データベースシステム 129－16
（２００３．１．２４）

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

江田 毅晴† 天笠 俊之† 吉川 正俊†† 植村 俊亮†

†奈良先端科学技術大学院大学情報科学研究科 〒630–0101 奈良県生駒市高山町 8916-5
††名古屋大学 情報連携基盤センター 〒464–8601 名古屋市千種区不老町

あらまし XML木のための動的範囲ラベル付け手法の評価とXRelへの適用について述べる. 以前我々は, XML問合せを高速化する範囲ラベル付け手法において, XMLデータに更新が起こった際に, 更新箇所周辺のラベルを局所的に付け直す, 動的範囲ラベル付け手法を提案した. 今回は, そのアルゴリズムについて実験をし, 提案手法が本当に大規模なラベルの付け直しを回避することができるのか評価した. また動的範囲ラベル付け手法の一つの実装として, XMLデータの関係データベースへの格納方法であるXRelへ適用することについて考察する. XRelでは, データを格納する際にファイル中での先頭からのバイト数の情報を用いていたため更新が難しかった. 動的範囲ラベル付け手法を用いることによって, 更新の際にラベルを付け直す箇所を少なくすることができる. また, 問合せ言語などによる更新を行う際に, 最新のXMLファイルを用意する必要がいらない結果構成アルゴリズムについて説明する.

Dynamic Range Labeling for XML: Evaluation and an Application to XRel

Takeharu EDA†, Toshiyuki AMAGASA†, Masatoshi YOSHIKAWA††, and Shunsuke UEMURA†

†Graduate School of Information Science, Nara Institute of Science and Technology 8916-5 Takayama, Ikoma, Nara 630–0101, Japan
††Information Technology Center Nagoya University Furo-cho, Chikusa-ku, Nagoya 464–8601, Japan

Abstract This paper describes an evaluation and an application to XRel of Dynamic Range Labeling for XML. We proposed Dynamic Range Labeling, in which labels around an update point are re-labeled locally in order to cope with updates to XML data. Some experimental results on the algorithms are reported. As one implementation of Dynamic Range Labeling, we discuss an application to XRel which is a novel method to store and retrieve XML data in RDB. In XRel, bytes in an XML file, which are kept in tables, makes updates difficult. By applying Dynamic Range Labeling to XRel, re-labeling times are reduced when updates occurs. We also describe the algorithm which reconstructs a result XML sub-tree without an original XML file.

1 はじめに

XMLがデータ交換形式としての確固たる地位を築くにつれ, インターネット上には大量のXMLデータがみられるようになった. 例えば, すべての真核生物に適用できるような統制された語彙体系の構築を目指しているGene Ontology Consortium[6]では, その語彙データをテキスト形式だけではなく, XML形式でも公開している.

しかし, Gene Ontology Consortium のテキスト形式のデータが日に何十箇所も更新されるのに対して, XML 形式のデータが更新されるのは, 月に一度だけである. これは, 数メガバイトに及ぶ XML データの更新管理の難しさに起因するものである. このように, XML の利用される場が増えるとともに, 単に XML としてデータを作成するだけでなく, それらのデータを更新をしつつ管理したいという要求がでてきているのである.

大量の XML データ, あるいは非常に大きな XML データをディスクに格納して管理する方法として, ネィティブ XML データベース [3, 14] が各種開発されている. また研究分野でも, ディスクに格納された XML データに対して, XPath[4] 等の包含問合せを高速化する手法が, 種々研究されている.

そうした XML 問合せを高速化する手法の一つとして, 範囲ラベル付け手法 (Range Labeling) [5, 17] を利用した構造結合 (Structural Join) [1, 8, 12] がある. 範囲ラベル付け手法によって付加したラベルだけで節点間の先祖子孫関係が判定可能であるため, "Book//author" や, また level を用いることによって "Book/*/person" などの正則経路式を含む問合せを高速に処理することができる.

しかしながら, 範囲ラベル付け手法は, 節点をラベル付けする際にラベル間に間隔を空けたとしても, XML データに対する更新操作を繰り返すうちに間隔を使い尽くしてしまうことが予想され, 頻繁に更新の起こる XML データのラベル付けには向かないとされている [9]. 今回, テストデータではあるが, そのような間隔を使い尽くす状況があることを実験にて確認した.

