論文 / 著書情報
Article / Book Information

論題(和文)	更新処理を考慮したXMLラベルにおける構造情報の抽出手法
Title(English)	Structural Information Extraction Method for Update-conscious XML Labeling
著者(和文)	高橋 昭裕, 梁 文新, 横田 治夫
Authors(English)	Akihiro Takahashi, Wenxin Liang, Haruo Yokota
出典(和文)	電子情報通信学会論文誌, Vol. J93-D, No. 6, pp. 795-804
Citation(English)	IEICE Transaction on Information & Systems, Vol. J93-D, No. 6, pp. 795-804
発行日 / Pub. date	2010, 6
URL	http://search.ieice.org/
権利情報 / Copyright	本著作物の著作権は電子情報通信学会に帰属します。 Copyright (c) 2010 Institute of Electronics, Information and Communication Engineers.

更新処理を考慮したXMLラベルにおける構造情報の抽出手法

高橋　昭裕[†a)]　　梁　　文新[††b)]　　横田　治夫[†c)]

Structural Information Extraction Method for Update-conscious XML Labeling

Akihiro TAKAHASHI[†a)], Wenxin LIANG[††b)], and Haruo YOKOTA[†c)]

あらまし　近年，XML データベースの格納や検索処理のために XML 木の各ノードに構造ラベルを付ける研究が注目されている．XML 文書の更新が起こる場合，単純なラベリング手法では更新コストが高くなる．そこで，文書更新時のコスト低減を目的とした DO-VLEI ラベリング手法と，DO-VLEI ラベルの格納容量削減を目的とした圧縮 bit 列 DO-VLEI が提案されている．本論文では圧縮 bit 列 DO-VLEI コードを用いた XML 問合せを実現するために，ラベルから構造問合せ処理に必要となる情報を抽出する手法を提案する．圧縮 bit 列 DO-VLEI コードでは先頭からの走査をせずに構造情報を取り出すことが可能であるため，高速な問合せ処理が可能となる．更に，同様のラベリング手法である ORDPATH と構造情報抽出処理の実行時間を比較し，その優位性を示す．

キーワード　XML，性能評価，ラベリング，構造情報

1. まえがき

大量の XML 文書を効率良く管理するために，RDB に格納する方法が注目されている [1]．XML 文書は，一組のタグをノードとして扱うことで木構造とみなすことができる [2]．図 1 に XML 文書を木構造とみなした XML 木を示す．XML 文書を RDB に格納する際，XML 文書の木構造情報を保持するための方法として XML 文書の各ノードにラベルを割り当て，ラベルとノード情報をタプルとして格納する方法がある．ラベルの規則性により各ノード間の包含関係など，ノードの XML 木における位置判定を可能にする．このようなノードにラベルを付けることによって構造情報を保持する手法をラベル付け手法と呼ぶ．

ラベル付け手法として，前順後順法 [3] や Dewey Order [4] などが提案されている．しかし，これらのラベル付けに単純な連続する整数を用いると，更新時に大規模なラベルの付け替えが必要となる．そこで，更新時のラベル付け替えのコストを抑える手法として，浮動小数点数法 [5] や範囲ラベリング法 [6]，範囲ラベリング法を改良した更新に強い節点ラベル付け手法 [2]，などが提案されている．しかし，これらの手法においても同じ箇所に多数のノードが挿入された場合にはラベルの付け替えが必要である．

このようなラベル付け替えが発生する問題に対して，我々は他ノードのラベルを変更することなく無制限にノードを挿入することができる VLEI コード (Variable Length Endless Insertable code) と，VLEI コードを用いたラベル付け手法を提案し，更新・検索処理に関する改良を行ってきた [7]〜[10]. [7] では VLEI コードを前順後順法に適用した PP-VLEI と，Dewey Order に適用した DO-VLEI を提案した. [8] では，包含関係の判定において，子孫ノードのラベルは必ず先祖ノードのラベルよりも長いという特徴を利用することで，検索処理の性能改善を行った. [9], [10] では，ラベル格納に必要な記憶容量低減を企図し，DO-VLEI コードを bit 列で表現する，圧縮 bit 列 DO-VLEI コード (Compressed-bit-string DO-VLEI code) を提案した.

XML の構造情報は，代表的なクエリ記述言語の XPath [11] や索引処理などに有用である．本論文では，圧縮 bit 列 DO-VLEI を用いて XPath の軸を用

† 東京工業大学，東京都
Tokyo Institute of Technology, 2–12–1 Ookayama, Meguro-ku, Tokyo, 152–8552 Japan
†† 大連理工大学，大連市
Dalian University of Technology, Development Zone, Dalian 116620, China
a) E-mail: akihiro@de.cs.titech.ac.jp
b) E-mail: wxliang@dlut.edu.cn
c) E-mail: yokota@cs.titech.ac.jp

いた構造に対するクエリを処理するために，XPath の各軸が表すノード集合を得るための手法を考案し，その際に必要となる各ラベルに埋め込まれている情報を高速に抽出する手法を提案する．具体的にはそれぞれのノードに割り当てられた圧縮 bit 列 DO-VLEI コードから木構造におけるノードの位置を知ることができ，また，二つのノードの位置関係を知ることが可能となる．例えば二つの圧縮 bit 列 DO-VLEI コードを比較することにより，二つのノードが親子関係にあるかを調べることが可能である．その際にワード長分の bit を同時処理することで高速に処理を行う．また，同様の情報を取得でき，DO-VLEI と同様に要素挿入に強いラベル付け手法である ORDPATH と処理時間について比較を行う．

本論文の構成は次のとおりである．まず，**2.** で DO-VLEI 及び，圧縮 bit 列 DO-VLEI について説明し，**3.** では関連研究として比較実験の対象となる ORDPATH について説明する．**4.** で圧縮 bit 列 DO-VLEI コードによる XPath 処理手法について述べ，**5.** では圧縮 bit 列 DO-VLEI コードによるラベルの構造情報抽出手法について説明する．**6.** で圧縮 bit 列 DO-VLEI と ORDPATH のラベル構造情報抽出性能について比較を行う．最後に **7.** でまとめと今後の課題を述べる．

