DEWS2006 7B-o4

算術符号化を利用したXMLデータへの高速な問い合わせ処理の改良

舞田　哲哉†　　坂本　比呂志†

† 九州工業大学情報工学部　〒 820-8502 福岡県飯塚市大字川津 680-4
E-mail: †{t_maita, hiroshi}@donald.ai.kyutech.ac.jp

あらまし　本研究では，半構造データに対する高速な XPath 処理法を提案する．これまでに，データを効率的に圧縮する手法として知られている算術符号化を半構造データの検索に応用した，逆算術符号化が提案されている．これは，木構造データ上のパスの依存関係を，データを圧縮したまま復号化することなく検査できる手法であり，この関係性を利用することで，パスによる問い合わせを高速に処理できる．しかしながら，この問い合わせで利用可能なパスの形式は限定されているため，一般の XPath の問い合わせは処理が困難である．そこで本研究では，このような逆算術符号化にノード間の先祖子孫関係を判定可能な範囲ラベルを導入することにより，より複雑な問い合わせ処理を高速に実現するための手法を提案する．評価実験の結果，300MB 程度の XML データに対してテキストを直接処理する既存の手法と比較し，数十から百倍の高速化を達成した．
キーワード　半構造データ，問合せ処理，情報検索

Improving the Effective Query Process for XML Data by Arithmetic Encoding

Tetsuya MAITA† and Hiroshi SAKAMOTO†

† Faculty of Engineering, Kyushu Institute of Technology　Kawazu 680-4, Iizuka-shi, Fukuoka 820-8502
E-mail: †{t_maita, hiroshi}@donald.ai.kyutech.ac.jp

Abstract　The method of reverse arithmetic encoding for effective compression and fast query process for semi-structured data was proposed. We apply this method and develop a new method for more complicated query process. By the reverse arithmetic encoding, we can check the dependency between any two paths without decoding and using this property, a fast query process over tree structures is obtained. However, it is difficult to process general queires since the type of query is very restricted. So, we improve the algorithm to realize such difficult queries by using extent labels, which are introduced in order to decide any ancestor-descendant relationship between nodes in a tree. By experiments, we show that our method is nearly one hundred times faster than other proposed methods when the data size is approximately 300MB.
Key words　Semi-Structured Data, Query Process, Information Retrieval

1. はじめに

本研究の目的は，大規模な半構造データの効率的な圧縮と複雑な問い合わせ処理を両立するために，圧縮データに対して直接問い合わせ処理を行うための手法を提案することである．

半構造データの圧縮形式には大きく分けて二通りの手法がある．一方は，圧縮データが元の木構造を保存しているもので同型圧縮と呼ばれ，もう一方は，木構造を保存しない手法で非同型圧縮と呼ばれている．これらの手法には一長一短があるが，構造を利用した検索には，同型圧縮が適していると考えられるため，本研究で提案する手法は，同型圧縮の中でも特に XPress [19] の手法に基づいている．その他の圧縮法や，XPath 処理法の関連研究については次の節でまとめる．

XPress で提案されている圧縮法は，データ圧縮法のひとつである算術符号化 [25] を応用したもので，特に” 逆算術符号化” と呼ばれている．このアイディアは，半構造データを木構造と見なしたときに，木に含まれるすべてのパスを，一つの数値データに変換することによって圧縮を行うというものである．逆算術符号化の名前は，データ (パス) を通常とは逆に後ろから符号化していくことに由来している．このように符号化されたパスは，ある有理数の半開区間 $[x, y)$ として表現される．このとき，元の 2 つのパスに，一方が他方の接尾辞になっている関係があるときまたそのときに限り，対応する区間は他方を含むという性質が成り立つ．この性質により，元のパスの情報を捨

てても，符号化された情報だけから，パスに関する問い合わせを処理することが可能になる．XPress の手法は，このような性質を利用している．

しかしながら，一般に接尾辞に関する情報だけでは，有用な問い合わせ処理を行うことはできない．例えば，“//section/subsection” のような検索は処理できるが，“//section//subsection” のような出現位置を限定しない中間的なノードを含むより複雑な処理はできない．なぜならば，“//section//” の部分には，接尾辞の情報が使えないからである．このような問題点を解決する方法として，例えばタグの出現に関するシグネチャファイルを利用することが考えられる [17]．この例では，各パス毎にそのパスに出現するタグに対応するビットが 1 でそれ以外のビットが 0 となるようなシグネチャファイルを準備する．問い合わせパターンに含まれるタグに対応するビットが 1 であるようなパスをすべて集めれば，目的のパスをすべて含むようなパスの集合が得られる．しかしこの方法では，タグの出現順序を無視しているため，得られるパスの集合は，不必要なパスを多く含んでいる可能性があり，最終的な出力の前に，絞り込み検索が必要となる．そこで本研究では，この問題を解決するため，範囲ラベル付けの手法を算術符号化と組み合わせることで，一般的な XPath 処理を高速に行うシステムを提案する．

