DEIM Forum 2012 E11-5

ビット列反転を利用した XML 木のコンパクトなラベリング手法の提案

大野　一樹† 　波多野賢治† 　横田　治夫††

† 同志社大学文化情報学部　〒 610-0394 京都府京田辺市多々羅都谷 1-3
†† 東京工業大学大学院情報理工学研究科計算工学専攻　〒 152-8552 東京都目黒区大岡山 2-12-1
E-mail: †bii0175@mail4.doshisha.ac.jp, †khatano@mail.doshisha.ac.jp, ††yokota@cs.titech.ac.jp

あらまし　XML データの管理では，各ノードへのアクセスを高速化するための手法として，ノードに対するラベリング手法が必要となっている. 世の中のあらゆるデータが XML で表現されつつあることを考慮すると，ラベルサイズを圧縮することは携帯デバイス等で XML データを扱う際には特に重要となる．そこで，本稿では XML 文書に対するラベルサイズの小さなラベリング手法を提案する．また，提案手法の有効性を確認するために XML データのパタン別に XML 木に対するラベルの付与に関する既存の手法との比較実験を行った．その結果，兄弟ノードの数が XML 木を占める割合が増加するにつれて，従来手法と比較して約 20

キーワード　XML , ノードラベリング，コンパクト

A Compact Labeling Scheme for XML Document using Inverted Bit Array

Kazuki OHNO†, Kenji HATANO†, and Haruo YOKOTA††

† Faculty of Culture and Information Science, Doshisha University
1-3 Tatara-Miyakodani, Kyotanabe, Kyoto, 610–0394 Japan
†† Department of Computer Science, Tokyo Institute of Technology, 2-12-2 Ookayama, Meguroku, Tokyo, 152–8552 Japan
E-mail: †bii0175@mail4.doshisha.ac.jp, †khatano@mail.doshisha.ac.jp, ††yokota@cs.titech.ac.jp

1. はじめに

近年， XML 形式のデータが増加しており，大量の XML 文書の管理を効率よく行うための研究が行われている．その中でも， XML 文書を木構造と見立てた図 1 のような XML 木 に対するノードへの高速なアクセスのための手法の一つとして，ノードにラベルを割り当てるラベリングと呼ばれる手法が研究されてきた [1] ．ノードにラベルを付与しておくことによりノードの位置情報をラベルから取得することができるため，高速に XML 木へのアクセスが可能となる．しかし， XML 文書は頻繁に更新が発生することが考えられるため，場合によっては更新のたびにすべてのラベルの付け替えが発生するといった問題が生じる．この問題を解決するために，ラベルの更新コストに着目した研究が行われている [2].

他にも， XML 文書に対するラベリング手法においては，ラベルのサイズも懸念される問題として挙げられる．その例として，記憶容量が小さい携帯端末等においても高速に XML 文書の処理を行うためには， XML 木のノードに対して小さなサイズのラベル付与を行うことが求められる．そのため，さまざまなラベリング手法に対するラベルサイズの圧縮に関する研究も

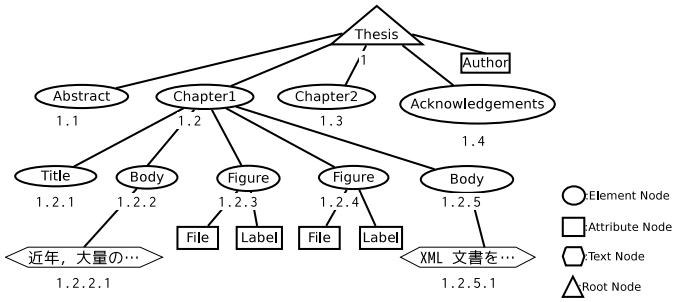

図 1　XML　木

行われている [3] [4].