2. 予 備 知 識

我々がこれまで提案してきた VLEI コードと，VLEI コードに Dewey Order を組み合わせた DO-VLEI，DO-VLEI コードを bit 列に変換する圧縮 bit 列 DO-VLEI，更に圧縮 bit 列 DO-VLEI の性質について説明する．

2.1 Dewey Order

Dewey Order では，親ノードのラベルの末尾に，デリミタ（区切り文字）と兄弟ノード間の順序を表現するコード（兄弟コード）とを加えて，子ノードのラベルを表現する．Dewey Order でラベル付けされた XML 木を図 1 に示す．Dewey Order の兄弟コードを単純に整数とすると，兄弟間にノードを挿入する際などにラベルの付け替えが必要となり，コストがかかるという問題がある．

2.2 VLEI コードの定義

VLEI コードは 1 から始まる 0,1 の可変長 bit 列であり，特殊な大小関係をもつ．以下に VLEI コードの定義を示す．

定義 1．VLEI コード

v を 1 から始まる {0,1} からなる bit 列とし，以下の大小関係を満足する場合に VLEI コードと呼ぶ．

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

すなわち，あるラベル v に対して，右に 0 の付くものは v よりも小さく，1 の付くものは v よりも大きい．ここで $\prec_v$ は VLEI コードによる比較を，$\cdot$ は文字列，数字の連結を意味する．

VLEI コードでは，任意の二つの VLEI コードの間に存在する新しい VLEI コードを無制限に作り出すことができるため，他ノードのラベルを付け替える必要がない．よって，更新コストの低いラベル付けが可能となる．

2.3 DO-VLEI

DO-VLEI では Dewey Order の兄弟コードに VLEI コードを用いる．

定義 2．DO-VLEI

（1） root ノードの DO-VLEI ラベルを $D_{root} = 1$ とする．

（2） あるノードの兄弟ノード間の順序を VLEI コード v_{child} の大小関係で表す．このとき，親ノードの DO-VLEI 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 DO-VLEI ラベル D_{child} は次のようになる．

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

図 2 に DO-VLEI によってラベル付けされた XML

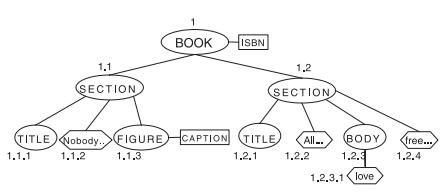

図 1 XML 木
Fig. 1 An example XML tree.

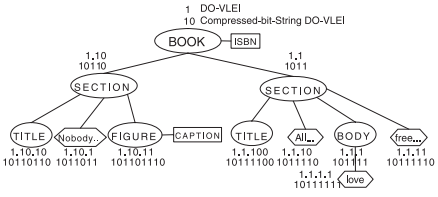

図 2 DO-VLEI コードによるラベル付け
Fig. 2 Labeling by DO-VLEI code.

表 1　XPath の軸
Table 1　XPath axes.

軸	軸の説明
ancestor	祖先ノード（親ノードを再帰的に取得することで得られる集合）
preceding	文書順でコンテクストノードより前 ∩ 祖先ノード以外
decendant	子孫ノード（子ノードを再帰的に取得することで得られる集合）
following	文書順でコンテクストノード以降 ∩ 子孫ノード以外
parent	親ノードが存在する場合，親ノード
preceding-sibling	コンテクストノードの前にあるすべての兄弟ノード
child	子ノード
following-sibling	コンテクストノードの後ろにあるすべての兄弟ノード

木を示す．

2.4　圧縮 bit 列 DO-VLEI

圧縮 bit 列 DO-VLEI は DO-VLEI コードをビット列表現に変換することにより，実際に扱うラベルのサイズを小さくし，また，bit 列による操作を可能にする．なお，以降「」で囲まれている 0，1 及び．の列は DO-VLEI コード，() で囲まれている 0，1 の列は bit 列であるとする．

2.4.1　DO-VLEI コードの構成要素

DO-VLEI コードは「.」と「1」と「0」の三つの記号から構成される文字列である．そして，三つの記号から構成される文字列が DO-VLEI コードであるためにはいくつかの条件が存在する．

（1）「.」は連続しない．
（2）「.」がラベルの末尾にあることはない．
（3）VLEI コードは「1」で始まる．

条件 (1)〜(3) から「.」の後には必ず VLEI コードの先頭の文字「1」が存在する．ゆえに，DO-VLEI コードは「.1」と「1」と「0」の三つの記号から構成されると言い換えることができる．

これら三つの記号にビット列を割り当てることにより，圧縮 bit 列 DO-VLEI コードを生成する．

定義 3．圧縮 bit 列 DO-VLEI
DO-VLEI コードにおいて,「.1」を bit 列 (10) に,「1」を bit 列 (11) に,「0」を bit 列 (0) に変換したものを，圧縮 bit 列 DO-VLEI コードと呼ぶ．

三つの記号「.1」「1」「0」それぞれに，語頭符号 [12] となる bit 列 (10)，(11)，(0) を割り当て，符号化する．図 2 の XML 木に圧縮 bit 列 DO-VLEI を示す．