範囲ラベルを用いた XPath 処理法として，特別なスキーマ等の情報を必要としない構造ジョインアルゴリズムが提案されている [3]．この構造ジョインアルゴリズムでは，範囲ラベルを用いてノード間の親子関係や先祖子孫関係を判定できるので，XML データの全探索を回避しながら，大規模な XML データに対する XPath 処理を行うことができる．ところがこの手法では，問い合わせのパターンに現れるタグの個数に比例した回数の構造ジョインを行う必要があるため，問い合わせパターンが長くなると，高速な問い合わせが困難になる．この問題を回避するために XML データの木構造の要約情報を利用した構造ジョインの回数を削減する手法が提案されている [27]．この手法では，枝分かれのある問い合わせパターンをシングルパスに分解し，それぞれのシングルパスを要約情報によって処理することで全体の構造ジョインの回数を削減することができ，その結果，100MB 程度のデータに対して，従来の手法と比較して数百倍の高速化を達成したと報告されている．

範囲ラベルを利用したこれらの手法は，パスをテキストとして処理するため，数値データの比較で処理を行う算術符号化と比べて処理が遅いと予想できる．そこで提案手法では，範囲ラベル付け手法と算術符号化による XPath 処理を組み合わせることで，検索のさらなる高速化を達成する．本研究では提案手法を実装し，様々な問い合わせパターンに対して XPath の処理時間を測定する実験を行った．実験データは DBLP の XML データ(注1) を用いた．その結果，300MB 程度のデータに対して，どの問い合わせに対しても数ミリ秒程度で XPath 処理を終えることを確認した．特に，算術符号化による問い合わせ処理では，問い合わせパターンのうち，絶対パスの部分の処理が問い合わせ時間にほとんど影響を与えないことがわかった．これは，一度符号化されてしまえば，パスの長さは無関係に処理できるためである．

このことから，提案手法の処理時間は，すでに提案されている算術符号化による符号化 [17] よりも数倍から数十倍高速である．また，他の既存の手法と比べても，同程度のサイズの XML データに対して，オンメモリ上の処理は提案手法の方が数十倍から 100 倍以上高速であると考えられる．また，提案手法のデータのサイズを増加させたときのインデックス作成時間も測定した．その結果，時間はデータの増加に対してほぼ線形に増加し，提案手法の規模耐性も確認した．

2. 関 連 研 究

ここでは，半構造データの高速な処理という観点から，データ圧縮やパスの要約情報についての関連研究をまとめる．

データ圧縮の研究は，従来の通信コスト削減やスペース節約などの他に，検索時間そのものを短縮するために有効な手法として注目されている [28]．また，XML に代表される半構造データに対して，冗長性を多く含むその特性から様々な圧縮法が提案されている．これらの手法は圧縮法と問い合わせの可否によって次のように分類される．

一方の分類は，xmlppm [7] のように，圧縮データが元データの木構造を保存しているもので，これらは同型圧縮と呼ばれる．同型圧縮にはその他に XGrind [24] や本研究が基礎としている XPress [19] などが知られている．もう一方の分類は，XMill [15] のように構造を保存しない手法で，非同型圧縮と呼ばれている．このタイプの圧縮法にはその他に XCQ [16]，XQueC [4]，SIX [13] などがある．これらのうち，圧縮データに直接問い合わせが可能なものは，XGrind，XPress，XCQ，および XQueC である．

また，元の木構造に現れるパスやノードの種類を要約して，XPath の処理を高速にするための研究も盛んに行われている．DataGuide [12] は XML 木に現れるパスを要約したものであり，データ構造上のパスの同値関係が，元の木構造のものと等価であるとき，特に Strong DataGuide と呼ぶ．その他にパスを要約する手法として，サイズが必ず元の XML 木以下になることが保証されている 1-Index [18] やデータマイニングの手法を用いて，問い合わせによって要約構造を動的に変更する APEX [8] などが提案されており，二次記憶上の XML データへのアクセスを抑制しつつ，要約情報のサイズ自体も可能な限り小さくする研究が進められている．

3. 準　　備

本論文で扱う基本的なデータ構造や概念について説明する．

3.1 XML 木と XPath 問い合わせ問題

XML テキストはある木構造データに対応し，本論文では，以下のような標準的なデータモデルを想定している．ただし，木の用語については省略する．ひとつの XML ファイルをページとよび，ある XML ページはノードにラベルを持つ根付き順

(注1)：http://dblp.uni-trier.de/xml/

序木に対応する．ノードのラベルは，元の XML ファイルのある要素に対応する．例えばあるノードのラベル name は，元の XML ページ中のある要素 name に対応する．図 1 に典型的な XML ページを示す．

```
<book>
   <author> author1 </author>
   <title> title1 </title>
   <section>
      <title> title2 </title>
      <subsection>
         <subtitle> title3 </subtitle>
         ...
      </subsection>
   </section>
<book>
```

図 1　XML ページの例

この XML ページに対応する XML 木の根は，ノードラベル book を持ち，根は author, title, section の 3 つの子を持っている．さらに section は，title, subsection の 2 つの子を持ち，subsection も自分の子を持っている．本研究の目的は，このような木構造のデータを，その構造をうまく保ったまま圧縮することで，検索時間の短縮や複雑な検索に利用することである．

本論文で扱う XPath 問い合わせの範囲は XP$(/, //, *, [\,])$ [23] である．すなわち，'/' と'//' はそれぞれ親子関係を指定する `child` 軸と先祖子孫関係を指定する `descendant` 軸であり，'$*$' は任意のタグを指定するノードテスト，'[ ]' は分岐を許可する．このような XPath 問い合わせは，ラベル付けされた木とみなせるので，問い合わせ木と呼ぶことにする．ただし，問い合わせ木中で，葉に相当するノードのうち 1 つをあらかじめ指定してあり，これをアウトプットノードと呼ぶ．

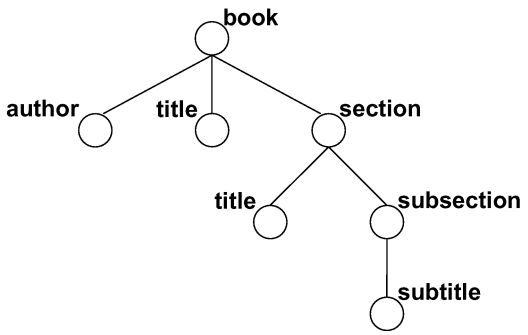

図 2　XML 木の例.