そこで, 我々は以前, 範囲ラベル付け手法のラベルを更新操作の際に局所的に付け直す, 動的範囲ラベル付け手法 [16] を提案した. 更新操作の際に, 更新箇所の周辺のラベルも小規模に付け直すことにより, 大規模なラベルの付け直しを避けながら, ラベル間の順序関係を維持することができ, 構造結合などによる効率の良い XML 問い合わせ処理を提供することが可能である.

実験結果により, 提案手法を利用すると, XML データの更新箇所が一様なときと, 更新箇所に偏りがある場合でも挿入と削除の割合が同じときには, 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.

そこで, 動的範囲ラベル付け手法の一つの応用として, 関係データベースへの XML データの格納法である XRel へ適用することにした. これによって, XML データに更新が起こった際の大規模なラベルの付け直しを避けることが可能となり, また XRel の効率の良い問合せ処理を提供することができる. また, 将来的な問合せ言語等による更新操作を見据えて, 問合せ結果部分 XML 木をデータベース内部の情報のみで構成する方法も構築した.

本稿は, まず動的範囲ラベル付け手法について説明し, その更新アルゴリズムの数値的評価について報告する. 次に XRel への適用について議論する. 最後にまとめと今後の課題を述べるが, その前に関連研究についても触れる.

2 動的範囲ラベル付け手法

動的範囲ラベル付け手法では, まず静的にもともとの XML 木にラベルを付加し, 更新操作が起こる度に, 挿入 XML 木にはラベルを付加し, 削除される XML 木はラベル毎削除する. その際に, ラベルの整合性を保つように周辺のラベルもつけ直す. 順を追って説明する.

2.1 静的なラベル付け

木構造における前置順と後置順の順序を保存した二つの数値の対[1]で XML 木中の節点をラベル付けする (図 1). これらをそれぞれ, 拡張前置順, 拡張後置順と呼ぶ. このようにラベル付けされた XML 木の節点に対して次のことが成り立つ.

I. *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

$$a_i < a_k \text{かつ} b_j > b_l$$

[1] 今回は整数値でラベル付けしているため, 前置順と後置順に対してそれぞれ狭義単調増加整数列となるような数値の対になる

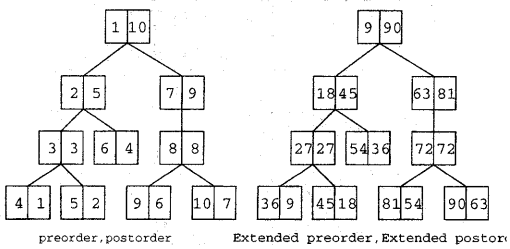

図 1: 前置順, 後置順及び拡張前置順, 拡張後置順の例

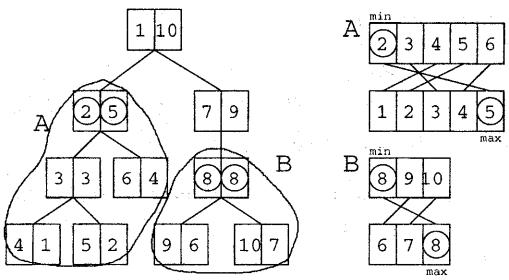

図 2: II, III

II. （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 (図 2)

III. 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 (図 2)

こうした性質を持つラベルを節点に付加した後, 分解された節点リストから, 下記のようなことが実現できる.

