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 obtained by structurally compressing 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 ENUMFREQMAXPATH, 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 ENUMFREQSUBTREE, for enumerating all frequent subtrees as characteristic subtrees appearing in S without decompressing Code(S). Finally, we implement the proposed algorithms ENUMFREQMAXPATH and ENUMFREQSUBTREE on a computer, explain the experimental results obtained by applying ENUMFREQMAXPATH to a compact coding of a synthetic edge-labelled ordered tree and ENUMFREQSUBTREE to a compact coding of a set of synthetic edge-labelled ordered trees and provide discussion on evaluations of ENUMFREQMAXPATH and ENUMFREQSUBTREE.

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

I. INTRODUCTION

Tree-structured data, such as Web documents, XML 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 kth [ and its subsequent ] are output if the index of a node is k. By taking [ to be “0” and ] 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 ≥ 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 ≥ 1), we present an efficient algorithm, called ENUMFREQMAXPATH, for enumerating all k-frequent maximal paths in T without decompressing Code(T). ENUMFREQMAXPATH finds k-frequent paths from each node toward the root on a level-wise strategy with respect to
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 $\sigma \ (0 < \sigma \leq 1.0)$, a subtree $t$ is said to be $\sigma$-frequent if $t$ appears in $[n \times \sigma]$ 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 $\sigma \ (0 < \sigma \leq 1)$ are given as input, using a rightmost expansion strategy [1], [18], we present an efficient algorithm, called EnumFreqSubtree, for enumerating all $\sigma$-frequent subtrees in $S$ without decompressing $Code(S)$. Finally, using experimental results obtained by applying the implemented EnumFreqMaxPath and EnumFreqSubtree 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 EnumFreqMaxPath and EnumFreqSubtree 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 given compression tree is provided by using its DFUDS. EnumFreqMaxPath can naturally use the succinct data structures in implementations that use the succinct data structure library (SDSL) [14]. Hence, EnumFreqMaxPath is time- and memory-efficient for enumerating all k-frequent paths from a given compression tree of an edge-labelled ordered tree.

In this section, we introduce a compression tree that is a hypertree obtained from an edge-labelled ordered tree by reference [6]. The hypertree is a set of first occurrences of repeated subtrees in the hypertree obtained from an edge-labelled ordered tree by reference [6]. The hypertree is a list of first occurrences of repeated subtrees in the hypertree obtained from an edge-labelled ordered tree by reference [6].

In this section, we introduce a compression tree that is a hypertree obtained from an edge-labelled ordered tree by reference [6]. The hypertree is a set of first occurrences of repeated subtrees in the hypertree obtained from an edge-labelled ordered tree by reference [6]. The hypertree is a list of first occurrences of repeated subtrees in the hypertree obtained from an edge-labelled ordered tree by reference [6].
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 $\Lambda$ 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 $w = w_1, w_2, \ldots, w_n$ be a sequence, $L = (\ell_1, \ell_2, \ldots, \ell_k)$ 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 $i, j$ $\left(1 \leq i \leq n, 1 \leq j \leq k\right)$, $w[i]$ and $L[j]$ denote the elements at $i$ and $j$ in $w$ and $L$, respectively, i.e., $w[i] = w_i$ and $L[j] = \ell_j$.

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, $T[U] = (U, \{e \in E(T) \mid \text{are endpoints of } e \in E(T) \text{are in } U\})$. 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 $(v_1, v_2, \ldots, v_n)$ of nodes satisfying the following conditions.

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

We denote the set of all references of $T$ as $R_T$. For a reference $R = (v, v_1, v_2, \ldots, v_n)$ of $T$, we denote a subgraph induced by the node set $\bigcup_{u \in W} V(P_{u,v})$ as $T(L)$, 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 $1 \leq i \leq n$. In Fig. 1, for reference $(10, 13, 20, 21)$, we give the reference tree $t = \left(\{(10, 13, 20, 21)\}, \{(10, 12, 21), (10, b, 12), (12, a, 13), \ldots, (10, b, 21)\}\right)$. A subset $D_T$ of $R_T$ is called a dictionary of $T$ if for any two distinct references $L_1 = (u, u_1, u_2, \ldots, u_n)$ and $L_2 = (v, v_1, v_2, \ldots, v_r) \in D_T$, $(V(T[L_1]) - \{u\}) \cap (V(T[L_2]) - \{v\}) = \emptyset$ 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 $T(10, 13, 20, 21), T(10, 25, 50, 65), T(29, 32, 39, 40)$, and $T(50, 53, 61, 63)$ are isomorphic.

