XML木のための動的範囲ラベル付け手法

江田毅晴[†] 櫻井保志[†] 天笠俊之[††]
吉川正俊[†††] 植村俊亮[††]

本論文は，動的 XML データのための新しい節点ラベル付け手法について述べる．今日，範囲ラベル付け手法に基づく構造結合は，XML 問合せ処理において最も重要な課題の 1 つと見なされている．XML 木中のそれぞれの節点は，木の中での前置順と後置順の順序関係を保存した数の対でラベル付けされる．このラベルを利用すると，任意の節点間の先祖子孫関係が判定できるため，XML 問合せ処理を効率的に行うことができる．しかしながら，XML データが更新されると，前置順，後置順それぞれの順序関係を保持するために，ラベルを付け直さなければならない．たとえ間隔を空けてラベル付けしたとしても，大量の挿入や大きな XML データの挿入があった場合には十分ではないと考えられる．頻繁に更新の起こる動的な XML データへの高速な問合せ処理を継続するためには，大規模なラベルの付け直しを避ける必要がある．そこで，本論文では大規模なラベル付け直しのコストを分散させ，小規模なラベルの付け直しでノードのラベルを維持する 2 つの動的節点ラベル付け手法を提案する．1 つは簡単な小規模ラベル付け直し手法であり，もう 1 つは近似ヒストグラムを用いて更新操作の情報を保持する，より洗練された手法である．これらの 2 つの手法により，節点のラベルを動的かつ小規模に管理することができる．実験結果により，更新操作を続けていってもコストの高い大規模なラベルの付け直しを回避できることを示す．

Dynamic Range Labeling for XML Trees

Takeharu Eda,[†] Yasushi Sakurai,[†] Toshiyuki Amagasa,[††]
Masatoshi Yoshikawa[†††] and Shunsuke Uemura[††]

This paper presents a new node labeling approach for data-oriented dynamic XML trees. Recently, structural joins based on range labeling schemes have been considered as one of the most important research topics for XML query processing. Each node in an XML tree is labeled with a pair of numbers which preserves the preorder and the postorder in the tree. The ancestor-descendant relationship between any two labeled nodes can be verified by using their labels; consequently, this property leads to efficient evaluation of XML queries. When an XML data set undergoes updating, however, the nodes have to be relabeled in order to keep their order relationships. Moreover, even if "gaps" are used in labels, the gaps may prove to be insufficient if there are many update operations or the insertion of a large XML tree is required. To avoid a "gap" shortfall, we propose two new dynamic node labeling schemes. One is simple local relabeling scheme and the other is more sophisticated in that it uses approximate histograms to keep approximated information of update operations. These two techniques allow node labels to be managed dynamically and locally. Experiments show that bulk relabeling, which is expensive, can be avoided while still permitting update operations.

1. はじめに

データベース技術に基づいた XML 木の構造をとらえる多くの問合せ言語が開発されてきた[4),12),17)]．XML データはラベル付き順序木であるため，子供節点を指定する child 軸や子孫節点を指定する descendant 軸がそうした問合せ言語の中核を占める．しかしながら，descendant 軸を含んだ問合せを大量の XML データに対してナイーブに実装すると非常に遅くなることが予想される．たとえば，XML データを DOM 木[15)] のように親子関係のリンク構造として扱っていたり，あるいは SAX[5)] のようなイベント形式で処理したりしている際には，XML データ全体に及ぶ検索という大きなコストがかかる．特に XML データが巨大ですべてを一次記憶に保持することができずに，一部を二次記憶におかなければならない場合には，二次

† 日本電信電話株式会社 NTT サイバースペース研究所
NTT Cyber Space Laboratories, NTT Corporation
†† 奈良先端科学技術大学院大学情報科学研究科
Graduate School of Information Science, Nara Institute of Science and Technology
††† 名古屋大学情報連携基盤センター
Information Technology Center, Nagoya University

記憶の I/O コストが処理時間に莫大な影響を与える．

昨今このコストの高い XML データの全検索を避けるために，範囲ラベル付け手法（*range labeling*[9),11),22),23)]）に基づいた，構造結合（*structural joins*[3),6)~8),10),14),16),19),21)]）アルゴリズムが各種開発されてきた．XML 木中の節点は，前置順と後置順を保存した数の対でラベル付けされ，XML 木は節点のリストへと分解される．さらに，XPath 式中の節点の包含関係は，ラベルに対する構造条件へと変換される．つまり，節点の包含関係を含む XML 木への問合せは，節点のラベルを使った構造結合へと変換される．要素名，テキストや付与されたラベルに索引を構築することによって，child 軸や，descendant 軸を含む問合せをコストのかかる XML データの全検索をせずに処理することができる．

一方，データ交換フォーマットとして XML が急速に普及したため，インターネット上には大量の XML データが見られるようになった．たとえば，すべての真核生物に適用できる動的に制御可能な語彙体系の構築を目指している Gene Ontology Consortium[13)] は，データをテキスト形式だけではなく，XML 形式でも提供している．しかしながら，1 日に何百カ所も語彙が更新され，テキスト形式のデータも更新されるのに対して，XML 形式のデータは月に 1 度しか更新されない．これは，数十メガにも及ぶ XML データの更新管理の難しさにも一因があるものと考えられる．更新を即座に XML データに反映させ，問合せ処理およびデータの提供を実現できれば，XML 形式であることの利点を用いて，まだ開拓されていない新たな語彙データの応用場面を生み出す可能性がある．

こうした可能性があるにもかかわらず，XML データが頻繁に更新される場合の範囲ラベル付け手法については，まだそれほど研究されていない．単純に前置順と後置順で XML 木をラベル付けすると，たった 1 つの節点の挿入に対してさえ，挿入箇所以降のすべての節点のラベルを付け直す必要がある．間隔を空けてラベルを付けることによって更新に備えることも可能であるが[9)]，間隔が挿入される XML 木の節点数よりも小さいことがありうる．また，更新の起こる箇所の偏りにより，利用可能な間隔が使いつくされてしまうことも予想され，永続的な範囲ラベル付け手法は，更新の頻繁に起こるデータ指向の XML データのラベル付けには向かないとされている[18)]．今回，テストデータであるが，間隔を使いつくしてしまう状況が起こりうることを，実験で確認した．つまり，更新が続くにつれ，いずれラベルを付け直さなければならないということである．

頻繁に更新の起こる動的な XML データへの高速な問合せ処理を継続するためには，大規模なラベルの付け直しを避ける必要がある．そこで，本研究では，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，動的な XML データのための，**動的範囲ラベル付け手法**を提案する．動的範囲ラベル付け手法では，更新が起こった際のラベルの付け直し回数を減らすために，間隔を空けて節点をラベル付けする．さらに，挿入の起こる箇所の間隔が，挿入 XML 木に対して狭い場合や，更新操作の偏りが，挿入箇所の間隔を使い果たしてしまった場合には，大規模な XML データ全体にわたる全ラベル付け直しをせずに，周辺のラベルを小規模に付け直す．動的範囲ラベル付け手法は，範囲ラベル付け手法における構造条件が，ラベルの値ではなく，順序関係のみによって成り立っているという事実[23)] を利用し，この順序関係を維持する形でラベルを付け直す．さらに，単純に小規模にラベルを付け直しただけでは管理が困難な[24)] 成長する XML データに対応するために，更新操作の統計情報を保持する近似ヒストグラムを導入する．近似ヒストグラムの値を利用することにより，更新操作の頻度に応じて，ラベルの間隔を適切に調節することができる．テストデータを用いた実験結果により，小規模なラベルの付け直しで，節点間の均衡を保ち続け，大規模なラベルの付け直しを避けられることが分かった．よって，更新が頻繁に起こる XML データに対しても，構造結合を利用した効率の良い XML 問合せ処理を提供し続けることができる．