本稿では ラベルサイズの圧縮を考慮した， XML 木に対するラベリング手法の一つである圧縮 bit 列 DO-VLEI が抱えている問題点を取り上げ，さらなるラベルサイズの圧縮を目的として新たなラベル圧縮手法であるビット列反転型圧縮 bit 列 DO-VLEI を提案し，比較実験を通して，ラベルサイズの圧縮性能を検証する．

2. 関 連 研 究

本節では関連研究として XML 木に対するラベリング手法のうち，更新コストの低いラベリング手法を中心に述べる．また，更新コストの低いラベリング手法の一つである DO-VLEI [2] および，圧縮 bit 列 DO-VLEI [4] について詳細とそれらの問題点を述べる．

2.1 更新コストの低いラベリング手法

XML 文書は頻繁に更新が発生するため，更新が発生するたびにノードの位置関係が変更される．それゆえ， XML 文書の更新のたびに， XML 木の各ノードに対しラベルの付け替えを行う必要が生じる．このような問題の解決のために，ラベルの付け替えにかかる更新コストの削減を目指した Dewey Order (注1) を利用する方法や前順後順法 [5] 等のラベリング手法が提案されてきた．

しかし， Dewey Order のようにラベリングの際に単純な連続する整数を用いると，新たなノードの挿入の際に兄弟関係の位置にあるノードについて再ラベリングを行わないとノードの挿入を反映できないため，更新に伴って大規模なラベルの付け替えが生じることになる．そのため，更新時の再ラベリングのコストを抑える手法として，範囲ラベリング法 [6] や範囲ラベリング法を改良した更新コスト削減のためのラベリング手法 [7]，などが提案されている．範囲ラベリング手法の改良 [7] では，小規模な再ラベリングを頻繁に行うことで大規模なラベルの付け替えを未然に防ぐことができるが，ラベルの付け替えが起こることに変わりはなく， XML 木に対して再ラベリングなしに，無制限にノードの挿入等の操作が行えるわけではないことが問題点として挙げられる．

その他にもラベルの更新コストを抑えるために，ラベル間の大小関係に各ラベルの先頭からの数値の大きさを利用することで再ラベリングを避ける QED が提案されており [8]，さらに，QED を利用しつつ QED によるラベルをビット列と “.” で表現することにより，ラベルの更新について高速に行える手法も提案されている [9]．しかし， QED はラベルの更新なしにノードの挿入が行えるもののラベリングの際にラベルの大きさについて最小，最大の値を持つラベルを定義しなければならないため，その範囲外の値を持つラベルを挿入するときにはラベルの付け替えが発生するという問題点が存在する．

このような再ラベリングが発生する問題に対して，既存の XML 木のノードの再ラベリングなしに無制限にノードを挿入することが可能なラベリング手法である Dewey Order と VLEI コードを組み合わせた DO-VLEI が提案されており [2]，その評価に，浮動小数点数法 [10] と範囲ラベリング手法の改良 [7] との比較が [2] 内にて行われている．両手法よりノードの検索，挿入の処理において再ラベリングの頻度が少なく，ラベルからのノードの XML 木おける位置情報についても高速に処理できるため，これらの点で有用性が高いことが示されている．

2.2 DO-VLEI

DO-VLEI は親ノードのラベルの末尾にデリミタとよばれる区切り記号である “.” を加えて親子関係を表現する Dewey Order を拡張し，兄弟ノード間の順序関係の表現に VLEI コードを採用した XML 文書のラベリング手法である．本節では，はじめに VLEI コードについて解説を行い，その特徴を述べた上で DO-VLEI を用いたラベリングについて述べる．

2.2.1 VLEI コード

VLEI コードは 1 から 開始される 0,1 の可変長のビット列であり，特殊な大小関係を持つ．v を 1 から始まり， 0,1 から構成されるビット列とし，以下の大小関係を満たす場合に以下の式において VLEI コードとして定義される．

$$v \cdot 0 \cdot \{0|1\}^* < v < v \cdot 1 \cdot \{0|1\}^*$$

ここで， ” < ” は VLEI コードの大小関係を意味し， “ · ” は文字列と数字の連結を意味する．Algorithm 1 に従えば，あるラベル v に対して，後方に 0 の付くものは v よりも小さく，同様に後方に 1 の付くものは v よりも大きいという大小関係が成り立つ．

VLEI コードを用いる利点として，任意の VLEI コード間に理論上は無制限に VLEI コードを生成することが可能であるという点が大きい．したがって，兄弟ノード間のノードの挿入において，既存のノードに対して再ラベリング処理を行うことなく，更新コストの低いノードの挿入処理が可能となる．また，VLEI コードを用いたノードの挿入は Algorithm 1 に基づいて行われる．

Algorithm 1 InsertVLEI (v_l, v_r)

```
if length(v_l) ≦ length(v_r) then
    v_i = v_r · 0
else
    v_i = v_l · 1
end if
return v_i
```

つまり，挿入される位置の前方のノードのラベルサイズ $length(v_l)$ が挿入される位置の後方のノードのラベルサイズ $length(v_r)$ と等しいとき，あるいは後方のノードのラベルサイズと比較して小さいときに後方のノードのラベル v_r の末尾に “0” を挿入してラベリングが行われ，それ以外のとき，つまり前方のノードのラベルサイズ v_l が後方のラベルサイズ v_r と比較して大きいときには前方のラベル v_l の末尾に “1” が挿入されてラベリングが行われ， XML 文書のラベルの更新が完了する．

(注1)：Online Computer Library Center. Dewey decimal classification. http://www.oclc.org/dewey/

2.2.2 DO-VLEI によるラベリング

DO-VLEI の定義を以下に記す.

