a node on the rightmost path of T. ENUFREQSUBTREE uses a trie structure to manage enumerated σ-frequent trees on the basis of the rightmost expansion strategy. In Fig. 6, we give a trie describing the enumeration process of σ-frequent trees with ENUFREQSUBTREE as an example.

From the definition of the frequency of a subtree in a set S of trees on FREQSUBTREEENU-PROBLEM, to determine the frequency of a subtree t appearing at an index i $(0 \leq i < |C_{T_S}|)$ in a coding C_{T_S} of a compression tree of T_S such that $Code(T_S) = \langle EL_{T_S}, D_{T_S}, C_{T_S} \rangle$ holds, it

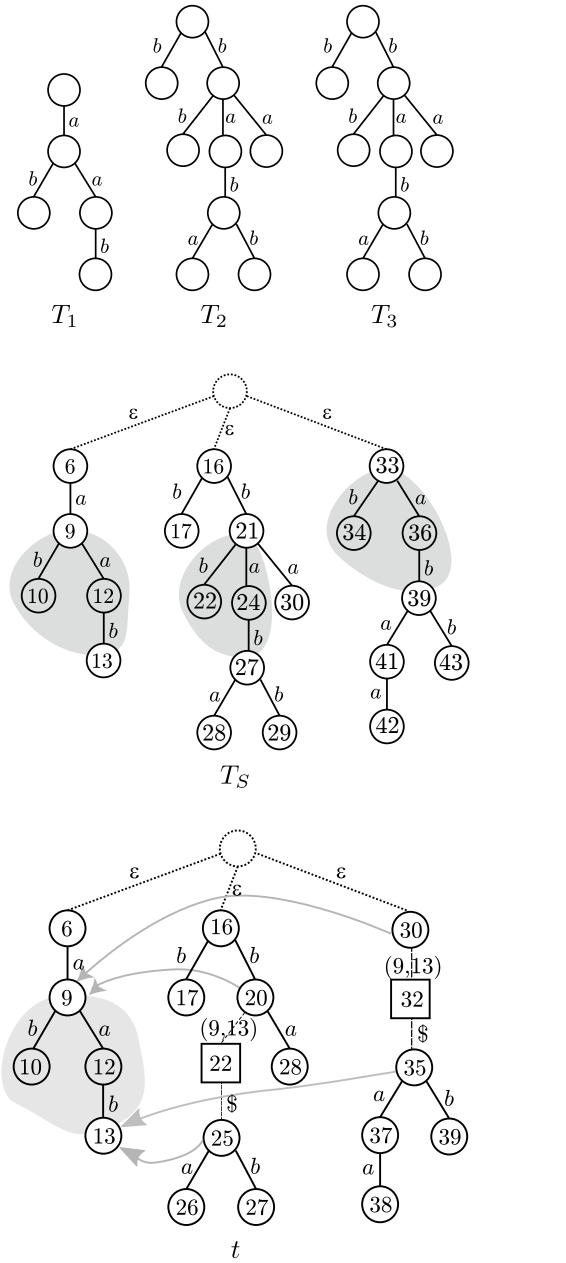

Fig. 5. Compression tree t of corresponding tree T_S for set $S = \{T_1, T_2, T_3\}$

is not necessary to count the number of subtrees that are isomorphic to t and that appear at index i in C_{T_S}, and it is only necessary to decide whether or not t appears at index i in C_{T_S}. Therefore, ENUFREQSUBTREE uses a triplet (p, i, OP_i), called an *occurrence point of* p *in* C_{T_S}, which means that, for each $k \in OP_i$, if $k = i$, the rightmost leaf of the subtree p appears at index i in C_{T_S}; otherwise, the rightmost leaf of the subtree p appears at index i in the reference tree corresponding to the reference $C_{T_S}[k]$. For example, the subtree f_7 in Fig. 6 appears in the compression tree t in Fig. 5 at indexes 12, 26, 28 and 37. Hence, the occurrence points of f_7 in C_{T_S} are $(f_7, 12, \{22\}), (f_7, 26, \{26\}), (f_7, 28, \{28\}), (f_7, 37, \{37\})$. ENUFREQSUBTREE (Algorithm 2) has two procedures: GENOCCPOINTSUBTREE (Procedure 3) and MAKECANDFREQSUBTREES (Procedure 4). Given an input set Z_{ln} of occurrence points of σ-frequent

Algorithm 2 ENUFREQSUBTREE

Require: A compact coding $\langle EL_{T_S}, D_{T_S}, C_{T_S}\rangle$ of a set S of trees and an integer σ $(0 < \sigma \leq 1.0)$.
Ensure: The set F of all σ-frequent trees in S.

$Z_1 = \text{GENOCCPOINTSUBTREE}(\langle EL_{T_S}, D_{T_S}, C_{T_S}\rangle, \sigma)$
$P_1 = \{p \mid (p, i, OP_i) \in Z_1\}$
$F = P_1$ **and** $ln = 1$
while $P_{ln} \neq \emptyset$ **do**
  $P_{ln+1} = \emptyset$
  $W_{ln+1} = \text{MAKECANDFREQSUBTREES}(Z_{ln}, Z_1)$
  for all $z = (p, i, OP_i) \in W_{ln+1}$ **do**
    if p is σ-frequent **then**
      $P_{ln+1} = P_{ln+1} \cup \{p\}$
      $Z_{ln+1} = Z_{ln+1} \cup \{z\}$
    end if
  end for
  $F = F \cup P_{ln+1}$
  ln++
end while
return F

Procedure 3 GENOCCPOINTSUBTREE

Require: The compact coding $\langle EL_T, D_T, C_T\rangle$ and a minimum support rate σ $(0 < \sigma \leq 1.0)$.
Ensure: The set Z_1 of all occurrence points of σ-frequent edge.

