# XBW 変換による圧縮 XML データ上の構造問合せの性能評価

東原 正智† 田島 敬史‡

†法政大学理工学部経営システム工学科

〒184-8584 東京都小金井市梶野町 3-7-2

‡京都大学情報学研究科社会情報学専攻社会情報モデル講座

〒606-8501 京都市左京区吉田本町

E-mail: †masanori.higashihara.83@adm.hosei.ac.jp, ‡tajima@i.kyoto-u.ac.jp

**あらまし** XML データなどのラベル付き木の圧縮手法の一つに XBW 変換がある．XBW 変換では，各ノードを，その親から根までの逆順ラベルパスでソートすることにより圧縮を行うため，XBW 変換により圧縮されたデータは，パスに基づく構造問合せを通常の非圧縮のデータより効率よく実行できる可能性がある．本論文では，XBW 変換された XML データに対する基本的な構造問合せの様々な評価手法を提案し，これらの性能評価を行った．

**キーワード** XML，XBW 変換，構造問合せ

## 1. はじめに

XML は、幅広く使われている汎用的なデータフォーマットである。XML は、タグでテキストデータを囲むためデータサイズが多くなり、そのためデータサイズの削減のため圧縮の研究が多く行われている。

XML を圧縮する研究はこれまでに数多くあり、特に 2000 年に発表された XMill が有名である[1][2]。当初は、圧縮のみが注目されていたが、XGrind[3]からは圧縮したデータの上での検索に関心が移っていった。圧縮したままで検索を行うことを本論文では圧縮検索と呼ぶ。圧縮検索に関する研究も、数多く提案されているが、データ構造そのものに起因する制約から問合せに限界があった。

XBW 変換は、近年研究されている簡潔データ構造[4]の一種で、データサイズは情報論的な下界で、基本的な関数(rank、select)が定義されている。また、Ferragina の研究では、上で定義されている関数(rank, select)から木構造を探索するための Tree Navigation 関数(GetParent, GetChildren, SubPathSearch など)を構成している[5][6][7]。

XML 上での問合せの研究には、これまでに数多くの研究がある。そのなかで、より効率的な問合わせとして構造問合せがある[8]。代表的な構造問合せとして structural join や holistic twig join などがあり、その改良を行う研究もある。構造問合せの特徴は、問合せを 2 項間の関係に分割し、インデックスによってその親子関係や先祖子孫関係を 2 項間の関係のみで判別することである。

本研究では、XML のモデルとして順序付ラベル木で表現し、そのデータを XBW 変換し、その上で定義されている関数を用いて、構造問合せを構成し、その性能評価をおこなった。XBW 変換は、2 つの配列と付加的なインデックスを用いる。また、こうした配列でのデータ操作について、包括的かつ理論的な研究も行われている[9]。

## 2. 関連研究

### 2-1. XBW 変換

XBW 変換は、順序付ラベル木を 2 つの配列に変換する。図 1 では、XBW 変換を具体的な例を用いて表している。

XBW の手順としては、

1) XML データを preorder によって探索する。探索したラベルやエレメントを、root を先頭にして順番に配列(Salpha)へ格納する。
2) その際、インデックスとして葉(leaf)や最後の子供(last child)について 1 を立てる。それ以外は 0 とする。そのインデックスは配列(Slast)に格納する。
3) 各配列の要素ごとに root からのパスを逆順に求めそれを配列(Spai)に格納する。
4) 最後に Spai を基準として安定ソートを行う。
5) 付加的なインデックスとして
   A:Spai でのパスの変わり目の先頭を 1 とし、それ以外は 0 とするインデックス。
   F:A から計算されるが、Salpha でのタグが先頭となる範囲の先頭の配列の番号を格納する。

上記の操作によって導かれる XBW 変換の配列の 3 つ組 S を S=<Slast, Salpha, Spai>と表す。元の順序付きラベル木を T とすると 2 つの間には次のような構造的な関係がある。

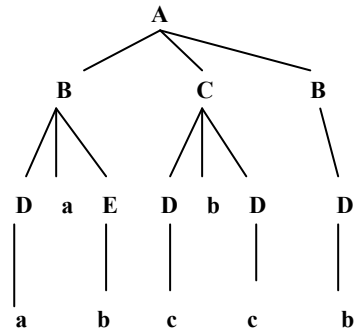
構造的性質.