2.5　XML のノード情報取得と軸

先述したとおり，XML 文書の構造に対するクエリ記述手法として XPath の軸 (axis) を利用しながら問合せを記述する方法がある．軸とは XPath において，選択するノードとコンテクストノードとの間の木構造関係を指定するための用語である．例えばあるノード A をコンテクストノードとしたとき，軸 ancestor によって A の祖先ノードすべてを選択することが可能である．主要な軸の説明を表 1 に示す．XML 文書の各ノードは軸 ancestor, preceding, decendant, following のいずれかに属する．また，軸 parent, preceding-sibling, child, following-sibling によって選択されるノード集合は，それぞれ ancestor, preceding, descendant, following によって選択されるノード集合のサブセットである．これに対してラベル付け手法による XML 文書格納では各ノードに付加されたラベルが構造の情報を保持しており，この情報を利用することによって XPath の軸によるクエリに対する応答を行う．各ノードが圧縮 bit 列 DO-VLEI を用いてラベル付けされている場合，すべてのコードを DO-VLEI コードにすることによってクエリの処理を行うことが可能であるが，すべての圧縮 bit 列 DO-VLEI コードを DO-VLEI コードに読み替えるのは効率的ではない．そこで，DO-VLEI コードを意識せずに利用可能である圧縮 bit 列 DO-VLEI コードの性質の説明をする．

2.5.1　圧縮 bit 列 DO-VLEI コードの Dewey Order に起因する性質

圧縮 bit 列 DO-VLEI における DO-VLEI コードから bit 列への変換では，DO-VLEI の構成要素を最小単位ごとに bit 列に変換を行っている．そのため，圧縮 bit 列 DO-VLEI コードは子ノードラベルが頭に親ノードラベルを含むという，Dewey Order の性質を保持している．具体的には，あるノードのラベルがそのノードの親ノードのラベルの右にデリミタとなる「.1」を表す記号と兄弟番号を示す VLEI コードを加えたものになっている．そのため，各ラベルから次の三つの情報を取得可能である．

DO-1　ノードの深さ
DO-2　親ノードのラベル

表 2　ビット操作の表記
Table 2　Bit operation notation.

表記	説明
$x \ll y$	左シフト
$x \overset{u}{\gg} y$	右シフト（論理シフト）
$\overline{x}$	ビット単位の否定
$x\&y$	ビット単位の積
$x\|y$	ビット単位の和

Algorithm 1
圧縮 bit 列 DO-VLEI コードの大小比較

入力: 圧縮 bit 列 DO-VLEI コード v,w
(length(v)≤length(w))
出力: v が w より大きいかどうか
1: $l' \leftarrow length(w) - length(v)$
2: $v' \leftarrow v \cdot 1 \cdot \{0\}^{l'-1}$
3: **if** $v' \geq w$ **then**
4:　　**return true**
5: **else**
6:　　**return false**
7: **end if**

DO-3　祖先ノードのラベル

Dewey Order ラベルの構成方法から，親ノードのラベルはラベルから最後に出現するデリミタ以降を削除することで得られる．祖先ノードは親ノードを再帰的に得ることによって漏れなく得ることが可能である．更に，深さは祖先ノードの数がそれに当たる．

2. 5. 2　圧縮 bit 列 DO-VLEI コードの数値比較による性質

圧縮 bit 列 DO-VLEI コードでは DO-VLEI コードの構成要素である「.1」「0」「1」にそれぞれ bit 列を割り当てることによって DO-VLEI コードを bit 列で表現しているので，これを「0」$= v_0$，「.1」$= v_d$，「1」$= v_1$ と表すとする．このとき，

$$1 \cdot v_0 \prec_v 1 \prec_v 1 \cdot v_d \prec_v 1 \cdot v_1 \qquad (1)$$

が成り立つならば Algorithm 1 を用いて圧縮 bit 列 DO-VLEI コードを大小比較をすることにより，昇順が XML 文書における文書順と等しくなる．これはすなわち，ある圧縮 bit 列 DO-VLEI コードがあったときに，それより大きい圧縮 bit 列 DO-VLEI コードのノードは descendant 若しくは following のどちらかに属するノードであり，同様に小さいとされたノードは ancesotor 若しくは preceding に属するノードである．証明は [13] を参照されたい．

なお，アルゴリズム中で出現するビット操作は表 2 のようになっている．以降，$v_0 = 0, v_d = 10, v_1 = 11$ とする．

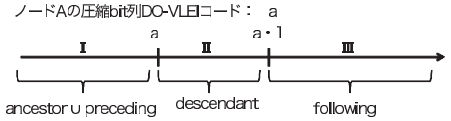

図 3　圧縮 bit 列 DO-VLEI コードの数直線
ancestor, preceding, decendant, following
Fig. 3　Code range for ancestor, preceding, decendant, following.

また，ある圧縮 bit 列 DO-VLEI コード a の child ノードの圧縮 bit 列 DO-VLEI コードは $a \cdot 10 \cdot \{11|0\}^*$ である．よって，圧縮 bit 列 DO-VLEI コード a の decendant ノードの圧縮 bit 列 DO-VLEI コード b は $a \cdot 10 \cdot \{0\}^* \preceq_v b \preceq_v a \cdot 10 \cdot \{1\}^*$ となる．更に最右辺に 1 を足すと，最後の 1 が何個続いているかにかかわらず $a \cdot 11 \cdot \{0\}^*$ となるので $a \cdot 10 \cdot \{0\}^* \preceq_v b \prec_v a \cdot 11 \cdot \{0\}^*$ となり，すなわち

$$a <_{CDV} b <_{CDV} a \cdot 1 \qquad (2)$$

である．ここで $<_{CDV}$ は Algorithm 1 を用いた大小比較の結果である．

以上の数値比較による性質をまとめると図 3 のようになり，XML 文書のラベルを，カレントノードのラベル a と $a \cdot 1$ を用いて，**Ⅰ** ancestor∪preceding，**Ⅱ** descendant，**Ⅲ** following，の三つの範囲に分けることができる．

3. 関 連 研 究

圧縮 bit 列 DO-VLEI と同様に，他ノードのラベルを付け替えることなく無制限にノードの挿入を行うことのできるラベル付け手法として，素数を用いた手法 [14] や Dewey Order を用いた手法 [15]〜[17] などが提案されている．本論文ではラベルの性能比較実験の比較対象としてよく用いられる ORDPATH を比較対象とする．

ORDPATH は内部では 0,1 の bit 列で表現される．これを圧縮 bit 列 ORDPATH(Compressed-bit-string ORDPATH) と呼ぶこととする．圧縮 bit 列 ORDPATH では表 3 の接頭辞符号化スキームを用いて，整数を bit 列に変換する．

圧縮 bit 列 ORDPATH では一つの整数を表すために，一組の bit 列コンポーネント L_i-O_i を用いる．i はデリミタで区切られた整数のラベル中における順番である．L_i は頭から逐次解析することで決定する bit

表 3　圧縮 bit 列 ORDPATH で用いる接頭辞符号化スキーム
Table 3　Prefix encoding scheme for compressed-bit-string ORDPATH.

