社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2004－AL－95 (2)
2004／5／21

順序木に対する逐次的な可逆圧縮手法

加藤 廣一郎[1], 内田 智之[2], 中村泰明[2]

[1] Department of Computer and Media Technologies, Hiroshima City University, Hiroshima 731-3194, Japan
katoon@toc.cs.hiroshima-cu.ac.jp
[2] Faculty of Information Sciences,
Hiroshima City University, Hiroshima 731-3194, Japan
{uchida@cs, nakamura@cs}.hiroshima-cu.ac.jp

Abstract. ネットワークや記憶装置技術の急激な発達とともに，厳密な構造を持たないが，木構造を持っている木構造データとよばれるデータが急増している．多くの巨大な木構造データを構造的に解析するには多くの時間を要するため，木構造データを構造を保持しながらできる限り可逆的に圧縮することができれば，その解析時間の短縮が望める．そこで，逐次的に与えられる木構造データを可逆的に圧縮する効率のよい手法を提案することを目的とする．本稿では，子に順番がつけられている根付き木である順序木で木構造データを表現する．そこで，逐次的に与えられる順序木に対する圧縮を定式化し，文字列上の圧縮手法の一つである LZSS 手法をもとに，順序木に対する逐次可逆圧縮アルゴリズムを提案する．さらに，高速な解凍アルゴリズムについても提案する．また，これらのアルゴリズムを計算機上に実装し，人工データを用いた評価実験の結果について報告する．

Sequential Algorithm for Compressing Ordered Tree without Loss

Koichiro Katoh[1], Tomoyuki Uchida[2], Yasuaki Nakamura[2]

[1] Department of Computer and Media Technologies, Hiroshima City University, Hiroshima 731-3194, Japan
katoon@toc.cs.hiroshima-cu.ac.jp
[2] Faculty of Information Sciences,
Hiroshima City University, Hiroshima 731-3194, Japan
{uchida@cs, nakamura@cs}.hiroshima-cu.ac.jp

Abstract. Due to the rapid growth of Information Technologies, electronic data which have no rigid structure but have tree structures have been rapidly increasing and each of them has become larger. Such data is called tree structured data. In general, analyzing large tree structured data is a time-consuming process in data mining. If we reduce the size of input data without loss of information including structural features, we can speed up such a heavy process. The purpose of this paper is to present an efficient lossless compression algorithm for sequential tree structured data. Tree structured data are represented by rooted trees each of whose internal node has ordered children. Such a tree is called an ordered tree. Firstly, we give a concept of lossless compression for an ordered tree. Secondly, based on a LZSS method which is one of lossless compression methods over strings, we present sequential algorithm for sequentially compressing a large ordered tree without loss. Moreover, we also present an efficient decompression algorithm for compressed ordered tree. Finally, in order to evaluate the performance of our algorithms, we report some experimental results.

1 はじめに

急速なネットワークや記憶装置の技術の発達とともに，HTML ファイルや XML ファイルのような電子データが急増している．これらのデータは厳密な構造を持っていないため，半構造化データと呼ばれている．また、木構造を持つ半構造化データは，木構造データと呼ばれ，Abiteboul ら [1] が提唱する Object Exchange Model を用いることにより，根付き木で表現することができる．特に、子供に順番がつけられて

いる根付き木は、順序木と呼ばれる．一般に構造を考慮した木構造データの解析には膨大な時間が必要である．解析対象の木構造データを構造を保持しながらできる限り圧縮することが，解析時間短縮には有用である．本研究の目的は，木構造データを表現する巨大な順序木が逐次的に与えられたとき，その順序木を可逆的に圧縮する効率のよい手法およびその解凍手法を提案することである。