1. ラベルが唯一であることにより, 問合せ言語内での節点同一性をラベルの比較だけで判定できる
2. 先祖子孫 (親子) 関係をラベルの比較だけで判定できるため, 巨大な XML データに対して正則経路式を評価する際に, 索引を構築することによってデータの全検索を避けることができる
3. 部分木の削除をラベルの情報を基に行うことができる [17]
4. ある節点を根とする部分木を, ラベルを用いて分解された節点リストから再構築できる

4. の性質は, 範囲ラベル付け手法においては直観的に明らかであるが, この性質が自明でない, あるいは簡単ではないラベル付け手法も考えられる. 巨大な XML データを更新する際には, 問合せ言語を用いて行うことが想定され, 最新のデータは, データベースの中にしか存在しない可能性が高い. その場合, 問合せの結果を構成するためにデータベース外の情報, (例えば最新の XML ファイル等) を保ち続けるのは困難である. 更新の多い動的な XML データを管理するには, ラベル付け手法によって, 4. の性質を実現できる必要がある. 今回, XRel へ動的範囲ラベル付け手法を適用する際に, 4. の性質を利用した部分木構成アルゴリズムについても述べる.

2.2 更新操作への対応

III の性質により, XML 木の挿入, 削除に対して, ラベルの連続した番号列の “一箇所に対する挿入” 及び, “連続した番号列の削除” を考えればよい. 図 1 の右側のように間隔をあけてラベルをつけたため, 間隔が充分に空いている場合には, そのまま番号列を挿入することによって, XML 木の挿入とすることができる. しかしながら一旦固定したラベルを与えてしまうと, 大きな XML 木の挿入が起こった場合や, 挿入操作が連続し間隔がつまってしまった場合には挿入操作を続けることはできなくなる [17].

そこで, なるべく大規模なラベルの付け直しを避けるため, 更新操作の度に周辺のラベルを小規模に均すことにする. 均し方の手順は, ラベルに使える幅を節点数で割った状態を最も良い状態 $BestInterval$ として, $0 < min \leq 1, 1 \leq max$ としたとき, 注目箇所の幅 x が, 挿入操作のときは,

$$min \cdot BestInterval < x$$

削除のときには,

$$x < max \cdot BestInterval$$

となるとき平衡状態とし, 挿入及び削除がおこった箇所の周辺が平衡状態になるまで前置順及び

後置順にラベルの番号を付け直すというものである.[2]

3 動的範囲ラベル付け手法の数値的評価

XML データを単純に数値のラベルの列とみなして挿入, 削除の操作列を作成し, 操作毎のラベルの付け直し回数を数える. これにより, 実際に動的範囲ラベル付け手法を実装した際の, 更新に対するその効果を予測することができる.

3.1 テストデータセットについて

提案手法は複数の XML データがある場合には, 1) 仮想的な根節点をつけて大きな木を作成する, 2) それぞれのデータ毎に別の ID を付加する, の二とおりの方法でラベル付けを同様に行うことができる. 2) の手法を考えるのは容易であるので, 今回は簡単のため, XML データは一つの木であるとして議論する.

各種 XML データベースベンチマーク [10, 13] の研究をみてみると, 挿入・削除の基本単位は, XML の一つのファイルそのものか, 一つの葉節点及びその連続であり, XML 木の部分木を単位としたものではない. 部分木単位の更新操作を一つの節点の更新操作の連続に置き換えることは可能であるが, 先読みをしていない範囲ラベル付け手法において, この置換えは本質的な範囲の無駄を生む. また, 提案手法は挿入・削除の単位を部分 XML 木として構築しており, 現実的な XML データの更新を考えても部分 XML 木単位でのデータ更新は必須と考えられる.

以上の議論をふまえ, 今回, 評価のための更新操作列を表 1 のようないくつかの場合について調べた. 正規分布は, 全体の 2 割の領域にほとんどの操作が起きるようにした. また, 更新操作回数は 3 万回であり, ラベルには 30 ビットの整数値を用いた. 初期節点数は 1 万である.

3.2 比較対象について

ラベルの付け直し計画については,

[2] [16][17] を参照

分布	一様分布 一か所に正規分布
(挿入):(削除)	1:1 1:0 2:1
操作単位の 節点数 (max)	2 10

表 1: テストデータセット

1. 動的範囲ラベル付け手法 (ExtendedNumberList)
2. 間隔が詰まるまで挿入し続け, 詰まったら, すべての節点のラベルを付け直す. (AllRelabelingList)
3. 動的範囲ラベル付けの max 条件無し (NoMaxList)

の三手法について比較した. 2. を調べることによって, 与えられたデータセットに対して, 間隔を空けるだけでは, 大規模なラベルの付け直しが発生する状況があることを確認できる. 3. は, 動的範囲ラベル付けの兄弟と考えられるが, max 条件が本当に効果があるか調べる必要があるため, 比較対象にいれた.