1. S の 3 つ組の先頭は T のルートを示す。
2. ノード u の 3 つ組は、次の条件の場合にノード u の三つ組に先行する。π[u]はノード u から親までの逆順のパスを表す。

```
<A>
  <B>
    <D>a</D>a<E>b<//E>
  </B>
  <C>
    <D>b</D> b <D>c</D>
  </C>
  <B>
   <D>b</D>
  </B>
</A>
```

pre-order

| id | Slast | Salpha | Spai | SP:EP | L |
|---|---|---|---|---|---|
| 1 | 0 | A | empty | 1:25 | 1 |
| 2 | 0 | B | A | 2:10 | 2 |
| 3 | 0 | D | BA | 3:5 | 3 |
| 4 | 1 | a | DBA | 4 | 4 |
| 5 | 0 | a | BA | 6 | 3 |
| 6 | 1 | E | BA | 7:9 | 3 |
| 7 | 1 | b | EBA | 8 | 4 |
| 8 | 0 | C | A | 11:19 | 2 |
| 9 | 0 | D | CA | 12:14 | 3 |
| 10 | 1 | c | DCA | 13 | 4 |
| 11 | 0 | b | CA | 15 | 3 |
| 12 | 1 | D | CA | 16:18 | 3 |
| 13 | 1 | c | DCA | 17 | 4 |
| 14 | 1 | B | A | 20:24 | 2 |
| 15 | 1 | D | BA | 21:23 | 3 |
| 16 | 1 | b | DBA | 22 | 4 |



木 T

Spai で
安定ソート

| id | Slast | Salpha | Spai | SP:EP | L | A | F |
|---|---|---|---|---|---|---|---|
| 1 | 0 | A | empty | 1:25 | 1 | 0 | 2 |
| 2 | 0 | B | A | 2:10 | 2 | 1 | 5 |
| 3 | 0 | C | A | 11:19 | 2 | 0 | 9 |
| 4 | 1 | B | A | 20:24 | 2 | 0 | 8 |
| 5 | 0 | D | BA | 3:5 | 3 | 1 | 12 |
| 6 | 0 | a | BA | 6 | 3 | 0 | -1 |
| 7 | 1 | E | BA | 7:9 | 3 | 0 | 16 |
| 8 | 1 | D | BA | 21:23 | 3 | 0 | 13 |
| 9 | 0 | D | CA | 12:14 | 3 | 1 | 14 |
| 10 | 0 | b | CA | 15 | 3 | 0 | -1 |
| 11 | 1 | D | CA | 16:18 | 3 | 0 | 15 |
| 12 | 1 | a | DBA | 4 | 4 | 1 | -1 |
| 13 | 1 | b | DBA | 22 | 4 | 0 | -1 |
| 14 | 1 | c | DCA | 13 | 4 | 1 | -1 |
| 15 | 1 | c | DCA | 17 | 4 | 0 | -1 |
| 16 | 1 | b | EBA | 8 | 4 | 1 | -1 |

**3 つ組 S=<Slast,Salpha,Spai>**

（注）

・SP:EP は、structural join における StartPos、EndPos を示す。

・L は、LevelNum を示す。

**図 1 .XBW 変換**

・辞書的な順番でπ[u]<π[v]（親から根まで逆順にさかのぼったパスが辞書的に先行する場合）

または、

・π[u]=π[v]のとき、T において u が preorder で v より先に探索される場合

3. T において、ノード u の子供 u1,u2, .., uz のとき、三つ組 s[u1],s[u2], ... , s[uz]は、S の中で各配列において連続してこの順番に現れる。さらに、その部分配列 Slast[u1, ... ,uz]をとるとノード u が最後の子供か否かを符号化している。つまり 1<=i<z において Slast[uz]=1 でそれ以外は Slast[ui]=0。

4. T において 2 つのノード u、v が同じラベル alpha[u]=alpha[v]とする。ノード u の三つ組が S の中で v の 3 つ組に先行する場合、u の連続する子供のブロックは、v の連続する子供のブロックに先行する。

ここで補助的なインデックスである F と A の関係について述べる。A は、ソートされた Spai の変わり目の先頭に 1 を立てる。注意点としては、ある項の親を求めたいとき､それまでの A の 1 の合計が親の id となる。上の例でいえば、id が 6 である a の親のタグは、A のフラグの合計の 2 が id となる B が親のラベルとなる。このときリーフはスキップしてカウントする。id が 16