（1） root ノードのラベルを “1” とする.

（2） ノードの親子関係は VLEI コードをデリミタで区切ることで表現される.

（3） ノードの兄弟関係は VLEI コードの大小関係で表される.

このとき，あるノードについて， DO-VLEI による親ノードのラベルを D_{parent} とすると，兄弟ノード間の順序を表す VLEI コードが v_{child} であるノードの D_{child} は次のように表される.

$$D_{child} = D_{parent} \bullet v_{child}$$

また， DO-VLEI の適用例として， DO-VLEI を用いてラベリングを行った XML 木を図 2 に示す.

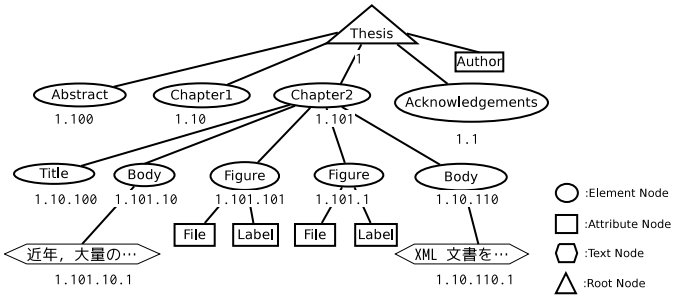

図 2 DO-VLEI による ラベリング

2.3 圧縮 bit 列 DO-VLEI

DO-VLEI によってラベリングされたラベルを圧縮するための手法として圧縮 bit 列 DO-VLEI が提案されている．圧縮 bit 列 DO-VLEI は DO-VLEI の構成条件を “.1” “1” “0” の 3 パターンに分けることで，次のように定義される.

（1） DO-VLEI “.1” をビット列 “11” に割り当てる.

（2） DO-VLEI “1” をビット列 “10” に割り当てる.

（3） DO-VLEI “0” をビット列 “0” に割り当てる.

圧縮 bit 列 ORD-PATH [3] との比較において，同ビット数において表現可能なラベルのパタンが圧縮 bit 列 DO-VLEI では多いことが示されており，圧縮 bit 列 DO-VLEI のラベル圧縮の有効性が実証されている.

また，圧縮 bit 列 DO-VLEI によって付与されたノードのラベルに対して，深さや親ノードなどの XML 木の構造情報を高速に取得できる構造情報抽出手法が提案されている [11]．ここで提案されている構造情報抽出手法を用いることにより，圧縮 bit 列 ORD-PATH との比較において問合せ処理においてアクセス速度の有効性が示されている．これらより，圧縮 bit 列 DO-VLEI は DO-VLEI に対してラベルサイズを圧縮したにもかかわらず，問合せ処理性能も向上できているということがわかる.

圧縮 bit 列 DO-VLEI の適用例として，図 2 に対して，圧縮 bit 列 DO-VLEI の適用例を図 3 に示す.

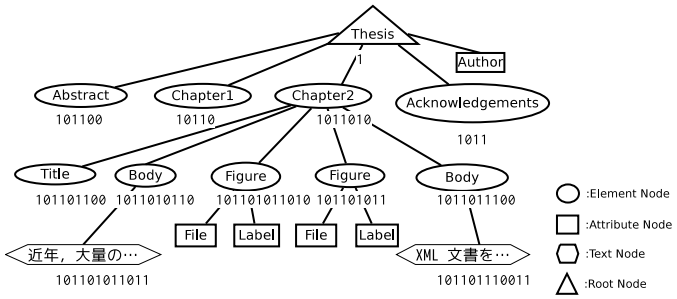

図 3 圧縮 bit 列 DO-VLEI による ラベリング

圧縮 bit 列 DO-VLEI の問題点は，その定義から VLEI コードの値が大きい，つまり， 右側に “1” を多く取るような VLEI コードラベリングが行われる兄弟ノードの挿入が頻繁に行われると， XML 木全体のラベルサイズが大きくなる傾向が存在する点である．そのため， “1” または “1” が 連続で出現するような DO-VLEI のコードに対してできるだけ短いビット列に置き換えることでラベルサイズを圧縮できると考えられる．しかし，単純にビット列 “1” を ビット列 “0” に対応する “0” 以外のもうひとつの 1 ビットで表現したところでは復号化の際に DO-VLEI におけるデリミタを解釈することができない.