一般のグラフ構造を有している半構造化データを対象とした圧縮手法として、Cook らの提案した Subdue システム [2] や糸川らの手法 [3] がある．また，山縣ら [4] は，松本ら [5] が提案した項木の概念を導入することにより，木構造データに対するグラフ文法にもとづく可逆的圧縮を提案し，さらにグラフ文法にもとづく最適な可逆的圧縮を求めることの困難性を示した．また，山縣らは，木構造データ内に頻出する部分構造をすべて検索し，その検索結果を用いて圧縮率のよい頻出部分構造を同じラベルを持つ超辺（変数という）で置き換える操作を繰り返すことで、圧縮対象である木構造データを表現するグラフ文法を作成する手法を提案している．本稿では，文字列を逐次的に圧縮する手法の一つである LZSS 手法 [6] を木構造データに応用した，可逆的な"逐次"圧縮手法および解凍手法を提案する．文字列上の LZSS 手法では、入力文字列 w 中に複数回出現する部分文字列 u が現れた場合，最初に u が出現した位置の先頭へのポインタと u の長さを示す値の組で置き換えることにより，w を逐次的に可逆圧縮している [7]．この文字列上の LZSS 手法をもとに，順序木に対する圧縮を，複数回出現する部分順序木へのポインタをもつハイパーツリーとその部分順序木を保存しておく外部辞書の組で定義する．つまり、順序木 T において、複数箇所に出現する部分順序木のうち、最初に出現する T の部分順序木 t へのポインタを外部辞書に保存し、t 以外の部分順序木は外部辞書内の t へポインタを参照する超辺で置き換えることにより，T からハイパーツリーを作成する．ことにより圧縮を実現する．外部辞書に保存される部分順序木 t へのポインタを，t の根へのポインタと t のすべての葉へのポインタの列で構成することにより，部分順序木を一意にかつ容易に抜き出すことが可能となる．これにより、ある順序木の圧縮を与えるハイパーツリーと外部辞書の組からもとの順序木を構成する解凍手法の高速化を行っている．また，本稿で提案した圧縮手法および解凍手法をコンピュータ上に実装し，人工的に作成した順序木を用いて評価実験を行ったので，その報告を行う．

本稿の構成は次の通りである．2 章で木構造データのデータモデルである順序木について定義し，さらに順序木の圧縮および解凍に関する諸定義を与える．また、3 章で、順序木に対する逐次圧縮および解凍アルゴリズムを与える．4 章では，人工的に作成された順序木を用いた評価実験の結果を報告する．5 章で結論を述べる．

2 諸定義

2.1 節に木構造データのデータモデルである順序木の定義を与え，2.2 節以降で順序木に対する可逆的な逐次圧縮および解凍に関する諸定義を与える．

2.1 順序木

XML ファイルなどの木構造を有する半構造化データは，葉に付随する辺にタグ間のテキストを，その他の辺には属性値を含むタグ情報を保持している根付き木で表現することができる [1]．特に，全ての内部頂

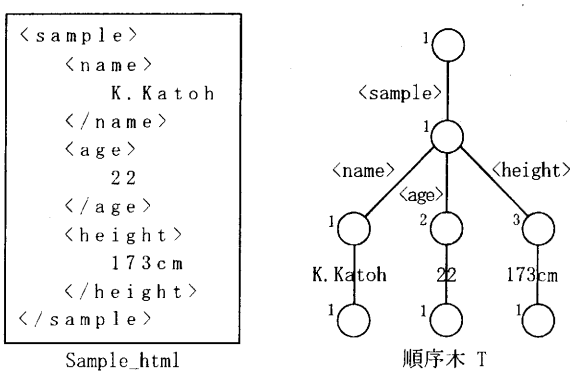


図 1. Sample.html のデータモデルである順序木 T

点に対して，その全ての子が順序もつような根付き木を**順序木**と呼ぶ．例として，図1にXMLファイルSample_htmlのデータモデルである順序木Tを示す．図1において，各頂点の左側にある番号は子の順序を表している．

Λを有限アルファベットとする．Λの各要素を辺ラベルと呼ぶ．Vを頂点集合，Eを辺集合とし，$T=(V,E)$を根付き木とする時，各辺はΛの要素を辺ラベルをとしてもつ．頂点$v,v',v''\in V$に対し，関係$<_v^T$をv',v''がvの子であり，v''はv'の弟ならば$v'<_v^T v''$と定義する．Tの各内部頂点vに対して2項関係$<_v^T$が成立している根付き木TをΛ上の**順序木**という．Λ上の順序木全体の集合をUで記す．