本論文の構成は下記のようになる．2 章で XML 木のための範囲ラベル付け手法について述べ，関連研究について触れる．3 章で，範囲ラベル付け手法において更新操作をどのようにして扱うかについて考察し，更新操作に対応させた静的な範囲ラベル付け手法が，XML 木に対する更新操作が継続して起こった場合不十分であることを示す．4 章で，2 つの小規模なラベル付け直し手法を提案する．1 つは，簡単な小規模な付け直し手法であり，もう 1 つは，より洗練された，更新操作の情報を保持する近似ヒストグラムを利用した手法である．5 章では，実験結果について述べる．6 章で関連する項目について議論し，7 章で本論文をまとめる．

2. 範囲ラベル付け手法

XML データの 1 つの入れ子構造は，木構造における親子関係に対応し，XML データ中の文書順は，木

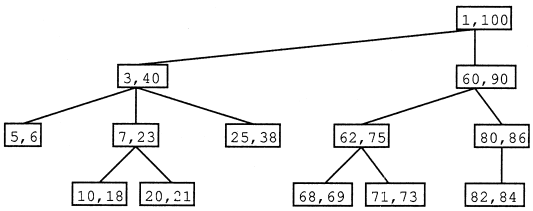

図 1 範囲ラベル付けの例
Fig. 1 An example of range labeling.

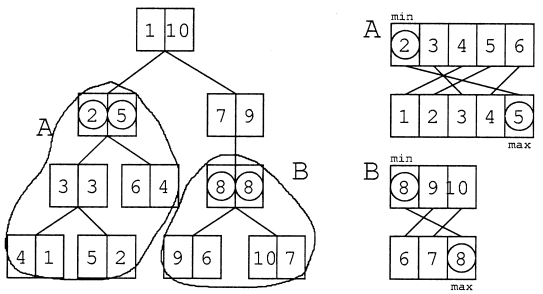

図 2 範囲ラベル付けのいくつかの性質
Fig. 2 Some properties of range labeling.

構造の子供の順序関係に対応する．つまり，XML データは，根付き順序木[2] に対応する．XML 木中のそれぞれの節点は，もともとの XML データから対応する要素名や属性名等でラベル付けされている．

XML 問合せの効率的な処理や各節点間の関係の情報の保持のために，XML 木中のすべての節点に新たに別のラベルを付与することが多い[1),11),18)]．XML 木はラベル付けされた後，節点のリストに分解して格納される．XML 木を単純にファイルとして格納する手法と比較すると，分解された節点リストに対して効率的なアクセス方法が構築されれば，XML 木の必要な部分にだけアクセスし，無駄なディスク I/O を減らすことが可能となる．

XML 木のためのラベル付け手法はいくつか研究されている[18),20)]．その中でも特に範囲ラベル付け手法は最もよく研究されている．**図 1** 中の簡単な例が示すように，節点のラベルは区間を表しており，その節点のすべての先祖のラベルに含まれるような区間になっている．よって，節点 $v = (i, j)$ と $w = (k, l)$ が次の包含関係を満たすときのみ，v は w の先祖になっている．

$$i < k < l < j$$

節点の名前と値と対応する開始と終了の数が結び付けられ，4 つの属性を持つテーブルをつくる☆．節点名属性に，B+木のような索引を構築することにより，“A//B” のような問合せは 2 段階で処理される．まず，索引を利用し高速に “A” 節点リストと “B” 節点リストを取り出す．次に，この 2 つの節点リストは，構造的に結合される[3),8)]．つまり，ある “A” 節点のすべての子孫の中からすべての “B” 節点を親子関係のリンク構造をたどって探し出したり，ある “B” 節点のすべての先祖の中からすべての “A” 節点を探し出したりする必要がなくなる．

図 1 中の例は，**区間ラベル付け**の 1 つの例と考えることができるが，一次元上の区間の包含関係が本質ではない．順序木中での前置順と後置順の順序関係が本質である☆☆．そこで，範囲ラベル付け手法の節点ラベルを次のように定義することができる．

定義 1 ［節点ラベル］
T を XML 木，v を T 中で前置順が i，後置順が j である節点のとき，v のラベルは，

$$(a_i, b_j)$$

とする．ただし，a_i，b_j はそれぞれ i，j に関して狭義単調増加正整数列である．

a_i，b_j をそれぞれ**拡張前置順**，**拡張後置順**と呼ぶ．さらに，$(a_1, a_2, a_3, \ldots, a_{n-1}, a_n)$，$(b_1, b_2, b_3, \ldots, b_{n-1}, b_n)$ をそれぞれ拡張前置順列，拡張後置順列と呼ぶ．

節点の前置順は XML データ中での開始タグの登場順であり，後置順は終了タグの登場順になっているため，ラベル付けの手続きは非常に単純である．スタックを用い，XML ファイルを 1 回走査するだけでラベル付けして節点リストに分解することができる☆☆☆．

上記のように定義された，ラベル付け手法の構造条件は，**図 2** から容易に導かれる．ある XML 木 T の根節点の前置順は，T に含まれるすべての節点の前置順の中で最も小さい．同様に，後置順は最も大きい（図 2）．

定理 1 ［構造条件］
$v = (a_i, b_j)$，$w = (a_k, b_l)$ を XML 木の 2 つの節点とする．
v は，

$$a_i < a_k, \text{ かつ } b_l < b_j$$

のときのみまたそのときに限り，w の先祖である．

本論文では，ラベルを表現するのに正整数値を用いているが，この制限は本質的ではない．たとえば，負

☆ 実際には親子関係の判定のために深さも使われる．

☆☆ つまり，条件 $k < l$ を除くことができ，前置順，後置順の二次元平面上での範囲どうしの包含関係になる．

☆☆☆ 同様に，前置順にソートされた節点リストの必要な部分を 1 回走査するだけで部分木を再構築できる[24)]．

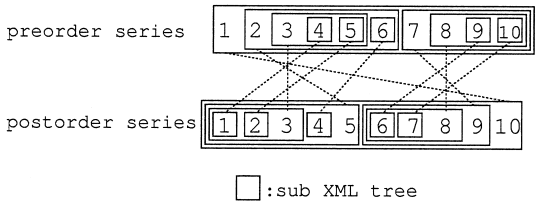

図 3 XML 部分木の前置順列および後置順列
Fig. 3 Preorder and postorder series of sub XML trees.