のbの親のラベルは、Aのフラグの合計が6となるが、idが6のaではなく、idが7のEが親となる。Fは、子どもの範囲の先頭のidを指す。

次にXBW変換の上の関数について説明する。基本的な関数として次のrankとselectが定義されている。

・rank(A,i,l):配列Aの中でのi番目までのラベルlの頻度を返す。

・select(A,i,l):配列Aの中でのi番目のラベルlの位置を返す。

また、XBWでは、上の2つの関数を用いて木を探索するTree Navigation関数が定義されている。

・GetParent(i):ノードuのSでの親の位置を返す。iはSでのノードuの配列の番号。

・GetChildren(i):ノードuのSの中での子供の配列の範囲を返す。

・GetRankedChild(i,k):ノードuのk番目の子供のSでの位置を返す。

・GetCharRankedChild(i,c,k):ラベルがcであるようなノードuの子供のk番目の子供の位置を返す。

・GetDegree(i):ノードuの子供の数を返す。

・GetCharDegree(i):ラベルがcであるようなノードuの子供の数を返す。

・SubPathSearch(pai):パスpaiをラベルc1,c2, ... ,ckで表す。この操作は、Tでは、パスpaiの中での中間の子孫のノードをS[First,Last]で表す。

以下では、上の中で代表的な3つの関数のアルゴリズムと例を示す。

・GetParent(i)

```
1.  If (i==1) then return -1;
2.  c = rank(A,i,1)
3.  y = select(A,c,1)
4.  k = rank(Slast,i-1,1)-rank(Slast,y-1,1)
5.  p = select(Salpha,k+1,c)
6.  return p;
```

(例)図1でGetParent(8)を求める。2.からc=2。(すなわち一つ上のラベルは、B) 3.からy=5。4.ではk=2-1=2。求める親はS[5,8]の中で2番目のブロックとなる。p=select(Salpha,2,B)=4。

・GetChildren(i)

```
1.  If (Salpha[i]∈ΣL) then return -1;
2.  c =Salpha[i];
3.  r = rank (Salpha,i,c);
4.  y = select (A,c,1);
5.  z = rank(Slast,y-1,1);
6.  FIRST=select(Slast,z+r-1,1)+1;
7.  LAST=select(Slast,z+r,1);
8.  return (FIRST,LAST)
```

(例)GetChildren(3)を求める。2.からc=Salpha[3]=C。r=1。y＝select(A,3,1)=F[C]=9。z=rank(Slast,8,1)=3。First=select(Slast,3,1)+1=9。Last=select(Slast,4,1)=11。S[9,11]がc=Salpha(3)=Cの子供の範囲。

・SubPathSearch(pai)

```
1.  FIRST=F(c1);LAST=F(c1+1)-1;
2.  If (FIRST>LAST) then return
    "Π is not a subpath of T";
3.  For (i=2,3,4,...,k) do
4.    k1=rank (Salpha, First-1,ci);
      z1=select (Salpha, k1+1,ci);
5.    k2=rank (Salpha, Last,ci);
      z2=select (Salpha, k2,ci);
6.  if(z1>z2) then return "Π is not a subpath of T";
7.  First=GetRankedChild (z1,1);
8.  Last=GetRankedChild (z2,GetDegree(z2));
9.  Return (First,Last);
```

(例) pai=BDとする。c1=B。F[B]=5。Last=F[c]-1=8。S[5,8]は、subpathB の子孫の存在するノード。次の段階は、c2=D。k1=0。k 2=2。Z1=5。Z2=8。S[5]の最初の子供は、S[12]。S[8]の最後の子供は、S[13]。l pれこれから答えはS[12,13]を返す。

### 2-2. Structural join

構造問合せの評価手法の一つである structural join においては, 問合せを2項間の親子(P-C)関係や祖先子孫(A-D)関係に分解し, これらの各関係を与えられた二つのノードのみを見て判定することが基本となる。Structural join や holistic twig join の論文では、各ノードを(DocID, StartPos, EndPos, LevelNum)の四つの属性で表し、DocID、StartPos、 EndPosによる索引付けを行う。

DocID:ドキュメントのNo.

