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

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

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

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

I. INTRODUCTION

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

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

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

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

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

II. PRELIMINARIES

In this section, we introduce a compression tree, which is a rooted edge-labeled ordered tree obtained from an edge-labeled ordered tree by replacing repeated occurrences of subgraphs having ordered-tree structures with references to a dictionary, which is a list of first occurrences of the repeated subgraphs in its depth-first traversal. Moreover, we define a succinct representation of a compression tree by extending the depth-first unary degree sequence (DFUDS)\cite{7, 9} for an ordered tree.
A child port of $h$. Two port lists $h = (u, L_h, u_1, u_2, \ldots, u_k)$ and $h' = (v, L_{h'}, v_1, v_2, \ldots, v_l)$ of an edge-labeled ordered tree $T$ are said to be disjoint if the following conditions are satisfied.

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

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

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

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

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

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

1. $V_t = V_T$.
2. $E_t = E_T \setminus \bigcup_{(v_0, v_1, \ldots, v_n) \in H_T} \{(v_0, a_1, v_1), \ldots, (v_n, a_n, v_n)\}$, and the label of each edge $e \in E_T$ is preserved in $E_t$.
3. $H_t = H_T$ and each port list in $H_t$ is labeled with a reference $(w, u_1, u_2, \ldots, u_k)$ of $D_T$ such that $w_k$ appears before $u_1$ on the depth-first traversal of $T$.

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

Definition 2: (Compression Tree) A compression tree is a hyper tree obtained from $T$ by replacing repeated oc-
currences of subgraphs having ordered-tree structures with references to the first occurrence of the subgraphs.

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

B. Succinct Representation of Compression Trees

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

The DFUDS for an ordered tree \( T \) of \( m \) edges is defined inductively as follows [7], [9]. The DFUDS of the tree consisting of only one node is \( [ ] \). The DFUDS of an ordered tree \( T \) that has \( k \) subtrees \( I_1, \ldots, I_k \) is a sequence of parentheses constructed by concatenating \( k + 1 \) \([ ]\) one \( \smile \) \( k \) DFUDSs of \( I_1, \ldots, I_k \) in this order (the initial \([ ]\) of the DFUDS of each subtree has been removed). The resultant DFUDS is a sequence of balanced parentheses of length \( 2m \). The sequence of parentheses, that is a DFUDS, can be interpreted as the result of visiting all nodes in preorder and outputting \( k \) \([ ]\) for each node, whose degree is \( k \), following the one \([ ]\).
The DFUDS is a succinct representation of an ordered tree with no edge labels. But since \([ ]\) occupies the rightmost position for each node in the DFUDS, we can modify the DFUDS of an ordered tree to the DFUDS of an edge-labeled ordered tree, using a hash function that returns the label of the edge incident to the node corresponding to each \([ ]\). To provide a succinct representation of a compression tree, we consider an underlying ordered tree for a compression tree. For a compression tree \( t = (V_t, E_t, H_t) \), we call by an underlying tree of \( t \) the edge-labeled ordered tree obtained from \( t \) by applying the following replacements to all port lists of \( t \). A port list \( h = (u, u_1, u_2, \ldots, u_k) \in H_t \) is replaced with an edge-labeled ordered tree as follows.

1. Remove \( h \) from \( t \).
2. Construct an edge-labeled ordered tree \( s = (u', v, u_1', u_2', \ldots, u_k', E_s) \) defined as follows.
   (a) \( v \) has \( u' \) as the parent and \( u_1', u_2', \ldots, u_k' \) as the children in that order.
   (b) The edge between \( u' \) and \( v \) is labeled with the reference of \( h \) and any edge between \( v \) and its child is labeled with the special symbol “\$”.
   (c) Identify the parent port \( u \) and each child port \( u_i \) (1 \( \leq \) \( i \) \( \leq \) \( k \)) with the root \( u' \) of \( s \) and each leaf \( u_i' \) (1 \( \leq \) \( i \) \( \leq \) \( k \)), respectively.

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

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

Definition 4: (Compact Coding for an Edge-Labeled Ordered Tree) Let \( T \) be an edge-labeled ordered tree. Let \( EL_T \) be the coding of the list of all edge labels in \( T \) which are sorted by occurrence order in the depth-first traversal of \( T \). Let \( DT \) be the coding of a list of references in \( T \) which are sorted by occurrence order in the depth-first traversal of \( T \). Let \( C_T \) be the coding of the compression tree \( t \) of \( T \). Then, a compact coding of \( T \) is given by a sequence of \( EL_T \circ \nu \circ DT \circ \nu \circ C_T \), denoted as \( Code(T) = (EL_T, DT, C_T) \), where \( \circ \) is an operator of two concatenating sequences.

Since we create \( EL_T, DT \) and \( C_T \) based on the depth-first traversal of \( T \), we can easily create hash functions that return the edge label or reference for each edge in the compression tree \( t \) of \( T \). Figure 1 shows the succinct representation of the compression tree \( t \).

III. Enumeration Algorithm for Extracting Frequent Paths in Compression Trees

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

Frequent-Paths Enumeration Problem

Instance: Compression tree \( t \) and minimum occurrence \( k \) (\( k \geq 1 \)).
Problem: Enumerate all \( k \)-frequent paths in \( t \) without decompresing \( t \).