$Z_1 = \emptyset$ **and** $W_1 = \emptyset$ **and** $OP_i = \emptyset$ for i $(0 \leq i < |C_T|)$
for all i $(0 \leq i < |C_T|)$ **do**
  if $C_T[i] \in EL_T$ **then**
    $OP_i = OP_i \cup \{i\}$
  end if
  if $C_T[i] \in D_T$ **then**
    for all an edge j of reference tree $T\langle C_T[i]\rangle$ **do**
      $OP_j = OP_j \cup \{i\}$
    end for
  end if
end for
for all i $(0 \leq i < |C_T|)$ such that $C_T[i] \geq 2$ **do**
  $W_1 = W_1 \cup (p, i, OP_i)$
  /* p is a tree consisting of an edge with label $EL_T[C_T[i]]$ */
end for
for all $z = (p, i, OP_i) \in W_{ln+1}$ **do**
  if p is σ-frequent **then**
    $Z_{ln+1} = Z_{ln+1} \cup \{z\}$
  end if
end for
return Z_1

subtrees in T_S whose sizes are ln and an input set Z_1 of occurrence points of σ-frequent edges in T_S, MAKECANDFREQSUBTREES returns the set Z_{ln+1} of all occurrence points of σ-frequent subtrees whose sizes are $ln + 1$. In Procedure MAKECANDFREQSUBTREES, when we construct the set Z_{ln+1} from Z_{ln} and Z_1 on the basis of the rightmost expansion strategy, for an occurrence point $(p, i, OP_i) \in Z_{ln+1}$ and $k \in OP_i$ in T_S, the following four cases with respect to indexes j and i of the expanded edge e and its parent edge must be considered, respectively.

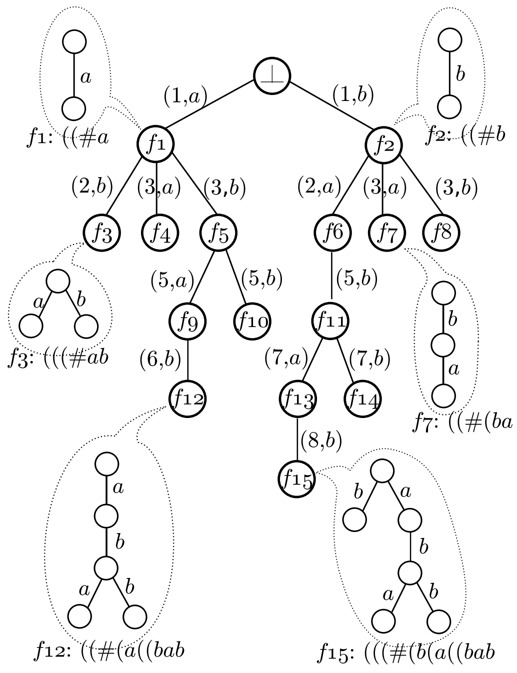

Fig. 6. TRIE F describing enumeration process of $\frac{2}{3}$-frequent subtrees. Symbol described in node denotes $\frac{2}{3}$-frequent subtree detected by path from node to root. Some subtrees and their DFUDSs are described around nodes managing them.

Case 1: Index k is not equal to i; that is, index i is in the reference tree corresponding to the reference $C_{T_S}[k]$, and index j is in a reference tree different from $C_{T_S}[k]$.

Case 2: Index k is not equal to i, and there exist no reference trees in which $C_{T_S}[j]$ is.

Case 3: Index k is equal to i, and there exists a reference tree in which $C_{T_S}[j]$ is.

Case 4: Index k is equal to i and $C_{T_S}[j]$ is in Λ, or there exists a reference tree in which $C_{T_S}[i]$ and $C_{T_S}[j]$ are.

These four cases are illustrated in Fig. 7. In MAKECANDFREQSUBTREES, for a reference $L = (v_0, v_1, \ldots, v_n)$ and an index i, $Arg(L, i) = k$ if $v_k = i$ $(0 \leq k \leq n)$; otherwise, $Arg(L, i) = -1$.

When a candidate tree t that has $ln+1$ edges is generated from a frequent tree s that has ln edges by the procedure MAKECANDFREQSUBTREES, we can see that the DFUDS of t can be made by inserting $\boxed{(}$ into an appropriate index and appending the edge label to the end of the DFUDS of s. To manage all frequent subtrees enumerated, ENUFREQSUBTREE use a trie structure, denoted as TRIE, which stores each frequent subtree in a path of TRIE. As an example of TRIE, we give an ordered tree F describing the enumeration process of $\frac{2}{3}$-frequent subtrees. For example, in Fig. 6, the edge label $(2, b)$ of the edge between the nodes f_1 and f_3 means that the DFUDS "(((#ab" of the subtree f_3 drawn near the node f_3 is obtained from the DFUDS "((#a" of the tree f_1 by inserting $\boxed{(}$ at index 2 and appending "b" to the end of "((#a". TRIE F stores the frequent subtree f_3 in the path from the root whose DFUDS is "(((#ab" to the node f_3. We explain the enumeration process of σ-frequent subtrees obtained by

Procedure 4 MAKECANDFREQSUBTREES

Require: The set Z_{ln} of occurrence points of σ-frequent subtrees whose lengths are ln and the set Z_1 of occurrence points of σ-frequent edges.

