4A1-4

構造圧縮された木構造データからの頻出部分木枚挙アルゴリズム

Algorithm for enumerating all frequent subtrees from structurally compressed tree-structured data

堀部 智也 *1 Tomoya Horibe　糸川 裕子 *2 Yuko Itokawa　内田 智之 *1 Tomoyuki Uchida　鈴木 祐介 *1 Yusuke Suzuki　宮原 哲浩 *1 Tetsuhiro Miyahara

*1広島市立大学大学院情報科学研究科
Graduate School of Information Sciences, Hiroshima City University

*2広島国際大学心理科学部
Faculty of Psychology, Hiroshima International University

Structurally compressing large tree-structured data without loss of information leads to reduction in running time needed to extract structural features. We first introduce a compression tree, which represents an edge-labeled ordered tree structurally compressed by a Lempel-Ziv compression scheme. Then, given a set D of compression trees, we propose a time- and memory-efficient algorithm for enumerating all frequent subgraphs, which have tree structures, in the set U_D without decompressing compression trees in D, where U_D is the set of edge-labeled ordered trees obtained by decompressing compression trees in D. Finally, we discuss the implementation of the proposed algorithm on a computer, explain the experimental results obtained by applying it to a set of synthetic edge-labeled ordered trees, and provide discussion on its evaluation.

1. はじめに

Web 文書，LaTeX ソースファイル，自然言語などの木構造データは順序木で表現されることが多い．また，それらの木構造データの数は日々増大している．それら巨大化した木構造データから共通する構造的特徴を抽出するには計算時間が非常にかかることが多い．そのため同じ特徴を繰り返し枚挙することのない効率的なマイニング手法が提案されている [Asai et al. 2004, Zaki 2005, Itokawa et al. 2010]．文字列上の圧縮手法である LZ 圧縮手法 [Ziv and Lempel 1977] は，繰返し出現する部分文字列を最初に出現する位置へのリファレンスで置き換えることでサイズの可逆圧縮を行う手法である．Itokawa ら [Itokawa et al. 2010] は，この LZ 圧縮手法を木構造データの構造的繰り返しの圧縮に拡張した可逆圧縮手法を提案している．木構造データ D を可逆圧縮したデータ $Compress(D)$ から，解凍することなく D に共通する構造的特徴を抽出するグラフマイニング手法の開発を本研究の目的とする．これにより抽出時間の短縮がもたらされ，より大きな木構造データの解析が可能となると考えられる．

この目的を達成するために，論文 [Horibe et al. 2017] において，木構造データを表現する辺ラベル付き順序木 T の構造的繰り返しを可逆的圧縮して得られる辺ラベル付き順序木である圧縮木を定義し，順序木に対する簡潔データ表現である DFUDS[Jansson et al. 2012, Raman et al. 2013] を圧縮木に拡張した簡潔データ表現を与えた．さらに，T に対する圧縮木の簡潔データ表現に基づく T の圧縮コーディングを提案した．図 1 に，辺ラベル付き順序木 T に対する圧縮木 t とその簡潔データ表現及び T の圧縮コーディングの例を示す．論文 [Horibe et al. 2017] では，辺ラベル付き順序木 T の圧縮コーディング $Code(T)$ と最小出現回数 k が与えられたとき，$Code(T)$ を解凍することなく k 回以上出現するパスを全て枚挙するアルゴリズム *EnuFreqPaths* を提案した．本稿では，論文 [Horibe et al. 2017] の結果を拡張し，辺ラベル付き順序木の圧縮コーディングの集合 $Compress(D)$ と実数 σ $(0 < \sigma \leq 1)$ が与えられたとき，$Compress(D)$ 中の圧縮コーディングを解凍することなく，$Compress(D)$ が表す辺ラベル付き順序木集合 D に含まれる $\lceil |Compress(D)| \times \sigma \rceil$ 個以上の辺ラベル順序木に共通して出現する順序木構造を持つ部分グラフを全て枚挙する効率的なグラフマイニングアルゴリズムを与える．最後に，提案アルゴリズムを計算機上に実装し，人工的なデータを用いた評価実験を行ったので，その評価実験結果の報告と考察を行う．