以上のように定義された XML 木と XPath 問い合わせに対して，XP$(/, //, *, [\,])$ の範囲の XPath の問い合わせ問題は，1 組の XML 木 T と問い合わせ木 P が与えられたとき，P のすべてのノードテストが T のあるノードにマッチし，かつマッチした T のノードが，P の枝が示すノード間の XP$(/, //, *, [\,])$ の範囲の関係を満たすような T の部分木をすべて見つける問題である．ただし，アルゴリズムは，見つけた部分木を返す代わりに，P のアウトプットノードにマッチした T のすべてのノードを出力すればよいものとする．

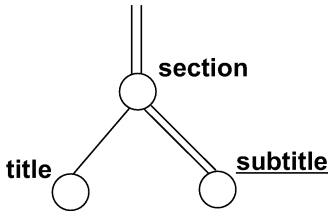

図 3　問い合わせ "`//section[./title]//subtitle`" に対応する問い合わせ木．先祖子孫関係は二重線で表されている．また，下線で示すノードがアウトプットノードである．

図 2 と図 3 にそれぞれ，図 1 の XML ページに対応する XML 木とある問い合わせ木の例を示した．この例では，問い合わせ木のすべてのノードが，目的の XML 木に定義に従って埋め込まれる．したがって，問い合わせ木のアウトプットノードが，XML 木の同じラベルを持つノードにマッチする．一般にこのような判定を行うためには，問い合わせ木の複雑さに依存した回数の構造ジョインを行わなくてはならないが，算術符号化を利用することで，その回数を大幅に削減することができ，また，その手法と範囲ラベルを組み合わせることで，複雑な問い合わせに対しても非常に高速に処理を行うことができる．次に算術符号化による XML の符号化を説明する．

3.2　逆算術符号化

ここでは，逆算術符号化による XML の符号化とその符号値を用いた XPath の処理の原理を説明する．例として，図 2 で示した XML 木を使って説明する．まず，XML 木に現れるノードパスを定義する．

XML 木のあるノードを指定すると，そのノードから根ノードに至るパスが一意に定まる．このとき，そのパスをそのノードのノードパスという．正確には次のように定義される．

［**定義 1**］ XML 木のあるノード v_n から根ノード v_0 へのパスに含まれるノードの列を，根に近いものから順に，$v_0, v_1, \ldots, v_n$ とする．また，v_i $(0 \leqq i \leqq n)$ のノードのラベルを ℓ_i とする．このとき，ノードとそのラベルを交互に並べた列 $v_0.\ell_0.v_1.\ell_1 \cdots v_n.\ell_n$ をノード v_n のノードパスと定義する．

このように XML 木中のノードを 1 つ指定することで，ノードパスはただひとつに定まる．ノードパスはこのように定義されるが，ノードとラベルを両方示すのは煩雑なので，以降の議論では，なるべくノードパスを単にラベルの列として表すことにしたい．例えば図 2 における *subsection* のラベルを持つノードのノードパスは，ノードを省略してラベルのみで示すと *book.section.subsection* である．ところが，タグの列として *book.section.subsection* となるようなノードパスは，複数存在することがあり得る．そこで，単にラベルの列なのか，ノードを省略したノードパスであるのかを区別する必要があるので，以下でその区別を付けるためにもう一つの概念であるラベルパスを定義する．

［**定義 2**］ 各ノード v_i のラベルが ℓ_i であるとき，ラベルの列

$\ell_1.\ell_2 \ldots \ell_n$ を v_n のラベルパスという．また，2つのラベルパス $p = p_i.p_{i+1} \ldots p_n$ と $q = q_j.q_{j+1} \ldots q_n$ に対して，$i \leqq j$ かつ $p_k = q_k$ $(j \leqq k \leqq n)$ であるとき，q は p の接尾辞であるという．

例えば，ラベルパス *section.subsection* や *subsection* は，いずれも *book.section.subsection* の接尾辞である．また，任意のラベルパス p は，それ自身の接尾辞である．同様に，上の定義の p, q の各ラベルを逆順に並べた列 p', q' に対して，q' は p' の接頭辞であるという．

次に，これらの概念を利用して XML 木のパスの逆算術符号化を定義する．目的は，与えられた XML 木のすべてのノードパスを半開区間 $[0.0, 1.0)$ 上のある部分区間に変換することである．ただし，ラベルパスとしては区別の付かないノードパスは同一の区間へ変換される．あるラベル T に対する符号化後の区間を $Interval_T$ と表す．符号化の初期段階として，XML 木に含まれるすべてのタグの頻度を計測する．このとき，各ラベルの $Interval_T$ は頻度の累積値として表現できる．このことを，図 2 の XML 木を用いて説明しよう．