3.3 結果

3.3.1 AllRelabelingList

AllRelabelingList では, 間隔が空いている限りは, 間隔を節点数で等分割して挿入し続ける. しかし空いていない場合には, すべての節点のラベルをつけ直す. この手法では, 毎回のラベルのつけ直し回数はほとんど 0 になるが, たまにバルクロードが起こる. データベースにおけるバルクロードは非常にコストがかかる. 3 万回の更新操作で起きたバルクロードの回数が多いほど, 提案手法の存在意義がある.

結果は, 一様分布の場合は, 操作単位の節点数が 2 個以下ではほとんどバルクロードは起こらなかった. しかし 10 個以下では, 5 回以上バルクロードが起きた. 正規分布にいたっては, 2 個以下でも 3 回以上バルクロードが起きており,

分布	挿入:削除	節点数	平均	最大値
一様	1:1	2	2.5	120
一様	1:1	10	15.6	780
一様	1:0	10	32.9	1142
一様	2:1	10	23.8	898
正規	1:1	2	4.0	180
正規	1:1	10	16.9	768

表 2: 提案手法のラベルの付け直し回数

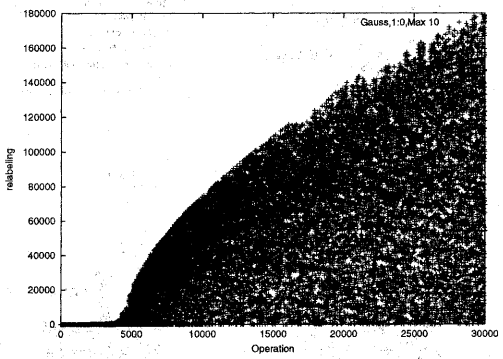

図 3: 更新操作の偏りにより間隔の均衡がくずれた例

10 個以下については, 15 回ほどバルクロードが起きている.

3.3.2 ExtendedNumberList

提案手法のアルゴリズムにそってラベルをつけ直すと, 表 2 のような結果になった. 一様分布の場合は, 少ないラベルの付け直し回数を保っている. 正規分布の場合には, (挿入:削除)=1:1 のときには, 表 2 のように, 現実的な付け直し回数だったが, 2:1 や 1:0 のときには, 図 3(横軸は操作回数, 縦軸はラベルの付け直し回数) のように最初はうまくいっているように見えるが, 途中から操作回数に比例して付け直し回数が増えていく. これは, 操作が集中している箇所の間隔が最小の間隔で詰まってしまい, 挿入操作の度に, 大規模にラベルを付け直しているためと考えられる.

	分布	節点数	平均	最大値
NoMaxList	一様	10	16.9	1020
ExtendedNumberList	一様	10	15.6	780
NoMaxList	正規	10	19.2	788
ExtendedNumberList	正規	10	17.0	768

表 3: (挿入):(削除)=1:1 のとき

3.3.3 NoMaxList

max 条件を外しても, 基本的な傾向は, ExtendedNumberList と変わらなかった. 挿入と削除の比率が同じ場合には, わずかながら, ExtendedNumberList のほうが良い結果を示した (表 3). 挿入操作の比率が削除より高い場合は, ExtendedNumberList とほぼ同じ回数になる. これは, 削除のときにしか使われない max 条件の適用される回数が少ないためである.

3.3.4 結果のまとめ

上記より, 動的範囲ラベル付け手法は更新操作の起こる箇所が一様の場合には, 大規模なラベルのつけ直しを避けられることが分かった. 更新操作が起こる箇所が偏っている場合でも, 挿入と削除が同じくらいの割合で起こるときには, 大規模なラベルのつけ直しを避けることができる. しかしながら, 挿入の割合が削除に比べて多い場合には, 大規模なラベルのつけ直しを引き起こすことも分かった. 現実的には更新操作の起こる箇所は偏る可能性が高く, また挿入が多い状況も充分考えられるので, 更新操作箇所の統計情報や XML の情報を利用して手法を改良する必要がある.

