

# XML 木のための更新に強い節点ラベル付け手法

## A Robust Node-Labeling Scheme for XML trees

江田 毅晴<sup>♡</sup> 天笠 俊之<sup>◇</sup>  
吉川 正俊<sup>▲</sup> 植村 俊亮<sup>◇</sup>

Takeharu EDA Toshiyuki AMAGASA  
Masatoshi YOSHIKAWA  
Shunsuke UEMURA

順序木の節点に、前置順と後置順の順序を保存した対からなるラベルを与えると、任意の節点間の先祖子孫関係を、ラベルの比較だけで判定することができる（これを範囲ラベル付け手法という）。この手法を XML 木に適用することによって、正規経路式を含む XML 問合せの処理を高速化することが可能となる。今日、大量かつ大容量の XML データが流通しており、これらの XML データを効率良く更新しながら管理したいという要求が高まっている。更新の起きる XML データに範囲ラベル付け手法を適用するには、ラベルを付け直す必要がある。本研究は、このラベルの付け直しに焦点を当てた。大規模なラベルの付け直しを避けながら、構造結合による高速な XML 問合せ処理を実現できる手法を提案する。

The Range Labeling, based on pre- and postorder of an ordered tree, speeds up the processing of XML queries including Regular Path Expressions. In the Rang Labeling, after each node in an XML tree is labeled with a pair of numbers, by comparing these labels we can detect the ancestor-descendant relationship between any two nodes. Today, huge amounts of XML data have already appeared, and developers have the needs to use such XML data that are updated frequently. Spaces for labels are limited. Therefore, when XML data is updated, we need to re-label in order to retain relationships among node labels. In this paper, we concentrate on the issue of re-labeling. By managing node labels, it becomes possible to update XML data continuously.

### 1. はじめに

大量の XML データを効率良く利用するために、XQuery や XQL といった、種々の XML 用問合せ言語が提案されてきた。それらの問合せ言語の核をなす部分として、XPath に代表される正規経路式 (Regular Path Expression) がある。

正規経路式を評価する際に、格納された大規模な XML データの木構造のリンクをたどるのは効率が悪い。そのため、問合せ評

<sup>♡</sup> 学生会員 奈良先端科学技術大学院大学情報科学研究科博士前期課程 [takeha-e@is.aist-nara.ac.jp](mailto:takeha-e@is.aist-nara.ac.jp)

<sup>◇</sup> 正会員 奈良先端科学技術大学院大学情報科学研究科 {amagasa,uemura} @is.aist-nara.ac.jp

<sup>▲</sup> 理事 名古屋大学情報連携基盤センター [yosikawa@itc.nagoya-u.ac.jp](mailto:yosikawa@itc.nagoya-u.ac.jp)

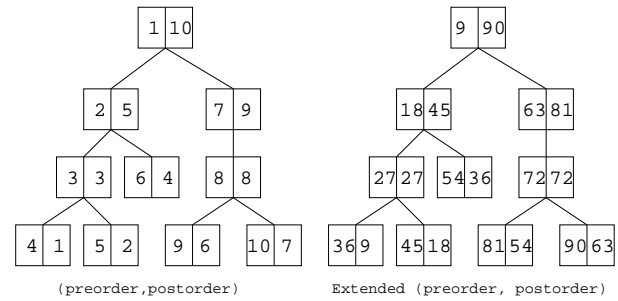


図 1: 前置順, 後置順及び拡張前置順, 拡張後置順の例

価を高速化する範囲ラベル付け手法 (Range Labeling)[3] が提案されている。範囲ラベル付け手法では、XML 木における前置順と後置順の順序を保存した対からなるラベルを節点に与え、XML 木を節点リストに分解する。さらに、正規経路式中の節点の包含関係を、ラベルに対する条件式に変換する。これにより、正規経路式による XML 木への問合せは、節点リストに対するラベルを用いた結合操作（構造結合）へと変換される。構造結合に関しては、既存の関係データベースでは、最適な結合アルゴリズムを選択しないことが分かっており [6], 種々の高速なアルゴリズムが研究されている [1][2][5]。