［**例 1**］ 図 2 の XML 木に現れるすべてのタグに対してある順序 ⟨ *book*, *author*, *title*, *section*, *subsection*, *subtitle* ⟩ を固定する．そして，各タグの頻度を計算すると以下のような値が得られる．このとき，各ラベルの $Interval_T$ はそれ以前のラベルの頻度を合計して得られる累積的頻度によって，表 1 のように計算される．

表 1　各ラベルの符号化

ラベル	頻度	累積的頻度	$Interval_T$
book	0.1	0.1	$[0.0, 0.1)$
author	0.1	0.2	$[0.1, 0.2)$
title	0.1	0.3	$[0.2, 0.3)$
section	0.3	0.6	$[0.3, 0.6)$
subsection	0.3	0.9	$[0.6, 0.9)$
subtitle	0.1	1.0	$[0.9, 1.0)$

このようにして定められた各ラベルの $Interval_T$ の値を用いて，すべてのノードパスを再帰的に符号化していく．この手続きは，各ラベルをアルファベット 1 文字と見なした場合の通常の算術符号化と同じものであり，図 4 のように記述される．

このアルゴリズムを用いて，あるノードパス $p = p_1.\cdots.p_n$ は，はじめに p_n の頻度によってある区間に変換され，この区間は各 p_i の頻度によって，次々に更新されていき，最終的にある一つの区間に符号化される．このことを次の例で具体的に説明する．

［**例 2**］ 表 2 にラベル T の区間値 $Interval_T$ と，T を接尾辞に持つラベルパス P とその区間値 $Interval_P$ の例を示す．ただし，符号化の方向は接尾辞から接頭辞の方向に進むものとする．このとき *book.section.subsection* の区間値は以下のように計算される．まず，*subsection* の区間値は $[0.6, 0.9)$，*section* の区間値が $[0.3, 0.6)$ であることから，*section.subsection* の区間値の左側は，$[0.6, 0.9)$ の区間を 100% としたときの，左から 30% であり，区間値の右側は，左から 60% であることがわかる．すなわち，$min_v := min_v + length * q_{min} = 0.6 + 0.3 * 0.3 = 0.69$ および $max_v := min_v + length * q_{max} = 0.6 + 0.3 * 0.6 = 0.78$ と計算できる．そこでこの $[0.69, 0.78)$ と *book* の区間値 $[0.0, 0.1)$ を用いて同様に計算することで，*book.section.subsection* の区間値 $[0.69, 0.699)$ が得られる．

Function $ReArithmetic(p)$
begin
1. $[min_v, max_v) := Interval_{p_n}$;
2. **if**$(n = 1)$ **return** $[min_v, max_v)$;
3. $length := max_v - min_v$;
4. $[q_{min}, q_{max}) := ReArithmetic(\tilde{p})$;
5. $min_v := min_v + length * q_{min}$;
6. $max_v := min_v + length * q_{max}$;
7. **return** $[min_v, max_v)$;

end

図 4　ノード v のノードパス $p = p_1.\cdots.p_n$ を区間 $[min_v, max_v)$ に変換する逆算術符号化アルゴリズム．ただし，$\tilde{p} = p_1.\cdots.p_{n-1}$.

表 2　ノードパスの符号化

要素 (T)	ノードパス (P)	$Interval_T$	$Interval_P$
book	*book*	$[0.0, 0.1)$	$[0.0, 0.1)$
section	*book.section*	$[0.3, 0.6)$	$[0.3, 0.33)$
subsection	*book.section.subsection*	$[0.6, 0.9)$	$[0.69, 0.699)$

XML 木に含まれる要素の数 n に対して，ノードパスに含まれるラベルの数は $O(n^2)$ であるが，以前計算した符号値を用いることで，すべてのノードパスの逆算術符号化は $O(n)$ 時間で計算可能である．算術符号化の詳細については論文 [21] などが詳しい．

このアルゴリズムによって得られる任意の区間は，以下の性質を満たすことが簡単にわかるが，この性質は，圧縮データに対して問い合わせ処理を高速に行う上で重要な役割を果たす．

［**補題 3**］ [19] あるノードパスを p とし，その符号化後の区間を I とする．このとき，p の任意の接尾辞を符号化した区間は I を含む．ここで，ある区間 $I' = [x', y')$ が $I = [x, y)$ を含むとは，$x' \leqq x$ かつ $y \leqq y'$ であることをいう．

この性質を用いることで，木のノードを巡回することなく目的のパスを高速に発見できる．例えば *section.subsection* の区間は $[0.69, 0.78)$ であり，*book.section.subsection* の区間は $[0.69, 0.699)$ であるが，前者は後者を必ず含み，かつ含んだ場合は必ず接尾辞になっていることが保障されるため，*/section/subsection* や，より複雑な *//section/subsection* のような問い合わせに対する処理が，区間の比較 (すなわち数値の比較) のみで高速に実現できる．本研究では，この検索手法をより複雑な XPath の問い合わせが実行可能となるように拡張し，実験によってその効果を確かめる．

4. 提 案 手 法

前節で説明した逆算術符号化によって，すべてのラベルパスは，ある区間値に変換される．ここで，異なるノードのノードパスであっても，そのラベルパスが同じであれば，同一の区間値に変換される．したがって，これらの区間値によって，もとの XML 木のノード群はクラスタ化されるので，できるだけ区間値の比較のみによってアウトプットノードが含まれるクラスを確定することで，二次記憶装置へのノードの参照を抑制することができる．