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

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

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 $\bigcup D_T = \emptyset$. Then, a triplet $t = (V_t, E_t, H_T)$ is called a hypertree obtained from $T$, $H_T$ and $D_T$, where $V_t, E_t, H_T$ are defined as follows.

1. $V_t = V_T$.
2. $E_t = E_T - \bigcup_{(v_0, a_1, v_1), \ldots, (v_0, a_r, v_r)} \in H_T \{\{v_0, a_1, v_1\}, \ldots, (v_0, a_r, v_r)\}$.

and the label of each edge $e \in E_T$ is preserved in $E_t$.

(3) $H_T = H_T$.

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 $t = \{(3, 8, \ldots, 49), (3, a, 8), (8, b, 10), \ldots, (3, a, 49)\}, \{(8, P, 25, 36, 47), (27, P, 32, 33, 34), (36, P, 41, 43, 45)\}$ 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 only one node is $\openone$. The DFUDS of an ordered tree $T$ that has $k$ subtrees $T_1, \ldots, T_k$ is a sequence of parentheses constructed by concatenating $k + 1$ $\openone$ and $k$ DFUDS of $T_1, \ldots, T_k$ in this order (the initial $\openone$ 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 $k$ $\openone$ for each node, whose degree is $k$, following the one $\openone$. The DFUDS is a succinct representation of an ordered tree with no edge labels.

However, since $\openone$ 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 $\openone$. 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

(Advance online publication: 10 February 2018)
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 $s = \{(u', v, u'_1, u'_2, \ldots, u'_N), E_s\}$ defined as follows. (a) The node $u'$ is the parent of $v$, and $u'_1, u'_2, \ldots, u'_N$ 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 $\text{DFUDS}(t)$.

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 $CT$, as follows. For each index $i \ (0 \leq i < n_t)$,

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

where $n_t$ is the number of nodes in the compression tree of $T$, $D_T = \text{DFUDS}(t)$ and, for a list $\text{Seq}$ of sequences and a sequence $\alpha$, $\text{Ind}(\text{Seq}, \alpha)$ is a function that returns the first index $k$ with $\text{Seq}[k] = \alpha$ if it exists, otherwise “$+$”. Then, a compact coding of $T$ is given by a sequence of $EL_T \circ \alpha \circ D_T \circ \alpha \circ CT$, denoted as $\text{Code}(T) = (EL_T, D_T, CT)$, where $\circ$ is an operator that concatenates two sequences.

Since we create $EL_T$, $D_T$ and $CT$ 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 $\text{Code}(T) = (EL_T, D_T, CT)$ be a compact coding of $T$. A length of $\text{Code}(T)$, denoted as $|\text{Code}(T)|$, is defined as the length of the sequence $EL_T \circ \alpha \circ D_T \circ \alpha \circ CT$. Fig. 1 shows the compact coding of the tree $T$.

### III. Enumeration Algorithm for Finding All k-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 path $p'$ in $T$ such that $p'$ has $p$ as a subpath. We define an enumeration problem, denoted as $\text{FREQMAXPATH}_{\text{ENU}}$-$\text{PROBLEM}$, for extracting all $k$-frequent maximal paths in a given compact coding of a tree as follows.

**FREQMAXPATH_{ENU}-PROBLEM**

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

**Problem**: Enumerate all $k$-frequent maximal paths in $T$ without decompressing $\text{Code}(T)$.

In Algorithm 1, we present our enumeration algorithm, denoted as $\text{ENUFREQMAXPATH}$, for solving $\text{FREQMAXPATH}_{\text{ENU}}$-$\text{PROBLEM}$. For a sequence $S$ of length $n$ on alphabet $\Lambda$, a character $c$ in $\Lambda$ and an integer $i \ (0 \leq i \leq n - 1)$, the two functions $\text{rank}$ and $\text{select}$ used in $\text{ENUFREQMAXPATH}$ are defined as follows.