L_i Bitstring	O_i length	O_i value range
000000001	20	[-1118485, -69910]
00000001	16	[-69909, -4374]
0000001	12	[-4373, -278]
000001	8	[-277, -22]
00001	4	[-21, -6]
0001	2	[-5, -2]
001	1	[-1, 0]
01	0	[1, 1]
10	1	[2, 3]
110	2	[4, 7]
1110	4	[8, 23]
11110	8	[24, 279]
111110	12	[280, 4375]
1111110	16	[4376, 69911]
11111110	20	[699912, 1118487]

列であり，続く O_i の bit 数と整数の範囲を接頭辞符号化スキームにより決定する．そして O_i は L_i によって指定される範囲の数を 2 進数によって表すことでラベルサイズの圧縮を行っている．

圧縮 bit 列 ORDPATH は圧縮 bit 列 DO-VLEI と同様に Dewey Order による性質と数値比較による性質を利用することができるものの，コード化の方式が異なることにより，**DO-1**〜**DO-3** の判定アルゴリズムが異なる．詳しくは [15] を参照されたい．

4. 圧縮 bit 列 DO-VLEI による XPath 処理

まず，XML 文書を四つに分割する ancestor, preceding, decendant, following のノード集合を取得する処理について説明し，そのあとにそれぞれのサブセットとなるノード集合を取得する parent, preceding-sibling, child, following-sibling について説明する．

図 3 が示すとおり数値比較により XML 文書を **I**〜**III** の三つの範囲に分けることが可能である．**II** の範囲にあるノードすべてを求めることで descendant が表すノード集合を得ることが可能であり，**III** の範囲にあるノードすべてを求めることで following が表すノード集合を得る．一方，**I** の範囲にあるノード集合は ancestor∪preceding であるためこの二つを分離する必要がある．ここで，**DO-3** の性質から ancestor ノードのラベルはカレントノードのラベルから生成が可能である．よって，preceding が表すノード集合を求める場合は **I** の範囲にあるノード集合のうち ancestor が表すノード集合を除外すればよい．

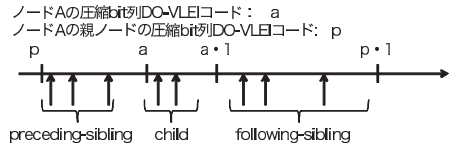

図 4　圧縮 bit 列 DO-VLEI コードの数直線 parent, preceding-sibling, child, following-sibling
Fig. 4　Code range for parent, preceding-sibling, child, following-sibling.

表 4　圧縮 bit 列 DO-VLEI コードによる各軸に対応するノード集合の取得方法
Table 4　XPath axis extraction methods.

軸	**圧縮 bit 列 DO-VLEI コードによるノードラベル取得方法**
ancestor	**DO-3** を用いる
preceding	（範囲 **III** の集合） - ancestor
decendant	範囲 **II** の集合
following	範囲 **III** の集合
parent	**DO-2** を用いる
preceding-sibling	ノードの範囲 **II**∩ 範囲 **I**∩ 深さがカレントノードと同じ
child	範囲 **II**∩ 深さがカレントノードの深さ +1 のノード
following-shibling	親ノードの範囲 **II**∩ 範囲 **III**∩ 深さがカレントノードと同じ

次に parent, preceding-sibling, child, following-shibling が表すノード集合を取得する方法について説明する．これらがが表すノード集合はそれぞれ ancestor, preceding, decendant, following が表すノード集合のサブセットである．このうち parent が表すノード集合は **DO-2** で得ることができることを既に説明している．カレントノードとその parent が表すノード集合のラベルを用いて他の三つの軸が表すノード集合を圧縮 bit 列 DO-VLEI コードの数直線上に表すと図 4 のようになる．更に，following-sibling, preceding-sibling が表すノード集合はカレントノードと深さが同じであり，child が表すノード集合はカレントノードより深さが 1 大きい．これらをまとめると表 4 の最右欄のようになる．

5. 構造情報抽出手法

2. 5. 1 で述べた **DO-1**〜**DO-3** の情報を圧縮 bit 列 DO-VLEI コードから抽出する手法を説明する．Dewey Order では各ノードのラベル構造は（親ノードラベル）·（デリミタ）·（兄弟番号）のようになっており，圧縮 bit 列 DO-VLEI コードで **DO-1**〜**DO-3** の情報を抽出する際に，デリミタの位置や数を知ること

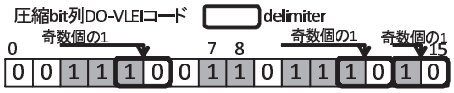

図 5 圧縮 bit 列 DO-VLEI コードの例
Fig. 5 An example fragment of a compressed-bit-string DO-VLEI code.

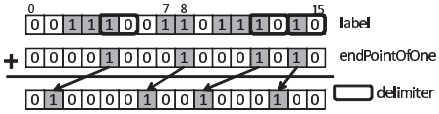

図 6 圧縮 bit 列 DO-VLEI コードと endPointOfOne の加算
Fig. 6 An example of delimiter detection.