しかし，逆算術符号化による手法では，接尾辞の関係が，符号化後の区間でも保存されるという性質を利用しているため，この手法では，接尾辞の関係にないような問い合わせパターンは処理できない．例えば $//section//subtitle$ というような，簡単ではあるがきわめて有用な問い合わせは，接尾辞の関係だけでは検索できない．したがって，逆算術符号化による区間値だけでは，XP$(/, //, *, [\,])$ の範囲の問い合わせのうち，中間ノード'//' と分岐'[ ]' を処理できない．そこで，ノードやパスに範囲ラベルを設定することで，ノードの先祖子孫関係などを判定し，それを利用して，高速な XPath 処理を行う．

4.1 範囲ラベル

ここでは，順序木の範囲ラベルとは，その木の各ノード v に，整数値のペア (n_v, m_v) を設定したものである．このとき，n_v, m_v はそれぞれ，その木を前置順と後置順で探索したときのノード v の順位を表す．範囲ラベルの設定は，後のデータの更新に備えて整数値同士の間隔を十分に空けることがよく行われるが，今回の場合は単に前置順と後置順の値そのものと考える(注2)．図 5 に，ある順序木の範囲ラベルの例を示す．

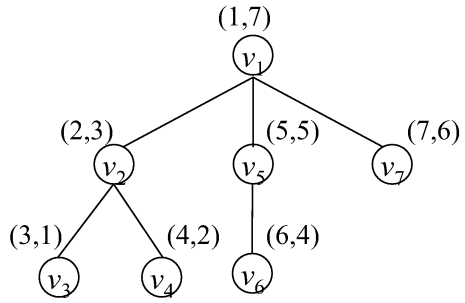

図 5　範囲ラベルの例．各ノードが自分の前置順と後置順の順位のペアを持っている．

実際には，このラベルに加えて，ノードの深さも付与することにより，ノード間の親子関係も判定できる．実際の判定には次の補題を用いることで，範囲ラベルのみから任意のノード間の関係を判定できる．

［**補題 4**］ 順序木中の任意のノード u, v に対して，それぞれの範囲ラベルを $(n_u, m_u), (n_v, m_v)$ とする．u が v の子孫であるのは，$n_v < n_u$ かつ $m_v > m_u$ であるときまたそのときに限る．

(注2)：今後の課題で述べるように，算術符号化によって符号化されたデータは更新が難しいため，本研究ではインデックスの動的な更新は考えない．

次にこの範囲ラベルと，算術符号化による区間値を組み合わせた XPath 処理法を説明する．

4.2 中間ノードの判定

アルゴリズムは，もとの XML 木に関する木構造を保持していない．アルゴリズムが保持しているのは，各パスのラベルパスやノードのラベルの算術符号化による区間値のみである．このような区間値のみから，上記の範囲ラベルによる判定を行うために，アルゴリズムは範囲ラベルに関する 2 種類のテーブルを作成する．一方をタグテーブル，他方をパステーブルという．タグテーブルは，順序木中のタグの出現に対して範囲ラベルを割り当てたテーブルであり，パステーブルは，異なるパスの出現に対して，その終端のノードに対して範囲ラベルを割り当てたテーブルである．

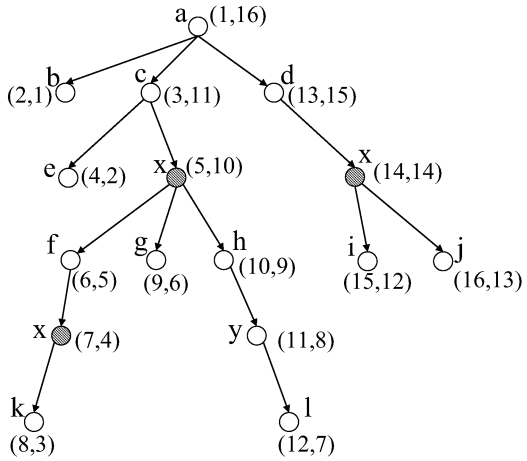

図 6　順序木のラベルと範囲ラベルの例．各ノードはアルファベットで示された自分自身のラベル (すなわちタグ名) と自分の前置順と後置順の順位のペアを持っている．

図 6 に，具体的な順序木を示した．アルゴリズムはこのようなラベルの情報から，表 3 のようなタグテーブルとパステーブルを作成する．特に，ラベル x は 3 回出現するため，x の範囲ラベルは各出現毎に用意する．また，この表には具体的なパスがそのままの形で示されているが，実際にはパスはその符号化された区間値で管理されていることに注意する．このテーブルを用いたシングルパス処理アルゴリズムを図 7 に示す．

Algorithm *SinglePath*
Input: タグテーブル T，パステーブル P，問い合わせ p.
Output: 問い合わせにマッチした範囲ラベルの列 (ノード列).
begin
1. $p = /p_1//p_2//\cdots//p_n$ とする．(p_i はラベルパス)
2. 各 p_i にマッチする範囲ラベルの集合 E_i を T から求める．
3. $E_p = \cap_{1 \leq i \leq n} E_i$ を出力する．
end