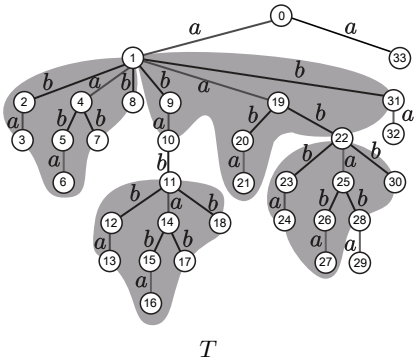
連絡先: 内田智之，広島市立大学大学院情報科学研究科，〒 731-3194 広島市安佐南区大塚東 3 丁目 4-1，uchida@hiroshima-cu.ac.jp

2. 準備

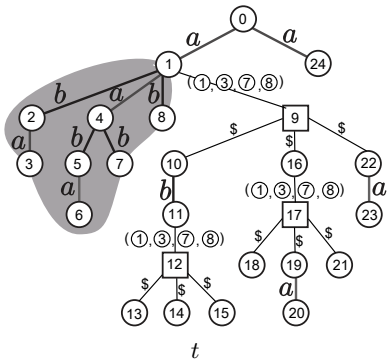
本節では，論文 [Horibe et al. 2017] で与えた，構造圧縮された木構造データのモデルである圧縮木について述べ，簡潔データ表現の 1 つである DFUDS(Depth-First Unary Degree Sequence, [Jansson et al. 2012, Raman et al. 2013]) を用いた圧縮木の簡潔データ表現を紹介する．

2.1 圧縮木

Λ をアルファベットとする．辺ラベル付き順序木とは，任意の内部ノードが順序付けられた子を持ち，任意の辺が Λ 内の要素でラベル付けられている根付き木をいう．辺ラベル付き順序木 T のノード集合と辺集合はそれぞれ $V(T)$ と $E(T)$ で表される．ノード u がノード v の親であるラベル a をもつ $E(T)$ 内の辺を 3 つ組 $(u, a, v) \in V(T) \times \Lambda \times V(T)$ で表す．T の内部ノード u と u の連続した子 u_1, u_2, ..., u_k からなるラベル ℓ をもつリスト $p = (u, \ell, u_1, u_2, \ldots, u_k)$ を**ポートリスト**という．このときの u を p の親ポート，u_1, u_2, ..., u_k を子ポートという．T の 2 つのポートリスト $p = (u, \ell_p, u_1, u_2, \ldots, u_k)$ と $p' = (u', \ell_{p'}, u'_1, u'_2, \ldots, u'_{k'})$ が互いに素であるとは，以下の 2 つの条件を満たすときをいう．(1) $\{u_1, u_2, \ldots, u_k\} \cap \{u'_1, u'_2, \ldots, u'_{k'}\} = \emptyset$. (2) もし u と u' が同じノードであれば，u_k は u'_1 より兄姉であるか，もしくは u_1 は $u'_{k'}$ の弟妹である．T のノード v に対し，v を根とする T の部分木を $T[v]$ で表す．さらに，$V(T)$ の部分集合 U に対して，U と両端点が U に含まれる辺からなる $E(T)$ の部分集合 E_U からなる T の誘導部分グラフを $T[U] = (U, E_U)$ と表す．T のノード u とその子孫 v に対して，v から u へのパスを $P_{v,u}$ で表す．以下の 3 条件を満たす T のノードリス



T



t

t の簡潔データ表現:

$$\underset{\textcircled{0}}{(}\ \underset{\textcircled{1}}{((\#}\ \underset{\textcircled{2}\textcircled{3}}{((((a}\ \underset{\textcircled{4}}{(b}\ \underset{\textcircled{5}}{a}\ \underset{\textcircled{6}}{((a}\ \underset{\textcircled{7}}{(b}\ \underset{\textcircled{8}}{a}\ \underset{\boxed{9}}{b}\ \underset{\textcircled{10}}{b}\ \underset{\textcircled{11}}{(((P}\ \underset{\boxed{12}}{(\$}\ \underset{\textcircled{13}\textcircled{14}\textcircled{15}\textcircled{16}}{(b}$$

$$\underset{\boxed{17}}{(((P}\ \underset{\textcircled{18}}{\$}\ \underset{\textcircled{19}}{\$}\ \underset{\textcircled{20}}{\$}\ \underset{\textcircled{21}}{(\$}\ \underset{\boxed{22}}{(((P}\ \underset{\textcircled{23}}{\$}\ \underset{\textcircled{24}}{(\$}\ a\ \$\ \underset{\textcircled{25}\textcircled{26}\textcircled{27}\textcircled{28}}{(\$}\ a\ a$$