が重要になる．ここで，圧縮 bit 列 DO-VLEI コードは語頭符号により bit 列化されている．よって，コードを頭から逐次読み込むことにより復号可能であり，デリミタとなる記号「.1」に割り当てられているビット列 (10) が出現する場所を検出することができる．しかし，この方法ではラベルの長さが長くなったときに非常に効率が悪い．そこで，レジスタ長の bit 列を一斉に処理し，デリミタの位置を調べる方法を示す．

まずはじめに，bit 列の中でどのようなところがデリミタになっているのかを考察する．デリミタに対応するビット列は (10) であるため，左から読んだときに (1) から (0) に変わる場所がデリミタの候補となる．しかし，DO-VLEI コードで「10」とならんでいるときのビット列も (110) となり，(10) が出現する．この二つを見分けるために (1) がいくつ並んだあとに (0) が出てくるかに着目する．具体的には図 5 のように，(1) が奇数個並んだあとに (0) が出てきた場合は最後の bit 列 (10) はデリミタを表し，偶数個並んだあとはデリミタではなく「10」である．よって，ビット列の中で 1 が奇数個並んでいる箇所を数えることによって深さを調べることができる．

本章では以降簡単のため，圧縮 bit 列 DO-VLEI コードは 2 Byte 以下の bit 列とし，2 Byte の固定長 bit 列として格納されているとする．なお，今回実験で用いる実装では，レジスタ長ごとに bit 列を分けて計算を行っている．

5.1 深さ判定

まず圧縮 bit 列 DO-VLEI コードを用いた深さ判定の方法を説明する．次の二つのステップで調べる．

D-1 奇数個 1 が並んでる箇所の数分ビットが ON になっているビット列を生成

D-2 ビット列の中で ON になってるビット数を数える

以下で **D-1**，**D-2** の説明をする．

D-1 奇数個ならんでいる箇所の数だけビット数が ON になっているビット列をつくるアルゴリズムを Algorithm 2 に示す．このアルゴリズムでは図 6 のように，ビット列中に存在する 1 の列の最後の 1 のところに 1 を足すと繰り上がりによって影響が伝搬され，1 の列の一つ左の 0 が 1 になることを利用している．なお，endPointOfOne は $x\&\overline{x \ll 1}$ とすることで得ることができる．

Algorithm 2
圧縮 bit 列 DO-VLEI によるデリミタ判定

入力: 圧縮 bit 列 DO-VLEI コード v
出力: v の深さ

```
1: endPointOfOne ← v&~(v ≪ 1)
2: delimiterEven ←~v&(v+(endPointOfOne&0x5555))&0xAAAA
3: v' ← v ≫ 1
4: endPointOfOne' ← endPointOfOne ≫ 1
5: delimiterOdd ←~v'&(v' + (endPointOfOne'&0x5555))&0xAAAA
6: delimiter ← delimiterEven|delimiterOdd
```

ここで，1 を足した場所からその足した影響で 0 が 1 になった場所までの距離が奇数であれば 1 が奇数個並んでいることになる．更にこれを場合分けすると，ビット列を頭から $0, 1 \cdots$ と番号付けしたときに

（1） 奇数番で足したときは偶数番が 0 から 1 に変化

（2） 偶数番で足したときは奇数番で 0 から 1 に変化

の二つのときに 1 が奇数個並んでいるといえる．ビット列全体で奇数番目の箇所だけで足したときは，0 から 1 に変化した箇所のうち偶数番目の箇所を抽出する．同様に偶数番目のときも行うことですべての 1 の列を検査することができる．なお，奇数番目，偶数番目の 1 のみを抽出するためにはそれぞれ即値で 0x5555, 0xAAAA と AND をとることで実現可能であり，0 から 1 に変化した箇所は $\overline{\text{before}}\&\text{after}$ とすることで得ることができる．

D-2 次にビット列の中で ON になっているビット数を数えるアルゴリズムである Algorithm 3 を説明する．この問題に対しては分割統治法を用いたアルゴリズムを利用する．まずビット列を 2 ビットごとに区切り，その中での ON になっているビット数を数え，もとの 2 ビットの中に格納する，次に 2 ビットの区

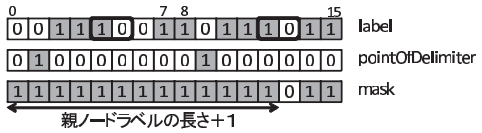

図 7　label と pointOfDelimiter の例
Fig. 7　An example label and pointOfDelimiter.

Algorithm 3
ON ビット数のカウント

入力: v (16 bit unsigned integer)
出力: v 中の ON ビットの数
1: $v \leftarrow v - ((v \overset{u}{\gg} 1)\&0x5555)$
2: $v \leftarrow (v\&0x3333) + ((v \overset{u}{\gg} 2)\&0x3333)$
3: $v \leftarrow ((v + (v \overset{u}{\gg} 4))\&0x0F0F)$
4: $v \leftarrow v + (v \overset{u}{\gg} 8)$
5: $v \leftarrow v + (v \overset{u}{\gg} 16)$
6: depth $\leftarrow v\&0x001F$
7: **return** depth

切り二つ分の数値を足し合わせて 4 ビットの中で ON になっている数を求める．これを繰り返すことによって全体で ON になっているビット数を数える．詳細は [18] を参照されたい．

5.2　親ノードラベルを生成する関数

親ノードラベルを得るためには最後のデリミタの場所を知る必要がある．ここで，深さを調べる際の最初のステップで Algorithm 2 を用いて生成したビット列（以降 pointOfDelimiter とする）を用いることを考える．最後のデリミタの影響によって pointOfDelimiter で ON になるビットは右から左へ見たときに最初に出現する ON ビットである．この ON ビットが i 番目のビットだったとするともとのラベルで $i+1$ 番目から始まる 1 の列の最後の 1 と続く 0 がデリミタである．ここが何番目かを知るために次のステップを用いる．