1. $\text{rank}_c(S, i)$ returns the number of occurrences of $c$ in the subsequence from index 0 to index $i$ of $S$.
2. $\text{select}_c(S, i)$ returns the $i$-th position of $c$ from the beginning of $S$.

By using the Succinct Data Structure Library (SDSL) [14], we can compute $\text{rank}$ and $\text{select}$ in constant time. When a compact coding $\text{Code}(T)$ of a tree $T$ and a minimum occurrence $k$ are given, $\text{ENUFREQMAXPATH}$ 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 $\text{Code}(T)$. $\text{ENUFREQMAXPATH}$ 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 $\text{ENUFREQMAXPATH}$ as an example.

$\text{ENUFREQMAXPATH}$ (Algorithm 1) has three procedures: $\text{GENOCCPOINT}$ (Procedure 1), $\text{MAKECANDFREQPATH}$ (Procedure 2) and $\text{MAXIMALCHECK}$ described later. For an edge $e$, $\text{parent}(e)$ returns the parent edge of $e$, and function $\text{childrank}(e)$ returns the index $i$ such that $e$ is the $i$-th child edge of the parent edge of $e$. By using the SDSL,

<Fig. 2. Tree expressing numeration process of $k$-frequent paths>
Algorithm 1  ENUFREQMAXPATH

Require: A compact coding \((EL_T, DT, CT)\) of a tree \(T\) and a minimum occurrence \(k (k \geq 1)\).

Ensure: The set \(FM\) of all \(k\)-frequent maximal paths in \(T\).

1: \(Z_1 = \text{GENOCCPOINT}(EL_T, DT, CT)\), \(k\)
2: \(P_1 = \{a | (a, i, OP_i) \in Z_1\}\)
3: \(CFM = P_1\) and \(ln = 1\)
4: while \(P_n \neq \emptyset\) do
5: \(P_{ln+1} = \emptyset\)
6: \(W_{ln+1} = \text{MAKECANDFREQPATH}(Z_n, P_{ln+1})\)
7: for all \(p \in P_n\) and \(a \in P_1\) do
8: if \(\sum_{(p, a, OP_i) \in W_{ln+1}} |OP_i| \geq k\) then
9: \(P_{ln+1} = P_{ln+1} \cup \{p \circ a\}\)
10: \(Z_{ln+1} = Z_{ln+1} \cup \{(p \circ a, i, OP_i)\}\)
11: end if
12: end for
13: \(CFM = CFM \cup P_{ln+1}\)
14: \(ln++\)
15: end while
16: \(FM = \text{MAXIMALCHECK}(CFM)\)
17: return \(FM\)

Procedure 1  GENOCCPOINT

Require: The compact coding \((EL_T, DT, CT)\) 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.

1: \(Z_1 = \emptyset\) and \(W_1 = \emptyset\) and \(OP_1 = \emptyset\) for \(i \leq i < |CT|\)
2: for all \(i \leq i < |CT|\) do
3: if \(CT[i] \in EL_T\) then
4: \(OP_i = OP_1\) \(\cup\) \{i\)
5: end if
6: if \(CT[i] \in DT\) then
7: for all an edge \(j\) of reference tree \(T[CT[i]]\) do
8: \(OP_j = OP_1\) \(\cup\) \{i\)
9: end if
10: end if
11: end for
12: for all \(i \leq i < |CT|\) such that \(CT[i] \in EL_T\) do
13: \(W_i = W_1 \cup \{(CT[i], i, OP_i)\}\)
14: end for
15: for all \(a \in EL_T\) do
16: if \(\sum_{(a, i, OP_i) \in W_1} |OP_i| \geq k\) then
17: \(Z_1 = Z_1 \cup \{(a, i, OP_i)\}\)
18: end if
19: end if
20: return \(Z_1\)

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 \(CT\), ENUFREQMAXPATH uses a triplet \((p, i, OP_i)\), where \(OP_i\) is a multi-set of indexes in the interval \([0, |CT|]\) and satisfies \(|OP_i| = m\). Such a triplet is called a path-count triplet at index \(i\). Given a compact coding \((EL_T, DT, CT)\) 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 \(|CT|\). 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 \(Z_{ln+1} = \{(p, i, OP_i) \in W_{ln+1} | p \in P_{ln+1}\}\).