T の圧縮コーディング:

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

図 1: 辺ラベル付き順序木 T, T を構造圧縮して得られる圧縮木 t, t の簡潔データ表現と T の圧縮コーディング

ト $(v,\ v_1,\ v_2, \ldots,\ v_n)$ を T の**リファレンス**という．(1) 各 i $(1 \le i \le n)$ に対して，v_i は v の子孫である．(2) 任意の i, j $(1 \le i, j \le n, i \ne j)$ に対して，v_i と v_j はお互い子孫関係にはない．(3) 任意の i, j $(1 \le i < j \le n)$ に対して，v_j は T の深さ優先走査において v_i より後に出現する．T のすべてのリファレンスの集合を $\mathcal{R}_T$ で表す．T のリファレンス $\ell = (v,\ v_1,\ v_2, \ldots, v_n)$ に対して，W_ℓ を T の深さ優先走査において v_1 から v_n の間で出現するが，任意の i $(1 \le i \le n)$ に対して部分木 $T[v_i]$ の根以外のノードではないすべての葉の集合とする．このとき，ノード集合 $\bigcup_{w \in W_\ell} V(P_{w,v})$ からなる誘導部分グラフを ℓ に対する**リファレンス木**といい，$T\langle\ell\rangle$ で表す．もし $\mathcal{R}_T$ の部分集合 $\mathcal{D}_T$ に対し，$\mathcal{D}_T$ 内の任意の 2 つのリファレンス $\ell_1 = (u, u_1, u_2, \ldots, u_\ell)$ と $\ell_2 = (v, v_1, v_2, \ldots, v_r)$ が $(V(T\langle\ell_1\rangle) - \{v\}) \cap (V(T\langle\ell_2\rangle) - \{v\}) = \emptyset$ を満たすとき，$\mathcal{D}_T$ を T の**辞書**という．図 1 において，リファレンス $(1,3,7,8)$ に対するリファレンス木は
$t\langle(1,3,7,8)\rangle = \big(\{1,2,\ldots,8\}, \{(1,b,2),(2,a,3),\ldots,(1,b,8)\}\big)$
である．

定義 1. $T = (V_T, E_T)$ を辺ラベル付き順序木，H_T を T のポートリストの集合，D_T を $\Lambda \cap D_T = \emptyset$ を満たす T の辞書とする．このとき，以下の 3 条件を満たす 3 つ組 $t = (V_t, E_t, H_t)$ を超木という．(1) $V_t = V_T$ (2) $E_t = E_T - \bigcup_{(v_0,v_1,\ldots,v_r) \in H_T} \{(v_0, a, v_1), \ldots, (v_0, a, v_r) \mid a \in \Lambda\}$ であり，E_T の各辺のラベルは E_t においても保持される．(3) $H_t = H_T$ であり，H_t のリスト長 k のポートリストは $\mathcal{D}_T$ 中のリスト長 k のリファレンスでラベル付けられている．

文字列上の圧縮法である LZ77 を順序木に拡張することで得られる超木である圧縮木を次で定義する．

定義 2. **圧縮木**とは，繰り返し出現する順序木構造の部分グラフを，深さ優先走査において最初に出現する位置を示すリファレンスで置き換えることで順序木から得られる超木をいう．