の値を含む整数値や浮動小数点，あるいは文字列さえもラベルとして用いることができる．ラベル間の順序関係さえ決定できればよい．正整数値を用いた理由は簡単のためと，可変長データ型を用いた場合と比較して，固定長データ型を用いた際，比較演算子が高速に計算できるためである．

3. 範囲ラベル付け手法の XML 木の更新への対応

3.1 XML 木の更新

XML データはラベル付き順序木であるため，4 つの更新操作：挿入，削除，修正，移動を考えることができる．修正は，節点の名前や値のみを修正し，木の構造は変えないため，節点に後から付与するラベルは変更する必要がない．移動は，挿入と削除の結合で実現することができる．よって本論文では，XML データの更新操作として，部分 XML 木の挿入と削除のみを考えることにする．

部分 XML 木とは，図 2 中の A や B のような木である．ある節点を根とする部分木は，その根節点のすべての子孫節点を含む．前置順に部分木を数える際には，根から始まって部分木の最右葉まで部分木内の節点のみを連続に数える．後置順の場合は，最左葉から始まって根節点まで連続に数える．つまり，部分 XML 木中の節点の前置順および後置順は両方とも連続している（**図 3**）．

本論文では，こうした挿入および削除の対象となる部分 XML 木を更新操作の**単位 XML 木**と呼ぶことにする．

3.2 静的範囲ラベル付け手法（static range labeling）

2 章で述べた範囲ラベル付け手法を XML データの更新操作に静的に対応させる．

バルクロード

もし，図 2 にあるようにラベル付けされた XML 木に挿入単位 XML 木を挿入しようとすると，挿入箇所より前置順，後置順が大きいすべての節点のラベルを付け直さなければならない．そこで，XML 木は間隔を空けて，**図 4** の左のようにラベル付けすることにする☆．

バルクロードの際の番号間の間隔は，図 4 のように，ラベルに用いる数値の最大値を節点数で割ったものとする．この例では，最大値は 100 でありバルクロードの際の間隔は 9 である．

挿　　入

挿入単位 XML 木の拡張前置順列，拡張後置順列の順序は連続であるため，挿入箇所は XML データベース全体の拡張前置順列および拡張後置順列内の 1 カ所の間隔になっている．そこで，利用可能な間隔を有効活用するために，挿入単位 XML 木を図 4 に示すように挿入箇所の間隔を等分割してラベル付けする．

削　　除

与えられたラベルを用いて，削除単位 XML 木に含まれているすべての節点の拡張前置順および拡張後置順を削除することができる．アルゴリズムを**表 1** に示す．$DeleteNumber(T, d)$ は，節点 d 自身と子孫のラベルを消去する．削除単位 XML 木の根節点である d の後置順は単位 XML 木に含まれるすべての節点の後置順の最大値であるので，d の子孫が含まれている後置順の範囲を知ることができる．さらに d の前置順は最小であるので，拡張前置順列を d から値が大きいほうへたどることによって，後置順の範囲を抜けるまで，単位 XML 木に含まれる節点のラベルを削除することができる．

以上，範囲ラベル付け手法を静的に XML 木の更新操作に対応させたが，節点間の間隔は有限であるため，この静的範囲ラベル付け手法は更新の頻繁に起こる XML 木にとっては不十分であると予想される．どれほど間隔を広げても，固定したラベルによって決まる間隔以上の部分 XML 木を挿入することはできない．**図 5** は，予想される拡張前置順列の番号の不均衡の例を示したものである．更新操作が偏って起きていた場合には，矢印の箇所に新たにデータを挿入しようとしても，図のように詰まっていることが予想される．

3.3 予 備 実 験

XML 節点のラベルが上述のような間隔を空けただけの静的な範囲ラベル付け手法で効率良く管理できるかどうか，つまり間隔を使いつくしてしまう状況が発生するかどうかを確認するため，XML 木の拡張前置（後置）順列を単純な整数値列と見なして実験を行っ

☆ 区間ラベル付けにおける間隔を空けたラベル付けは，文献 9) で導入された．

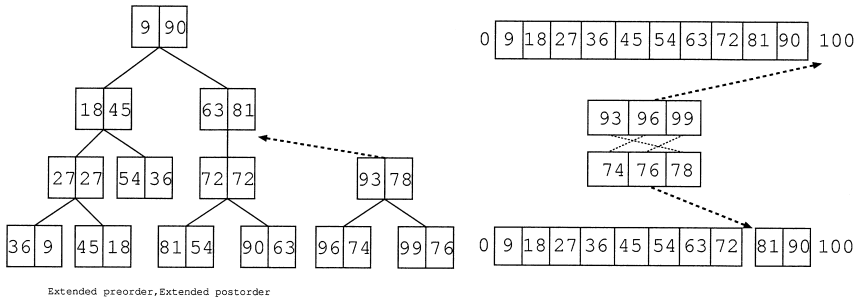

図 4 単位 XML 木を挿入
Fig. 4 Inserting an unit (a sub XML tree).

表 1 拡張前置順列と拡張後置順列から削除単位の節点のラベルを削除する
Table 1 Deleting nodes from extended preorder series and extended postorder series.

Algorithm *DeleteNumber*(T, d)

```
1  i := preorder of d;
2  j = max := postorder of d;
3  while (0 < j ≤ max, and i ≤ n)
4        delete the node whose preorder is i;
5        i++;
6        j := postorder of the node whose preorder is i;
7  endwhile
```

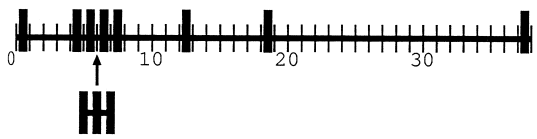

図 5 拡張前置順列における番号列の不均衡の例
Fig. 5 Unbalance of an extended preorder series.

た．今回，人工的な挿入と削除からなる更新操作列を作成した．

それぞれの挿入は，挿入箇所を示す数値と挿入単位 XML 木に含まれる節点数の 2 つの数値の対で示される．それぞれの削除は，削除される部分木の開始の数値と終了の数値で示される．挿入の場合は，もし挿入される単位 XML 木の節点数のために十分な間隔が挿入先にある場合は，挿入箇所の間隔を挿入単位 XML 木に含まれる節点に等分割して割り当て，ラベル付けする．もし間隔が足りない場合は，XML データベース中のすべての節点のラベルを付け直す．すべての節点のラベルの付け直しは，非常にコストが高い．

3.3.1 実験操作列とデータセット

それぞれのテストデータセットのパラメータを**表 2** に示す．更新操作の位置の分布に関して，3 種類のパターンをつくった．挿入操作と削除操作の分布は独立である．正規分布の際の位置の平均はラベル列の 30%と 70%の位置の場合を行い，標準偏差は 10%で行った．

表 2 テストデータのパラメータ
Table 2 The parameters of test data.

挿入の分布	一様
	正規（30%）
	正規（70%）
削除の分布	一様
	正規（30%）
	正規（70%）
（挿入）:（削除）	1:1
	2:1
更新単位 XML 木の節点数	2
	10
	30

挿入と削除の割合に関しては，1:1 と 2:1 について行った．2:1 という割合は，実際の成長する XML データである go.xml [13] の更新操作の割合に近い値である．

更新操作単位 XML 木に含まれる節点数は最大値を決めてランダムに生成した．以降，単位 XML 木の節点数は，このランダムに生成する節点数の最大値を表すものとする．単位 XML 木の節点数として，2，10，30 について行った．更新操作回数は 30,000 回，32 ビット正整数を使ってラベルを表現した．更新操作を始める前の初期節点数は，10,000 とした．

3.3.2 実験結果

更新操作を続け，発生した全ラベル付け直しの回数を数えた．結果を**表 3** に示す．たとえば，更新単位 XML 木の節点数が 30 のときには全ラベル付け直しは，439(= $\frac{30000}{68.3}$) 回の更新操作のたびに 1 度起こっていることになる．更新単位 XML 木の節点数が増えるにつれ，全ラベル付け直しの回数が非常に増えていることも分かる．

3.4 静的ラベル付けの問題点

もしも XML データが静的であれば，すなわち更新がそれほど起こらず問合せもそれほど頻繁でない場合，データベースへのトラフィックが少ないときを見計らっ

表 3 30,000 回の更新操作の中で発生した全ラベル付け直しの回数の平均値
Table 3 The average times of bulk loads in 30,000 operations.