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

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

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

\( ENUFREQPATHS \) (Algorithm 1) has two procedures \( GENOCPPINT \) (Procedure 2) and \( MAKECANDFREQPATHS \) (Procedure 3). For an edge \( e \), \( parent(e) \) returns the parent edge of \( e \), and function \( childrank(e) \) returns the index \( i \) such that \( e \) is the \( i \)-th child of the parent edge of \( e \). For reference \( L = (v_1, v_2, \ldots, v_k) \in DT \) and an integer \( n \) (\( 1 \leq n \leq k \)), a function \( dic(L, n) \) returns the index \( v_n \).

By using the SDSL, these operations on a compression tree can be executed in constant time.

To count the number of paths, which is isomorphic to a path and terminates at an index \( i \) of \( C_T \), \( ENUFREQPATHS \) uses a triplet \( (u, p, OP_u) \), where \( OP_u \) is a multi-set of indexes in the interval \( [0, |C_T|] \) and satisfies \( |OP_u| = m \). Such a triplet is called a \( path \)-\( count \) \( triplet \) at index \( i \). Given a compact coding \( (EL_T, DT, C_T) \) of an edge-labeled ordered tree \( T \), \( ENUFREQPATHS \) first generates the set \( Z_1 \) of \( path \)-\( count \) triplets at all indexes between 0 and \( |C_T| \). Then, it constructs the set \( P_1 \) of all \( k \)-frequent edges from \( Z_1 \). That
is, $P_1$ is the set of all $k$-frequent paths, the length of each is 1. Next, by using Procedure MAKECANDFREQPATHS, ENUMFREQPATHS recursively generates the set $C_{ln+1}$ of all path-count triplets having candidate paths, the length of each is $ln+1$ from set $Z_1$ and set $Z_{ln+1}$, each of which has a path whose length is $ln$. From $C_{ln+1}$, ENUMFREQPATHS constructs the set $P_{ln+1}$ of $k$-frequent paths, the length of each is $ln+1$ and set $Z_{ln+1} = \{(i,p,OP_i) \in C_{ln+1} | \ p \in P_{ln+1}\}$. Finally, ENUMFREQPATHS returns the set $F$ of all $k$-frequent paths appearing in Tree $t$.

In Procedure MAKECANDFREQPATHS, when we construct a candidate path whose length is $ln+1$ by expanding the $k$-frequent path whose length is $ln$, four cases are considered (see Fig. 3). If we try to expand a path by appending an edge to the edge $e$ incident to the root of a reference tree, we must consider two cases in which the label of parent($e$) is 'S' (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 'S' (case 3) or is included in $EL_T$ (case 4).

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

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

...
Procedure 3 MAKECANDFREQPATHS

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

Ensure: The set $Z_{ln+1}$ of path-count triplets each of which has a path whose length is $ln + 1$

1: $OP_{i} = \emptyset$ for $i (0 \leq i < |C_{T}|)$
2: $U = \emptyset$
3: for all $p \in P_{ln}$ do
4: for all $(i, p, OP_{i}) \in Z_{ln}$ do
5: if $(rank(i) + 1, v_{1}, \ldots, v_{k}) \in D_{T}$ then
6: for all $w \in OP_{i}$ do
7: $u = parent(w)$
8: if $C_{T}[u] = \emptyset$ then
9: $P^{*}$ case 1 */
10: $q = dic (C_{T}[parent(u)], childrank(u))$
11: $U = U \cup \{q\}$
12: $OP_{q} = OP_{q} \cup \{parent(u)\}$
13: else
14: $P^{*}$ case 2 */
15: $U = U \cup \{u\}$
16: $OP_{u} = OP_{u} \cup \{u\}$
17: end if
18: end for
19: else if $C_{T}[parent(i)] = \emptyset$ then
20: $P^{*}$ case 3 */
21: for all $w \in OP_{i}$ do
22: $j = parent(i)$
23: $u = dic (C_{T}[parent(j)], childrank(j))$
24: $U = U \cup \{select(u + 1)\}$
25: $OP_{u} = OP_{u} \cup \{select(u + 1)\}$
26: end for
27: else
28: $P^{*}$ case 4 */
29: $e = parent(i)$
30: $U = U \cup \{e\}$
31: $OP_{e} = OP_{e} \cup OP_{i}$
32: end if
33: end for
34: for all $w \in U$ do
35: $Z_{ln+1} = Z_{ln+1} \cup \{(w, p \circ C_{T}[w], OP_{w})\}$
36: end for
37: return $Z_{ln+1}$

in $T$.

IV. EXPERIMENTAL RESULTS AND DISCUSSION

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

A. Experimental Environments and Setting

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

B. Experimental Results and Discussion

Figure 4 shows a comparison of the running time of ENUFREQPATHS when each $T \in D$ and $t \in CT(D)$, which is the compression tree of $T$, are given as inputs. We can see that ENUFREQPATHS given the compression tree $t \in CT(D)$ of $T$ is always faster than that given an edge-labeled ordered tree $T \in D$. Figure 5 shows that the running time of ENUFREQPATHS is proportional to the number of occurrence points of frequent paths. Figure 6 shows a comparison of the running time needed to find all frequent paths. For $i \geq 1$, ENUFREQPATHS constructs candidate paths $p^{i+1}$, whose lengths are $i + 1$, from all frequent paths $p'$ whose lengths are $i$ by expanding at the occurrence points of $p'$. From Fig. 5, we can see that ENUFREQPATHS is a sequential linear time algorithm w.r.t. the number of found occurrence points. From Fig. 6, for a compression tree $t$ of an edge-labeled ordered tree $T$, we can see that the running time...
required for extracting all \( k \)-frequent paths from \( t \) is faster than that from \( T \) by about 4.11 times.

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

V. CONCLUSION

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

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

ACKNOWLEDGMENTS

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

REFERENCES