Third, ENUFREQMAXPATH constructs the set \(CFM\) of
must consider two cases in which the label of an argument does not exist in the root of the parent
edge (see Fig. 3). If we try to extend a path by appending an edge to the root of a path, then
an occurrence of the path will not be found by the TRIE and the extension will be discarded. If we
append an edge to a path at a point where the label of an argument exists, then the TRIE will return
the path in reverse sequence. In either case, 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 \( w^R = w_k, w_{k-1}, \ldots, w_1 \). 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 \( w = xy \) and \( |x| + |y| > 0 \). If 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 \( w = p g (g \in \Lambda^+) \), 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 \( PATH^R = \{ aba, bab \} \). Then, Procedure MAXIMALCHECK outputs the set \( \{ bab \} \) of the reverse sequence of the path "abab" stored in the leaf \( g_4^R \) 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 GENOC-
cPOINT 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, ENUFREQMXPATH generates $P_1 = \{a, b\}$ as the set of 5-frequent paths whose length is 1. By recursively applying Procedure MAKEANDFREQPATH, ENUFREQMXPATH 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 $CFM = \{a, b, ab, ba, aba, bab, baba\}$, ENUFREQMXPATH 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 $\sigma (0 < \sigma \leq 1)$, a tree $T$ is $\sigma$-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 $\sigma$ 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 $\varepsilon$ not in $A$. Then, a compact coding of $S$ is defined as a compact coding $Code(T_S) = (EL_{T_S}, D_{T_S}, C_{T_S})$ 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 $\sigma$-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 $\sigma (0 < \sigma \leq 1)$.

**Problem:** Enumerate all $\sigma$-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, 11, 18] for trees, ENUFREQSUBTREE enumerates all $\sigma$-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 $\sigma$-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 $\sigma$-frequent trees on the basis of the rightmost expansion strategy. In Fig. 6, we give a trie describing the enumeration process of $\sigma$-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 $T_S$ such that $Code(T_S) = (EL_{T_S}, D_{T_S}, C_{T_S})$ holds, it 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 $(p, i, OP)$, called an occurrence point of $p$ in $C_{T_S}$, which means that, for each $k \in OP$, 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: GENOCPPSINTREE (Procedure 3) and MAKEANDFREQSUBTREES (Procedure 4). Given an input set $Z_{in}$ of occurrence points of $\sigma$-frequent
Algorithm 2 \texttt{ENUFREQSUBTREE}

Require: A compact coding \((EL_{T_S}, D_{T_S}, C_T)\) of a set \(S\) of trees and an integer \(\sigma (0 < \sigma \leq 1.0)\).

Ensure: The set \(F\) of all \(\sigma\)-frequent trees in \(S\).
1: \(Z_1 = \text{GENOCCPOINTSUBTREE}(\langle EL_{T_S}, D_{T_S}, C_T, \sigma \rangle)\)
2: \(P_1 = \{p \mid (p, i, OP_i) \in Z_1\}\)
3: \(F = P_1\) and \(ln = 1\)
4: while \(P_{ln} \neq \emptyset\) do
5: \(P_{ln+1} = \emptyset\)
6: \(W_{ln+1} = \text{MAKECANDFREQSUBTREES}(Z_{ln}, Z_1)\)
7: for all \(z = (p, i, OP_i) \in W_{ln+1}\) do
8: if \(p\) is \(\sigma\)-frequent then
9: \(P_{ln+1} = P_{ln+1} \cup \{p\}\)
10: \(Z_{ln+1} = Z_{ln+1} \cup \{z\}\)
11: end if
12: end for
13: \(F = F \cup P_{ln+1}\)
14: \(ln++\)
15: end while
16: return \(F\)

Procedure 3 \texttt{GENOCCPOINTSUBTREE}

Require: The compact coding \((EL_T, D_T, C_T)\) and a minimum support rate \(\sigma \) (\(0 < \sigma \leq 1.0\)).