図 1 に，辺ラベル付き順序木 T に対する圧縮木 $t = (\{0,1,\ldots,24\}, \{(0,a,1),(1,b,2),\ldots,(0,a,24)\}, \{(1,P,10,16,22),(11,P,13,14,15),(16,P,18,19,21)\})$ を与える．ここで，T 中の灰色領域は同じ順序木構造をもつ部分グラフを表し，t 中の灰色領域は辞書の要素を表す．t の簡潔データ表現において，丸数字は t のノード ID を，四角数字はポートリストを，記号 “P” はリファレンス $(1,3,7,8)$ を表す．T の 4 つのリファレンス $(1,3,7,8)$, $(1,10,22,31)$, $(11,13,17,18)$, $(22,24,28,30)$ に対するリファレンス木 $T\langle(1,3,7,8)\rangle, T\langle(1,10,22,31)\rangle, T\langle(11,13,17,18)\rangle, T\langle(22,24,28,30)\rangle$ はすべて同型である．よって，深さ優先走査により最初に現れるリファレンス木 $T\langle(1,3,7,8)\rangle$ 以外をリファレンス $(1,3,7,8)$ でラベル付けられたポートリストで置き換えることで T を情報の欠落なく圧縮することができる．このときの T の辞書は $\{(1,3,7,8)\}$ となる．

2.2 圧縮木の簡潔データ表現とそのコーディング

順序木の簡潔データ表現の 1 つに DFUDS[Jansson et al. 2012, Raman et al. 2013] がある．順序木 T の DFUDS は以下のように再帰的に定義される．1 ノードのみからなる順序木の DFUDS は $\boxed{(}, \boxed{)}$ である．T の根の子を根としてもつ部分木 T_1, $T_2, \ldots,$ T_k をこの順に持つ順序木 T の DFUDS は，$k+1$ 個の $\boxed{(}$，1 つの $\boxed{)}$，k 個の T_1, $T_2, \ldots,$ T_k の DFUDS の各先頭の $\boxed{(}$ を除いて順に並べた括弧列である．T が n 個のノードを持つとき，T の DFUDS は長さ $2n$ の括弧列となる．この括弧列の $\boxed{)}$ に辺ラベルを対応させることにより，辺ラベル付き順序木に対する DFUDS に拡張することができる．よって，辺ラベル付き順序木に対する DFUDS を用いて，圧縮木に対する簡潔データ表現を以下に定義する．

定義 3. 圧縮木 $t = (V_t, E_t, H_t)$ の簡潔データ表現とは，次のように t から得られる辺ラベル付き順序木 $s = (V_s, E_s)$ の DFUDS をいう．(1) $V_s = V_t \cup H_t$. (2) $E_s = E_t \cup \{(u, \ell_h, h), (h, '' \$'', u_1), \ldots, (h, '' \$'', u_k) \mid h = (u,\ u_1,\ u_2, \ldots,\ u_k) \in H_t\}$. ここで，$\ell_h$ とはポートリスト h のラベルであるリファレンスである．

図 1 に，圧縮木 t の簡潔データ表現を与える．

圧縮木の簡潔データ表現を用いた辺ラベル付き順序木のコンパクトなコーディングを以下に定義する．

定義 4. T を辺ラベル付き順序木とする．EL_T と D_T をそれぞれ T の深さ優先走査順に出現する辺ラベルの全種類のリス

トとリファレンスの全種類のリストのコーディングとする．また，C_T を T の圧縮木 t の簡潔データ表現のコーディングとする．このとき，T の**圧縮コーディング** $Code(T)$ とは，区切り記号 $''!''$ で連結されたコーディング列 $EL_T \circ ''!'' \circ D_T \circ ''!'' \circ C_T$ をいう．なお，$\circ$ は文字列の連結を行う 2 項演算を表す．以後，T の圧縮コーディングを $Code(T) = \langle EL_T, D_T, C_T \rangle$ で表す．

EL_T, D_T, C_T は T の深さ優先走査に基づき作成されるため，容易に辺ラベル，リファレンスを返すハッシュ関数を作ることができる．図 1 に辺ラベル付き順序木 T の圧縮コーディングを与える．なお，C_T のコーディングにおいて，$\boxed{(}$, "#", "a", "b", "c" はそれぞれ 0, 1, 2, 3 に、"$", "P" はそれぞれ "-1", "-2" にコード化されている．