Ensure: A set W_{ln+1} of occurrence points of candidate subtrees whose lengths are $ln+1$.

for all $f \in P_{ln}$ **do**
  /* P_{ln} is the set of σ-frequent subtrees whose lengths are ln */
  $OP'_i = \emptyset$ for i $(0 \leq i < |C_T|)$
  $U = \emptyset$
  for all $(f, i, OP_i) \in Z_{ln}$ **do**
    for all $w \in OP_i$ **do**
      for all $Path^f_w$ 上の辺 e **do**
        /* $Path^f_w$ is the rightmost path from the edge at the index w to the edge corresponding to the root */
        if $e = w$ **then** $index = 0$, **otherwise** $index = childrank(c) + 1$ **end if**
        /* c is the child of e on $Path^f_w$ */
        for $r = index$ to $r < degree(e)$ **do**
          $u = child(e, r)$
          if $\exists L \in D_T$ s.t. $Arg(L, i) \geq 1$ **then**
            $dollar = child(w, Arg(L, i))$
            for $r' = 0$ to $r' < degree(dollar)$ **do**
              $u' = child(dollar, r')$
              if $C_T[u'] < -1$ **then**
                /* case 1 */
                $d_w = D_T[|C_T[w] + 2|]$
                $p = d_w[0]$
                for $r" = 0$ to $r" < degree(p)$ **do**
                  $u" = child(p, r")$
                  $U = U \cup \{(f'_e, u")\}$
                  /* f'_e is a subtree obtained by the rightmost expansion at the index e */
                  $OP'_{u"} = OP'_{u"} \cup \{u'\}$
                end for
              else
                /* case 2 */
                $U = U \cup \{(f'_e, u')\}$
                $OP'_{u'} = OP'_{u'} \cup \{u'\}$
              end if
            end for
          else
            if $C_T[u] < -1$ **then**
              /* case 3 */
              $d_w = D_T[|C_T[w] + 2|]$
              $p = d_w[0]$
              for $r' = 0$ to $r' < degree(p)$ **do**
                $u' = child(p, r')$
                $U = U \cup \{(f'_e, u')\}$
                $OP'_{u'} = OP'_{u'} \cup \{u\}$
              end for
            else
              /* case 4 */
              $U = U \cup \{(f'_e, u)\}$
              if $i \neq w$ **then** $OP'_u = OP'_u \cup \{w\}$, **otherwise** $OP'_u = OP'_u \cup \{u\}$ **end if**
            end if
          end if
        end for
      end for
    end for
  end for
  for all $(f', w) \in U$ **do** $W_{ln+1} = W_{ln+1} \cup \{(f', w, OP'_w)\}$ **end for**
end for
return W_{ln+1}

applying a compact coding $Code(S) = \langle EL_S, D_S, C_S \rangle$ of $S = \{T_1, T_2, T_3\}$ in Fig. 5 to ENUFREQSUBTREE as a running example. Let t be the compression tree of $Code(S)$ in Fig. 5. We set σ to $\frac{2}{3}$. Let F be a trie, denoted by TRIE, having only one node labelled with $\perp$. In line 1 of Algorithm ENUFREQSUBTREE, when the compact coding $Code(S) = \langle EL_S, D_S, C_S \rangle$ and the minimum support rate σ are given, GENOCCPOINTSUBTREE outputs the set of the occurrence points of all $\frac{2}{3}$-frequent subtrees whose size is one as follows. By executing the for loop from lines 2 to 11 of Procedure GENOCCPOINTSUBTREE, for each index i $(0 \leq i < |C_S|)$, GENOCCPOINTSUBTREE creates the set OP_i of indexes that represent subtrees consisting of one edge. That is, $OP_6 = \emptyset, OP_9 = \{9\}, OP_{10} = \{10, 22, 32\}, \ldots, OP_{16} = \emptyset, \ldots, OP_{39} = \{39\}$. Let f_1 and f_2 be subtrees consisting of one edge labelled with a or b, respectively. Next, by executing the for loop from 12 to 14 of Procedure GENOCCPOINTSUBTREE, for each index i $(0 \leq i < |C_S|)$, GENOCCPOINTSUBTREE creates the set W_1 of occurrence points of all subtrees consisting of one edge. That is, $W_1 = \{(f_1, 9, \{9\}), (f_1, 12, \{12, 22, 32\}), \ldots, \ldots, (f_2, 39, \{39\})\}$. Here, we remark that $(f_2, 10, \{10, 22, 32\})$ in W_1. Since the edge $(9, a, 10)$ is in the reference tree referenced by the reference $(9, 13)$, indexes 22 and 32 of the port lists having the reference $(9, 13)$ in the DFUDS of t are added to the occurrence point $(f_2, 10, \{10\})$. In the same way as the edge $(9, b, 10)$, for the edges $(9, a, 12)$ and $(12, b, 13)$, we also remark that $(f_1, 12, \{12, 22, 32\})$ and $(f_2, 13, \{13, 22, 32\})$ in W_1. Then, by executing the for loop from 15 to 19 of Procedure GENOCCPOINTSUBTREE, from W_1, GENOCCPOINTSUBTREE creates the set Z_1 of all occurrence points of the rightmost leaves of $\frac{2}{3}$-frequent subtrees f_1 and f_2. That is,