StartPos:preorderでの最初に探索した順番

EndPos:preorderで最後に探索した順番

LevelNum:階層の番号。Rootは1から始まる。

例として図1で

A [B=‘a’]//D [. =‘c’]

という問合せを考える。この問合わせは

A-B (親子関係の判定)

B-a (親子関係の判定)

A=D(先祖子孫関係の判定)

D-c (親子関係の判定)

に分解される。ここで二つのノードとして

ノード1：(D1,S1:E1,L1)

ノード2：(D2,S2:E2,L2)

があったとする。判定基準は

1) 先祖子孫関係は

D1=D2,S1<S2 かつ E2<E1 の時

2) 親子関係は

D1=D2,S1<S2,E2<E1 かつ L1+1=L2 の時である。

A：(1,1:25,1)、D:(１,21:23,3)のとき

D1=1=1=D2、S1=1<21=S2,かつ E2=23<25=E1 からこれは先祖子孫関係の条件を満たす。

B:(1,2:10,2)

D1=1=1=D2、S1=1<2=S2、E2=10<25=E1 かつ

L1=1+1=2=L2

から A と B は親子関係を満たす。

代表的なアルゴリズムとして次の４つがある。

- Tree-Merge-Anc
- Tree-Merge-Desc
- Stack-Tree-Desc
- Stack-Tree-Anc

## 3. 提案手法

親子関係、祖先子孫関係について XBW 特有の性質を利用するアルゴリズムを評価する。図 2 で提案手法を図示する。親子関係(a/b)では、XBW 変換は XBW 変換されたデータ表現上では、ある親ノードに対応する解となる子ノードが一箇所に固まって現れる。また、ある親ノードは、ねじれの関係は生じない。そのため、深さ優先探索より有利と考えられる。この性質を利用した XBW 変換されたデータ表現上での a/b の評価手法として以下の手法を考え、これらの性能評価・比較を行う。

提案手法1. Salphaをスキャンしてaを見つけてgetChildrenを行う。

提案手法 2. F を使い、Salpha の中の「a の子供」に対応する範囲を計算し、そのなかから Salpha=b のものを抽出し、b 側から getParent を行う。

| id | Slast | Salpha | Spai | SP | EP | L | A | F |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | A | empty | 1 | 25 | 1 | 0 | 2 |
| 2 | 0 | B | A | 2 | 10 | 2 | 1 | 5 |
| 3 | 0 | C | A | 11 | 19 | 2 | 0 | 9 |
| 4 | 1 | B | A | 20 | 24 | 2 | 0 | 8 |
| 5 | 0 | D | BA | 3 | 5 | 3 | 1 | 12 |
| 6 | 0 | a | BA | 6 | 6 | 3 | 0 | -1 |
| 7 | 1 | E | BA | 7 | 9 | 3 | 0 | 16 |
| 8 | 1 | D | BA | 21 | 23 | 3 | 0 | 13 |
| 9 | 0 | D | CA | 12 | 14 | 3 | 1 | 14 |
| 10 | 0 | b | CA | 15 | 15 | 3 | 0 | -1 |
| 11 | 1 | D | CA | 16 | 18 | 3 | 0 | 15 |
| 12 | 1 | a | DBA | 4 | 4 | 4 | 1 | -1 |
| 13 | 1 | b | DBA | 22 | 22 | 4 | 0 | -1 |
| 14 | 1 | c | DCA | 13 | 13 | 4 | 1 | -1 |
| 15 | 1 | c | DCA | 17 | 17 | 4 | 0 | -1 |
| 16 | 1 | b | EBA | 8 | 8 | 4 | 1 | -1 |

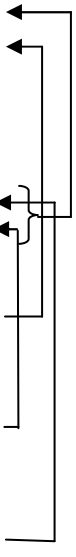


図２．提案手法

一方，祖先子孫関係(a//b)においては，XBW変換されたデータ表現では、ある祖先ノードに対応する解となる子孫ノードは、一箇所には固まっていない.また、祖先ノード間の前後関係と、対応する子孫ノードの間の前後関係については、全体としてはねじれているが、各祖先ノードをその親ノードから根までの逆順パスが同じであるようなものにグループ分けして考えた場合、同じグループの中ではねじれが生じない。祖先子孫関係の判定は、親子関係の判定を繰り返す。