2.2 ハイパーエッジとハイパーツリー

Dを有限アルファベットとする．Dの各要素を辞書ラベルと呼ぶ．また，WをUの部分集合とする．γをD中の1つの辞書ラベルに対して，W中の1つの順序木を対応させる1対1写像とする．このとき，Wをγに関する**辞書**と呼ぶ．

$t=(V_t,E_t)$を順序木とする．以下の条件(1)〜(3)を満たす順序木を，$s=(V_s,E_s)$の**部分順序木**という．

(1) $V_s\subseteq V_t$
(2) $E_s\subseteq E_t$
(3) sにおける2項関係はtにおいても保持される．つまり，$v,v',v''\in V_s$に対し，$v'<_v^s v''$ならば，そのときに限り$v'<_v^t v''$である．

$s=(V_s,E_s)$と$t=(V_t,E_t)$を順序木とする．このとき，次の(1)と(2)を満たす全単射$\psi:U_s\to U_t$が存在する時，sとtは**同形**であると言う．

(1) $(\psi(u),\psi(u'))\in E_t$ならばそのときに限り，$(u,u')\in E_s$である．また，同じ辺ラベルをもつ．
(2) 2つ以上の子を持つsの頂点uと，uの2つの子u',u''に対して，$u'<_u^s u''$ならばそのときに限り，$\psi(u')<_{\psi(u)}^t\psi(u'')$である．

V_tを頂点集合とし，頂点$v_1,\ldots,v_n\in V_t$に対し，$\{v_1,\ldots,v_n\}\cup\{p(v_1),\ldots,p(v_n)\}=\phi$を満たす1対1写像$p:\{v_1,\ldots,v_n\}\to V_t$が存在するとする．このとき，頂点リスト$(v_1,\ldots,v_n)\in V_t^n$と$(p(v_1),\ldots,p(v_n))\in V_t^n$および辞書ラベル$d\in D$からなる3つ組み，

$$h=\left\langle (v_1,\ldots,v_n),d,(p(v_1),\ldots,p(v_n))\right\rangle$$

をハイパーエッジという．

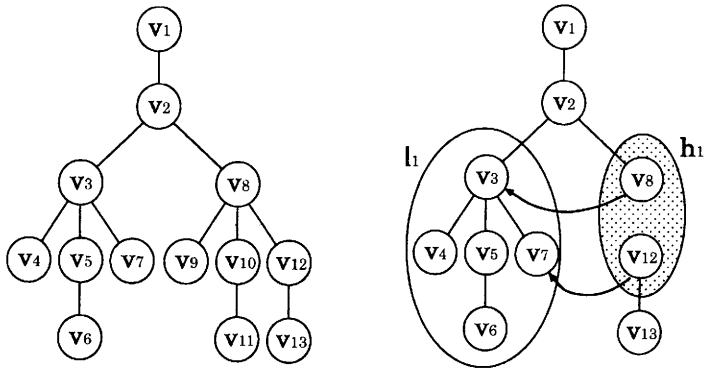
あるΛ上の順序木$t=(V_t,E_t)$に対し，次の条件(1)〜(5)を満たす，$g=(V_t,E,F)$を(Λ,D)上のハイパーツリーという．

(1) FはDの要素を辞書ラベルとしてもつハイパーエッジの有限集合であり，F中の任意の相異なる2つのハイパーエッジ$h_1=\left\langle (v_1,\ldots,v_n),d_x,(p(v_1),\ldots,p(v_n))\right\rangle$と$h_2=\left\langle (w_1,\ldots,w_n),d_y,(p(w_1),\ldots,p(w_n))\right\rangle$に対して，$\{v_2,\ldots,v_n\}\cap\{w_2,\ldots,w_n\}=\phi$である．
(2) $E=E_t-\bigcup_{h\in F}\left\{(v_1,v_i)|h=\left\langle (v_1,\ldots,v_n),d,(p(v_1),\ldots,p(v_n))\right\rangle,2\le i\le n\right\}$.
(3) tにおける2項関係はgにおいても保持される．つまり，$v,v',v''\in V_t$に対し，$v'<_v^t v''$ならばそのときに限り$v'<_v^g v''$である．
(4) 各$d\in D$に対し，tは順序木$\gamma(d)$と同形な部分順序木を持つ．
(5) 任意のハイパーエッジ$h=\left\langle (v_1,\ldots,v_n),d,(p(v_1),\ldots,p(v_n))\right\rangle\in F$において，$g$は，$p(v_1)$を根として持ち，$p(v_2),\ldots,p(v_n)$をこの順序で葉として持つ$\gamma(d)$と同形の部分順序木を持つ．