しかしながら、範囲ラベル付け手法の XML データの更新への対応については、まだそれほど研究されていない。例えば、単純に前置順と後置順の対で XML 木の節点をラベル付けすると、一回の挿入に対してさえ、大規模なラベルの付け直しが必要である。前置順と後置順を拡張し、番号間の間隔を空けて数えたとしても、その間隔をこえた数の節点を持つ XML 木を挿入することはできないので、更新を続けていくうちに、いずれはラベルを付け直ししなければならない。

そこで本研究は、範囲ラベル付け手法を XML データの更新に対応させることを目標とした。基本的アイデアは、範囲ラベル付け手法における包含定理が、ラベル間の順序関係しか利用していないことに着目したことである [7]。提案手法では、節点番号を間隔を空けて数えることにより、挿入操作の頻度が低い場合には、ラベルを付け直す必要がなくなる。また、挿入箇所の間隔よりも挿入 XML 木の節点数が多い場合や、挿入、削除の繰返しにより極端に番号と番号の間隔が狭まり、挿入ができなくなった場合には、挿入箇所の両端の番号を少しずつ広げることによって、小規模に番号の偏りを均していく。提案するアルゴリズムは、挿入先 XML 木と挿入 XML 木の節点数の合計が、ラベルに用いるビット数で表現可能であれば、必ず挿入が可能である。小規模なラベルの付け直しで、ラベル間の間隔の均衡を保ち続け、大規模なラベルの付け直しを避けながら、構造結合による効率の良い XML 問合せ処理を実現できる。

### 2. 範囲ラベル付け手法

XML データは、ラベル付順序木とみなせる。順序木とみなして議論するときには、XML データを XML 木と呼ぶことにする。さらに、順序木の部分木に対応させて XML 木の部分木を部分 XML 木と呼ぶ。部分 XML 木は XML 木である。範囲ラベル付け手法では、XML 木中の節点を前置順及び後置順で数えており、その際に間隔をあけて数えることによって、後々の更新操作に対応する基盤を与える。議論を分かりやすくするため、以後 XML 木の各節点の型は区別しないものとする。つまり要素節点も、文節点も一つの節点としてそれぞれ唯一に番号をふる。属性節点は今回は無視することにする。

$T$  を XML 木とする。 $T$  が  $n$  個の節点から成ることを  $|T| = n$  と書く。 $T$  の根節点を  $root(T)$  で表す。節点  $e$  の前置順を  $pre(e)$ 、後置順を  $post(e)$  で表す。 $root(T)$  に関しては、図 2 のように、次の定理が成り立つ。

## 定理 1

(部分) XML 木  $T$  中のすべての節点の中で  $root(T)$  が最も前置順の値が小さく、後置順の値が大きい。

次に節点番号を定義する。

## 定義 1 節点番号

XML 木  $T$  において、 $(pre(e), post(e))=(i, j)$  である節点  $e$  の節点番号は、

$$(a_i, b_j)$$

とする。ただし、 $a_i, b_j$  はそれぞれ  $i, j$  に対して狭義単調増加数列である。

$a_i, b_j$  をそれぞれ拡張前置順、拡張後置順と呼び、 $epre(e), epost(e)$  で表す。つまり、節点をラベル付けすると、図 1 のように、XML 木中を、前置順及び後置順に何らかの間隔を空けて数えた番号の対を与えることである。

$T$  中のすべての節点に節点番号を与えることにより、 $T$  の拡張前置順列を  $(a_1, a_2, a_3, \dots, a_{n-1}, a_n)$  と定義する。同様に  $T$  の拡張後置順列は  $(b_1, b_2, b_3, \dots, b_{n-1}, b_n)$  と定義される。今後、この節点番号と拡張前置順列及び拡張後置順列を用いてラベルの節点番号管理を説明する。