3. 提 案 手 法

本節では XML 文書のラベリング手法のうち DO-VLEI によって生成されたラベルの圧縮手法である圧縮 bit 列 DO-VLEI について，前節で挙げた問題点を解決することを目的としてビット 列反転型圧縮 bit 列 DO-VLEI の提案を行う.

ビット列反転型圧縮 bit 列 DO-VLEI の VLEI コード “1” については圧縮 bit 列 DO-VLEI が “11” と符号化していたのに対し， “1” で符号化する．これを行うことで，DO-VLEI において “1” が大量に出現する場合に圧縮 bit 列 DO-VLEI において ラベルサイズが増加するという問題に対応できると考える．つまり， DO-VLEI において “1” が出現すればした数だけ，提案手法においてラベルサイズを圧縮できる.

また，デリミタとその後に必ず出現する VLEI コードの “1” の並びである DO-VLEI コード “.1” については，圧縮 bit 列 DO-VLEI ではビット列 “11” と 2 ビットで表現されているため，ラベルサイズの観点から考え，ビット列反転型圧縮 bit 列 DO-VLEI においても，同時に “11” を 2 ビットで表現する．そのため， DO-VLEI コード “.1” には bit 列 “11” を割り当てるため， DO-VLEI コード “11” に対しては bit 列 “1111” を割り当てることになる.

ところで，以上の割り当てを行うと，デリミタを表現することが困難となることは容易に想像できる．そのため，デリミタごとに VLEI コードの反転を用いることで，通常のデリミタに相当するビット列とそれを反転したビット列の二つのパタンでデリミタを表現する.

この手順は Algorithm 2 で表現でき，図 1 に対して， Algorithm 2 を適用した例を図 4 に示す.

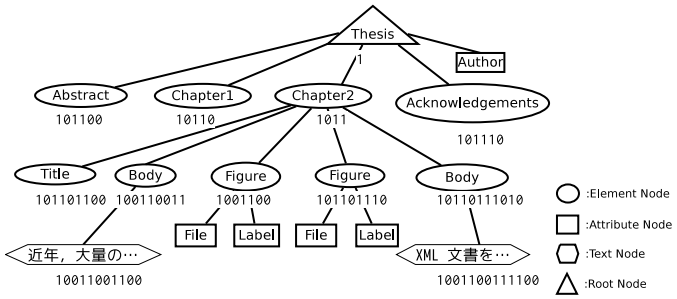

図 4　ビット列反転型圧縮 bit 列 DO-VLEI による ラベリング

Algorithm 2 ビット列反転型圧縮 bit 列 DO-VLEI

関数 $split(s,x)$: 文字列 s を文字 y で区切った文字列を要素とした配列を返す
$replace(s,x,y)$: 文字列 s 中の x を y に置換
$substr(s,x)$: 文字列 s の x 文字目以降を取り出す
入力 $dovlei$: $DO-VLEI$ によるラベル
dovleils ← $split(dovlei,$".")reversebitdovlei ← "1"
for $i \leftarrow (1 \rightarrow dovleils[length(dovleils)]1$ **do**
  if $reversebitdovlei[length(reversebitdovlei)] =$ "1" **then**
    $reversebitdovlei \leftarrow reversebitdovlei$ ・"00"
  else
    $reversebitdovlei \leftarrow reversebitdovlei$ ・"11"
  end if
  if $dovleils[i][length(dovleils)] =' 1'$ **then**
    if $length(dovleils[i]) \geqq 2$ **then**
      $dovleils[i] \leftarrow$ "0"・$dovleils[i]$
      $dovleils[i] \leftarrow replace(dovleils[i],$"1","$a$")
      $dovleils[i] \leftarrow replace(dovleils[i],$"0","1")
      $dovleils[i] \leftarrow replace(dovleils[i],$"1$aa$","10000")
      $dovleils[i] \leftarrow replace(dovleils[i],$"$a$","0")
      $reversebitdovlei \leftarrow reversebitdovlei \cdot substr(dovleils[i],1)$
    end if
  else
    if $length(dovleils[i]) \geqq 2$ **then**
      $dovleils[i] \leftarrow$ "0"・$dovleils[i]$
      $dovleils[i]$
    end if
    $reversebitdovlei \leftarrow$
      $reversebitdovlei \cdot substr(dovleils[i],1)$
  end if
end for
return $reversebitdovlei$