順序木における根と葉の定義と同様に，親を持たない頂点$v\in V_t$をgにおける根と呼び，子を持たない頂点$v\in V_t$をgにおける葉と呼ぶ．

2.3 圧縮と解凍

tをΛ上の順序木とする．$g=(V,E,F)$を(Λ,γ)上のハイパーツリーとする．g上のハイパーエッジ$h=\left\langle (v_1,\ldots,v_n),d,(p(v_1),\ldots,p(v_n))\right\rangle$を，以下の手順で$\gamma(d)$と同形な部分順序木と置き換えることを$(\Lambda,\gamma)$上の**置換**という．



(a) 順序木 t　　(b) ハイパーツリー g

図 2. 圧縮と解凍の関係

(1) $\gamma(d)$ のコピーを作る．コピーした木中の $p(v_1),\ldots,p(v_n)$ に対応する頂点をそれぞれ $v_1,\ldots,v_n$ と同一視し，h とコピーした木を置き換え，新しいハイパーツリー g' を作る．

(2) V_t をコピーした木の頂点集合とする．このとき，g' における頂点 v の子に対する順序関係 $<_v^{g'}$ を，以下の条件を満たすように修正する．

(a) $v,v',v''\in V_g$ の時，$v'<_v^g v''$ ならば $v'<_v^{g'} v''$．

(b) $v,v',v''\in V_t$ の時，$v'<_v^t v''$ ならば $v'<_v^{g'} v''$．

(c) $v=v_1\big(=p(v_1)\big),v'\in V_g-\{v_2,\ldots,v_r\},v''\in V_t$ の時，$v'<_v^g v_2$ ならば $v'<_v^{g'} v''$．

(d) $v=v_1\big(=p(v_1)\big),v'\in V_g-\{v_2,\ldots,v_r\},v''\in V_t$ の時，$v_r<_v^g v'$ ならば $v''<_v^{g'} v'$．

全ての g 上のハイパーエッジに対して，(Λ,γ) 上の置換を行って得られる順序木を g の**既置換木**という．

また，辞書 W は部分順序木を保持している．辞書 W は部分順序木 $t=(V_t,E_t)$ の頂点へのポインタを，$\{v_1,v_2,\ldots,v_n\}$ として保持する．頂点リストの左端の頂点 v_1 が t の根へのポインタをもち，頂点リストの左から 2 番目から順に，t の左端の葉へのポインタ，左端から 2 番目の葉へのポインタ，と順にもつようにする．よって，参照したい部分順序木の根と葉へのポインタを順番に並べて保持することで，少ない情報から部分順序木を一意に定ることができる．

t を順序木，g をハイパーツリーとする．このとき，g の既置換木と t が同形であり，g と辞書 W のサイズの和が t のサイズより小さければ，組 (g,W) は t の**圧縮**という．また，(g,W) から t と同形な既置換木を作成する手続きを**解凍**という．g の既置換木と t が同形になることにより，可逆的な圧縮となる．

$h=\Big\langle(v_1,\ldots,v_n),d,\big(p(v_1),\ldots,p(v_n)\big)\Big\rangle$ をハイパーエッジとする．ハイパーエッジのサイズ $|h|$ を

$$|h|=n+1+n=2n+1$$

と定義する．また，ハイパーツリー $g=(V,E,F)$ のサイズを，

$$|g|=|V|+2\times|E|+\sum_{h\in F}|h|$$

と定義する．さらに辞書 W のサイズを

$$|W|=\sum_{d\in W}\sharp d$$

と定義する．ここで，$\sharp d$ とは，d 中の要素の数である．

順序木 t と，ハイパーツリー g に対して，次式を t の g における**圧縮率**という．木 t が g へどの程度圧縮されたかを示す値である．

$$\left(1-\frac{|g|+|W|}{|t|}\right)\times 100$$

Algorithm: Tree_Compression

入力: 順序木 t, 最小頂点数 Min, 最大頂点数 Max

出力: ハイパーツリー g, 外部辞書 $exDic$

手法:

1. $g \leftarrow t$, $exDic = \phi$, $tmpDic = \phi$
2. g の各頂点 v_i を帰りがけ順で読み込み，以下の処理を行う．
3. $S = Make\_SubTree(g, v_i, Min, Max)$
4. $tmpDic, exDic$ がともに空ならば，$tmpDic \leftarrow S$ とし，2 に戻る．
5. S 内の各順序木 f に対し，頂点数が多い順に以下の処理を行う．
6. f と同形の順序木が $exDic$ に存在すれば圧縮する．2 に戻る．
7. f と同形の順序木が $tmpDic$ に存在すれば圧縮する．その後，f を $exDic$ に登録し，2 に戻る．
8. ハイパーエッジ g, 外部辞書 $exDic$ を出力する．

図 3. *Tree_Compression*

図 2 に圧縮と解凍の例を示す．図 2-(a) に順序木 $t = (V_t, E_t)$ を与える．図 2-(b) に順序木 t を圧縮して得られるハイパーツリー $g = (V_g, E_g, F_g)$ を与える．ここで，

$$V_g = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_{12}, v_{13}\},$$
$$E_g = \{\{v_1, v_2\}, \{v_2, v_3\}, \{v_3, v_4\}, \{v_3, v_5\}, \{v_5, v_6\}, \{v_3, v_7\}, \{v_2, v_8\}, \{v_{12}, v_{13}\}\}$$
$$F_g = \{\langle (v_8, v_{12}), d_1, (v_3, v_7)\rangle\}$$

である．辞書ラベル d_1 をもつ部分順序木 $t_{d_1} = (V_{d_1}, E_{d_1})$, はまた，辞書 $W = \{(v_3, v_4, v_6, v_7)\}$ から，

$$V_t = \{v_3, v_4, v_5, v_6, v_7\},$$
$$E_t = \{\{v_3, v_4\}, \{v_3, v_5\}, \{v_5, v_6\}, \{v_3, v_7\}\}$$

と，一意に定まる．

辞書ラベル d_1 をもつ部分順序木 t_{d_1} のコピーを，ハイパーツリー g 上のハイパーエッジ h_1 と，置き換えることにより，順序木 t を得ることができる．

3 逐次圧縮および解凍アルゴリズム

この節では，順序木に対する逐次圧縮および解凍アルゴリズムについて述べる．

3.1 逐次圧縮アルゴリズム

順序木 t が与えられたときに，t の圧縮を与えるハイパーツリー g と外部辞書 $exDic$ の組を出力するアルゴリズム Tree_compression を図 3 に記す．

ハイパーツリー g, g の頂点 v_i, 値 Min, Max が与えられたとき，関数 *Make_Subtree* は，頂点 v_i を根の右端の子としてもつ，以下の条件を満たす g の部分順序木 $f = (V_f, E_f)$ 全体の集合を返す．

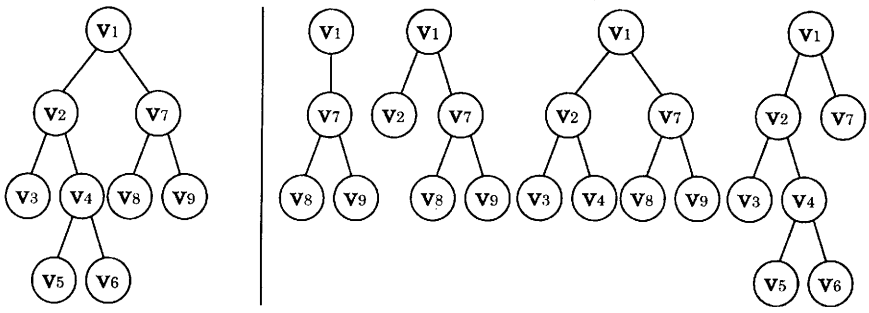


図 4. 順序木 t において，v_7 における Make_SubTree の実行例

Algorithm: HyperTree_Decompression

入力: ハイパーエッジ g, 外部辞書 $exDic$

出力: 順序木 t

手法:

1. $t \leftarrow g$
2. t の各頂点 v_i を幅優先で読み込み以下の処理を行う.
3. 読み込まれた頂点 v_i から派生する辺がハイパーエッジの場合,
   ハイパーエッジ中の辞書ラベルに対応する部分順序木のコピーを生成し,
   ハイパーエッジに生成された部分順序木のコピーを代入する.
4. 順序木 t を出力する.