## 定理 2 包含定理

XML 木  $T$  中の任意の節点  $x = (a_i, b_j), y = (a_k, b_l)$  に対して次の条件を満たすときだけ、 $x$  は  $y$  の先祖となる。

$$a_i < a_k, \text{ かつ } b_j > b_l$$

(拡張)前置順と(拡張)後置順を軸とした二次元平面を考えると、ある節点の先祖、子孫は、その節点の左上の範囲、右下の範囲にそれぞれ対応する。

包含定理により、任意の節点間の先祖子孫関係がラベルの比較だけで判定可能になり、正則経路式を含む XML 木への問合せは、包含定理の不等式を述語に含む構造結合 (Structural Join) へと変換される。

## 2.1 XML 木の更新操作

XML 木の更新操作には、下記の基本的な操作が必要とされる

- 挿入 (n, k, X) … 節点  $n$  の  $k$  番目の子供に XML 木  $X$  を挿入する。
- 削除 (X) … 部分 XML 木  $X$  を削除する。
- 修正 (n, v) … 節点  $n$  の値を  $v$  に修正する。
- 移動 (n, k, X) … 部分 XML 木  $X$  を節点  $n$  の  $k$  番目の子供に移動する。

修正の際には、XML 木の構造は変化しないので、節点番号は修正する必要がない。また移動は挿入と削除によって実現可能であるので、今回は考慮しないことにした。よって以下では、挿入と削除について考える。

ここで、挿入と削除の基本単位は XML 木とする。XML 木に関しては図 2 のように、次の性質が成り立つ。

## 定理 3

任意の挿入する XML 木及び削除する部分 XML 木は連続した前置順及び後置順を持つ。

定理 3 により、挿入する際には、拡張前置順列及び拡張後置順列のある一箇所の間隔に XML 木、つまり番号列を挿入すればいいことが分かる。また、削除に関しても、ある連続した部分拡張前置順列及び部分拡張後置順列を削除すればよい。

## 2.2 問題点

範囲ラベル付け手法では、間隔を空けて XML 木中の節点を数えているため、十分な間隔が空いている箇所には、連続した番号列を挿入 XML 木の各節点に与えることによって、そのまま XML 木を挿入することができる。しかしながら、一旦固定したラベルを与えてしまうと、隣り合う拡張前置順 (拡張後置順) の間隔以

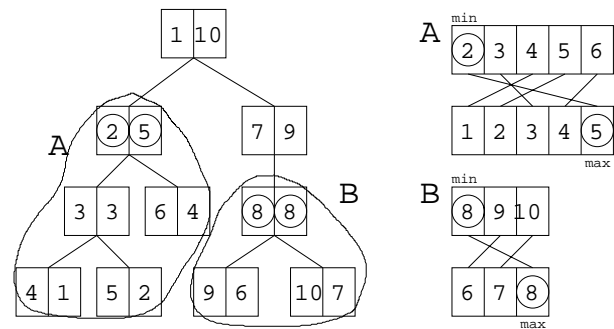


図 2: 部分 XML 木は連続した前置順、後置順を持つ



図 3: 不均衡に番号を使い尽くした結果、矢印のところに挿入できない例

上の節点数を持つ XML 木を挿入することはできない。たとえ、非常に大きな間隔を空けてラベルを与えたとしても、挿入、削除を繰り返すことによって、図 3 のように番号が極端に偏ってしまうことがある。

こうした問題に対して、我々は節点に与えたラベルの番号を小規模に付け直す、節点番号管理を提案する。

## 3. 節点番号管理

提案するラベルの節点番号管理の基本的流れを説明する。拡張前置順列及び拡張後置順列についてそれぞれ独立におこなう。

まず、XML 木への操作を行う際に、更新箇所の両端の節点番号間の間隔に XML 木が挿入可能か判定する。可能であれば、その間隔を等分割して挿入 XML 木に番号を割り当て、更新後の間隔が平衡状態 (定義 3) になるかどうか確認する。間隔が平衡状態でなければ、拡張前置 (後置) 順列を一段外側に辿ることによって、挿入箇所の間隔を両側に広げ、挿入 XML 木の節点数を 2 増やす。平衡状態になるまで確認しするところまで戻り、同じことを平衡状態になるまで繰り返す。更新後に発生する間隔が広すぎる場合も狭すぎる場合も同じ操作で、節点番号の間隔の分布を均していく。これは、全ての節点数で等分割した状態を最も良い状態とし、その上下定数倍までを平衡 (定義 3) としているためである。これにより、常に節点番号の間隔の平衡状態を保ち、大規模なラベルの付け直しを避けることができる。