4. 評 価 実 験

本節では提案手法の有効性を示すため，さまざまな XML 文書を用いて評価実験を行う．

具体的にはビット列反転型圧縮 bit 列 DO-VLEI と圧縮 bit 列 DO-VLEI を用いて，実際に XML 文書にラベルの付与を行い，それらを別々のデータベースに格納する．そして，データベースに格納されているラベルの長さの総和を求めることでラベルサイズの比較を行った．実験環境は表 1 のとおりである．

表 1　実 験 環 境

OS	Arch Linux
CPU	Intel Core2 Quad CPU Q9450 2.66 GHz
RAM	4GB
RDB	SQlite 3.7.8

4.1　ラベル付け

実際に XML 文書にラベルを付与してデータベース上で扱うため，評価実験では XML 文書を RDB に格納する場合を想定し，XML 文書を DOM (Document Object Model) 木として取り込み，表 2 〜 4 で表される要素ノードテーブル，属性ノードテーブル，テキストノードテーブルの三種類のテーブルに分けて RDB に格納する．テキストノードは要素ノードの子ノードとして定義され親ノードが判別できれば，親ノードのラベルからラベルは計算することが可能でありラベル付を行う必要がない．属性ノードについても, 一つの要素に付随する複数の属性間には順番が存在せず，どの要素に付随しているかのみが重要であるので，属性ノードへのラベル付けも行う必要がない．ゆえに，三種類のテーブルの中でラベルサイズに依存してテーブルのデータサイズが変化するのは要素ノードテーブルのみである．

XML 木についての初期のラベルの付与は XML 文書に対する初期状態のラベルサイズを最小に抑えるため Algorithm 3 を用いて行う．DO-VLEI を用いてラベリングを行う際において，Algorithm 3 は最初に兄弟ノード郡のラベリングの基準ノードの VLEI コードである “1” を親ノードのラベルと結合したラベルを付与する要素ノードを決定する．さらに，VLEI コード “1” が付与される要素ノードである兄弟ノード郡における基準ノードが決定されることで，基準のラベルが付与された要素

ノードとラベルが付与される要素ノードとの順序関係によって兄弟ノード全体のラベルサイズが最小になるようにラベルを付与していくことができる．

データベースサイズの比較は要素ノードのラベルの長さ，つまり表 2 の label より計算した値の合計を用いて行う．なぜなら，提案手法の目的はラベルサイズの削減にあるからである．

Algorithm 3 初期のラベルの決定

入力 a_i: ノードの出現位置
　　N: 兄弟ノード数
int $m \leftarrow \lceil log_2 N \rceil$
int $P \leftarrow 2^m$
string $v_i \leftarrow 1$
int $x \leftarrow a_i - P$
while $x \neq 0$ **do**
　int $P \leftarrow \frac{1}{2}P$
　if $x > 0$ **then**
　　$v_i \leftarrow v_i$・1
　　$x \leftarrow x - P$
　else
　　if $x < 0$ **then**
　　$v v_i$・0 $x \leftarrow x + P$
　　end if
　end if
end while
return v_i

表 2　要素ノードテーブル

id	Integer	自身の id
label	Varchar	自身のラベル
name	Varchar	要素名

表 3　テキストノードテーブル

id	Integer	自身の id
element	Integer	親要素の id
text	Varchar	テキストの内容

表 4　要素ノードテーブル

element	Integer	親要素の id
name	Varchar	属性名
value	Varchar	属性の値

4.2　一般的な XML データにおける実験

初めに一般的な XML データを対象として実験を行う．ラベリングの対象データとして XML 文書における代表的なベンチマークである XMark [12] の XML 文書生成プログラムである xmlgen (注2) を用いて生成した XML 文書を用いる．xmlgen では文書の規模を表す scale factor の値を設定することで，構造の定まった任意の規模の XML 文書を生成することができるため，実験では scale factor の値を変化させて，規模の異なる XML 文書を生成した．生成する XML 文書の規模を scalefactor の値を 0.001 , 0.01 , 0.1 の三つに分けて生成する．