Ensure: The set \(Z_1\) of all occurrence points of \(\sigma\)-frequent edge.
1: \(Z_1 = \emptyset\) and \(W_1 = \emptyset\) and \(OP_i = \emptyset\) for \(i (0 \leq i < |C_T|)\)
2: for all \(i (0 \leq i < |C_T|)\) do
3: if \(C_T[i] \in EL_T\) then
4: \(OP_i = OP_i \cup \{i\}\)
5: end if
6: if \(C_T[i] \in D_T\) then
7: for all an edge \(j\) of reference tree \(T'[C_T[i]]\) do
8: \(OP_j = OP_j \cup \{i\}\)
9: end for
10: end if
11: end for
12: for all \(i (0 \leq i < |C_T|)\) such that \(C_T[i] \geq 2\) do
13: \(W_1 = W_1 \cup (p, i, OP_i)\)
14: \(*\) \(p\) is a tree consisting of an edge with label \(DL_{C_T[i]}\)
15: end for
16: for all \(z = (p, i, OP_i) \in W_{ln+1}\) do
17: if \(p\) is \(\sigma\)-frequent then
18: \(Z_{ln+1} = Z_{ln+1} \cup \{z\}\)
19: end if
20: end for
21: return \(Z_1\)

subtrees in \(T_S\) whose sizes are \(ln\) and an input set \(Z_1\) of occurrence points of \(\sigma\)-frequent edges in \(T_S\). \texttt{MAKECANDFREQSUBTREES} returns the set \(Z_{ln+1}\) of all occurrence points of \(\sigma\)-frequent subtrees whose sizes are \(ln + 1\). In Procedure \texttt{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.

\begin{figure}
\centering
\includegraphics[width=\textwidth]{fig6.png}
\caption{TRIE \(F\) describing enumeration process of \(\frac{1}{2}\)-frequent subtrees. Symbol described in node denotes \(\frac{1}{2}\)-frequent subtree detected by path from node to root. Some subtrees and their DFUDSs are described around nodes managing them.}
\end{figure}

\begin{enumerate}
\item \textbf{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 \(C_{T_S}[k]\).
\item \textbf{Case 2}: Index \(k\) is not equal to \(i\), and there exist no reference trees in which \(C_{T_S}[j]\) is.
\item \textbf{Case 3}: Index \(k\) is equal to \(i\), and there exists a reference tree in which \(C_{T_S}[j]\) is.
\item \textbf{Case 4}: Index \(k\) is equal to \(i\) and \(C_{T_S}[j]\) is in \(A\), or there exists a reference tree in which \(C_{T_S}[i]\) and \(C_{T_S}[j]\) are.
\end{enumerate}

These four cases are illustrated in Fig. 7. In \texttt{MAKECANDFREQSUBTREES}, for a reference \(L = \langle v_0, v_1, \ldots, v_n \rangle\) 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 \texttt{MAKECANDFREQSUBTREES}, we can see that the DFUDS of \(t\) can be made by inserting \(\emptyset\) into an appropriate index and appending the edge label to the end of the DFUDS of \(s\). To manage all frequent subtrees enumerated, \texttt{ENUFREQSUBTREEBASE} use a trie structure, denoted as \texttt{TRIE}, which stores each frequent subtree in a path of \texttt{TRIE}. As an example of \texttt{TRIE}, we give an ordered tree \(F\) describing the enumeration process of \(\frac{1}{2}\)-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 \(\emptyset\) at index 2 and appending "b" to the end of "((#a)".
Procedure 4 MAKE CandFreqSubtrees

Require: The set $Z_n$ of occurrence points of $\sigma$-frequent subtrees whose lengths are $ln$ and the set $Z_1$ of occurrence points of $\sigma$-frequent edges.