$Width$  をラベルに割り当てるビット数で表現できる数値の場合の数とする。  $T(|T| = n)$  を更新される XML 木、  $ExPreList(T)$ 、  $ExPostList(T)$  をそれぞれ  $T$  の拡張前置順列、拡張後置順列、  $X(|X| = m)$  を挿入 XML 木とする。  $Width$  を等分割した幅  $BestInterval := \left\lceil \frac{Width}{|T|+|X|+1} \right\rceil$  間隔で番号がふられているときを最も良い均衡状態とする。挿入可能条件として、この幅が 1 以上でなければならない。

## 定義 2 挿入可能

$BestInterval \geq 1$  のとき、挿入可能である。

$BestInterval$  に対して、ある許容範囲までは、均衡が保たれているとしてラベルの付け直しをしない。この範囲を定める定数を  $min, max$  とする。  $0 < min \leq 1, 1 \leq max$  である。

## 定義 3 平衡

現在注目している間隔  $x$  が、次の条件を満たしているとき、 $x$  は平衡である。

$$min \cdot BestInterval < x < max \cdot BestInterval$$

節点番号の均衡の確認及び調整をするタイミングについては種々考えられるが、今回は最も基本的なタイミングとして一括読み、挿入、削除の際にアルゴリズムを適用することを想定した。

### 3.1 挿入, 一括読み込みアルゴリズム

ここでは, 拡張前置順列についてだけ説明するが, 拡張後置順列についても同様である. 挿入する際のアルゴリズム *Insert* は次のようになる.

#### Algorithm *Insert*

入力: 拡張前置順列:  $\mathbf{T}$ , 挿入 XML 木の節点数:  $m$ , 挿入箇所の直前の前置順:  $k$

出力: 挿入を完了した  $n + m$  個の拡張前置順列

```

1 if (挿入可能でない) exit;
2 if ( $k = |T|$ )  $b := \frac{Width - a_k}{m + 1}$ ;
3 if ( $k = 0$ )  $b := \frac{a_1}{m + 1}$ ;
4 else  $b := \frac{a_{k+1} - a_k}{m + 1}$ ;
5 if ( $b$  が平衡でない)
6   if ( $k = 1$ )
7      $\mathbf{T} := (a_3, \dots, a_{n-1}, a_n)$ ;
8      $Insert(\mathbf{T}, m + 2, 1)$ ;
9   if ( $k = |T|$ )
10     $\mathbf{T} := (a_1, a_2, \dots, a_{n-3})$ ;
11     $Insert(\mathbf{T}, m + 2, |T|)$ ;
12  else
13     $\mathbf{T} := (a_1, a_2, \dots, a_{k-1}, a_{k+2}, \dots, a_n)$ ;
14     $Insert(\mathbf{T}, m + 2, k - 1)$ ;
15 else
16  return  $\mathbf{T} := (a_1, a_2, \dots, a_k, a_k + [b], a_k + 2[b], \dots,$ 
 $a_k + m[b], a_{k+1}, \dots, a_n)$ ;

```

$k$  は, 挿入箇所の間隔を構成する拡張前置順の小さい方の前置順である. この  $k$  を求めるのは簡単ではない. 挿入する際には, 親節点からみて何番目の子節点かを特定する必要があるが, 今回議論している範囲ラベル付け手法では, この問合せを高速に処理することができないからである. 子 XML 木を先頭からすべてたどっていけば, 検索することは可能であるが, 非常にコストがかかると予想される. 解決策としては, 兄弟節点同士をリンクでつなげることが考えられるが, 詳細については今後の課題である.