今回の実験では要素ノードの挿入を繰り返した後のラベルサイズの比較は行っておらず，XML 文書に対し Algorithm 3 を用いてラベルの付与を行っている．実験結果を図 5 に示す．図 5 の縦軸が, 圧縮 bit 列 DO-VLEI を基準とした場合のビット列反転型圧縮 bit 列 DO-VLEI のラベルサイズの総和の割合を表しており，このことから従来手法を圧縮率の点で上回っていることが分かる．

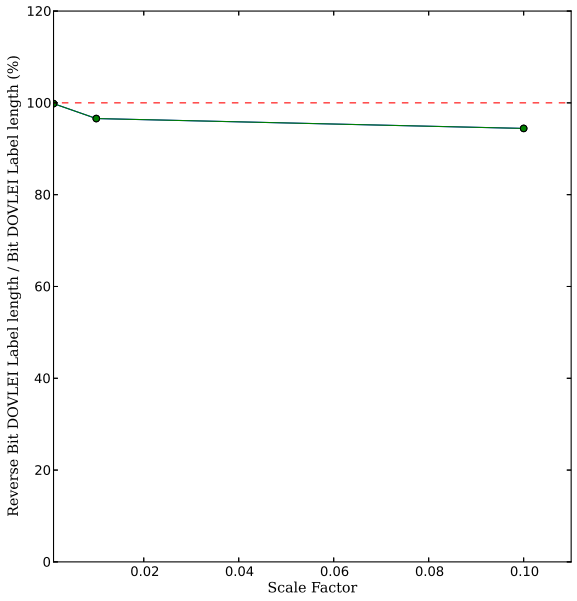

図 5　xmlgen による XML 文書についての ラベルサイズ比較

4.3　ノードの出現位置に偏りのある XML 文書

4.2 節では一般的な XML 文書として xmlgen によって生成された XML 文書を対象にラベルサイズ比較実験を行った．しかし，xmlgen の scale factor 増加による XML 文書の規模の増加は兄弟ノード， XML 木の深さがともに増大していき，要素ノードの出現が偏った XML 文書についてはラベルサイズについて評価ができていない．そのため，要素ノードの出現位置に偏りのある XML データを用いて，さらに実験を行った．

図 6 のような完全 4 分木を初期の XML 木として，兄弟ノードあるいは葉ノードに対して部分木を挿入していくことにより変化していくラベルサイズ変化を観察する．対象となる XML データは深さの増加に伴うノード数を考慮し，深さ 4 の 4 分木に対して要素ノードの挿入を繰り返し実験を行った．要素ノードの挿入方法については後述する既存の要素ノード間への挿入と既存の葉ノードの下への挿入の二つのパタンに分け，各パタンにおいてラベルサイズの変化を測定した．