4 動的範囲ラベル付け手法の XRel への適用について

XRel[15] は, XML データを関係データベースに格納する一手法であり, その特徴として, XML 木における根からのパスと XML ファイル中での先頭からのバイト数で開始と終了を表したリージョンを各節点に付与している. パスをテーブルに格納することにより, XPath による問合せを高速に評価することができる. パス

だけではXML木の構造を保存することができないため, リージョンを用いている.

しかしながら, リージョンはXMLデータに更新が起きるとファイル中の更新箇所以降のすべての節点について書き換えられなければならない. この操作は非常にコストがかかるため, 各種リージョンの改良手法が提案されてきた [11, 2].

Kha等 [11] は, ファイルの先頭からのバイト数ではなく, 親節点から数えたバイト数を用いることによって, 大幅な更新箇所数の削減を達成した. Amagasa等 [2] は, 挿入が起きた際に, 浮動小数点を用いて新しい節点のリージョンを表現することによって, ある段階までは, リージョンを書き換える必要がない手法を提案した. この二つはいずれも, 最新のXMLファイルの存在を仮定して, そのファイルから問合せの結果をリージョンを用いて切り出す手法である.

しかし, XMLデータベースが更新される状況というのは, 必ずしも最新のファイルが入手可能なわけではない. 非常に大きなXMLデータを扱うときには, 利用者はXMLデータベースに対して何らかの更新操作を記述可能な言語を用いて, XML木の挿入や削除を行う. この過程において, XMLファイルはどこにも登場しない. つまり, ファイルのバイト数を利用することもできないし, そのような外部ファイルを同時に構築することも現実的ではない.

範囲ラベル付け手法は, XML木の構造情報を保持するのに充分な手法である. 今回我々は, 動的範囲ラベル付け手法をXRelのリージョンの代わりに利用することを考えた. これによって, 大規模なラベルの付け直しを避けながら, XRelを更新可能にする. また後述する結果部分木構成アルゴリズムによって, 一次記憶に収まらないような巨大なXMLデータを問合せ言語等で直接データベースに更新を行った際にも, 最新のXMLファイル無しで結果を構成することができる.

4.1 テーブルの変更点

テーブルの変更点としては, リージョンの代わりに, 拡張前置順 (epre) と拡張後置順 (epost) を用いて, 要素テーブルには要素名, 属性テーブルには, 属性名も持つようにする. 変更点はこれだけである (図4).

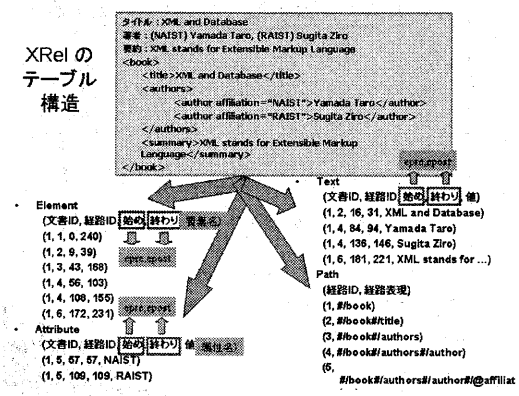

図4: XRelテーブルの変更点

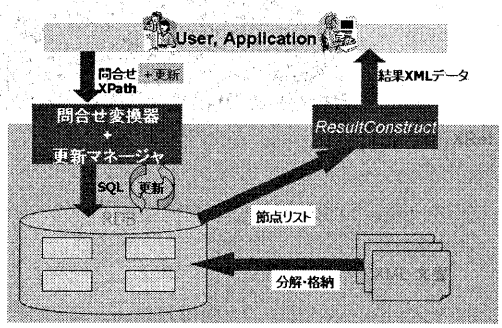

図5: システム

4.2 システムの変更点

変更後のシステムは, 図5のようになる. おおもとのXMLファイルをバルクロードした後は, 利用者は, 更新操作を記述可能な問合せ言語等を用いて論理的なXML木に更新をおこなう. 動的範囲ラベル付け手法を適用したXRelでは, 最新のXMLファイルは必要無い. テーブルにXMLデータの情報をすべて格納し, テーブルの中の情報だけで, 結果XML木を構成する. 結果XML木再構成アルゴリズムを4.3節で説明する.