$$Z_1 = \left\{ \begin{array}{l} (f_1, 9, \{9\}), (f_1, 12, \{12, 22, 32\}), \\ (f_1, 26, \{26\}), (f_1, 28, \{28\}), (f_1, 37, \{37\}), \\ (f_1, 38, \{38\}), (f_2, 10, \{10, 22, 32\}), \\ (f_2, 13, \{13, 22, 32\}), (f_2, 17, \{17\}), \\ (f_2, 20, \{20\}), (f_2, 27, \{27\}), (f_2, 39, \{39\}) \end{array} \right\}.$$

As a result, GENOCCPOINTSUBTREE outputs Z_1. To manage $\frac{2}{3}$-frequent subtrees f_1 and f_2 in P_1 by TRIE, EN-

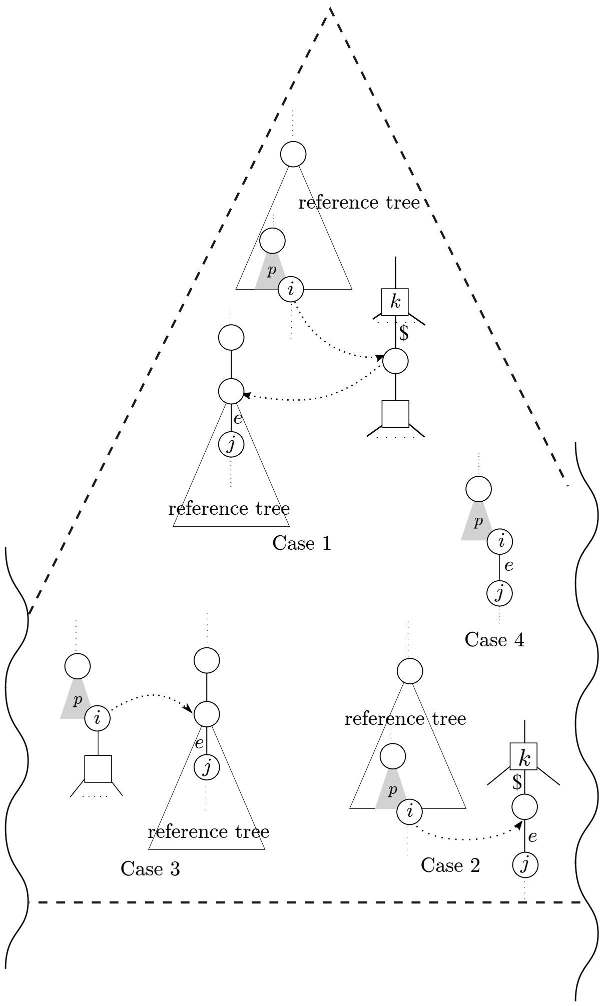

Fig. 7. Illustration of four cases of expanding subtrees

uFreqSubtree appends two edges $(\perp, (1, a), f_1)$ and $(\perp, (1, b), f_2)$ from the node $\perp$ in TRIE. For each edge label $\alpha \in \{a, b\}$, the DFUDS "((#α" of the tree f in Fig. 6 is generated by inserting $\boxed{(}$ into index 1 of DFUDS "(#" and appending the edge label α to the end of DFUDS "((#". Hence, the nodes f_1 and f_2 in TRIE correspond to the subtrees f_1 and f_2 whose DFUDSs are "((#a" and "((#b" by using the edge labels (1, a) and (1,b), respectively.

Next, by using a running example on the compression tree t of corresponding tree T_S in Fig. 5 and TRIE F in Fig. 6, we explain the making process of the set Z_2 of occurrence points of all $\frac{2}{3}$-frequent subtrees with two edges obtained from Z_1 by executing the while loop from lines 4 to 15 in Algorithm EnuFreqSubtree. In line 6 of EnuFreqSubtree, MakeCandFreqSubtrees creates the set W_2 of occurrence points of all subtrees generated by applying the rightmost expansion to each subtree in P_1 in the following way. Let U be a set of pairs, each of which consists of a candidate frequent subtree f and an index at which the rightmost leaf of f occurs.

As an example of the rightmost expansion in Case 4, consider the rightmost expansion of f_1 appearing at index 12 of t for the occurrence point $(f_1, 12, \{12, 22, 32\}) \in Z_1$. Since edge 9 has no child as the next sibling of edge 12 but edge 12 has child 13, the pair $(f_5, 13)$ is added to U, where f_5 is a tree isomorphic to the subtree $T_S[\{(9, a, 12), (12, b, 13)\}]$ of T_S. Moreover, since index 12 is in the reference tree corresponding to the reference $(9, 13)$ at indexes 22 and 32, the occurrence point $(f_5, 13, \{13, 22, 32\})$ is added to W_2.