(注2)：xmlgen: http://www.monetdb.cwi.nl/xml/downloads.html

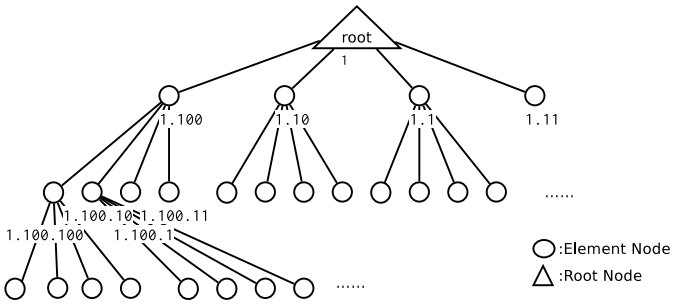

図 6　完全 4 分木

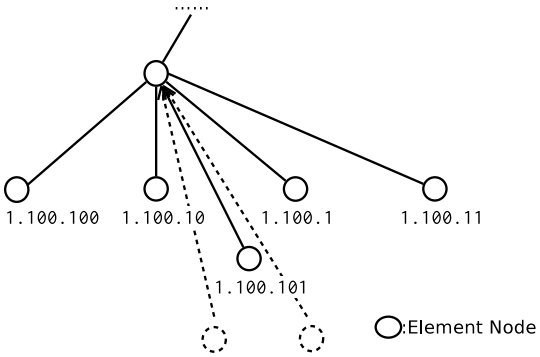

図 7　兄弟ノード間へのノード挿入

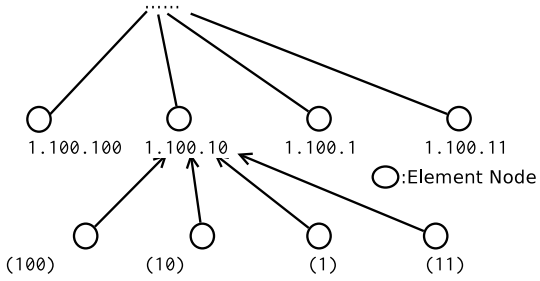

図 8　葉ノードの下へのノード挿入

4.3.1　ノードの挿入処理

提案手法は, 兄弟ノードの数が増加するほど従来手法と比較してラベルサイズが小さくなるように設計を行っている．そのため，ノードの挿入パタンとして兄弟ノードを増加させる挿入処理とそれとは対称的な葉ノードの挿入処理の二パタンにノードの挿入処理パタンを分類し，実験を行う．

一つ目の要素ノードの挿入処理は， 4 分木の深さを変化させることなく， VLEI コードの挿入アルゴリズムである Algorithm 1 を用いてラベリングを行った後に既存の要素ノード間に挿入していく．しかし，ノード挿入アルゴリズムの性質上，既存の要素ノードの両端に要素ノードの挿入を行うと，隣の要素ノードのラベルに数字が付加されてラベリングが行われるため，XML 木全体のラベルサイズが増加することは自明である．ゆえに本実験では要素ノードの挿入箇所を既存の要素ノードの中央に固定する．ランダムに要素ノードを様々な箇所の兄弟ノード間に挿入していくだけでは平等に要素ノードが挿入されるため，一度に四つの要素ノードを既存の要素ノードの中央に挿入する．

例えば，図 7 のような 4 分木の場合，中央の位置である 2 番目と 3 番目の間に要素ノードを挿入し，その後は初期の状態における 2 番目と 3 番目の間のいずれかのランダムな箇所への要素ノードの挿入を 3 回繰り返すことで要素ノードの挿入を行う．

二つ目のパタンは 4 分木の兄弟ノード数を変化させることなく，既存の XML 木の末端に要素ノードの挿入を行っていく．既存の XML 木の葉ノードを親ノードとして，要素ノードを挿入していくために図 8 のようにラベリングを行った四つの要素ノードによって構成される部分木を挿入する処理を繰り返すことで， 4 分木を維持して挿入を行う．挿入される要素ノードのラベルは事前に部分木のなかでラベル付けされた括弧の中の数字の前にその要素ノードの親ノードのラベルを結合すればよい．

4.3.2　ラベルサイズ比較実験

4.2 節の実験と同様に要素ノードのラベルの長さの合計を従来手法と提案手法で比較を行ったところ，図 9 のようなグラフが得られた．図 9 の縦軸は図 5 と同様に従来手法を基準とした場合の提案手法のラベルサイズの総和の割合であるため，兄弟ノードの数が増えていく場合に限り，提案手法が従来手法より圧縮率の点で優位であることがわかった．その一方で, 葉ノードに要素ノードが挿入されていく場合については，ラベルサイズに大きな変化はみられなかった．

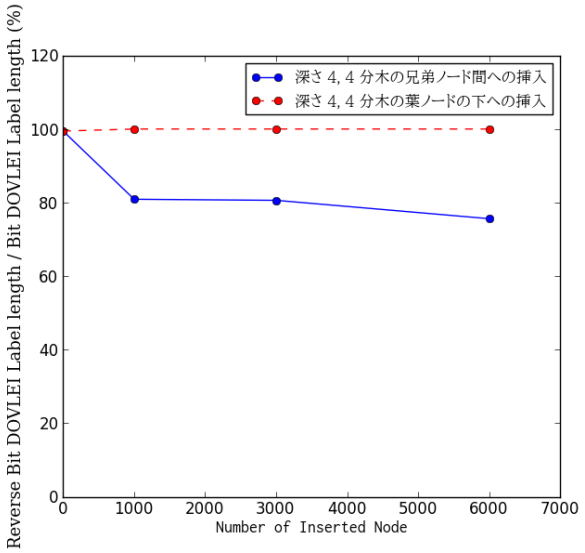

図 9　深さ 4 の 4 分木に対する二パタンのノード挿入

5.　おわりに

本稿では XML 文書のラベリング手法である DO-VLEI によって付与されたラベルに対してのラベル圧縮手法として， ビット列反転型圧縮 bit 列 DO-VLEI の提案を行った．