図 7　シングルパスの XPath 処理アルゴリズム

以下では表 3 を使用して，図 7 のアルゴリズムを説明する．

表 3　各ラベルとラベルパスの出現に関する範囲ラベル．

ラベル	範囲ラベル	ラベルパス (区間値)	範囲ラベル
a	$(1,16)$	a	$(1,16)$
b	$(2,1)$	ab	$(2,1)$
c	$(3,11)$	ac	$(3,11)$
d	$(13,15)$	ace	$(4,2)$
e	$(4,2)$	acx	$(5,10)$
f	$(6,5)$	$acxf$	$(6,5)$
g	$(9,6)$	$acxfx$	$(7,4)$
h	$(10,9)$	$acxfxk$	$(8,3)$
i	$(15,12)$	$acxg$	$(9,6)$
j	$(16,13)$	$acxh$	$(10,9)$
k	$(8,3)$	$acxhy$	$(11,8)$
l	$(12,7)$	$acxhyl$	$(12,7)$
x	$(5,10)$	ad	$(13,15)$
x	$(7,4)$	adx	$(14,14)$
x	$(14,14)$	$adxi$	$(15,12)$
y	$(11,8)$	$adxj$	$(16,13)$

入力問い合わせが'$//x//y$' であるとき，アルゴリズムは根に近い部分のパターンから処理していく．この場合はまず，中間にタグ x を含んだパスの範囲を決定する．タグテーブルより，x の範囲ラベルは $(5,10)$，$(7,4)$，$(14,14)$ であるので，各範囲ラベルに対して補題 4 を用いて，前置順が自分よりも大きく，後置順が自分よりも小さい範囲ラベルを持つラベルパスをパステーブルから見つければよい．ただしこの場合，$(7,4)$ は $(5,10)$ の子孫であるのであらかじめ除外してよい．したがって求める範囲は，$(5,10)$ に対しては，ラベルパス acx から $acxhyl$ までであり，$(14,14)$ に対しては最後の 3 つのラベルパスが条件を満たす．さて，このようにして絞り込まれたラベルパスの範囲から，残りのタグ y を接尾辞に持つパスを確定すればよい．このとき，補題 3 を用いて，範囲内から接尾辞が y であるようなパスを区間の比較によって求めることで，$acxhy$ が得られる．これは確かに'$//x//y$' にマッチしてかつそれ以外にこの問い合わせにマッチするパスは存在しない．

中間ノードを含んだより複雑なシングルパスも同様の手法で処理できる．入力は一般に'$/p_1//p_2//\cdots//p_n$' の形をしていると考えてよい(注3)．ただし，各 p_i はラベルパスである．この場合，最初の $/p_1$ の部分は区間値の比較によって確定できる．またそれ以外の処理は，上述したタグテーブルとパステーブルによって絞り込み検索をしていくことで，目的のパスを決定できる．ただし，p_i はいくつかのラベルの連接すなわち親子関係を持っているので，その判定には，範囲ラベルに別に持たせてある根からの深さの値を用いればよい．以上のようにして，中間ノードを持つシングルパスについては効率よく処理できる．

4.3　分岐の判定

分岐を含んだ問い合わせの判定は，まず，問い合わせをシングルパスに分解して，これまでの手法を使って各シングルパスにマッチするノードを確定する．そしてそれらのノードに対して，範囲ラベルを用いて構造ジョインを行い，最終的にマッチしたノード列を取り出す．このような構造ジョインは，一般に，2 つのノード x,y の分岐ノード z を見つけることと同等であり，具体的には，補題 4 からただちに導かれる次の補題を利用することで，分岐ノードを効率的に判定できる．

［**補題 5**］ 順序木中の任意のノード x,y,z に対して，z が x,y の共通祖先であるのは，$n_z < n_x, n_y$ かつ $m_z > m_x, m_y$ であるときまたそのときに限る．

この補題と各ノードの根からの深さの値によって，共通祖先のうちもっとも深い最近共通祖先を確定できる．この最近共通祖先が求める分岐ノードである．

5. 評 価 実 験

本研究で提案したアルゴリズムのうち，シングルパスの XPath 処理に関して実装し，評価実験を行った．実験環境は，Celeron 2.40GHz，512MB メモリ上の Windows XP であり，算術符号化と範囲ラベルによる構造ジョイン処理を C 言語で実装した．

実験で使用したデータは，DBLP の論文データを XML 化したアーカイブを `http://dblp.uni-trier.de/xml/`から入手した．データサイズは非圧縮で 294MB である．このデータを算術符号化によって符号化し，典型的な問い合わせを入力として，その応答時間を測定した．この実験で，提案手法の処理時間と算術符号化とシグネチャファイルによる XPath 処理法 [17] の処理時間を比較した．

応答時間は，XPath 問い合わせ開始から，マッチするラベルパスを確定するまでの時間を測定している．実際の処理時間は，これに対応するアウトプットノードを二次記憶上の XML 木から取り出し，結果をソートする時間が加わる．また，分岐を含んだ問い合わせの処理には，さらにそのための構造ジョインの手間が加算される．

5.1　応答時間の測定

提案手法と [17] の手法との応答時間を比較した．表 4 に今回使用した問い合わせパターンを示す．ここで，各問い合わせについて説明する．問い合わせ 1 は絶対パスであり，問い合わせ 2 から 6 までは通常の中間ノードを含んだシングルパスである．また，問い合わせ 7 および 8 の'$*$' は，すべてのラベルにマッチするワイルドカードを表す．さらに，問い合わせ 9 は AND (&) を含んでおり，このパスの意味は'/dblp//sub//sup ∪ /dblp//sup//sub' と等価である．同様に，問い合わせ 10 は OR (|) を含み，'/dblp//sub ∪ /dblp//sup' と等価である．