図 5. *HyperTree_Decompression*

(1) $Min \leq \sharp V_f \leq Max$

(2) $\forall v \in V_t, \forall u \in V_g - V_t$ に対し, もし $\{v, u\} \in E_g$ ならば v は t の根かあるいは葉である.

値 Min, Max により, 関数 Make_SubTree により返される部分順序木の個数を制限する. サイズの大きな部分順序木は入力となる順序木 t に頻出する可能性は低い上, 同形判定に要する時間も増加してしまう. 逆に, 頂点数が少ない部分順序木を圧縮候補としても, 圧縮効率の向上は望めない. よって, 入力データにより Min, Max を適切な値に設定する必要がある.

例として順序木 t と, t の頂点 v_7, Min が 4, Max が 7 として与えられたときに関数 *Make_Subtree* から返される部分順序木全体の集合 S を図 4 に示す.

アルゴリズム Tree_Compression の 6, 7 行目の圧縮するという部分では以下の処理を行う, S 中の部分順序木と同形な, 辞書中の部分順序木の根と葉を対応させる. このとき, S 中の部分順序木の葉である頂点に対応する, 入力となる順序木の頂点に子がいない場合, その頂点を削除する. さらに S 中の部分順序木の, 根でも葉でもない頂点も削除し, 圧縮を行う. また, 削除できる頂点が 1 つも存在しない場合は圧縮を行わない.

アルゴリズム Tree_compression は 2 種類の辞書をもつ. $exDic$ は外部辞書で, 圧縮の対象となった部分順序木を保持しており, 圧縮終了後にハイパーツリーとともに出力される. $tmpDic$ は一時的な仮の辞書であり, 圧縮候補の部分順序木を保持する. これらの部分順序木は, 圧縮終了時に圧縮の対象となっていないため, 破棄される.

また, 入力データにより, 圧縮候補の部分順序木の個数が膨大になる場合がある. この場合, 同形判定に要する時間が増加してしまう. そのため, 圧縮候補の部分順序木を保持する $tmpDic$ のデータ構造をあらかじめ設定した大きさのキューで実現する. これにより, 圧縮候補の部分順序木の個数を制限でき, 一杯になった $tmpDic$ に新しい部分順序木が登録されると, $tmpDic$ 中の最も古い部分順序木が削除されることになる.

アルゴリズム Tree_Compression では順序木の頂点を深さ優先に逐次読み込んで行く. そして新しい頂点を読み込むごとに関数 *Make_Subtree* を呼び出し圧縮候補の部分順序木を生成し, その後ハイパーエッジとの置き換えを行うことにより, 逐次的な圧縮を実現している.

3.2 解凍アルゴリズム

ハイパーツリーと辞書が与えられたとき, 解凍した結果である順序木を出力するアルゴリズム HyperTree_Decompression を図 5 に示す.

このアルゴリズムは, ハイパーツリーを, 幅優先で読み込んでいく. 読み込まれた頂点が, ハイパーエッジ中の頂点であるときは, そのハイパーエッジ中の辞書ラベルに対応する部分順序木のコピーを生成し, 生成された部分順序木のコピーを対応するハイパーエッジ中の頂点に置き換えて新しいハイパーツリーを作成する. この置換をハイパーツリー内のハイパーエッジがなくなるまで行い, 既置換木を作成する.