一括読み込みは, 挿入アルゴリズムの入力で,  $T$  を空順列,  $m$  を読み込む XML 木の節点数,  $k$  を 0 とすることによって実現可能である.

### 3.2 削除アルゴリズム

削除は, 拡張前置順列と拡張後置順列を往復する必要があるため, 引数は XML 木  $T$  と削除する部分 XML 木  $X$  になる. 番号を削除した後に発生した間隔が, 平衡かどうか判定しラベルを付け直す, その際には *Insert* アルゴリズムの挿入節点数を 0 とすることによって実現している.

#### Algorithm *Delete*

入力: XML 木  $T$ , 削除する部分 XML 木  $X$

出力: 削除後の  $T$  の拡張前置順列, 拡張後置順列

```

1 DeleteNumber( $T$ , root( $X$ )); // root( $X$ ) 以下の番号を消す.
2 Insert( $ExPreList(T)$ , 0, pre(root( $X$ )) - 1);
3 Insert( $ExPostList(T)$ , 0, post(root( $X$ )) - 1);

```

*DeleteNumber*( $T, d$ ) は,  $ExPreList(T)$ ,  $ExPostList(T)$  それぞれの  $d$  節点以下の番号を削除する. 片方だけでは, 削除する範囲が分からないため定理 1 及び定理 3 の性質を用いて同時に処理している.

*deletenodePre*( $i$ ) は, 前置順が  $i$  番目の節点番号 (前置順, 後置順も含めて) を削除する関数である.

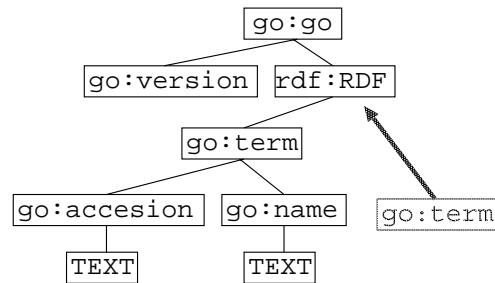


図 4: XML 木

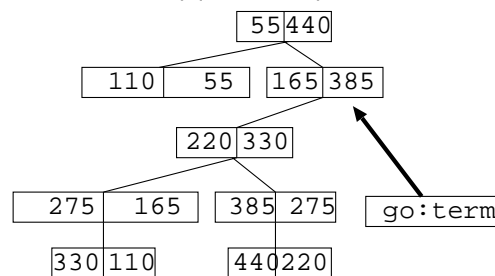


図 5: 初期節点番号

#### Algorithm *DeleteNumber*( $T, d$ )

```

1  $i := pre(d)$ ;
2  $j := post(d)$ ;
3  $b_{max} := epost(d)$ ;
4 while ( $0 < b_j \leq b_{max}$  かつ,  $i \leq n$ )
5    $deletenodePre(i)$ ;
6    $i++$ ;
7    $j :=$  前置順が  $i$  の節点の後置順;

```

## 4. アルゴリズムの具体的動作

[4] で公開されている XML データを利用して, 提案手法の具体的な動作を説明する. 今, 図 4 のように XML 木  $T(|T| = 8)$  が与えられたとし,  $Width = 500$ ,  $min = \frac{1}{2}$ ,  $max = 2$  とする.

### 4.1 一括読み込み

$BestInterval = \lfloor \frac{500}{8+1} \rfloor = 55$  であるので,  $ExPreList(T) = (55, 110, \dots, 440)$ ,  $ExPostList(T) = (55, 110, \dots, 440)$  と図 5 のように初期節点番号を与える.