更新単位 XML 木の節点数	全ラベル付け直しの回数（平均）
2	1.0
10	14.2
30	68.3

てシステムを停止し，すべての節点のラベルを付け直すことは可能であろう．しかしながら，XML は近年新しい役割を持ち始めている．つまり，単にデータ交換の共通形式として利用されるだけではなく，大量のデータの基本的構造を表現する手段として利用されている．そうすると，全ラベル付け直しはシステムをしばらく止める必要があるため深刻な問題となる．こうしたシステムの停止を回避し，XML 問合せ処理を効率良く提供し続けるため，ラベルの番号間のバランスを保ち全ラベル付け直しを避ける 2 つの動的範囲ラベル付け手法を提案する．

4. 動的範囲ラベル付け手法

XML 木の様々なバターンの更新操作に対応させるため，2 つの動的ラベル付け手法を提案する．これにより更新操作の度に周辺のラベルを小規模に付け直すことにより，ラベルの番号間のバランスが保たれ，全ラベル付け直しの回数をより少なくすることができる．

1 番目の動的ラベル付け手法は，更新箇所の間隔が狭すぎる場合は拡大し，広すぎる場合には伸縮すればいいという単純な観察に基づいた動的ラベル付け手法である．2 番目の手法は，1 番目の手法のナイーブな拡大伸縮判定条件を近似ヒストグラムを用いて，より更新操作の分布を反映させる形で改良し洗練させた動的ラベル付け手法である．

管理するラベルの番号列は，拡張前置順列と拡張後置順列の 2 つあるが，それぞれの順列における番号の順序関係は独立しているため，それぞれの順列の番号を独立に付け直すことが可能である．よって，以降では XML 木 T の拡張前置順列 $ExPreList(T) = (a_1, a_2, \cdots, a_n)$ についてだけ説明する．$Width$ は，ラベルの番号に当てることのできる正整数値の最大値である．また説明のため，$a_0 = 0$，$a_{n+1} = Width$ と仮定する．

4.1 簡単な動的範囲ラベル付け手法

図 5 に示された問題を解決するため，更新箇所の節点のラベルを簡単かつ小規模に付け直す．範囲ラベル付け手法の構造条件は，ラベルの番号の順序関係のみに基づいているため，ラベルの番号の順序関係が変わらなければ，近隣の節点のラベルを付け直すことが可能である．

表 4 挿入アルゴリズム
Table 4 Insertion algorithm.

Algorithm *Insert*
input: extended preorder series: $\mathbf{T} = (a_1, \ldots, a_n)$,
the number of nodes in a unit to be inserted: $|X|$,
the preorder of inserted position: k
output: extended preorder series with $n + |X|$ nodes

```
1  l := k;
2  m := k + 1;
3  while ((a_m − a_l) is not balanced)
4      |X| = |X| + 2;
5      if (l = 1)
6          m = m + 2;
7      else if (m = |T|)
8          l = l − 2;
9      else
10         l = l − 1;
11         m = m + 1;
12     endif
13 endwhile
14 Δx := ⌊(a_m − a_l)/(|X|+1)⌋;
15 return T := (a_1, a_2, ..., a_l, a_l + Δx, a_l + 2Δx,
       ..., a_l + |X| · Δx, a_m, ..., a_n);
```

4.1.1 簡単なラベルの付け直し方法

X を挿入単位 XML 木とする．$|X|$ は X に含まれる節点の数を表す．節点のラベルの間隔が最も均衡のとれた状態であることを示す定数，つまり XML データ中の全節点に対して幅 $Width$ を最大限に利用して与えられる均等な幅 $BestInterval$ を次のように定義する．

$$BestInterval := \left\lfloor \frac{Width}{|T| + |X| + 1} \right\rfloor$$

X を間隔 x の幅を等分割してラベル付けしたときに発生する間隔が $BestInterval$ の min 倍より大きい場合は，周辺のラベルは付け直さない．

定義 2 ［balanced］（挿入の場合）
ある間隔 x が次の条件を満たす場合，x は **balanced** である．

$$min \cdot BestInterval \cdot (|X| + 1) < x$$

ただし，min は 1 未満の定数である．

この条件は，間隔 x が挿入単位 XML 木 X にとって十分かどうか確認する．挿入箇所の間隔をこの条件が満たされるまで広げることができる．$BestInterval$ に対して許容する下限の間隔を min が定めている．

挿入アルゴリズムを**表 4** に示す．はじめに挿入箇所の間隔の長さが **balanced** かどうか確認する（3 行

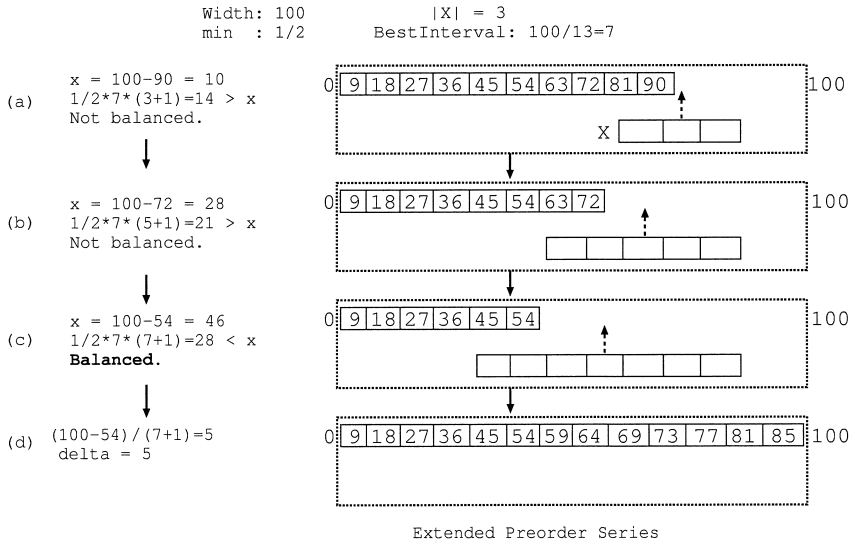

図 6 簡単なラベルの付け直しの具体例
Fig. 6 An example behavior of simple relabeling.