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

1: for all $f \in P_n$ do
2: /* $P_n$ is the set of $\sigma$-frequent subtrees whose lengths are $ln */
3: $OP'_i = \emptyset$ for $i$ ($0 \leq i < |C_T|$)
4: $U = \emptyset$
5: for all $(f, i, OP_i) \in Z_n$ do
6: for all $w \in OP_i$ do
7: for all $Path'_w$ \& $\forall j \in e$ do
8: /* $Path'_w$ is the rightmost path from the edge at the index $w$ to the edge corresponding to the root */
9: if $e = w$ then index = 0, otherwise index = childrank($e$) + 1 end if
10: /* $e$ is the child of $e$ on $Path'_w$ */
11: for $r = index$ to $r < \text{degree}(w)$ do
12: $u = \text{child}(e, r)$
13: if $\exists L \in D_T$ s.t. Arg($L, i$) $\geq 1$ then
14: dollar = $\text{child}(w, \text{Arg}(L, i))$
15: for $r' = 0$ to $r' < \text{degree}(\text{dollar})$ do
16: $u' = \text{child}(\text{dollar}, r')$
17: if $C_T[u'] \neq -1$ then /* case 1 */
18: $d_w = D_T[|C_T[w] + 2|]$
19: $p = d_w[0]$
20: for $r'' = 0$ to $r'' < \text{degree}(p)$ do
21: $u'' = \text{child}(p, r'')$
22: $U = U \cup \{(f', u'')\}$
23: /* $f'$ is a subtree obtained by the rightmost expansion at the index $e$ */
24: $OP'_w = OP'_w \cup \{u\}$
25: end for
26: else /* case 2 */
27: $U = \emptyset$
28: $OP'_w = OP'_w \cup \{u\}$
29: end if
30: end for
31: end for
32: end for
33: end for
34: end for
35: end for
36: end for
37: end for
38: end if
39: if $i \neq w$ then $OP'_w = OP'_w \cup \{w\}$, otherwise $OP'_w = OP'_w \cup \{u\}$
40: end if
41: end for
42: end for
43: end for
44: end for
45: end if
46: end if
47: end for
48: end for
49: end for
50: end for
51: end for
52: for all $(f', w) \in U$ do $W_{n+1} = W_{n+1} \cup \{(f', w, OP'_w)\}$ end for
53: end for
54: return $W_{n+1}$

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 ENUMFreqSubtree as a running example. Let $t$ be the compression tree of $Code(S)$ in Fig. 5. We set $\sigma$ to $\frac{1}{2}$. Let $F$ be a trie, denoted by TRIE, having only one node labelled with $\perp$. In line 1 of Algorithm ENUMFreqSubtree, when the compact coding $Code(S) = \langle EL_S, D_S, C_S \rangle$ and the minimum support rate $\sigma$ are given, GENOCPCPointsSubtree outputs the set of the occurrence points of all $\frac{1}{2}$-frequent subtrees whose size is one as follows. By executing the for loop from lines 2 to 11 of Procedure GENOCPCPointsSubtree, for each index $i$ ($0 \leq i < |C_S|$), GENOCPCPointsSubtree creates the set $OP_i$ of indexes that represent subtrees consisting of one edge. That is, $OP_b = \emptyset, OP_9 = \{9\}, OP_{10} = \{10, 22, 32\}, \ldots, OP_{29} = \emptyset, \ldots, OP_{39} = \{39\}$. 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 GENOCPCPointsSubtree, for each index $i$ ($0 \leq i < |C_S|$), GENOCPCPointsSubtree creates the set $W_i$ of occurrence points of all subtrees consisting of one edge. That is, $W_i = \{(f_1, 9, \{9\}), (f_1, 12, \{12, 22, 32\}), \ldots, (f_2, 39, \{39\})\}$. 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 GENOCPCPointsSubtree, from $W_1$, GENOCPCPointsSubtree creates the set $Z_1$ of all occurrence points of the rightmost leaves of $\frac{1}{2}$-frequent subtrees $f_1$ and $f_2$. That is, $Z_1 = \{ (f_1, 9, \{9\}), (f_1, 12, \{12, 22, 32\}), (f_2, 10, \{10, 22, 32\}), (f_2, 13, \{13, 22, 32\}), (f_2, 17, \{17\}), (f_2, 20, \{20\}), (f_2, 27, \{27\}), (f_2, 39, \{39\}) \}$. As a result, GENOCPCPointsSubtree outputs $Z_1$. To manage $\frac{1}{2}$-frequent subtrees $f_1$ and $f_2$ in $P_1$ by TRIE, EN-
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 $(f_5, 13, [12, 22, 32])$ 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 $(f_2, 13, [12, 22, 32])$ 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 $(f_2, 26)$ 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 $f_2$ 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 $(f_6, 10)$ is added to $U$, and the occurrence point $(f_6, 10, [22])$ is added to $W_2$, where $f_6$ 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 $U = \{ (f_3, 27), (f_3, 39), (f_4, 12), \ldots, (f_5, 13), \ldots, (f_7, 26), (f_7, 37), \ldots, (f_9, 10), \ldots, (f_9, 39), (f', 28) \}$ and $W_2 = \{ (f_3, 27, [27]), (f_3, 39, [39]), (f_4, 12, [12]), \ldots, (f_5, 13, [12, 22, 32]), \ldots, (f_7, 26, [26]), (f_7, 37, [37]), \ldots, (f_9, 10, [22]), \ldots, (f_9, 39, [39]), (f', 28, [28]) \}$, 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 $P$ in $F$ in Fig. 6. 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 $\frac{4}{3}$-frequent.