### 4.2 挿入

矢印で示された部分に *go:term* 要素 (図 6) が挿入されるものとする. 挿入箇所の左端を与える前置順及び後置順は,  $k = 8$  (前置順),  $k = 7$  (後置順) となる.  $|X| = 13$ ,  $\mathbf{T} = ExPreList(T)$  とし

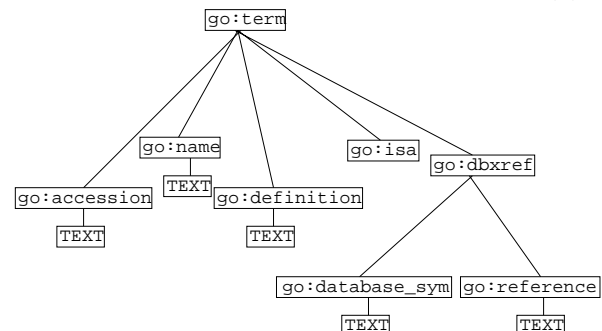


図 6: 挿入 XML 木

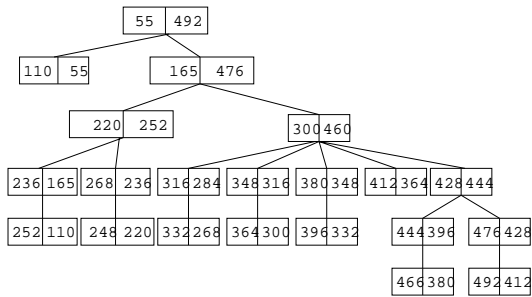


図 7: 挿入後

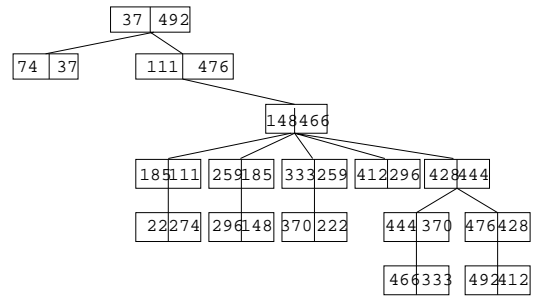


図 9: 削除後

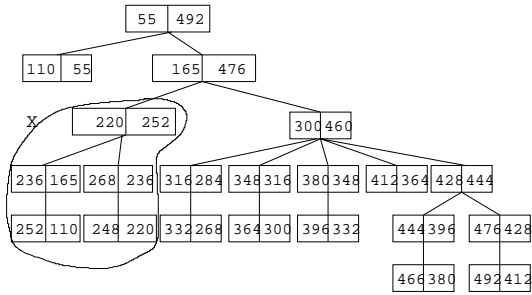


図 8: 削除部分 XML 木

て  $Insert(T, 13, 8)$  を適用すると,

$$BestInterval = \left\lfloor \frac{500}{8 + 13 + 1} \right\rfloor = 22 \geq 1$$

であるので、挿入可能である。今、 $|T| = 8 = k$  より、

$$b = \frac{500 - 440}{14} = 4 < \min \cdot BestInterval = 11$$

より平衡でないので、 $T = (55, 110, \dots, 300)$  として  $Insert(T, 15, 6)$  を呼び出す。

$$b = \frac{500 - 330}{16} = 10 < 11$$

より平衡でないので、 $T = (55, 110, 165, 220)$  として  $Insert(T, 17, 4)$  を呼び出す。

$$\begin{aligned} \max \cdot BestInterval &= 44 > \\ b &= \frac{500 - 220}{17} = 16 > 11 \end{aligned}$$

より平衡となる。よって  $T = (55, 110, 165, 220, 220 + 16, 220 + 2 \times 16, \dots, 220 + 17 \times 16)$  を返し、挿入が完了する。  $ExPostList(T)$  についても同様であり、図 7 のようになる。