目）．もしそうでない場合は T の挿入箇所の間隔を両脇に（あるいは片方に）たどることにより広げ，挿入単位 XML 木の節点数を 2 増やす（4〜12 行目）．この手順を挿入箇所の間隔の長さが **balanced** になるまで続ける．**図 6** は図 4 の挿入操作の際の，この簡単なラベルの付け直し手法の振舞いの具体例を示したものである．今，$min = 1/2$ とする．(a) ステップにおいて，間隔 $x = 10$ は，3 節点の挿入には不十分であると判断される．そこで，挿入箇所の間隔は値の小さいほうへ 2 広げられ，挿入単位 XML 木の節点数に 2 加える（(a) から (b)）．(c) ステップにおいて，間隔 $x = 46$ は 7 節点の挿入に対して十分であると判定する．そこで，間隔 x を $7+1$ で割り，7 個の挿入節点をその値の間隔でラベル付けする．

次に，削除のために **balanced** の定義を書き換える．削除は挿入の場合とは逆に，削除箇所の間隔が $BestInterval$ の max 倍より小さい場合には周辺のラベルは付け直さない．

定義 3 ［balanced］（削除）
もし間隔 x が次の条件を満たすなら，x は **balanced** である．

$$x < max \cdot BestInterval$$

ただし，max は 1 以上の定数である．

削除の際には，まず最初に T から $deleteNumber$ アルゴリズム（表 1）を利用して節点を削除する．その後，挿入アルゴリズム（表 4）を利用して削除された箇所の間隔が定義 3 を満たすようにする．削除の際に挿入アルゴリズムを利用するときには，X 中の節点の数を 0 とする（つまり，$|X| = 0$）．

削除のための **balanced** の定義は，削除された後に発生した間隔が狭くなることを要求する．削除の後に発生した偏った間隔が挿入と同じアルゴリズムで均されるのは，許容する範囲である **balanced**（定義 2，3）を $BestInterval \times min$ より小さく，$BestInterval \times max$ より大きく定義しているからである．よって，拡張前置順列中の間隔の均衡が粗く保たれ，大規模なラベルの付け直しが避けられる．

4.2 近似ヒストグラムを用いた動的ラベル付け手法

より洗練された動的ラベル付け手法は，近似ヒストグラムを保持して実現する．もし挿入がある箇所に集中的に発生した場合には，その箇所の間隔を広くとっておけば，ラベルを付け直すコストが減らせると予想され，削除が集中的に起こった箇所は，間隔を狭めておけば同様にコストが減らせると予想される．更新操作の位置と回数を記録することによって，挿入が多く起こる箇所を広げ，削除が多く起こる箇所を狭めることができる．XML データ中の節点の数は変動するため，我々は近似手法を採用した．

近似ヒストグラム

拡張前置順列・拡張後置順列に対してそれぞれ独立にヒストグラムを作成し，更新操作の近似位置に対する更新回数を数える．それぞれのヒストグラムは大きさ $size$ の整数値配列であり，ヒストグラムに許す更新操作回数を決めるパラメータ $delta$ を持つ．もし XML データに対する更新操作が $delta$ 回を超えた場

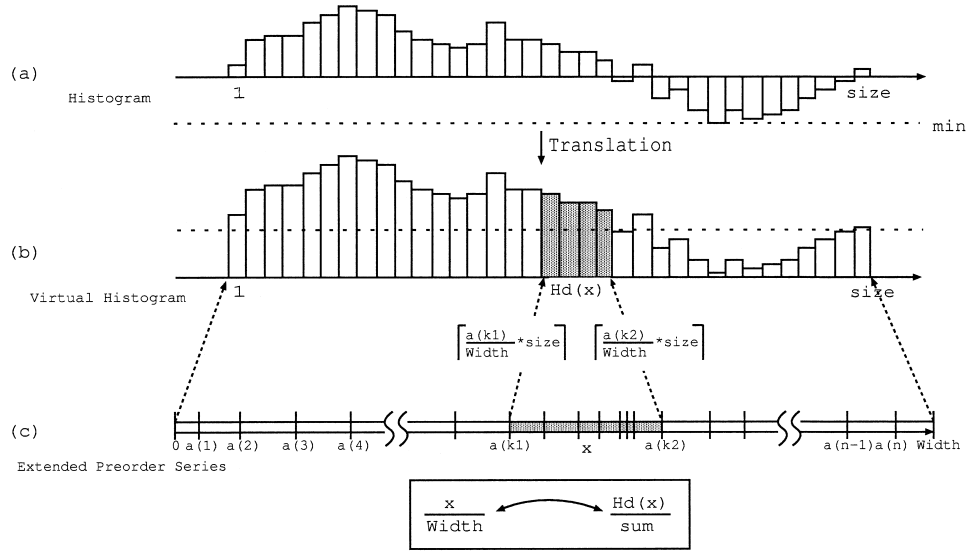

図 7 平行移動と近似対応
Fig. 7 Translation and approximated correspondence.

合には，ヒストグラム中のすべての値を消去して最初から新しい統計情報を数え直すことができる．H をヒストグラムとする．

$$H = (h_1, h_2, \cdots, h_{size})$$

最初は，更新操作が起きていないので，$h_i = 0 (1 \leq i \leq size)$ である．さらにパラメータとして，h_{min}，sum，$count$ を保持する．h_{min} は，ヒストグラム中の値の最小値であり，sum はヒストグラム中の値の和である．$count$ は更新操作の回数である．

4.2.1 ヒストグラムの更新

拡張前置順列を $(a_1, a_2, \cdots, a_n)$ とする．挿入操作が $a_k (1 \leq k \leq n)$ と a_{k+1} の間に起こったとすると，$0 < \frac{a_k}{Width} \leq 1$ であるので，

$$0 < \frac{a_k}{Width} \cdot size \leq size$$

であり，$p_k = \lceil \frac{a_k}{Width} \cdot size \rceil$ とし，h_{p_k} を位置 a_k に挿入が 1 回起こった記録として，1 加える．今回の手法では，更新操作単位 XML 木の節点数の情報はヒストグラムには反映させない．a_k から a_{k+m} までの削除の場合には，位置は挿入の場合と同様に決定され，削除の起こった記録として h_{p_k} から 1 を引く．sum は次のようにして更新される．

$$sum = \begin{cases} sum + 1 & \text{（挿入の場合）} \\ sum - 1 & \text{（削除の場合）} \end{cases}$$

$count$ は更新操作回数を単純に数えるだけである．sum と $count$ が更新操作が起こった際にヒストグラム中の値に実際にアクセスせずに更新できるのに対して，h_{min} はヒストグラム中の値にアクセスせずに正確な値を保つことはできない．そこで，ヒストグラム中の値をスキャンし h_{min} を更新するメソッドを用意する．

利用された計算機の資源に応じて，ヒストグラムを作成する際に $size$ を 1000 や，3000 等，$delta$ を 5000 等として設定できる．ヒストグラムのデータ構造は単純であり容量は小さく実装が容易である．XML データベースのサイズが変わったとしても，固定長のヒストグラムで更新操作の近似情報を記録できる．

4.2.2 ヒストグラムを用いたラベル付け直しアルゴリズムの改良

基本となるアイディアは，更新操作の分布の情報を拡張前置順列（あるいは拡張後置順列）のラベルの分布に反映させることである．2 つの基本的な方法で行う．

1 つは，ヒストグラム値の仮想的な**平行移動**である．削除操作の情報もヒストグラム中に保持するため，ヒストグラム値に 0 や負の値をとるものがある．そこで h_{min} の値を利用してヒストグラム中のすべての値を正値に平行移動する（**図 7** 中の (a) から (b)）．

$$(h_1, h_2, \cdots, h_{size}) \Longrightarrow$$
$$(h_1 - h_{min} + 1, h_2 - h_{min} + 1, \cdots, h_{size} - h_{min} + 1)$$