3. 頻出部分順序木枚挙アルゴリズム

辺ラベル付き順序木の集合 D と実数 σ $(0 < \sigma \leq 1)$ に対して，順序木 s が D における σ 頻出であるとは，s を部分グラフとして持つ D 中の辺ラベル付き順序木の数が $\lceil |D| \times \sigma \rceil$ 以上であるときをいう．辺ラベル付き順序木に対する圧縮コーディングの集合が与えられたとき，順序木構造を持つ σ 頻出な部分グラフを全て枚挙する問題を以下に定義する．

圧縮コーディング集合からの頻出部分順序木枚挙問題
インスタンス：圧縮コーディング集合 U と
実数 σ $(0 < \sigma \leq 1)$
問題：U の各圧縮木を解凍することなく D_U における σ 頻出な順序木構造を有する部分グラフを全て枚挙せよ．なお，D_U は U の各圧縮木を解凍して得られる辺ラベル付き順序木の集合である．

この問題を解く *EnuFreqSubTrees* アルゴリズムを Algorithm 1 に示す．*EnuFreqSubTrees* アルゴリズムは，最右拡張 [Asai et al. 2004, Zaki 2005, Itokawa et al. 2010] により順序木を枚挙する手法を用い，辺数に関しインクリメントしながら頻出な部分順序木を枚挙するアルゴリズムである．*EnuFreqSubTrees* アルゴリズムでは，j 番目のコーティング上の位置 i に順序木 p の最右葉に付随する辺が出現することを保持するために 4 つ組 (j, i, p, OP_i^j) で表現される出現位置管理構造を用いる．ここで，OP_i^j は位置 i がリファレンス木内のとき，そのリファレンス木へのリファレンスをもつ辺のリストである．*EnuFreqSubTrees* アルゴリズムの関数 makeCandFreqSubTrees は，辺数 ln の σ 頻出な順序木の出現位置管理構造の集合 Z_{ln} と辺数 1 の σ 頻出な順序木の出現位置管理構造の集合 Z_1 が入力として与えられたとき，辺数 $ln+1$ の順序木の出現位置管理構造の集合 Z_{ln+1} を返す関数である．makeCandFreqSubTrees では，Z_{ln} 内の各出現位置管理構造 (j, i, p, OP_i^j) に対して，j 番目の圧縮コーディングの位置 i から始まる p の最右パス上の各ノードから子 u を伸ばしてできるそれぞれの順序木について，次の 4 つの場合に分けて処理を行う．

(Case 1) p から u を伸ばすことで i が含まれる辞書内のリファレンス木から u が出てしまう場合，
(Case 2) 辞書内のリファレンス木内に位置 i が含まれかつ p' がそのリファレンス木内に止まる場合，
(Case 3) 位置 i を含むリファレンス木が辞書内にないが，u が辞書内のリファレンス木に入ってしまう場合，
(Case 4) 位置 i を含むリファレンス木が辞書内になく u を伸ばしても，どのリファレンス木にも入ることがなく，ポートリストが貼り付けられている辺にもならない場合．

Algorithm 1 *EnuFreqSubTrees*

Require: 圧縮コーディング集合 U と実数 σ $(1 < \sigma \leq 1)$
Ensure: 辺ラベル付き順序木集合 $D = \{T \mid$ 辺ラベル付き順序木 T の圧縮コーディングが U に含まれる $\}$ における全 σ 頻出順序木集合 F

1: P_1 を D における 1 辺のみからなる σ 頻出順序木の集合;
2: Z_1 を P_1 中の辺ラベルの出現位置管理構造の集合;
3: $F = P_1$ **and** $ln = 1$
4: **while** $P_{ln} \neq \emptyset$ **do**
5: $\quad P_{ln+1} = \emptyset$
6: $\quad C_{ln+1} = makeCandFreqSubTrees(Z_{ln}, Z_1)$
7: $\quad$ **for all** $z = (j, i, p, OP_i^j) \in C_{ln+1}$ **do**
8: $\quad\quad$ **if** p が D において σ 頻出 **then**
9: $\quad\quad\quad P_{ln+1} = P_{ln+1} \cup \{p\}$
10: $\quad\quad\quad Z_{ln+1} = Z_{ln+1} \cup \{z\}$
11: $\quad\quad$ **end if**
12: $\quad$ **end for**
13: $\quad F = F \cup P_{ln+1}$
14: $\quad ln{+}{+}$
15: **end while**
16: **return** F