図 8 にこれらの問い合わせに対する応答時間の結果を示す．横軸の数字が表 4 の各問い合わせパターンに対応する．時間 A が提案手法による応答時間であり，時間 B はシグネチャファイルを用いた場合の応答時間である．この結果より，シグネチャファイルを使う場合には，問い合わせ 7 や 10 のように，マッチするパスの数が増加するにつれて過剰にマッチしたパスを絞り込まなければならず，応答時間が増加しているのに対し，提

(注3)：問い合わせの接頭辞が相対パスで始まる場合は p_1 の長さが 0 と考えればよい．

表 4　問い合わせに用いたシングルパス

	問い合わせパターン
1	/dblp/inproceedings/title/i/sup
2	/dblp/inproceedings//sup
3	/dblp//title//sup
4	/dblp//title/i//sup
5	/dblp//inproceedings//i//sup
6	//title//sup
7	//title//*
8	/dblp/inproceedings//title//*
9	/dblp//sub&sup
10	/dblp//sub|sup

案手法では，過剰なマッチが起こらないため，すべての問い合わせについて処理時間が改善されている．この結果から，すでに提案されている算術符号化とシグネチャファイルによる処理法よりも本研究で提案した手法は十分に高速であることが示された．

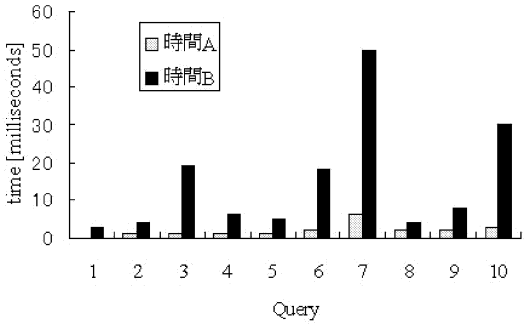

図 8　応答時間の比較

5.2　規模耐性の測定

応答時間と共に，提案手法によるインデックス構築のデータの増加に対する規模耐性を測定した．結果を図 9 に示す．測定時間はデータの増加に対してほぼ線形に増加しており，この結果から，提案手法は少なくとも数百 MB 程度の XML データに対しては十分な規模耐性を持つことがわかる．

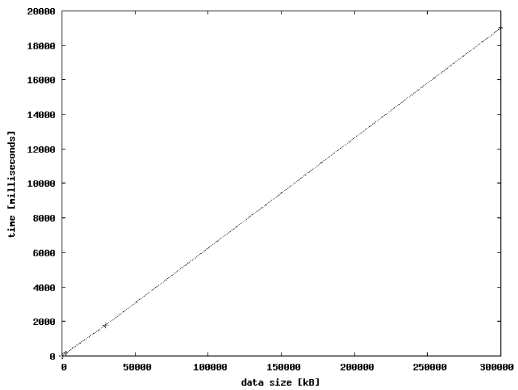

図 9　インデックス構築の規模耐性

6.　まとめと今後の課題

本研究では，算術符号化と範囲ラベルを組み合わせた半構造データに対する効率的な XPath 処理法を提案し，実験によって，素朴に構造ジョインを行う手法に対してその優位性を示した．

本研究で提案した手法は，中間ノードを含んだシングルパスについては数ミリ秒程度で処理を終える．特に問い合わせ 1 や 2 のような簡単なパスについてはその効果は顕著であり，応答時間はミリ秒のオーダーを下回ることがある．この速度は，ラベルや木構造をテキストを保存したまま比較する既存の手法では実現が困難である．この値は，例えば，要約情報 [27] を用いた手法に比べて数十倍高速であり，より多くの構造ジョインを行う手法 [3], [8], [26] に対してはそれ以上の差があると考えられる．

また，提案手法は，既存の算術符号化を用いた手法よりも数十倍高速であり，データの増加に対する規模耐性も十分にあることを示した．しかしながら，算術符号化によるインデックスは，次のような問題点を抱えており，その克服が今後の課題である．

（1） XML 木の深さに対する耐性：計算機は無限精度の実数を扱えないため，実際の算術符号は計算可能な桁以内で丸められており，その精度には限界がある．今回の実験では問題は起こらなかったが，ある程度深いノードを含む木を扱う場合には，長いパスを符号化する必要がある．現在の算術符号化は，この問題を適応型アルゴリズムによって解決している．これは，実数値が有効桁からあふれてしまう場合に，それ以降の符号化を最初の区間値からリセットして行う手法である．しかしながら，本研究における XPath 処理の問題ではこれは解決策にならない，なぜならば，パスの符号化を途中でリセットしてしまうと，接尾辞の関係が保たれなくなってしまうからである．

（2） データの更新：要約情報や通常の DOM 木のような実際に木を構築する手法では，データの一部分を更新 (削除や追加など) を行って，それをインデックスに反映させることは容易である．ところが提案手法では，あるノードを編集した場合，XML 木の中でそのノードの子孫の関係あるノードすべてに修正を加える必要がある．この場合もやはり接尾辞の関係を保存するためである．したがって，提案手法では，インデックスの更新はかなり大がかりな修正が必要となる．

以上のような問題点を解決し，最終的な XML データベースを構築することが今後の目標である．

文　　献

[1] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web.* Morgan Kaufmann, 2000.