As an example of the rightmost expansion in Case 2, consider the rightmost expansion of f_2 appearing at index 13 of t for the occurrence point $(f_2, 13, \{13, 22, 32\})$ in Z_1. Since the edge at index 13 has no child but index 13 is in the reference tree corresponding to the reference $(9, 13)$ at indexes 22 and 32, the pairs $(f_7, 26)$ and $(f_7, 37)$ are added to U, and the occurrence points $(f_7, 26, \{26\})$ and $(f_7, 37, \{37\})$ are added to W_2, where f_7 is a tree isomorphic to the subtree $T_S[\{(24, b, 27), (27, a, 28)\}]$ of T_S.

As an example of the rightmost expansion in Case 3, consider the rightmost expansion of f_2 appearing at index 20 of t for the occurrence point $(f_2, 20, \{20\}) \in Z_1$. Since the edge at index 20 has the edge at index 22 that is labelled with the reference $(9, 13)$ as a child, the pair $(f_8, 10)$ is added to U, and the occurrence point $(f_8, 10, \{22\})$ is added to W_2, where f_8 is a tree isomorphic to the subtree $T_S[\{(24, b, 27), (27, b, 29)\}]$ of T_S.

By applying the rightmost expansions for other occurrence points in Z_1, EnuFreqSubtree finally obtains $U = \{ (f_3, 27), (f_3, 39), (f_4, 12), \ldots, (f_5, 13), \ldots, (f_7, 26), (f_7, 37), \ldots, (f_8, 10), \ldots, (f_8, 39), (f', 28)\}$ and $W_2 = \{(f_3, 27, \{27\}), (f_3, 39, \{39\}), (f_4, 12, \{12\}), \ldots, (f_5, 13, \{13, 22, 32\}), \ldots, (f_7, 26, \{26\}), (f_7, 37, \{37\}), \ldots, (f_8, 10, \{22\}), \ldots, (f_8, 39, \{39\}), (f', 28, \{28\})\}$, where f' is the subtree isomorphic to the subtree $T_S[\{(21, a, 24), (21, a, 30)\}]$ in T_S in Fig. 5 and f_3, f_4, f_5, f_6, f_7 and f_8 are the subtrees corresponding to the nodes f_3, f_4, f_5, f_6, f_7 and f_8 described in F in TRIE F in Fig. 6, respectively. By executing the for loop from lines 7 to 12 in EnuFreqSubtree, we obtain $P_2 = \{f_3, f_4, \ldots, f_8\}$ and $Z_2 = W_2 - \{(f', 28, \{28\})\}$ because the subtree f' is not $\frac{2}{3}$-frequent.

In the same way as the construction of Z_2, for each i ($i \geq 3$), EnuFreqSubtree recursively constructs the set Z_i of occurrence points of all $\frac{2}{3}$-frequent subtrees with i edges.

V. Experimental Results and Discussion

In this section, we discuss the efficiencies of EnuFreqMaxPath and EnuFreqSubtree by explaining the experimental results obtained by applying EnuFreqMaxPath and EnuFreqSubtree implemented on a PC to synthetic large data, which were randomly generated.

A. Experimental Environments and Synthetic Data Set

We implemented EnuFreqMaxPath and EnuFreqSubtree on a computer with macOS 10.12 Sierra, and 32 GB of memory and a 4 GHz Intel Core i7 by using C++. In implementing EnuFreqMaxPath and EnuFreqSubtree, we used the SDSL [14] to implement operations on succinct data structures for compression trees.

Let $Code(T) = \langle EL_T, D_T, C_T \rangle$ be a compact coding of a tree T. A *compression ratio* of $Code(T)$ is defined as $\dfrac{|Code(T)|}{|T|}$, where $|T|$ denotes the length of the DFUDS of T. Moreover, a *compression tree size* of $Code(T)$ is defined as the half length of C_T, i.e.,$|C_T|/2$. Let N be an integer in $\{1000, 2000, 3000, 4000, 5000, 6000, 7000\}$,

r a compression ratio in $(0, 1.0]$ and K an integer in $\{100, 200, 300, 400, 500\}$. For each i $(1 \leq i \leq 100)$, we randomly created the compact codings $Code(T_i) = \langle EL_{T_i}, D_{T_i}, C_{T_i} \rangle$ satisfying the following conditions (1)-(4).

(1) The compression tree size of $Code(T_i)$ is about N.
(2) A compression ratio of $Code(T_i)$ is about r.
(3) The number of edge-labels of T_i is 3, i.e., $|EL_{T_i}| = 3$.
(4) The number of references in $Code(T_i)$ is at most 5, i.e., $|D_T| \leq 5$.

Then, in the experiments, the set $C_r(N) = \{Code(T_1), Code(T_2), \ldots, Code(T_{100})\}$ was used as synthetic data. Moreover, $D(C_r(N))$ denotes the set of 100 trees obtained from $C_r(N)$ by decompressing all compact codings in $C_r(N)$.

Let $S = \{T_1, T_2, \ldots, T_K\}$ be a set of K trees. A *compression ratio* of S is defined as $\dfrac{|Code(S)|}{\sum_{k=1}^{K} |DFUDS(T_k)|}$. We randomly created a compact coding $Code(T_S) = \langle EL_{T_S}, D_{T_S}, C_{T_S} \rangle$ of a corresponding tree T_S for S such that T_S satisfies the following conditions (1)-(4).