従来手法である圧縮 bit 列 DO-VLEI は VLEI コードの “1” について “10” を割り当てているため，兄弟関係にある要素

ノードの数が多くなるにつれて，ラベルサイズが大きくなる傾向があったが，提案手法では従来手法とは異なり， ビット列の反転を用いて VLEI コードの符号化を行っていくことで，デリミタに同じ出現パターンのビット列を割り当て， VLEI コードの “1” について， 圧縮 bit 列 DO-VLEI が “11” の 2 ビットで表現していたのに対し，提案手法のビット列反転型圧縮 bit 列 DO-VLEI では “1” の 1 ビットで表現することでラベルの圧縮率の向上を目指した．実験結果より，兄弟ノードが XML 木に対して占める割合が大きいほど，従来手法に比べ提案手法の方がラベルサイズの削減に貢献できるということが示せた．

今後の課題として，圧縮 bit 列 DO-VLEI はビット演算による高速な構造情報抽出手法が提案されているため [11]， 提案したビット列反転型圧縮 bit 列 DO-VLEI についても対応する別の構造情報抽出手法を提案したうえで問合せ処理性能を評価する必要がある．

謝　　辞

本研究の一部は日本学術振興会科学研究費補助金基盤研究 (A)(課題番号: 22240005) および若手研究 (B)(課題番号: 22700248) によるものである．ここに記して謝意を表す．

文　　献

[1] 清水敏之, 鬼塚真, 江田毅晴, 吉川正俊. XML データの管理とストリーム処理に関する技術. 電子情報通信学会論文誌. Vol. D90, No. 2, pp. 159–184, 2007.

[2] 小林一仁, 小林大, 横田治夫. 挿入制限のない XML 範囲ラベリング用コード. 情報処理学会研究報告. Vol. 2003, No. 71, pp. 41–48, 2003.

[3] P. O'Neil, E. O'Neil, S. Pal, I. Cseri, G. Schaller, and N. Westbury. ORDPATHs: insert-friendly XML node labels. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, SIGMOD '04, pp. 903–908, New York, NY, USA, 2004. ACM.

[4] 村上翔一, 小林大, 横田治夫. DO-VLEI を用いた XML 格納におけるラベルサイズと問い合わせ性能. 第 17 回 電子情報通信学会データ工学ワークショップ (DEWS2006) 論文集, 7B-o2, March 2006.

[5] I. Tatarinov, S. D. Viglas, K. Beyer, J. Shanmugasundaram, E. Shekita, C. Zhang, Storing and querying ordered XML sing a relational database system. *Proceedings of the 2002 ACM SIGMOD international conference on Management of data, SIGMOD 02*, pp. 204–215, 2002.

[6] E. Cohen, H. Kaplan, and T. Milo. Labeling dynamic XML trees. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '02, pp. 271–281, New York, NY, USA, 2002. ACM.

[7] 江田毅晴, 櫻井保志, 天笠俊之, 吉川正俊, 植村俊亮. XML 木のための更新に強い節点ラベル付け手法. *In DBSJ Letters.* No.1 in Vol.1, 2002.

[8] C.Li, T. W. Ling, QED : A novel quaternary encoding to completely avoid re-labeling in XML updates. In *Proceedings of the 14th ACM international conference on Information and knowledge management* pp. 501 – 508, New York, NY, USA, 2005. ACM.

[9] C. Li, T. W. Ling, M. Hu, Efficient Processing of Updates in Dynamic XML Data. In *Proceedings of the 22nd International Conference on Data Engineering*, pp. 03 – 07, April 2006.

[10] T. Amagasa, M. Yoshikawa, and S. Uemura. QRS: a robust numbering scheme for XML documents. In *Proceedings of the 19th International Conference on Data Engineering*, pp. 705 – 707, March 2003.

[11] 高橋昭裕, 梁文新, 横田治夫. 要素挿入に強い XML ラベルの構造情報抽出手法の提案と評価. 第 19 回 電子情報通信学会データ工学ワークショップ (DEWS2008) 論文集, C8-4, March 2006.

[12] A. R. Schmidt, F. Waas, M. L. Kersten, D. Florescu, I. Manolescu, M. J. Carey, and R. Busse. The XML benchmark project. Technical report, Amsterdam, The Netherlands, 2001.