提案手法3 Salphaをスキャンしてaを見つけ、リストを作る。そのリストに対してgetchildrenをして子供をみつけ、子のリストを作る。子のリストに対して、スキャンを行い、bを探す。子全員に対してGetchildrenを行い、孫のリストを作成し、 b のスキャンを行う。以下全てリーフになるまで繰り返す。

提案手法4 Salphaをスキャンして、bを探し、リストを作成する。リスト全員に対して、getparentを行う。その親のリストを作成、aを探す。更にそのリスト全てに対してgetparentを行い、rootになるまでgetparentを行い、先祖のリストに対してaを探す。

**XBW変換での親子関係の判定**

親子関係B/Dの判定

提案手法1.

1. Salphaで先頭からBを探索する。
   Salpha[2]とSalpha[4]でBを得る。
2. これに対してGetChildrenを行う。
   GetChildren(2)=[5,7]
   GetChildren(4)=[8,8]
3. 上で得られた範囲でSalpha[5]=Dを探索する。

提案手法2.

1. Salphaの中からDを探す。idが、5、8、9、11のリストができる。
2. 1.で得られた範囲でgetparentを行う。親のリストは、それぞれ、idが5→2、8→４、9→3、11→3となる。その範囲からBをさがす。
3. 抽出した候補からBを探す。２、４が答えとなる。

**XBW変換での祖先子孫関係の判定**

祖先子孫関係B//bの判定

提案手法3.

1. 祖先のリスト作成
   Salphaで先頭からBを探索しリスト作成する。
2. 子孫のリスト作成
   Bの子供の範囲からGetChildrenから子孫のリストを作成し、そのリストでbを探す。
3. 更に、子のリストに対してgetchildrenを行いBを探す。

提案手法4

1.子孫のリストの作成

Salphaをスキャンして、bを探す。子孫リストを作成する。idの10,13,16がリストとなる。

2.親リストの作成

リストに対してgetparentを行う。

10→3、13→8、16→7が親である。Bはない。

3.先祖のリスト

リストをgetparentすると、7→2、8→2となりidが2のBが答えとなる。

最後に先祖子孫のねじれの関係について述べる。図3では、親子関係では矢印は交差していないが、祖先子孫関係では、7→16、8→13で交差している。これがねじれの関係である。深さ優先探索では、逆に祖先子孫関係では、ねじれの関係がないが、親子関係では、ねじれが生じる。

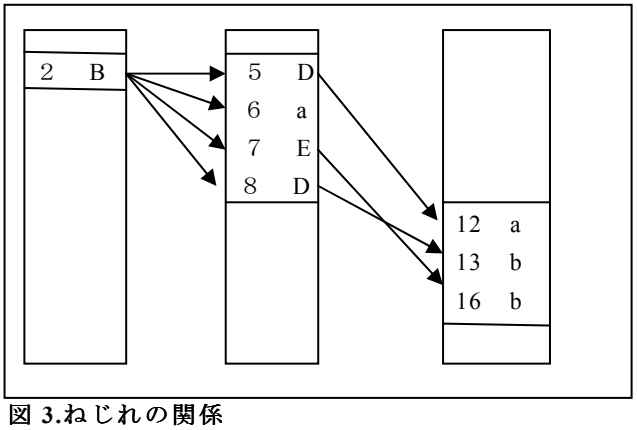
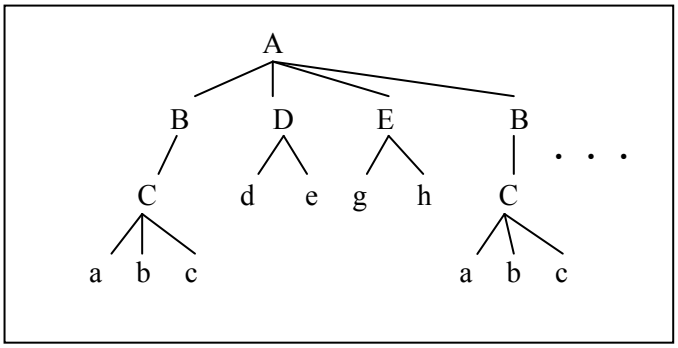