### 4.3 削除

図 8 が示す XML 部分木 X を削除する。  $|T| = 21$ 、 $|X| = 5$  である。

$Delete(T, X)$  を適用すると、まず、 $DeleteNumber(T, root(X))$  を呼びだし、 $i := pre(d) = 4$ 、 $j := post(d) = 6$ 、 $b_{max} := epost(d) = 252$  であり、 $deletenodePre(4)$  で  $(220, 252)$  を消去し、次に  $i = 4 + 1 = 5$ 、 $j = 3$  とし、 $b_3 = 165 \leq b_{max}$ 、 $5 \leq 21$  であるので、 $deletenodePre(5)$  で  $(236, 165)$  を消去する。次に  $i = 6$ 、 $j = 2$  となり、 $b_2 = 110 \leq b_{max}$ 、 $6 \leq 21$  であるので、 $deletenodePre(6)$  で、 $(252, 110)$  を消去する。次に  $i = 6 + 1 = 7$ 、 $j = 5$  とし、 $b_5 = 236 \leq b_{max}$ 、 $7 \leq 21$  であるので、 $deletenodePre(7)$  で  $(268, 236)$  を消去する。次に  $i = 7 + 1 = 8$ 、 $j = 4$  とし、 $b_4 = 220 \leq b_{max}$ 、 $8 \leq 21$  であるので、 $deletenodePre(8)$  で  $(284, 220)$  を消去する。次に  $i = 8 + 1 = 9$ 、 $j = 19$  とするが、 $b_{19} = 460 > b_{max}$  であるので、while ループが止まり、 $DeleteNumber(T, root(X))$  が終了する。次に、発生した間隔を調整するため、 $Insert(ExPreList(T), 0, 3)$  及び、 $Insert(ExPostList(T), 0, 1)$  を呼びだし、調整した後、結果を出力する (図 9)。

## 5. まとめ

本論文では、正則経路式を含む問合せ処理を高速化する範囲ラベル付け手法において、XML 木の更新に対応させるため、節点のラベルの番号を管理する手法を提案した。提案手法では、更新操作をおこなうときに、更新箇所周辺の番号の偏り具合を判定して、狭すぎる場合は広げ、広すぎる場合は縮めて番号の分布が平衡状態にあるよう、小規模にラベルを付け直す。これにより、大規模なラベルの付け直しを避けながら、後々の更新操作に柔軟に対応することができる。

## [文献]

- [1] Shurug Al-Khalifa, H. V. Jagadish, et al. Structural Joins: A Primitive for Efficient XML Query Pattern Matching. In *Proc. ICDE*, 2002.
- [2] Shu-Yao Chien, Zografoula Vagena, et al. Efficient Structural Joins on Indexed XML Documents. In *Proc. VLDB*, 2002.
- [3] Edith Cohen, Haim Kaplan, and Tova Milo. Labeling Dynamic XML Trees. In *Proc. PODS*, 2002.
- [4] Gene Ontology Consortium. <http://www.geneontology.org>.
- [5] Quanzhong Li and Bongki Moon. Indexing and Querying XML Data for Regular Path Expressions. In *The VLDB Journal*, 2001.
- [6] Chun Zhang, Jeffrey F. Naughton, et al. On Supporting Containment Queries in Relational Database Management Systems. In *Proc. SIGMOD*, 2001.
- [7] 江田毅晴, 天笠俊之, 吉川正俊, 植村俊亮. 更新に強い XML 節点数え上げ手法とその管理. 情報処理学会研究報告, データベースシステム 2002-DBS-128, 2002.

### 江田 毅晴 Takeharu EDA

奈良先端科学技術大学院大学情報科学研究科博士前期課程在学中. XML データベースの研究に従事. 情報処理学会学生会員. 日本データベース学会学生会員.

### 天笠 俊之 Toshiyuki AMAGASA

奈良先端科学技術大学院大学情報科学研究科助手. データベースシステムの研究に従事. 情報処理学会正会員. 電子情報通信学会正会員, 日本データベース学会正会員.

### 吉川 正俊 Masatoshi YOSHIKAWA

名古屋大学情報連携基盤センター教授. データベースシステムの研究に従事. 情報処理学会正会員. 電子情報通信学会正会員, 日本データベース学会理事.

### 植村 俊亮 Shunsuke UEMURA

奈良先端科学技術大学院大学情報科学研究科教授. データベースシステムの研究に従事. 情報処理学会フェロー. 電子情報通信学会フェロー, 日本データベース学会正会員. 著書に「データベースシステムの基礎」(オーム社) など.