P-1　奇数個 1 が並んでる箇所の数分ビットが ON になっているビット列を生成

P-2　左から読んだときにデリミタの箇所で初めて 0 が出現するビット列を生成

P-3　頭から続く 1 の数を数える

最初のステップは深さを調べる際と同様なのでここでは省略する．

P-2 最初のステップでは図 7 の mask を作る．

$$\overline{(\text{pointOfDelimiter}\& - \text{pointOfDelimiter}) - 1}|label$$

とすることで得ることができる．$x\& - x$ とすることで x の中で最右の 1 のみを抽出可能であり，これから 1 を引くことで最右の 1 より右のビットのみが 1 となる．これを反転させることによって最右の 1 を含むそれより左のビットがすべて 1 となる．更にこのビット列ともとのラベルと OR を取ることにより，最右のデリミタの箇所までが 1 で，そこで初めて 0 が出現する bit 列を得ることができる．

Algorithm 4
先頭から続く 1 の長さ判定

入力: v (16 bit unsigned integer)
出力: 頭から続く 1 の数
1: $v \leftarrow \overline{v}$
2: $n \leftarrow 16$
3: $c \leftarrow 8$
4: **repeat**
5:　$x \leftarrow v \overset{u}{\gg} c$
6:　**if** $x \neq 0$ **then**
7:　$n \leftarrow n - c$
8:　$v \leftarrow x$
9:　**end if**
10:　$c \leftarrow c \overset{u}{\gg} 1$
11: **until** $c \neq 0$
12: **return** $n - x$

P-3 次のステップでは先頭から続く 1 の数を数えることにより，親ノードラベルの長さを得る．アルゴリズムの例を Algorithm 4 に示す．このアルゴリズムではレジスタ長に対する二分探索法を用いている．レジスタ長を固定とすることでループを排除し，より高速なアルゴリズムとすることが可能である．詳細は [18] を参照されたい．

5.3　祖先ノードラベルを生成する関数

祖先ノードラベルは親ノードラベルを再帰的に計算することにより得ることができる．Algorithm 2 によって得られた pointOfDelimiter を $x\&(x-1)$ とすることにより，最右の 1 が 0 になるので，これを利用し，前節の **P-2**，**P-3** を繰り返す．

6.　実装と実験

2.〜**4.** で述べたとおり，圧縮 bit 列 ORDPATH は圧縮 bit 列 DO-VLEI と同様に Dewey Order による性質と数値比較による性質を利用することができ，この二つの性質の組合せにより，XPath の軸が表すノード集合を得る処理を網羅できる．圧縮 bit 列 ORDPATH と圧縮 bit 列 DO-VLEI の XPath 処理において異なる点は，Dewey Order に起因する性質である **DO-1**〜**DO-3** の判定アルゴリズムである．深さ及び親ノードラベル判定においては圧縮 bit 列 DO-VLEI がラベルの深さによらず一定の実行速度となる一方，圧縮

表 6　XML 文書一覧
Table 6　XML documents.

	size (byte)	elements	max-depth	avg-depth	max-fanout	avg-fanout
SwissProt.xml	114820211	2977031	5	3.556711	50000	2.034511
dblp.xml	133862772	3332130	6	2.902279	328858	1.943555
ebay.xml	35525	156	5	3.75641	380	5.391305
item0.xml (xmlgen SF=1)	118552713	1666315	12	5.547796	25500	2.10692
item1.xml (xmlgen SF=0.1)	11875066	167865	12	5.548244	2550	2.103751
item2.xml (xmlgen SF=0.01)	1182547	17132	12	5.506012	255	2.102092
item3.xml (xmlgen SF=0.001)	118274	1729	12	5.717756	25	2.04965
lineitem.xml	32295475	1022976	3	2.941175	60175	1.941175
nasa.xml	25050288	476646	8	5.583141	2435	2.000728
orders.xml	5378845	150001	3	2.899987	15000	1.899987
part.xml	618181	20001	3	2.899905	2000	1.899905
partsupp.xml	2241868	48001	3	2.833295	8000	1.833295
psd7003.xml	716853012	21305818	7	5.15147	262526	1.818513
reed.xml	283619	10546	4	3.199791	703	1.809579
treebank_e.xml	86082517	2437666	36	7.872788	56384	1.571223
uwm.xml	2337522	66729	5	3.952435	2112	1.952276
wsu.xml	1647864	74557	4	3.157866	3924	2.077534

bit 列 ORDPATH はラベルの深さによって実行速度が遅くなる性質がある．そこで，圧縮 bit 列 DO-VLEI コードと圧縮 bit 列 ORDPATH コードのラベルの深さ，親ノードラベル，祖先ノードラベル判定の実行速度比較し，圧縮 bit 列 DO-VLEI の優位性を示す．

6.1 実　　装

圧縮 bit 列 DO-VLEI を用いて RDB に XML 文書を格納し，各ラベルから **DO-1**〜**DO-3** の情報を抽出できるよう，前節で説明したアルゴリズムをもとに実装を行った．RDB には PostgreSQL を利用し，ラベルは可変長 bit 列 (bit varying) で格納した．各情報抽出手法は PostgreSQL のユーザ定義関数として C 言語による実装を行った．また，比較実験を行うために ORDPATH についても圧縮 bit 列 DO-VLEI 同様に各ラベルから深さ，親ノードラベル，祖先ノードラベルを得られるようにした．

6.2 実　　験

5. で説明した関数の実行時間を測定し ORDPATH と DO-VLEI の比較を行った．実験では，作成した XML 文書のラベルすべてに対し，深さ，親ノードラベル，祖先ノードラベルを抽出する関数を実行し，実行時間を計測した．実験環境は表 5 のとおりである．XMark [19] 用の XML 文書作成ツールである xmlgen [20] によって作成した XML 文書と，XMLData Repository [21] の XML 文書を利用した．実際に実験に使用した XML 文書の情報を表 6 に示す．これらの XML 文書を DO-VLEI 及び ORDPATH でラベル付けを行い，ラベルのみを RDB に格納した．また，各

表 5　実験システムの構成
Table 5　Experimental environment.