**図 3.ねじれの関係**

## 4. 計算機実験

計算機実験は、構造問合せの応答時間を対象とする。データは人工データを用いる。下記の XML を 1 単位として反復してデータを作成する。

**図.3 XML データの 1 単位**




1 単位のノード数は、12 ノードであり、この単位を繰り返して、1002 ノード、3004 ノード、5006 ノード、7008 ノード、10011 ノードのデータを作成した。比較対象として[8]の論文のアルゴリズムである Stack-tree-anc、Tree-merge-anc、Tree-merge-desc の 3 つのアルゴリズムを実装して比較した。

計算機環境は、CPU:Core 2 Duo 2.53GHz, メモリ 4GB、OS:winXP を用いる。

**表 1．実験で用いた問合せ**

| | |
|---|---|
| A/B | A//C |
| B/C | B//a |
| C/a | A//g |
| D/e | |
| E/h | |

実験結果としては、提案手法 1,2 はオリジナルの structural join の論文[8]の実行結果から 10 倍から 200 倍ほど遅かった。データサイズは、2 倍ほど大きい。提案手法 1 と提案手法 2 を比較してみると、親子関係の判定では、提案手法 1 の方が提案手法 2 より早く。祖先子孫関係の判定では、提案手法 2 の方が早い。計測時間が 0 となっている時間は、計測値そのままである。オリジナルの structural join の論文[8]の手法は、親子関係と祖先子孫関係に関係なく一様に高速である。これは、親子関係では、StartPos と EndPos による関係と LevelNum による木構造の制約が判定条件となるが、祖先子孫関係の判定では、StartPos と EndPos による関係による制約のみで判定するため、ノード数が多くなっても判定の速度に変化がない理由である。

## 5. まとめ

今回は、構造問合せのなかで親子関係の判定、祖先子孫関係の判定の性能評価を行った。実験結果は、オリジナルの論文より遅かったが、XBW 変換の特徴は、高い木構造上での高いナビゲーション機能である。今後の課題としては、今回 2 項でのみ問合せについて計算実験を行ったが、3 項以上での問い合わせでの実験、また、実データでの計算機実験、そのほか holistic twig Join など問合せの実装とその性能評価があげられる。また、基本的な関数である rank,select に関する検討も必要であると考えている。さらに、計算量からの考察も今回は不十分である。プログラム変換という立場からも計算量が変化する珍しいプログラム変換の例であると言われている。より高速化には、XBW の解の集合状態を生かすアルゴリズムが必要だが、今後の課題としたい。XBW は簡潔データ構造の 1 つだが、簡潔データ構造は圧縮などの研究は活発に行われているが、その上の問合せの研究はあまりやられていない。本研究の目的は、その問合せについての方向性を示すことでもある。