また, XPath問合せのSQLへの変換アルゴリ

ズムも position() 関数以外は基本的に変更点は無い. position() 関数の評価は, XPath の grouping による問合せなども考えると, もともとの XRel で採用されている親節点からみた順番をテーブルに格納する手法というのは, 必ずしも効果的とは言えないので, 今回は, すべて一度結果を構成して順番を数える方法をとることにした.

4.3 アルゴリズム *ResultConstruct*

リージョンを使っていないため, テーブルに格納された情報から結果を構築する必要がある. これから説明するアルゴリズム *ResultConstruct* は, スタックを用いて, SQL 問合せの結果である節点リストを一回走査するだけで木構造を復元する.

XRel の問合せ処理の枠組を利用すると, XPath による問合せを SQL 問合せに変換し, SQL 問合せの結果として, 結果部分 XML 木の根節点の拡張前置順と拡張後置順が返ってくる. 今, 根節点の拡張前置順, 拡張後置順を (a_i, b_j) とすると, XRel のそれぞれのテーブルに対して,

select * from Element where epre>= a_i and epost <= b_j order by epre

のような問合せをかけ, それらを拡張前置順でマージすることによって, 結果部分 XML 木に含まれるすべての節点のリストを拡張前置順にソートされた状態で得ることができる.

前置順にソートされているので, 節点リストを一回走査することは, 結果 XML 木を前置順に走査していることにあたる. アルゴリズムは, 根節点からスタートし, 次の節点が前の節点の子節点か判定しながら動く. 子節点の場合には, 子節点として木をつくり, 拡張後置順をスタックにつんで次の節点にいく. 子節点でない場合には, スタックに格納された結果木の根から現在の節点までの経路上の節点の後置順を順次取り出すことによって, どの節点の子供なのか判定することができる. 子節点になるまでスタックから値をとりだし, 子節点になった場合には, 再び同じ手続きを繰り返す.

ResultConstruct により, DOM 木のような木構造を復元することができる. また, XML 形式の開始タグと閉じタグを含むテキストを出力するアルゴリズムも, スタックに拡張後置順だけでなく要素の名前も格納することにより実現できる. 4 行目の **while** ループをスタックが空になるまでまわして, 子供と決まったときに開始タグ, 属性, あるいはテキストの値を出力し, スタックから値を取り出すときに閉じタグを出力する.

Algorithm *ResultConstruct*
入力 :preorder soted nodes list *PreList*
出力 :result XML sub tree
1 x is a first node in PreList;
2 *now* is the current node and the root of a result XML tree;
3 push x.epost in *Stack*;
4 **while**(true)
5 **if**(x.epost < top of *Stack*)
6 x is child of *now*;
7 *now* is x;
8 **if**(x doesn't have next)
9 **exit**;
10 **end if**
11 push x.epost in Stack;
12 x = x.next;
13 **else**
14 pop *Stack*;
15 *now* is parent of *now*;
16 **end if**
17 **end while**

---

5 関連研究

昨今, 範囲ラベル付け (Range Labeling) に基づく構造結合 (Structural Join) アルゴリズムの改良は特に研究されている [8, 12]. [8] では, XR-tree という索引を構築し, 構造結合にあらわれない余分な先祖と子孫を回避している. [12] では, 問題設定を単一の問合せから複数の問合せに拡張し, 問合せ共有をおこなって冗長な問合せ処理を回避している.

また, ネイティブ XML データベースの研究ではミシガン大学の TIMBER Project が有名である. 彼らは [7] の中で, 局所的なラベルの付け直しについて触れているが, 未解決の問題であるとしている. 本研究の成果は, その問題に対する一つの解である.

6 まとめと今後の課題

本稿では, 動的範囲ラベル付け手法の評価について報告し, またその一つの実装として, XRelへの適用について議論した.