全ての場合において，OP_i^j で管理されている各位置 k に対して，位置 i が辞書のリファレンス木内に含まれている，または u を伸ばすことでリファレンス木に入ったり，リファレンス木を参照するポートリストを表す辺に対応する場合には，同様の場合分け処理が必要である．Case 2 や Case 3 の場合には，解凍して得られる辺ラベル付き順序木において最右拡張した順序木が出現するか否かを確認する時間より余分に時間がかかる．しかし，Case 1 や Case 4 の場合で，は位置 i の処理と OP_i^j 内の位置 k の処理が同一となり，位置 k の処理を省くことができる．これにより計算時間が短縮できる．また，*EnuFreqSubTrees* を簡潔データ構造を用いて実装することにより，SDSL(Succinct Data Structure Library)[SDSL] を使用することができ，*EnuFreqSubTrees* の高速化と省メモリ化を図ることができる．

4. 実験と考察

本節では，前節で提案したアルゴリズム *EnuFreqSubTrees* を計算機上に実装し，人工データを用いた評価実験を行ったので，その実験結果を報告し考察する．

EnuFreqSubTrees を，主記憶メモリ 32GB，OS が MacOS Sierra で，4GHz の Intel Core i7 プロセッサを持つ計算機上に C++で実装した．辺ラベル付き順序木の圧縮コーディング集合に対するデータ構造として簡潔データ構造を採用し，各種関数は SDSL[SDSL] を使用した．また，圧縮木に現れる辞書のサイズを 5，辺ラベル種を 2 種類，解凍時のノード数を約 1000，圧縮率 (=ノード数減少率) を C に設定した 5 個の圧縮木を根で連結させた圧縮木 $D_C^5(1000)$ を，C が 20 から 80 まで 20 刻みでそれぞれランダムに作成した．さらに，圧縮率を 50 とし，ノード数を約 N に設定した圧縮木 $D_{50}^5(N)$ を，N が 1000 から 3000 まで 500 刻みでそれぞれランダムに作成した．加えて，上記の圧縮木 D に対応する非圧縮木もそれぞれ用意した．まず，4 つの圧縮木 $D_C^5(1000)$ $(C = 20, 40, 60, 80)$ とそれぞれ対応する非圧縮木に *EnuFreqSubTrees* を入力して与えた時の実行時間と圧縮木の圧縮率との関係を評価した．そ