| node | query | Method1,3 | Method2,4 | STD | MTA | MTD |
|---|---|---|---|---|---|---|
| 1002 | | | | | | |
| | A/B | 0.01 | 0.02 | 0 | 0 | 0 |
| | B/C | 0.03 | 0.02 | 0 | 0 | 0 |
| | C/a | 0.03 | 0.02 | 0 | 0 | 0 |
| | D/e | 0.01 | 0.01 | 0 | 0 | 0 |
| | E/h | 0.02 | 0.03 | 0 | 0 | 0 |
| | A//C | 0.05 | 0.04 | 0 | 0 | 0 |
| | B//a | 0.03 | 0.03 | 0 | 0 | 0 |
| | A//g | 0.05 | 0.03 | 0 | 0 | 0 |
| 3004 | | | | | | |
| | A/B | 0.04 | 0.07 | 0 | 0 | 0 |
| | B/C | 0.05 | 0.07 | 0 | 0.01 | 0 |
| | C/a | 0.08 | 0.07 | 0.01 | 0.01 | 0.01 |
| | D/e | 0.05 | 0.07 | 0.01 | 0.01 | 0.01 |
| | E/h | 0.05 | 0.06 | 0.01 | 0.01 | 0.01 |
| | A//C | 0.26 | 0.13 | 0 | 0 | 0 |
| | B//a | 0.11 | 0.14 | 0.01 | 0.01 | 0.01 |
| | A//g | 0.27 | 0.15 | 0 | 0 | 0 |
| 5006 | | | | | | |
| | A/B | 0.02 | 0.13 | 0.01 | 0 | 0.01 |
| | B/C | 0.14 | 0.15 | 0.02 | 0.02 | 0.02 |
| | C/a | 0.15 | 0.14 | 0.02 | 0.01 | 0.02 |
| | D/e | 0.11 | 0.14 | 0.02 | 0.02 | 0.01 |
| | E/h | 0.11 | 0.16 | 0.02 | 0.01 | 0.01 |
| | A//C | 0.66 | 0.36 | 0.01 | 0.01 | 0.01 |
| | B//a | 0.26 | 0.36 | 0.03 | 0.02 | 0.01 |
| | A//g | 0.65 | 0.37 | 0 | 0 | 0.01 |
| 7008 | | | | | | |
| | A/B | 0.05 | 0.23 | 0 | 0.01 | 0.01 |
| | B/C | 0.18 | 0.25 | 0.02 | 0.02 | 0.02 |
| | C/a | 0.28 | 0.32 | 0.02 | 0.01 | 0.02 |
| | D/e | 0.18 | 0.28 | 0.02 | 0.01 | 0.01 |
| | E/h | 0.17 | 0.28 | 0.02 | 0.02 | 0.01 |
| | A//C | 1.26 | 0.67 | 0 | 0.01 | 0 |
| | B//a | 0.45 | 0.7 | 0.02 | 0.01 | 0.02 |
| | A//g | 1.26 | 0.69 | 0 | 0.01 | 0 |
| 10011 | | | | | | |
| | A/B | 0.05 | 0.52 | 0.01 | 0.01 | 0.01 |
| | B/C | 0.33 | 0.49 | 0.04 | 0.02 | 0.02 |
| | C/a | 0.51 | 0.59 | 0.04 | 0.03 | 0.02 |
| | D/e | 0.34 | 0.57 | 0.04 | 0.02 | 0.03 |
| | E/h | 0.34 | 0.54 | 0.05 | 0.02 | 0.04 |
| | A//C | 2.53 | 1.27 | 0.01 | 0.01 | 0.01 |
| | B//a | 0.89 | 1.34 | 0.04 | 0.03 | 0.02 |
| | A//g | 2.53 | 1.39 | 0.01 | 0.01 | 0.01 |

**表 2.実験結果(単位:秒)**

| | XBW+A | XBW+F | STJ[8] |
|---|---|---|---|
| 1000 | 14 | 15 | 14 |
| 3000 | 43 | 48 | 21 |
| 5000 | 73 | 81 | 35 |
| 7000 | 102 | 104 | 49 |
| 10000 | 146 | 163 | 70 |

**表 3.データサイズ(単位:KB)**

## 謝辞

XBW 変換に関する論文を御教示して頂ました定兼邦彦先生に深く感謝いたします。また、コメンテーターの先生方には、コメントと激励のお言葉を頂き、誠に有難うございます。まだまだ不十分な研究ですが、今後の研究の糧としていきたいと考えています。

## 参 考 文 献

[1] 東原正智，田島敬史，“問い合わせ処理に適したＸＭＬデータ圧縮形式に関する研究”，DEWS 2004.

[2] LIEFKE, H. AND SUCIU, D.: XMill: an efficient compressor for xml data. In Proc. ACM SIGMOD International Conference on Management of Data, 2000.

[3] TOLANI, P. M. AND HARITSA, J. R.: XGRIND: A query-friendly XML compressor. In Proc. 18th International Conference on Data Engineering (ICDE).

[4] 定兼 邦彦,” 大規模データ処理のための簡潔データ構造”, 情報処理 48(8), 899-902, 2007.

[5] FERRAGINA, P., LUCCIO, F., MANZINI, G., AND MUTHUKRISHNAN, S., : Structuring labeled trees for optimal succinctness, and beyond. In Proc. 46th IEEE Symposium on Foundations of Computer Science, FOCS, 2005.

[6] FERRAGINA, P., LUCCIO, F., MANZINI, G., AND MUTHUKRISHNAN, S.: Compressing and searching XML data via two zips. In Proc. 15th International World Wide Web Conference (WWW). 2006.

[7] FERRAGINA, P., LUCCIO, F., MANZINI, G., AND MUTHUKRISHNAN, S.: Compressing and indexing labeled trees, with applications. Journal of the ACM, 57(1), 2009.

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

[9] Atsuyuki Morishima, Keishi Tajima, Masateru Tadaishi: Optimal tree node ordering for child/descendant navigations. ICDE, 2010.