(1) The number of nodes in T_S is about $N \times K + 1$.
(2) A compression ratio of $Code(T_S)$ is about r.
(3) The number of edge-labels in T_S is just 3, i.e., $|EL_{T_S}|$=3.
(4) The number of references in D_{T_S} is at most $5 \times K$.

To clearly show K, r and N, $Code(T_S)$ is denoted as $Code_r^K(N) = \langle EL_S, D_S, C_S \rangle$, and T_S is denoted as $D(Code_r^K(N))$.

B. *Experimental Results of* ENUFREQMAXPATH *for Solving* FREQMAXPATHENU-PROBLEM

We measured the running times needed to solve FREQMAXPATHENU-PROBLEMS for $C_{50}^{100}(N)$ and $D(C_{50}^{100}(N))$ by using the implemented ENUFREQMAXPATH while varying the value of a compression tree size N from 1000 to 7000, respectively. The number of nodes in T is called a *decompression size* of $Code(T)$. Figs. 8, 9 and 10 show experimental results with respect to the following three items (a)-(c) for a compact coding $d \in C_{50}^{100}(N)$.

(a) The running times vs. the compression tree size and decompression size of d.
(b) The running time vs. the number of occurrence points of all frequent paths appearing in the tree obtained by decompressing d.
(c) The running time vs. the number of all frequent maximal paths in the tree obtained by decompressing d.

Because, in this experimental setting, if the compression tree size N increased, both the decompression size of each compact coding and the number of occurrence points of all frequent paths increased in general, Figs. 8, 9 and 10 show that the running times were proportional to the compression tree sizes N. Moreover, ENUFREQMAXPATH was faster when given a compact coding $d \in C_{50}^{100}(N)$ than when the tree corresponding to d was given, and a difference in running time appeared as the compression tree size N increased. Since the compression ratio was fixed to 50, if the compression tree size N of a compact coding in $C_{50}^{100}(N)$ increased, the size of the reference trees in the compact

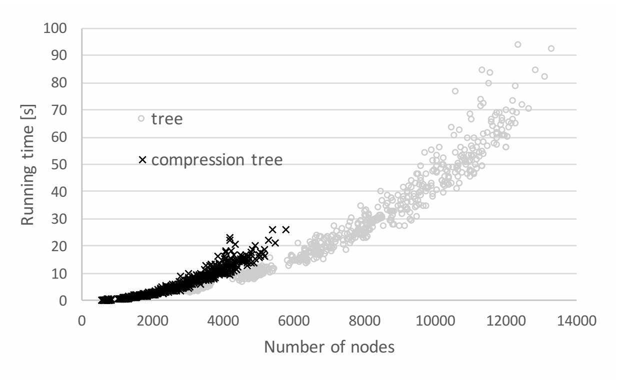

Fig. 8. Running times vs. compression tree size and decompression size given by numbers of nodes

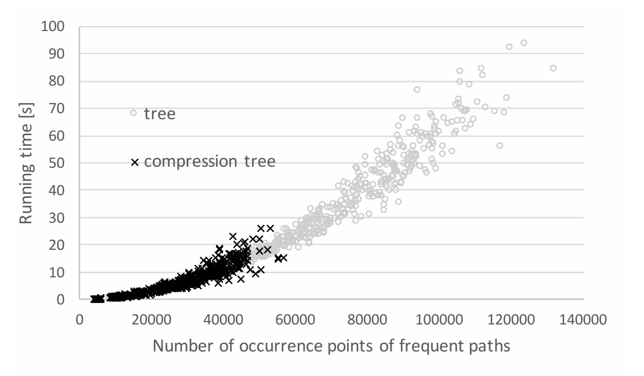

Fig. 9. Running time vs. number of occurrence points of frequent paths

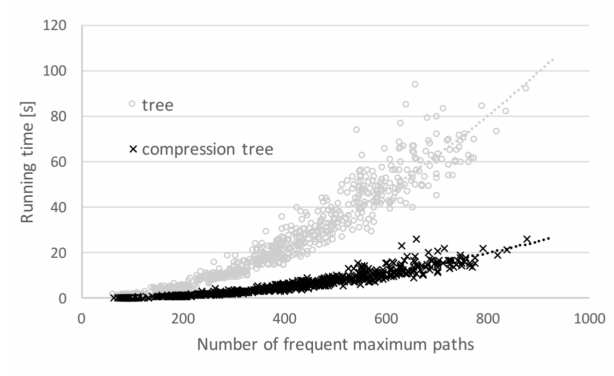

Fig. 10. Running time vs. number of frequent maximal paths

coding increased. Moreover, since a frequent path appearing in a reference tree appeared in all subtrees represented by port lists having the same reference as a label, as the compression ratio increased, the number of occurrence positions of the frequent paths in the compact coding decreased. This lead to a reduction in memory usage and increase in enumeration speed of all frequent maximal paths. Therefore, these experimental results show that ENUFREQMAXPATH has the advantage of extracting all frequent maximal paths from large trees having repeated subtrees.