In the same way as the construction of $Z_2$, for each $i \geq 3$, $ENUFREQSUBTREE$ recursively constructs the set $Z_i$ of occurrence points of all $\frac{4}{3}$-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 $Code(T) = (ELT, DT, CT)$ be a compact coding of a tree $T$. A compression ratio of $Code(T)$ is defined as $\frac{|Code(T)|}{|T|}$, 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 $CT$, i.e., $|CT|/2$. Let $N$ be an integer in $\{1000, 2000, 3000, 4000, 5000, 6000, 7000\}$.

(Advance online publication: 10 February 2018)
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 \(\text{Code}(T_i) = \langle EL_{T_i}, D_{T_i}, C_{T_i} \rangle\) satisfying the following conditions (1)-(4).

1. The compression tree size of \(\text{Code}(T_i)\) is about \(N\).
2. A compression ratio of \(\text{Code}(T_i)\) 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 \(\text{Code}(T_i)\) is at most 5, i.e., \(|D_{T_i}| \leq 5\).

Then, in the experiments, the set \(C_r(N) = \{\text{Code}(T_1), \text{Code}(T_2), \ldots, \text{Code}(T_{100})\}\) 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

\[
\frac{\text{running time} \text{ of } \text{Code}(S)}{\sum_{k=1}^{K} |\text{DFUDS}(T_k)|}
\]

We randomly created a compact coding \(\text{Code}(T_S) = \langle EL_{T_S}, \mu(D_{T_S}), C_{T_S} \rangle\) 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 \(\text{Code}(T_S)\) 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 \(N\), \(\text{Code}(T_S)\) is denoted as

\(\text{Code}^{K}_r(N) = \langle EL_{S}, S_{\mu(D_{S})}, C_{S} \rangle\), and \(T_S\) is denoted as

\(D(\text{Code}^{K}_r(N))\).

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

We measured the running times needed to solve FREQMAXPATHENU-PROBLEMS for \(C^{100}_{100}(N)\) and \(D(C^{100}_{50}(N))\) 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 \(d \in C^{100}_{50}(N)\).

(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 \(d \in C^{100}_{50}(N)\) 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 \(C^{100}_{50}(N)\) increased, the size of the reference trees in the compact
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 FREQSUBTREENU-PROBLEM**

We measured the running times needed to solve FREQSUBTREENU-PROBLEMS for \(C^{100}_{100}(1000),\)
\(C^{100}_{50}(1000)\) and \(C^{100}_{60}(1000)\) 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 ENUFREQSUBTREE for $Code_{100}^{100}(1000)$ and for $D(Code_{100}^{100}(1000))$ 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 ENUFREQSUBTREE started to quickly expand because the higher the compression ratio, the greater the difference between the numbers of nodes of the compression tree of $Code_{100}^{100}(1000)$ and $D(Code_{100}^{100}(1000))$.

Second, Fig. 12 shows the difference in the running times between ENUFREQSUBTREE for $Code_{50}^{100}(1000)$ and for $D(Code_{50}^{100}(1000))$ 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 $Code_{50}^{100}(1000)$.

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 ENUFREQSUBTREE 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.

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 FREQMAXPATH|ENU-PROBLEM and FREQSUBTREE|ENU-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 FREQMAXPATH|ENU-PROBLEM and FREQSUBTREE|ENU-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.

(Advance online publication: 10 February 2018)
REFERENCES


(Advance online publication: 10 February 2018)