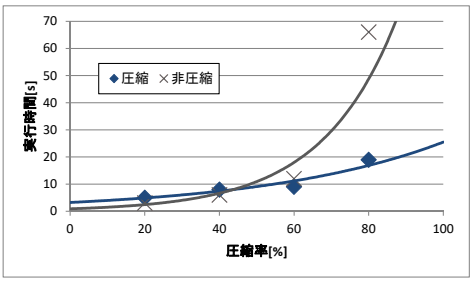


図 2: 実行時間と圧縮率の関係

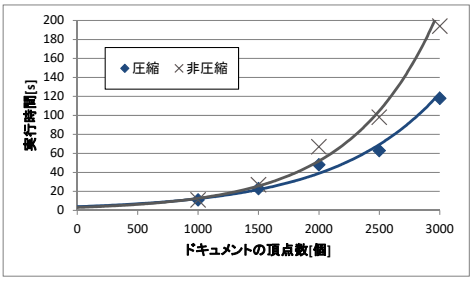


図 3: 実行時間とノード数の関係

の実験結果として，*EnuFreqSubTrees* の実行時間と圧縮率の関係を示したグラフを図 2 に示す．図 2 により，低圧縮率の圧縮木を与えた場合には，圧縮する前の辺ラベル付き順序木 (非圧縮木) を与えた場合より低速であるが，圧縮率が高くなるにつれて圧縮木を与えた方が非圧縮木の場合より高速なことがわかる．これは，低圧縮の場合，非圧縮木と比較してノード数の減少が少なく，圧縮木中のリファレンスに対する操作にかかる時間の方が実行時間に大きな影響を与えたものと考えられる．高圧縮率によるノード数の減少が実行時間に影響し，圧縮木に対する *EnuFreqSubTrees* アルゴリズムの実行時間が非圧縮木の場合より高速となった．さらに，6 つの圧縮木 $D^5_{50}(N)$ $(N = 1000, 1500, \cdots, 3000)$ とそれに対応する非圧縮木を用いて，ノード数と実行時間の関係を示したグラフを図 3 に示す．図 3 により，ノード数の増加に従って実行時間の差が開いていくことがわかる．圧縮率を固定してノード数を増やすと，辞書にあたるリファレンス木が大きくなる．このため，圧縮木の場合の頻出部分グラフの出現位置数と非圧縮木の場合の出現位置数の差が大きく開くことになり，圧縮木の場合の方が，非圧縮木の場合より高速になっていくと考えられる．これらの実験により，大きな繰返し構造を持つ巨大な辺ラベル付き順序木に対して *EnuFreqSubTrees* はより有効であることが実証された．

5. おわりに

本稿では，LZ 圧縮法を辺ラベル付き順序木の構造圧縮に拡張し，辺ラベル付き順序木 T を可逆的圧縮した圧縮木 t を紹介し，簡潔データ表現 DFUDS を拡張した圧縮木の簡潔データ表現を与えた．また，圧縮木 t の簡潔データ表現を用いた T の圧縮コーディングを与えた．さらに，辺ラベル付き順序木の圧縮コーディングの集合 $Compress(D)$ から解凍することなく $Compress(D)$ に対応する辺ラベル付き順序木集合 D に頻出する順序木構造を有する部分グラフを枚挙するアルゴリズム *EnuFreqSubTrees* を与えた．

今後の課題としては，実データを用いた評価実験を行うことや構造的変数を持つ順序木パターン p[Suzuki et al. 2015] と圧縮木 t が与えられたとき，t を解凍することなく t が表す辺ラベル付き順序木と p がマッチするかを判定するパターンマッチング手法への拡張が考えられる．

参考文献

[Asai et al. 2004] Asai, T., Abe, K., Kawasoe, K., Sakamoto, H., Arimura, H. and Arikawa., S.: Efficient substructure discovery from large semi-structured data. *IEICE TRANSACTIONS on Information and Systems* E87-D(12): pp.2754-2763 (2004).

[Horibe et al. 2017] Horibe, T., Itokawa, Y., Uchida. T., Suzuki. Y. and Miyahara, T.: Algorithm for enumerating all frequent paths from structurally compressed tree-structured data. *2017 IAENG International Conference on Computer Science* [2017 年 3 月 15-17 日, 香港, 発表予定] (2017).

[Itokawa et al. 2010] Itokawa, Y., Katoh, K., Uchida, T. and Shoudai, T.: Algorithm using expanded LZ compression scheme for compressing tree structured data. *Lecture Notes in Electrical Engineering*, Springer, pp.333–346 (2010).

[Itokawa et al. 2010] Itokawa, Y., Miyoshi, J., Wada, M. and Uchida, T.: Succinct Representation of TTSP Graphs and its Application to the Path Search Problem. *Proc. 6th IASTED International Conference on Advances in Computer Science and Engineering (ACSE 2010)*, pp.33-40 (2010)

[Jansson et al. 2012] Jansson, J., Sadakane, K. and Sung, W.-K.: Ultra-succinct representation of ordered trees with applications. *Journal of Computer and System Sciences* 78, pp.619-631 (2012)

[Raman et al. 2013] Raman, R. and Rao, S.S.: Succinct representations of ordinal trees. *Space-efficient data structures, streams, and algorithms*, *Proc. LNCS* Vol. 8066, pp.319-332 (2013)

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

[Suzuki et al. 2015] Suzuki, Y., Shoudai, T., Uchida, T. and Miyahara, T.: An efficient pattern matching algorithm for ordered term tree patterns. *IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences*, E98-A(06):1197–1211 (2015)

[Zaki 2005] Zaki, M. J.: Efficiently mining frequent trees in a forest: algorithms and applications. *IEEE Transactions on Knowledge and Data Engineering*, 17(8):1021–1035 (2005)

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