C. *Experimental Results of* ENUFREQSUBTREE *for Solving* FREQSUBTREEENU-PROBLEM

We measured the running times needed to solve FREQSUBTREEENU-PROBLEMS for $Code_r^{100}(1000)$, $Code_{50}^{K}(1000)$ and $Code_{60}^{100}(1000)$ by using the implemented

ENUFREQSUBTREE while varying the compression ratio r from 0.2 to 0.8, the number K of compact codings from 100 to 500 and the minimum support rate from 1.0 down to 0.4, respectively.

First, Fig. 11 shows the difference between the running times of ENUFREQSUBTREEs for $Code_r^{100}(1000)$ and for $D(Code_r^{100}(1000))$ as inputs for when the compression ratio r was set from 0.2 to 0.8 and the minimum support rate was set to 1.0. From Fig. 11, we can see that, as the compression ratio r increased higher and higher, the difference in the running times of ENUFREQSUBTREEs started to quickly expand because the higher the compression ratio, the greater the difference between the numbers of nodes of the compression tree of $Code_r^{100}(1000)$ and $D(Code_r^{100}(1000))$.

Second, Fig. 12 shows the difference in the running times between ENUFREQSUBTREEs for $Code_{50}^{K}(1000)$ and for $D(Code_{50}^{K}(1000))$ for when the number K of trees was varied from 100 to 500 and the minimum support rate was set to 1.0. Due to the fixed compression ratio, the sizes of reference trees in the input compact coding increased as the number of trees increased. With the enumeration method used in ENUFREQSUBTREE, the running time depends on the number of occurrence points of subtrees whose frequency must be checked. Therefore, from Fig. 12, we can see that the expansion of the difference in the running time was due to the expansion of the differences in the number of nodes and in the number of occurrence points in $Code_{50}^{K}(1000)$.

Finally, Fig. 13 shows the running time of ENUFREQSUBTREE for $Code_{60}^{100}(1000)$ for when the minimum support rate was varied from 1.0 down to 0.4. Because the number of subtrees whose frequency must be checked increased extremely as the minimum support rate became smaller, the running time of ENUFREQSUBTREE became longer, and the difference between the running times of ENUFREQSUBTREEs for $Code_{60}^{100}(1000)$ and for $D(Code_{60}^{100}(1000))$ expanded.

These experimental results clearly show the advantage of enumerating all frequent paths and all frequent subtrees without decompression in structurally compressed trees.

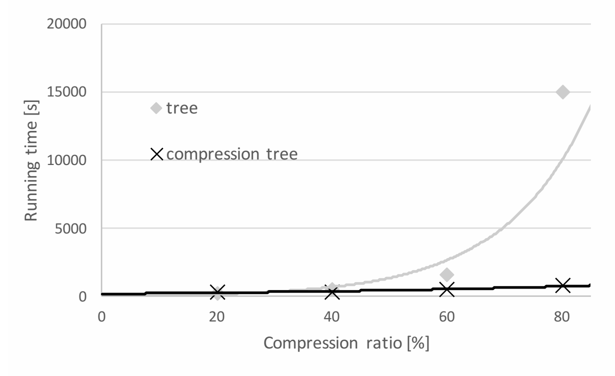

Fig. 11. Running time vs. compression ratio

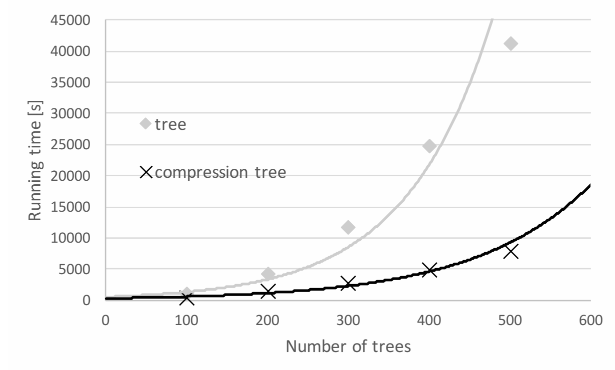

Fig. 12. Running time vs. number of trees

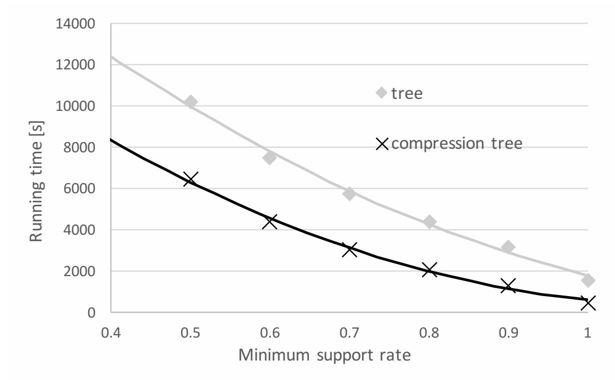

Fig. 13. Running time vs. minimum support rate

VI. CONCLUSION

We introduced a compression tree that is obtained by replacing repeated occurrences of subgraphs having ordered tree structures with references to the first occurrence point on the basis of an Lempel-Ziv compression scheme, and we presented a succinct representation of a compression tree by using DFUDS. We considered problems, denoted as FREQMAXPATHENU-PROBLEM and FREQSUBTREEENU-PROBLEM, for enumerating all frequent maximal paths from a given compression tree without decompression and enumerating all frequent subtrees from a given set of compression trees without decompression, respectively. Then, by using SDSL [14], we presented time- and memory-efficient algorithms, denoted as ENUFREQMAXPATH and ENUFREQSUBTREE, for solving FREQMAXPATHENU-PROBLEM and FREQSUBTREEENU-PROBLEM, respectively. We discussed the efficiency of the proposed algorithms ENUFREQMAXPATH and ENUFREQSUBTREE by using experimental results that were obtained by applying the algorithms to randomly generated synthetic large data.