平行移動により，すべての値が正値をとる仮想的なヒストグラムを構成できる．sum は同様に $sum - size \times (h_{min} - 1)$ と仮想ヒストグラムの sum に変換できる．よって，ある位置 k における仮想ヒストグラムの全体中での割合を，h_p/sum と得る．明らかに，

$0 < h_p/sum < 1$ と $\sum_{p=1}^{size} h_p = sum$ が成り立っている．拡張数値順列中のラベル間の間隔の列を考えると，その間隔列を近似的にこの仮想的な正値ヒストグラムの分布に近付けるという操作は，挿入が多く起こる箇所を広げ，削除が多く起こる箇所を狭めるという操作に対応することが分かる．

もう 1 つは，ヒストグラムと拡張前置順列との**近似対応**である．拡張前置順列中の位置は，ヒストグラムを更新する際と同じ方法で対応付ける．つまり，ある節点 a_k のヒストグラム中で対応付けられた位置 p_k は，

$$p_k = \left\lceil \frac{a_k}{Width} \cdot size \right\rceil$$

となる．次に，拡張前置順列中の間隔 $x = a_{k2} - a_{k1}$ に対応するヒストグラム中の間隔 $Hd(x)$ を定義する．

$$Hd(x) := \sum_{k=k1}^{k2} h_{p_k}$$

これにより，次の近似対応をえることができる．

$$\frac{x}{Width} \Longleftrightarrow \frac{Hd(x)}{sum} \qquad \text{近似対応}$$

この近似対応は，拡張数値列中のある箇所の $Width$ に対する割合と，対応するヒストグラム中の箇所の sum に対する割合の対応関係を示している．この割合が同じくらいであれば，拡張前置順列のその箇所がヒストグラムの統計量の情報を反映していると考えることができる．

この近似対応を利用して，**balanced** を次のように再定義する．

定義 4 ［ヒストグラムを用いた balanced］

もし間隔 $x = a_{k2} - a_{k1}$ が次の条件を満たすとき，x は **balanced** であるとする．

$$\frac{Hd(x)}{sum} \cdot min < \frac{x}{Width} \quad \text{（挿入の場合）}$$
$$\frac{x}{Width} < \frac{Hd(x)}{sum} \cdot max \quad \text{（削除の場合）}$$

ただし，max は 1 以上の定数であり，min は 1 以下の定数である．

この条件は，間隔 x の拡張前置順列中の割合が，ヒストグラム中で対応する箇所のヒストグラム中での割合を反映しているかどうかを確認する．定義 2，3 との違いは，節点の個数ではなくラベルの位置情報で平衡状態を定義している点である．

表 4 の挿入アルゴリズムと同じアルゴリズムをヒストグラムを用いた手法においても，挿入と削除の場合両方とも定義 **balanced** を置き換えることによって利用している．

5. 実験結果

2 つの提案する動的ラベル付け手法を表 2 と同じデータセットを用いて評価した．動的範囲ラベル付け手法では更新操作ごとに周辺のラベルを小規模に付け直すことによって大規模なラベルの付け直しを回避する．更新操作の際にラベルが付け直された節点の個数を**コスト**とする．

今回，次の 2 つの評価項目を用いて評価した．

(1) 全ラベル付け直し（に匹敵するラベルの付け直し）が発生した回数
(2) 更新操作ごとのラベルの付け直しコストの平均値

全節点数の 80 %の節点のラベルの付け直しが起きた場合，平衡条件判定等のコストも考慮し，全ラベル付け直しと見なした．また，たとえ全ラベル付け直しが頻繁に起こらなくても，毎回の小規模なラベルの付け直しのコストが平均して高い場合は，更新操作の処理に時間がかかり結果として問合せを高速に処理できないためコストの平均値も評価項目として用いた．

30,000 回の更新操作を行った際に発生した全ラベルの付け直しのコストを**表 5** に示す．単位 XML 木の節点数が 2 のときには，簡単なラベル付け直し手法は全ラベル付け直しが 1 回も発生しなかった．10 と 30 のときには，全ラベル付け直しはしばしば継続的に発生した（このことは，図 9 と 10 に現れている）．ヒストグラムを用いた手法の場合は，単位 XML 木の節点数が 2 のときには，全ラベルの付け直しは静的手法よりも数多く発生したが，単位 XML 木の節点数が増加しても全ラベルの付け直しの回数はそれ程増えなかっ

表 5 全ラベルの付け直しの回数（平均）
Table 5 The time of bulk loads (average).

単位 XML 木の節点数	Static	Dynamic (Simple)	Dynamic (Histogram)
2	1.333	0	4.833333
10	8.5	1067.5	0
30	30	2233.167	0.166667
total	13	1100.222	1.665

表 6 安定したデータ（1:1）のコスト
Table 6 Stable data set (1:1).

	Simple	Histogram
Average	17	38
Maximum	1035	5110

表 7 成長するデータ（2:1）のコスト
Table 7 Growing data set (2:1).

	Simple	Histogram
Average	35336	327
Maximum	103118	95091

た．この結果から，必ずしもヒストグラムを用いた手法が良い結果を示すわけではないことが分かる．つまり，簡単な動的手法とヒストグラムを用いた手法それぞれに適した更新操作列のタイプがあると予想される．

そこで，テストデータセットを 2 つのタイプ：安定したデータ（1:1）と成長するデータ（2:1）に分けてラベルの付け直しコストの最大値と平均を分析した．安定したデータとは，更新操作列が同じくらいの頻度（1:1）の挿入と削除からなっているデータのことで，データ全体の大きさが，更新が続いていっても著しく増えたり減少したりしない．成長するデータは挿入の割合が削除よりも多く（2:1），更新が続くにつれ，大きさが増えるデータを指す．

安定したデータ（1:1）

表 6 は，安定したデータセットの際の毎回の更新操作ごとのラベルの付け直しコストの平均値と最大値を示している．簡単な動的手法は，ラベルの付け直しコストを低く抑えているが，ヒストグラムを用いた手法は抑えていない．すなわち，安定したデータセットの場合には簡単な手法はヒストグラムを用いた手法よりもコストを抑えることができるといえる．

成長するデータ（2:1）

表 7 は，成長するデータセットの結果を示している．簡単な手法が成長するデータを効率良く管理できていないのに対して，ヒストグラムを用いた手法は，ラベルの付け直しコストを大幅に下げている．データセットが成長するときには，ヒストグラムを用いた手法が簡単な手法よりも適しているといえる．

このようなデータの種類に対して手法の得手，不得手がある理由は次のように考えられる．提案した簡単な手法は，ヒストグラム中の値をすべて等しくした際のヒストグラムを用いた手法と見なすことができる．そうすると，安定したデータに関しては，理想的なヒストグラムを用いた手法と見なすことができる簡単な手法は効率良くコストを分散し，ラベルの付け直し回数を減らすことができるということになる．すなわち逆に考えると今回のヒストグラムを用いた手法は安定したデータの際には必ずしも理想的な統計情報を保持できていないということである．

上記 2 種類のデータセットに対する結果をより詳しく見るために，**図 8**，**図 9**，**図 10** に典型的なラベルの付け直しコストの結果のグラフを示す．グラフは，横軸が更新操作回数，縦軸が更新の際に発生した周辺のラベルの付け直し回数（コスト）である．図 8 は，挿入と削除の箇所の分布が同じ一様分布であり，比率が 2:1 であり，単位 XML 木の節点数は 30 の場合である．コストを平均すると，ヒストグラムを用いた手法は，簡単な手法よりも良く感じられるが，最大値を比較すると，ヒストグラムを用いた手法は簡単な手法の 10 倍のコストがかかっている．