実験結果により,

- 更新操作箇所の分布が一様な場合
- 分布が偏った場合でも, 挿入と削除の割合が同じくらいの場合

には, 提案手法により大規模なラベルの付け直しを回避できることが分かった. しかし, 分布が偏り, かつ挿入の割合が削除より大きい場合には, 大規模なラベルの付け直しを引き起こす.

また, XRelへの動的範囲ラベル付け手法の適用を議論することによって, 問合せ言語等によって巨大なXMLデータを管理する必要があるときには, 効果を発揮する可能性があることが分かった.

今後の課題としては, スキーマ情報や, XMLに対する更新操作の統計情報を範囲設計(Range Design)に役立てて, ラベルのつけ直し回数を減らしていくことがあげられる. また, XRelへの適用に関しては現在実装中である.

謝辞

本研究の一部は, 文部科学省科学技術研究費補助金(課題番号: 12680417, 13780236, 14019064), ならびに科学技術振興事業団(JST)の戦略的基礎研究推進事業(CREST)「高度メディア社会の生活情報技術」プログラムの支援によるものである. またDao Dinh Kha氏には, 日頃から本研究に関して有意義な議論をして頂いた. ここに記して謝意を示す.

参考文献

[1] Shurug Al-Khalifa, H. V. Jagadish, Nick Koudas, Jignesh M. Patel, Divesh Srivastava, and Yuqing Wu. Structural Joins: A Primitive for Efficient XML Query Pattern Matching. In *Proc. ICDE*, 2002.

[2] Toshiyuki Amagasa, Masatoshi Yoshikawa, and Shunsuke Uemura. QRS: A Robust Numbering Scheme for XML Documents. In *Proc. ICDE*, 2003. To appear.

[3] Apache Software Foundation. Xindice1.0 (Zeen-dee-chay). http://xml.apache.org/xindice/.

[4] James Clark and Steve DeRose. XML Path Language(XPath) Version 1.0. http://www.w3.org/TR/xpath.

[5] Edith Cohen, Haim Kaplan, and Tova Milo. Labeling Dynamic XML Trees. In *Proc. PODS*, 2002.

[6] Gene Ontology Consortium. http://www.geneontology.org.

[7] H. V. Jagadish and Shurug Al-Khalifa and Adriane Chapman and Laks V. S. Lakshmanan and Andrew Nierman and Stelios Paparizos and Jignesh M. Patel and Divesh srivastava and Nuwee Wiwatwattana and Yuqing Wu and Cong Yu. TIMBER: A Native XML Database. In *The VLDB Journal*, 2003. To appear.

[8] Jiang Haifeng, Hongjun Lu, Wang Wei, and Beng Chin Ooi. XR-Tree: Indexing XML Data for Efficient Structural Join . In *Proc. ICDE*, 2003. To appear.

[9] Haim Kaplan and Tova Milo and Ronen Shabo. A comparison of labeling schemes for ancestor queries. In *Proc. SODA*, 2002.

[10] Kanda Runapongsa and Jignesh M. Patel and H.V. Jagadish and Shurug Al-Khalifa. The Michigan Benchmark: Towards XML Query Performance Diagonostics, 2002. http://www.eecs.umich.edu/db/mbench/bmLongV.pdf.

[11] Dao Dinh Kha, Masatoshi Yoshikawa, and Shunsuke Uemura. An XML Indexing structure with Relative Region Coordinate. In *Proc. ICDE*, 2001.

[12] Nicolas Bruno and Luis Gravano and Nick Koudas and Divesh Srivastava. Navigation- vs. Index-Based XML Multi-Query Processing. In *Proc. ICDE*, 2003. To appear.

[13] Timo Bohme and Erhard Rahm. XMach-1: A Benchmark for XML Data Management. http://dbs.uni-leipzig.de/en/projekte/XML/XmlBenchmarking.html.

[14] Wolfgang M. Meier. eXist. http://exist.sourceforge.net/.

[15] Masatoshi Yoshikawa, Toshiyuki Amagasa, Takeyuki Shimura, and Shunsuke Uemura. XRel:A Path-Based Approach to Storage and Retrieval of XML Documents Using Relational Databases, 2001.

[16] 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 . In *DBSJ Letters*, No. 1 in Vol.1, 2002.

[17] 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.