For future work, we will apply the proposed algorithms to real-world large data. Moreover, we have plans to extend the pattern matching algorithms proposed by Itokawa et al. [8] and Suzuki et al. [16] for edge-labelled ordered trees to pattern matching algorithms for compression trees. That is, by extending the proposed algorithms ENUFREQMAXPATH and ENUFREQSUBTREE to the pattern matching algorithms for compressed trees, we will propose time- and memory-efficient enumeration algorithms for extracting all characteristic term tree patterns [16] having structured variables common to given compressed trees without decompression.

ACKNOWLEDGMENTS

This work was partially supported by Grant-in-Aid for Scientific Research (B) (Grant Number 26280087) and (C) (Grant Number JP15K00313) from the Japan Society for the Promotion of Science (JSPS), Japan.

References

[1] T. Asai, K. Abe, K. Kawasoe, H. Sakamoto, H. Arimura, and S. Arikawa, "Efficient substructure discovery from large semi-structured data," *IEICE TRANSACTIONS on Information and Systems* Vol. E87-D, No. 12, pp.2754-2763, 2004.

[2] J. Barbay, L. C. Aleardi, M. He, and J.I. Munro, "Succinct representation of labeled graphs," *Algorithmica*, Vol. 62, No. 1-2, pp.224-257, 2012.

[3] D. Benoit, E. D. Demaine, J. I. Munro, R. Raman, and S. S. Rao, "Representing trees of higher degree," *Algorithmica*, Vol. 43, No. 4, pp.275-292, 2005.

[4] P. Ferragina, F. Luccio, G. Manzini, and S. Muthukrishnan, "Compressing and Indexing Labeled Trees, with Applications," *Journal of the ACM*, Vol.57, No. 1, pp.4:1-4:33, 2009.

[5] R. F. Geary, N. Rahman, R. Raman, and V. Raman, "A simple optimal representation for balanced parentheses," *Theoretical Computer Science*, Vol. 368, No. 3, pp.231-246, 2006.

[6] T. Horibe, Y. Itokawa, T. Uchida, Y. Suzuki, and T. Miyahara, "Algorithm for enumerating all frequent paths from structurally compressed tree-structured data," *Lecture Nodes in Engineering and Computer Science: Proceedings of The International MultiConference of Engineers and Computer Scientists 2017*, 15-17 March, 2017 Hong Kong, pp.69-74, 2017.

[7] Y. Itokawa, K. Katoh, T. Uchida, and T. Shoudai, "Algorithm using expanded LZ compression scheme for compressing tree structured data," *Lecture Notes in Electrical Engineering: Intelligent Automation and Computer Engineering*, Vol. 52, Springer, pp.333-346, 2009.

[8] Y. Itokawa, M. Wada, T. Ishii, and T. Uchida, "Pattern matching algorithm using a succinct data structure for tree-structured patterns," *Lecture Notes in Electrical Engineering: Intelligent Control and Innovative Computing*, Vol. 110, Springer, pp.349-361, 2012.

[9] Y. Itokawa, T. Uchida, and M. Sano, "An algorithm for enumerating all maximal tree patterns without duplication using succinct data structure," *Lecture Nodes in Engineering and Computer Science: Proceedings of The International MultiConference of Engineers and Computer Scientists 2014*, 12-14 March, 2014 Hong Kong, pp.156-161, 2014.

[10] J. Jansson, K. Sadakane, and W.-K. Sung, "Ultra-succinct representation of ordered trees with applications," *Journal of Computer and System Sciences*, Vol. 78, No. 2, pp.619-631, 2012.

[11] H.-I Lu and C.-C. Yeh, "Balanced parentheses strike back," *ACM Transactions on Algorithms*, Vol. 4, No. 3, pp.28:1-28:13, 2008.

[12] J. I. Munro and V. Raman, "Succinct representation of balanced parentheses and static trees," *SIAM Journal on Computing*, Vol. 31, No. 3, pp.762-776, 2001.

[13] R. Raman and S.S. Rao, "Succinct representations of ordinal trees," *Lecture Notes in Computer Science: Space-efficient data structures, streams, and algorithms*, Vol. 8066, Springer, pp.319-332, 2013.

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

[15] J. A. Storer and T. G. Szymanski, "Data compression via textual substitution," *Journal of the ACM*, Vol. 29, No. 4, pp.928–951, 1982.

[16] Y. Suzuki, T. Shoudai, T. Uchida, and T. Miyahara, "An efficient pattern matching algorithm for ordered term tree patterns," *IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences*, Vol. E98-A, No. 6, pp.1197-1211, 2015.

[17] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Transactions on Information Theory*, Vol. IT-23, No. 3, pp.337-343, 1977.

[18] M. J. Zaki, "Efficiently mining frequent trees in a forest: algorithms and applications," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 17, No. 8, pp.1021-1035, 2005.