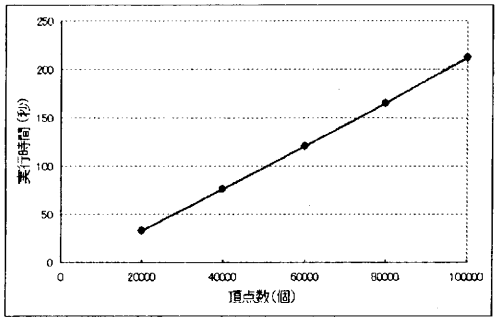


図 6. 頂点数と圧縮時間の関係

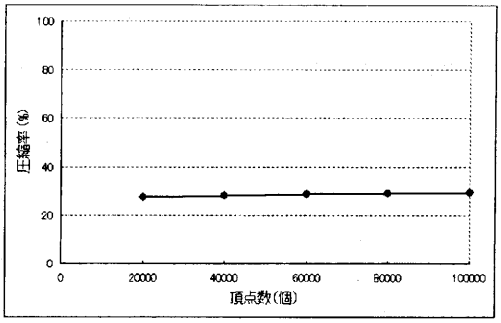


図 7. 頂点数と圧縮率の関係

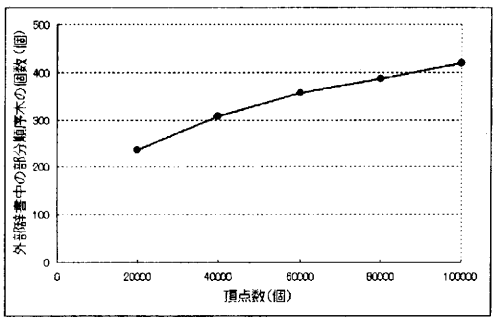


図 8. 頂点数と外部辞書の部分順序木の個数の関係

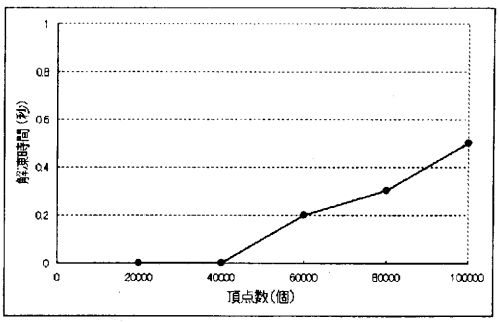


図 9. 頂点数と解凍時間の関係

4 実験および実験結果

4.1 実験環境

前述したアルゴリズム Tree_Compression と HyperTree_Decompression をコンピュータ上に実装し，人工データを用いた評価実験を行った．設定は，各頂点の次数が 4 以下，辺ラベルが 2 種類の木で，頂点数がそれぞれ 20,000，40,000，60,000，80,000，100,000 個のランダムに生成した木を入力データとして用いた．各頂点数での圧縮率，圧縮時間，外部辞書に登録された部分順序木の数と解凍時間について，それぞれの値を異なる木を用いて 10 回ずつ計測した．

本実験では以下の実験装置を用いた．

- OS : Red Hat Linux
- 計算機 : Intel XEON(TM)2.40GHz，主記憶容量 1.00GB

図 7 に頂点数と圧縮時間の関係を，図 8 に頂点数と圧縮率の関係を，同様に図 9 と図 10 に頂点数と解凍時間，保存辞書の個数との関係をそれぞれ示す．それぞれのデータは，各頂点数に対して 10 回ずつ計測して得られた値の平均値である．

4.2 実験結果

図 6 において，圧縮時間が頂点数が増えるにつれ線形に推移していることが分かる．これは本システムは圧縮候補の部分順序木を一時的に保存しておく仮の辞書の容量を制限したため，圧縮する部分順序木の検索の範囲がほぼ一定に保たれたからだと考えられる．また，圧縮候補の部分順序木のうち，圧縮対象となった部分順序木は，圧縮後もデータが保持される外部辞書に順次登録されていく．本システムでは圧縮候補となる部分順序木の最小頂点数を 4，最大頂点数を 7 と制限した．また，辺ラベルも 2 種類と制限していることから，圧縮候補の部分順序木の種類が限られてくる．

このことに対して，本システムでの仮の辞書に登録できる部分順序木の個数を10,000個と，外部辞書の容量よりもはるかに大きくなるように設定した．このために外部辞書の容量を考慮しても，圧縮する部分木の検索の範囲がほぼ一定に保たれ，結果として圧縮時間が線形に推移したと考えられる．

さらに頂点数が多い場合では，ほとんど全ての種類の圧縮対象の部分順序木が仮の辞書に登録される．その後は，仮の辞書の容量が増加しなくなり，圧縮する部分順序木の検索の範囲が一定に保たれる．このような場合では，圧縮時間が線形に推移するようになると考えられる．