正規分布の場合（図 9，10），ヒストグラムを用いた手法が，ラベルの付け直しコストを効率良く毎回の更新操作に分散させているの対して，簡単な手法のコストは莫大であり，続けて全ラベルの付け直しを発生させている．これは，ラベル番号の分布が更新操作が頻繁に起こる箇所に集中してしまったためであると考えられる．ヒストグラムを用いた手法ではこの状況を回避し，平均と最大コストを下げることができる．

6. 議　　論

6.1 永 続 性

動的範囲ラベル付け手法を用いると，節点のラベルは**永続性**を満たさなくなる．文書のヴァージョニングには，ヴァージョンの違いを区別するために，ラベルの**永続性**が要求される[9]．しかしながら，近年データよりの XML がインターネット上に登場し，我々はこうしたデータに対して問合せおよび更新を行う必要がある．さらに，データの格納領域が限られているときには，更新の頻繁に起こる XML データを利用可能な領域内で管理する必要性もあると考えられる．動的範囲ラベル付け手法はラベルの**永続性**を放棄するかわりに連続的な更新操作を可能にし，**構造結合**を利用した高速な問合せ処理を実現できる．どうしても**永続性**が必要となりかつ更新が頻繁に起こる場合には，別に永続性を満たすラベルを用意することも考えられる．

6.2 索　　引

構造結合は，節点の名前，値や与えられたラベルに構築された索引を利用して実現される．たとえば，B+木，R 木やその他新たな索引も開発されている[7),8),14)]．これらの索引において重要なことは，前置順と後置順の順序関係である．**動的範囲ラベル付け手法**は，節点のラベル間の順序関係を維持するため，これらの構

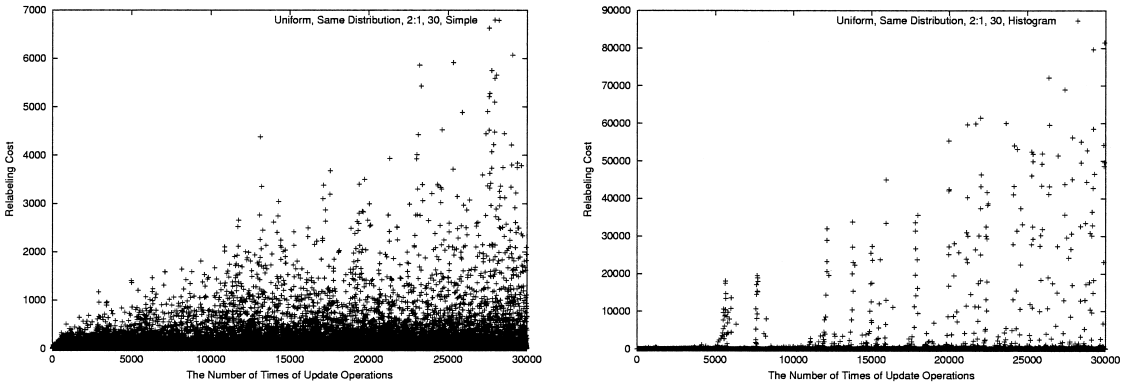

図 8 簡単な手法がヒストグラムを用いた手法より優れている場合（一様分布，2:1，挿入と削除が同じ分布）
Fig. 8 Simple scheme superior to histogram-based scheme (uniform, growing, same distribution).

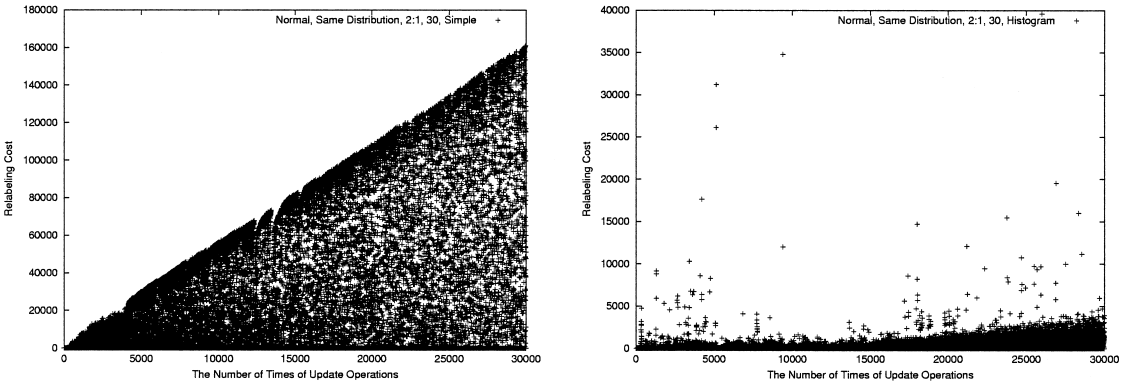

図 9 ヒストグラムを用いた手法が簡単な手法より優れる場合（正規，2:1，挿入と削除が同じ分布）
Fig. 9 Histogram-based scheme superior to simple scheme (normal, growing, same distribution).

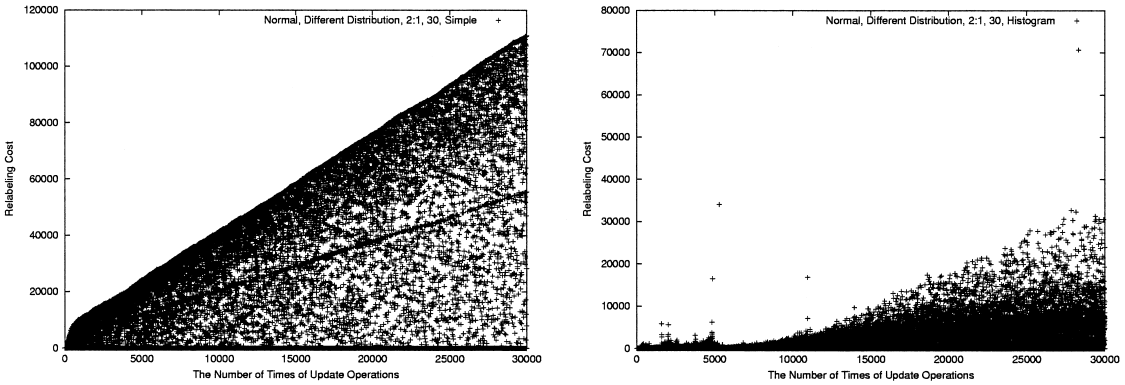

図 10 ヒストグラムを用いた手法が簡単な手法より優れる場合（正規，2:1，挿入と削除が違う分布）
Fig. 10 Histogram-based scheme superior to simple scheme (normal, growing, different distribution).