CPU:	Dual-Core Intel Xeon 5110 (1.60 GHz)
Memory:	DDR2 ECC FB-DIMM 9 GB (2 GB×4, 512 MB×2)
Storage:	D-RAID RAID 0 + 1
HDD:	SEAGATE ST3750640AS (750 GB, 7200 rpm, 3.5 inch)
OS:	Linux 2.6.18 CentOS 5 (Finlal)
Compiler:	gcc (Red Hat 4.1.1-52)
RDB:	PostgreSQL 8.2.4

文書単位での実行速度と，ラベルの深さに起因する実行速度を明確にするために，文書ごとのテーブルと各ラベルの深さごとのテーブルを作成した．例えば深さ 1 のラベルを集めたテーブルには今回利用した XML 文書の深さ 1 のノードに付加されるラベルがすべて格納されている．

なお，今回の実験においてはテーブルごとに実行時間を計測した．例えば深さの実行時間を計測する際には，ある一つのテーブルすべてのラベルに対し深さを判定する時間を計測した．この場合，関数実行以外の時間が含まれてしまうため，何も行わない関数をすべてのラベルに対して実行した時間も計測し，深さ判定時間より引くことで深さ判定に必要な時間とした．

6.3 実 験 結 果

実験結果は図 8，図 9 のグラフのようになった．なお，それぞれのグラフでは圧縮 bit 列 ORDPATH の値によって正規化している．

図 8 はラベルの深さごとにすべてのラベルに対して

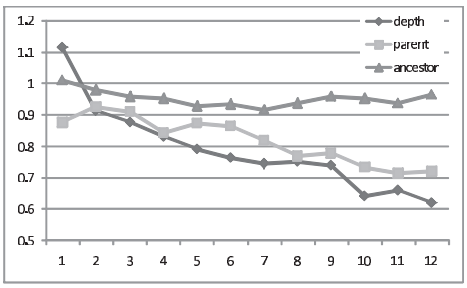

図 8　圧縮 bit 列 ORDPATH を 1 としたときの圧縮 bit 列 DO-VLEI のクエリ実行時間（横軸：ラベルの深さ）

Fig. 8　Execution time ratio of compressed-bit-string DO-VLEI to compressed-bit-string ORDPATH (horizontal axis: depth of node).

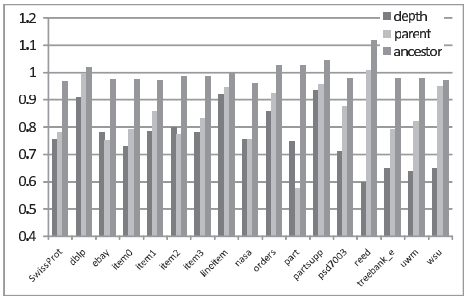

図 9　圧縮 bit 列 ORDPATH を 1 としたときの圧縮 bit 列 DO-VLEI のクエリ実行時間（横軸：ドキュメント）

Fig. 9　Execution time ratio of compressed-bit-string DO-VLEI to compressed-bit-string ORDPATH (horizontal axis: XML document).

処理を行ったときの実行時間である．このグラフでは，ラベルの深さが深くなると圧縮 bit 列 DO-VLEI の depth 及び parent の結果が良くなっている．これは，圧縮 bit 列 DO-VLEI の情報抽出アルゴリズムが深さによらず一定の処理時間で行えるのに対し，圧縮 bit 列 ORDPATH では接頭辞符号化スキームを用いているため，デリミタを知る際にコードの頭から逐次解析しなくてはならず，深さの分だけビット列コンポーネント $L_i - O_i$ を読み取って偶数なのか奇数なのかを判定しなくてはならないためである．一方 ancestor はすべての祖先ノードを深さの数だけ出力するため，圧縮 bit 列 ORDPATH と圧縮 bit 列 DO-VLEI の差があまり出ない．

図 9 は各文書ごとにすべてのラベルに対して処理を行ったときの実行時間である．圧縮 bit 列 DO-VLEI における depth と parent の平均処理時間はそれぞれ圧縮 bit 列 ORDPATH の処理時間の約 75%と 85%となった．一方 ancestor の結果は横軸を深さとしたとき同様圧縮 bit 列 ORDPATH と圧縮 bit 列 DO-VLEI の結果はほぼ同等である．今回は初期状態のラベルを用いて実験を行っているが，圧縮 bit 列 ORDPATH において要素挿入が起こると深さより多くのビット列コンポーネントを解析する必要があるため，更に性能が低下すると考えられる．

実験の結果，深さが深いノード程圧縮 bit 列 DO-VLEI の方が性能がよく，平均で depth 及び parent の実行時間がそれぞれ圧縮 bit 列 ORDPATH の 75%と 85%程度になることを確認できた．一方 ancestor に関しては圧縮 bit 列 DO-VLEI と圧縮 bit 列 ORDPATH であまり差が出なかった．

7. むすび

本論文では圧縮 bit 列 DO-VLEI を用いた XPath 処理方法を述べたあと，その際に必要な圧縮 bit 列 DO-VLEI から情報を抽出する手法を提案した．更に同様の処理が可能である ORDPATH と情報抽出処理の実行時間比較を行い，depth，parent の情報を抽出する操作において圧縮 bit 列 DO-VLEI を用いた場合の方が圧縮 bit 列 ORDPATH を用いた場合に比べて実行時間がそれぞれ 25%と 15%程度短くなることを示した．これは，圧縮 bit 列 ORDPATH においてはコードを頭から逐次解析する必要があるのに対し，圧縮 bit 列 DO-VLEI ではレジスタ長の bit 列を一斉に処理するためである．また，他の XPath に利用する操作においては操作性能に違いはないことから XPath の処理性能において圧縮 bit 列 DO-VLEI が圧縮 bit 列 ORDPATH に対し優位であることを示した．