図7において，圧縮率は頂点数の増加とは関係なくほぼ一定であることが分かる．本システムは圧縮候補の部分順序木を圧縮する際，その圧縮対象となった部分順序木のデータを圧縮終了後も保持する．そのため，ある部分順序木が一回しか圧縮されなければ，むしろ圧縮後のサイズが増加する場合がある．逆に同じ部分順序木が圧縮する木構造データ中に頻出する場合は，圧縮後のサイズがより小さくなるという特徴がある．実験では，読み込んだ頂点数が少ない時，すでに，ある程度頻出する部分木が発見されている．しかし，読み込んだ頂点数が多くなるにつれ，頻出する部分順序木も圧縮されていくが，あまり頻出しない部分順序木も発見され圧縮される．また，図8では頂点数が20,000個のとき，圧縮対象となった外部辞書内の部分順序木がすでに約230個存在している．しかし，頂点数が5倍の100,000個の時でも，外部辞書内の部分順序木の個数は約410個までしか増加していない．そのため，頻出する部分順序木の圧縮により減少したサイズが，あまり頻出しない部分順序木の圧縮によるサイズの増加により，相殺されたためだと考えられる．しかし，前述した通り，読み込んだ頂点数がさらに大きくなると，新しく発見される部分順序木のほとんどがあまり頻出していないためだと考えられる．その後，仮の辞書に新しく登録される部分順序木がほぼなくなるため，徐々に圧縮率が改善されていくと考えられる．

図9からは，頂点数が100,000個の時でさえ，実行時間が約0.5秒と高速であることが分かる．これは，本提案手法がLZSS手法の特徴を有していることを示している．

5 まとめ

本研究では，LZSS手法[6]を基に，順序木の可逆的な逐次圧縮アルゴリズムと解凍アルゴリズムを提案した．さらに圧縮及び解凍アルゴリズムを計算機に実装し，人工データを用いた評価実験を行った．本手法は，入力の順序木において，同形の部分順序木が頻出する場合に圧縮率が向上する．特定の条件の元でランダムに生成された順序木を用いたため，満足な圧縮率が得られなかった．しかし，実データでは，ランダムデータに比べデータにかたよりがある場合が多く，データ量も膨大である．ゆえに実データを用いることで，圧縮率が向上すると考えられる．本手法は，圧縮時間が頂点数に対し線形に推移し，解凍時間も非常に高速であることから，充分高速な圧縮手法と解凍手法であると言える．

References

1. S. Abiteboul, P. Buneman, and D. Suciu, Data on the Web : From Releations to Semistructured Data and XML, Morgan Kufmann, 2002.
2. D. J. Cook and L. B. Holder : Graph-Based Data Mining, IEEE intelligent Systems, Vol.15, No.2, pp.32-41, 2000.
3. Y. Itokawa, T. Uchida, T. Shoudai, T. Miyahara and Y. Nakamura "Finding Frequent Subgraphs from Graph Structured Data with Geometric Information and Its Application to Lossless Compression", Proc. (PAKDD 2003), LNAI 2637, pp.582-594, 2003.
4. K. Yamagata, T. Uchida, T. Shoudai and Y. Nakamura, "An Effective Grammar-Based Compression Algorithm for Tree Structured Data", Proc. 13th International Conference on Inductive Logic Programming (ILP 2003), pp.383-400, LNAI 2835, Springer, 2003.
5. S. Matsumoto, Y. Hayashi, and T.Shoudai Polynormal time inductive inference of regular term tree languages from positive data. *Proc. ALT-97, Springer-Verlag, LNAI 1316*, pp.212-227, 1997. T. Uchida, T. Shoudai and S. Miyano : Parallel Algorithms for Refutation Tree Problem on Formal Graph Systems, IEICE Trans. Info. & Syst. Vol.E78-D, No.2, pp.99-112, 1995.
6. James A. Storer and Thomas G. Szymanski : The Macro Model for Data Compression, 10th Annual Symposium of the Theory of Computing , pp.30-39, 1978.
7. Jacob Ziv and Abraham Lempel: A Universal Algorithm for Sequential Data Compression, IEEE Transactions on Information Theory, Vol. IT-23, No.3, pp.337-343, 1977.