造結合用の索引を更新するコストも低いと予想される．しかしながら，詳細は未解決であり，今後の課題である．

7. まとめと今後の課題

今日 XML はその役割を広げ，様々な環境でデータを交換するために利用されている．文書よりの XML では，参照できる永続的なラベルが要求されるが，デー

タよりの XML では，高速に問合せおよび更新を処理できる XML データベースが必要である．

本論文では，データよりの動的 XML 木のために 2 つの**動的範囲ラベル付け手法**を提案した．1 つは，簡単な小規模なラベルの付け直し手法であり，もう 1 つは更新操作の統計情報を保持する近似ヒストグラムを用いてラベルを付け直すより洗練された方法である．実験結果により，

(1) 簡単な動的範囲ラベル付け手法は，安定したデータセット（挿入と削除が同じくらいの割合で起こる）のラベルを効率良く管理することができる．

(2) ヒストグラムを用いた動的範囲ラベル付け手法は，成長するデータセット（挿入が削除よりも頻繁に起こる）のラベルを効率良く管理することができる．

固定長のデータ型で表現されたラベルを更新に対して効率良く管理することによって，descendant 軸を含む XML 問合せは**構造結合**（*structual joins*）を利用して高速に処理することができる．

本研究の今後の課題としては，

(1) パラメータの調整
実験において，ラベルの付け直しコストがあまり低くならない場合があったが，パラメータを調整することによってコストの振舞いは少し変わった．最適なパラメータの選択はコストを下げるために必要である．

(2) ラベル管理機構
今回は，単純に更新操作ごとに，ラベルを付け直すことを考えたが，実際は，更新操作とラベルの付け直しは独立して行うことが可能である．つまり，システムの負荷情報等から判断してラベルを付け直すことができる．こうした一連のラベル管理機構を構築することが望まれる．

をあげる．

謝辞 本研究をすすめるにあたり，多くのご意見と助言をいただいた，奈良先端科学技術大学院大学情報科学研究科植村研究室の皆様に深く感謝いたします．

参 考 文 献

1) Abiteboul, S., Kaplan, H. and Milo, T.: Compact Labeling schemes for ancestor queries, *Proc. SODA* (2001).
2) Aho, A.V., Hopcroft, J.E. and Ullman, J.D.: *The Design and Analysis of Computer Algorithms*, Addison-Wesley (1974).
3) Al-Khalifa, S., Jagadish, H.V., Koudas, N., Patel, J.M., Srivastava, D. and Wu, Y.: Structural Joins: A Primitive for Efficient XML Query Pattern Matching, *Proc. ICDE* (2002).
4) Boag, S., Chamberlin, D., Fernandez, M.F., Florescu, D., Robie, J. and Sime'on, J.: XQuery 1.0: An XML Query Language. http://www.w3.org/TR/xquery/ W3C Working Draft 15 November (2002).
5) Brownell, D.: Simple API for XML (SAX). http://www.saxproject.org/
6) Bruno, N., Gravano, L., Koudas, N. and Srivastava, D.: Navigation- vs. Index-Based XML Multi-Query Processing, *Proc. ICDE* (2003).
7) Bruno, N., Koudas, N. and Srivastava, D.: Holistic Twig Joins: Optimal XML Pattern Matching, *Proc. SIGMOD* (2002).
8) Chien, S-Y., Vagena, Z., Zhang, D. and Tsotras, V.J.: Efficient Structural Joins on Indexed XML Documents, *Proc. VLDB* (2002).
9) Chien, S-Y., Zaniolo, C., Tsotras, V.J. and Zhang, D.: Storing and Querying Multiversion XML Documents using Durable Node Numbers, *Proc. WISE* (2001).
10) Choi, B., Mahoui, M. and Wood, D.: On the Optimality of Holistic Algorithms for Twig Queries, *Proc. DEXA* (2003).
11) Cohen, E., Kaplan, H. and Milo, T.: Labeling Dynamic XML Trees, *Proc. PODS* (2002).
12) Deutsch, A., Fernandez, M., Florescu, D., Levy, A. and Suciu, D.: XML-QL: A Query Language for XML. http://www.w3.org/TR/1998/ NOTE-xml-ql-19980819/
13) Gene Ontology Consortium. http://www.geneontology.org/
14) Haifeng, J., Lu, H., Wei, W. and Ooi, B.C.: XR-Tree: Indexing XML Data for Efficient Structural Join, *Proc. ICDE* (2003).
15) Le Hegaret, P., Whitmer, R. and Wood, L.: Document Object Model (DOM). http://www.w3.org/DOM/
16) Jiang, H., Wang, W. and Lu, H.: Holistic Twig Joins on Indexed XML Documents, *Proc. VLDB* (2003).
17) Robie, J.: XQL FAQ. http://www.ibiblio.org/xql/
18) Kaplan, H., Milo, T. and Shabo, R.: A comparison of labeling schemes for ancestor queries, *Proc. SODA* (2002).
19) Li, Q. and Moon, B.: Indexing and Querying XML Data for Regular Path Expressions, *The VLDB Journal* (2001).
20) Wei, W., Haifeng, J., Lu, H. and Yu, J.X.: PBiTree Coding and Efficient Processing of

Containment Join, *Proc. ICDE* (2003).
21) Zhang, C., Naughton, J.F., DeWitt, D.J., Luo, Q. and Lohman, G.M.: On Supporting Containment Queries in Relational Database Management Systems, *Proc. SIGMOD* (2001).
22) 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, No.1 (2002).
23) 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).
24) 江田毅晴，天笠俊之，吉川正俊，植村俊亮：XML のための動的範囲ラベル付け手法：その評価および XRel への適用について，情報処理学会研究報告，データベースシステム DBS-129/BCC-4 (2003).

(平成 15 年 9 月 20 日受付)
(平成 16 年 2 月 2 日採録)

（担当編集委員　福田 剛志）

江田　毅晴（正会員）

2003 年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修了．現在，NTT サイバースペース研究所に所属．データベースに関する研究開発に従事．

櫻井　保志（正会員）

1991 年同志社大学工学部電気工学科卒業．同年日本電信電話株式会社入社．1996 年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修了．1999 年同大学院博士後期課程修了．工学博士．現在，NTT サイバースペース研究所に所属．2004 年よりカーネギーメロン大学客員研究員（2005 年までを予定）．索引技術，情報検索に関する研究開発に従事．

天笠　俊之（正会員）

1994 年群馬大学工学部情報工学科卒業．1999 年同大学大学院工学研究科修了．博士（工学）．同年から奈良先端科学技術大学院大学情報科学研究科助手．XML データベース，装着型コンピュータにおけるデータベース応用等の研究に従事．電子情報通信学会，ACM，IEEE Computer Society，日本データベース学会各会員．

吉川　正俊（正会員）

1980 年京都大学工学部情報工学科卒業．1985 年同大学大学院工学研究科博士後期課程修了．工学博士．京都産業大学，奈良先端科学技術大学院大学を経て，2002 年から名古屋大学情報連携基盤センター教授．The VLDB Journal および Information Systems（Elsevier/Pergamon）の編集委員．XML データベース，多次元空間索引等の研究に従事．

植村　俊亮（フェロー）

1964 年京都大学工学部電子工学科卒業．1966 年同大学大学院工学研究科修士課程修了．同年電気試験所（産業技術総合研究所）．1970 年マサチューセッツ工科大学電子システム研究所客員研究員，1981 年電総研ソフトウェア部プログラム研究室長，1988 年東京農工大学教授を経て，1993 年から奈良先端科学技術大学院大学情報科学研究科教授．データ工学，データベースシステムの研究に従事．工学博士．IEEE Fellow，電子情報通信学会フェロー．現在，情報処理学会理事，日本情報考古学会理事，データベース振興センター評議員等．