今後，XPath の処理を実装し，ORDPATH 以外のラベル付け手法との実行時間比較を行う必要がある．また，今回は述べなかったが，偏った要素挿入が起こった場合，ラベルが長くなってしまう可能性があるので，それに対する対策を考案する必要がある．

謝辞　本研究の一部は，文部科学省科学研究費補助金特定領域研究 (19024028)，独立行政法人科学技術振興機構 CREST，及び東京工業大学 21 世紀 COE プログラム「大規模知識資源の体系化と活用基盤構築」の助成により行われた．

文　献

[1] I. Tatarinov, S.V.K.S. Beyer, J. Shanmugasundaram, E.J. Shekita, and C. Zhang, “Storing and querying

ordered XML using a relational database system," Proc. SIGMOD 2002, pp.204–215, 2002.
[2] 江田毅晴，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，" DBSJ Letters, vol.1, pp.35–38, 2002.
[3] P.F. Dietz, "Maintaining order in a linked list," Proc. STOC1982, pp.122–127, 1982.
[4] Online Computer Library Center. Introduction to the Dewey Decimal Classification. http://www.oclc.org/oclc/fp/about/about_the_ddc.htm
[5] T. Amagasa, M. Yoshikawa, and S. Uemura, "QRS: A robust numbering scheme for XML documents," ICDE, pp.705–707, 2002.
[6] E. Cohen, H. Kaplan, and T. Milo, "Labeling dynamic XML trees," Proc. 21st ACM SIGMOD-SIGACT-SIGART, pp.271–281, 2002.
[7] K. Kobayashi, W. Liang, D. Kobayashi, A. Watanabe, and H. Yokota, "VLEI code: An efficient labeling method for handling XML documents in an RDB," Proc. ICDE2005, 2005.
[8] 長良香子，小林一仁，小林　大，横田治夫，"XML データベースのラベル付け手法 VLEI の評価，" 第 16 回電子情報通信学会データ工学ワークショップ (DEWS2005) 論文集，2005.
[9] 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，2006.
[10] W. Liang, A. Takahashi, and H. Yokota, "A low-storage-consumption xml labeling method for efficient structure information extraction," Proc. DEXA 2009, LNCS 5690, pp.7–22. 2009.
[11] World Wide Web Consortium. XML path language. http://www.w3.org/TR/xpath, 1999.
[12] 韓　太舜，小林欣吾，金谷文夫，山本博資，横尾英俊，鈴木譲，森田啓義，佐藤　創，伊藤秀一，情報源符号化無ひずみデータ圧縮，情報理論とその応用シリーズ 1-1，培風館，1998.
[13] 高橋昭裕，梁　文新，横田治夫，"挿入によって他ラベル変更を起こさない可変長 XML ラベルの容量評価，" 情処学研報，DBS-143(17), pp.97–102, July 2007.
[14] X. Wu, M.L. Lee, and W. Hsu, "A prime number labeling schema for dynamic ordered XML trees," ICDE, pp.66–78, 2004.
[15] P. O'Neil, E. O'Neil, S. Pal, I. Cseri, G. Schaller, and N. Westbury, "ORDPATHs: Insert-friendly XML node labels," Proc. SIGMOD2004, pp.903–908, 2004.
[16] M.P. Haustein, T. Härder, C. Mathis, and M. Wagner, "Deweyids — The key to fine-grained management of XML documents," SBBD, pp.85–99, 2005.
[17] C. Li and T.W. Ling, "QED: A novel quaternary encoding to completely avoid re-labeling in XML updates," Proc. CIKM2005, pp.501–508, 2005.
[18] Jr. Guy L Steele, Hacker's Delight, Addison-Wesley Professional, 2003.
[19] A. Schmidt, F. Waas, M. Kersten, and D. Florescu, "The XML benchmark project," Technical Report INS-R0103, 2001.
[20] xmlgen. http://monetdb.cwi.nl/xml/downloads.html
[21] XML Data Repository. http://www.cs.washington.edu/research/xmldatasets/.

（平成 21 年 9 月 7 日受付，22 年 1 月 4 日再受付）

高橋　昭裕

平 18 東工大・工・情報工学卒．平 20 同大大学院・情報理工・計算工・修士課程了．現在（株）エヌ・ティ・ティ・データ．XML インデキシングに関する研究に従事．

梁　　文新

平 10 中国西安交通大・工・電子情報卒．平 13 同大大学院・電子情報理工学研究科・修士課程了．平 18 東工大・大学院情報理工学研究科・博士課程了．CREST 研究員．東工大・特別研究員．平 21 中国大連理工大・准教授，現在に至る．博士（工学）．主として XML データベース，Web に関する情報検索，マルチメディア等に関する研究に従事．日本データベース学会，IEEE，ACM SIGMOD 各会員．

横田　治夫　（正員：フェロー）

昭 55 東工大・工・電物卒．昭 57 同大大学院・情報・修士課程了．同年富士通（株）．同年 6 月（財）新世代コンピュータ技術開発機構研究所 (ICOT)．昭 61（株）富士通研究所．平 4 北陸先端大・情報・助教授．平 10 東工大・大学院情報理工・助教授．平 13 東工大・学術国際情報センター・教授，平 22 東工大大学院情報理工学研究科教授，現在に至る．工博．主として分散インデキシング，データ工学向けアーキテクチャ，高機能ストレージシステム，ディペンダブルシステム等に関する研究に従事．元本会データ工学研究専門委員会委員長．ACM SIGMOD 日本支部長．日本データベース学会理事．情報処理学会フェロー．人工知能学会，IEEE，ACM 各会員．