[2] A. Aboulnaga, A. R. Alameldeen, and J.F. Naughton. Estimating the Selectivity of XML Path Expressions for Internet Scale Applications. In *Proceedings of 27th International Conference on Very Large Data Bases,* pp. 591-600, 2001.

[3] S. Al-Khalifa, H. V. Jagadish, N. Koudas, J. M. Patel, D. Srivastava, and Y. Wu. Structural Joins: A Primitive for Efficient XML Query Pattern Matching. In *Proc. ICDE2002.*

[4] A. Arion, A. Bonifati, G. Costa, S. D'Aguanno, I. Manolescu, and A. Pugliese. Xquec: pushing queries to compressed XML data. In *Proceedings of the Twenty-Ninth International Conference on VLDB*, pp. 1065-1068, 2003.

[5] S. Boag, D. Chamberlin, M. F. Fernandez, D. Florescu, J. Robie, and J. Simeon. XQuery 1.0: An XML Query Language. `http://www.w3.org/TR/2002/WD-xquery-20020816`, 2002.

[6] T. Bray, J. Paoli, C. M. Sperberg-McQueen, and E. Maler. Extensible Markup Language (XML) 1.0 W3C Recommendation, `http://www.w3.org/TR/REC-xml`, 1998.

[7] J. Cheney. Compressing XML with Multiplexed Hierarchical PPM Models, In *Proceedings of Data Compression Conference,* pp. 163-172, 2001.

[8] c.-W. Chung, J.-K. Min, and K. Shim. APEX: Adaptive Path Index for XML Data. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data,* pp. 121-132, 2002.

[9] J. Clark and S. DeRose. XML Path Language (XPath) Version 1.0 `http://www.w3.org/TR/xpath`, 1999.

[10] M.F. Fernandez and D. Suciu. Optimizing Regular Path Expressions Using Graph Schemas. In *Proceedings of the 14th International Conference on Data Engineering,* pp. 13-23, 1998.

[11] D. Florescu and D. Kossman. Storing and Querying XML Data using an RDMBS. *IEEE Data Engineering Buletin,* 22(3):27-34, 1999.

[12] R. Goldman and J. Widom. DataGuides: Enable Query Formulation and Optimization in Semistructured DataBases. In *Proceedings of 23rd International Conference on Very Large Data Bases,* pp. 436-445, 1997.

[13] T. J. Green, A. Gupta, G. Miklau, M. Onizuka, D. Suciu, Processing XML Streams with Deterministic Automata and Stream Indexes *ACM TODS*, vol. 29, no. 4, 2004.

[14] P. G. Howard and J. S. Vitter. Analysis of Arithmetic Coding for Data Compression. In *Proceedings of the IEEE Data Compression Conference,* pp. 3-12, 1991.

[15] H. Liefke and D. Suciu. XMill: An Efficient Compressor for XML Data. In *Proceedings of the 2000 ACM SIGMOD Inernational Conference on Management of Data,* pp. 153-164, 2000.

[16] W.-Y. Lam, W.-N. Peter, and M. Levene. XCQ: XML Compression and Querying System. In *WWW Conference, poster*, 2003.

[17] T. Maita and H. Sakamoto. A Simple Extension of Queriable Compression for XML Data. In *Proc. of the 2005 International Conference on Active Media Technology (AMT2005)*, pp. 91-95, 2005.

[18] T. Milo and D. Suciu. Index structures for path expressions. In *Proc. of the Int´ l Conf. on Database Theory*, pp. 277-295, 1999.

[19] J.-K. Min, M.-J. Park, and C.-W. Chung. XPRESS: A Queriable Compression for XML Data. In *Proceedings of the 2000 ACM SIGMOD Inernational Conference on Management of Data,* pp. 122-133, 2003.

[20] C.-W. Park, J.-K. Min, and C.-W. Chung. Structural Function Inlining Technique for Structually Recursive XML Queries. In *Proceedings of the 28th International Conference on Very Large Data Bases,* pp. 83-94, 2002.

[21] D. Salomon. Data Compression, the complete reference. Springer-Verlag New York, Inc, 1998.

[22] A. Schmidt, F. Waas, M. Kersten, M. Carey, I. Manolescu, and R. Busse. Xmark: A benchmark for XML data management. In *Proceedings of the 28th International Conference on Very Large Databases (VLDB)*, pages 974–985, 2002.

[23] T. Schwentick. XPath Query Containment. *ACM SIGMOD Record*, 2004.

[24] P. M. Tolani and J. R. Haritsa. XGRIND: A Query-friendly XML Compressor. In *Proceedings of the 18th International Conference on Database Engineering,* 2002.

[25] I. H. Witten, R. M. Neal, and J. G. Cleary. Arithmetic Coding for Data Compression. *Communications of the ACM,* 30(6):520-540, 1987.

[26] Y. Wu, J. M. Patel, and H. V. Jagadish. Structural Join Order Selection for XML Query Optimization. In *Proc. ICDE2003.*

[27] 江田毅晴，鬼塚真，山室雅司．XML データの要約情報を用いた高速な XPath 処理法．DEWS2005, 6B-o4, 2005.

[28] 竹田正幸，篠原歩．圧縮されたテキスト上のパターン照合〜データ圧縮とパターン照合の新展開〜．情報処理学会誌, 43(7):763-769, 2002.

[29] 山本博資，植松友彦，横尾英俊，古賀弘樹，星守．データ圧縮における最新アルゴリズム [I]〜[V]，電子情報通信学会誌，vol.86, No.2〜7，2003.