社団法人 情報処理学会 研究報告
IPSJ SIG Technical Reports

2007－DBS－143 (16)
2007／7／2

木直列化を用いたXMLデータの類似結合

文 連子† 天笠 俊之†† 北川 博之††

† 筑波大学大学院システム情報工学研究科コンピュータサイエンス専攻
〒305-8573 茨城県つくば市天王台 1-1-1
† 筑波大学計算科学研究センター
E-mail: moon@kde.cs.tsukuba.ac.jp, {amagasa, kitagawa}@cs.tsukuba.ac.jp

あらまし 本稿では，XMLデータの木構造をノードの系列に直列化する手法を用い，系列同士の類似度によってXMLデータの類似結合を行う手法を提案する．近年のXMLデータの急速な普及により，多くのデータがXML形式で記述されるようになっている．その結果，類似した内容であるにも関わらず，異なるマークアップ語彙や異なる構造を持つXMLデータが増加している．これらの相補的な情報を統合するための手段の一つが類似結合 (similarity join) である．本研究で提案する類似結合の概要は以下のとおりである．1) 結合しようとする二つのXMLデータを，ノードの系列として直列化する，2) 得られた系列から，構造および内容に関して意味的にまとまりのある部分系列を抽出する，3) 部分系列同士のテキスト情報を用いて，マッチする可能性のある部分系列のペアを抽出する，4) 得られたペアから，構造的に類似している系列を抽出する．テキスト同士の類似性を比較するために，確率的な集合要素判定手法であるbloomフィルタを用いる．また，系列化されたXML部分木の構造の類似性を判定するために編集類似度を用いる．
キーワード XML，類似結合，XMLデータの直列化，bloomフィルタ

An Approach for XML Similarity Join using Tree Serialization

Lianzi WEN†, Toshiyuki AMAGASA††, and Hiroyuki KITAGAWA††

† Department of Computer Science
Graduate School of Systems and Information Engineering
University of Tsukuba
1-1-1 Tennodai, Tsukuba 305-8573, Japan
† Center for Computational Sciences, University of Tsukuba
1-1-1 Tennodai, Tsukuba 305-8577, Japan
E-mail: moon@kde.cs.tsukuba.ac.jp, {amagasa, kitagawa}@cs.tsukuba.ac.jp

Abstract In this paper we propose a scheme for similarity join over XML data based on XML data serialization and subsequent similarity matching over XML node subsequences. Due to the recent explosive diffusion of XML, large amount of electronic data are now marked up with XML. As a consequence, there are growing number of XML data which represent similar contents, but have dissimilar structures. In order to extract as much information as possible from those heterogeneous information, *similarity join* has been used. Basically, our (proposed) similarity join for XML data can be summarized as follows: 1) we serialize XML data as XML node sequences; 2) we extract semantically/structurally coherent subsequences; 3) we filter out dissimilar subsequences using textual information; and 4) we extract pairs of subsequences as final result by checking structural similarity. We make use of the *bloom filter*, which is a probabilistic method for set member test, to measure similarities over texts, and the *edit similarity* to measure structural similarities over node sequences.
Key words XML, similarity join, XML data serialization, bloom filter

1. はじめに

XML (Extensible Markup Language) [1] は，1998 年に World Wide Web Consortium (W3C) によって勧告された，データ交換のためのメタ言語である．タグによって，文書やデータの意味や構造を階層的にマークアップし，木構造やグラフ構造を記述することができる．仕様が簡明であることとクリアテキストでデータ構造を記述できることから，現在，その応用範囲は急速に広がっており，ウェブデータや文書だけではなく，ビジネスデータ，科学データ，ログなど多くの応用で利用されるようになっている．今後も，XML の応用範囲は広がることが予想される．

このような背景から，類似した内容を含みながら異なる構造を持つ XML データが増えつつある．例えば，ACM，IEEE (CS)，情報処理学会，電子情報通信学会など，多くの学会で行なわれている電子図書館サービスを考える，そこで提供される文献データベースは多くの場合，XML 形式でも公開されている．このとき，各文献情報について，多くの情報は重複しているが，あるデータベースにしか含まれていない固有の情報も存在する．図 1，2 の二種類の XML データを考えると，(1) には出版社の情報 (**publisher**) が含まれており，(2) にはウェブサイトの情報が含まれている．これらのデータベースからより多くの情報を引き出すためには，相補的な情報を統合することが重要である．

この目的のために用いられる演算が，類似結合 (similarity join) である．類似結合は，主に関係データベースの分野で活発に研究が行なわれてきたが，最近では XML データに応用した研究がいくつか見られる．このため，今後，XML データが広く普及するに従い，その重要度が増すことが予想される．XML データの類似結合を実現するに当たっては，1) XML データから適切な単位で情報を抽出する，2) XML データの内容および構造の異種性の取り扱いが技術的課題になる．1) については，構造が固定していて同様の構造を持った要素が繰り返し表れるような，いわゆるデータ系の XML であれば比較的容易であるが，構造がゆるやか，あるいあ再帰的な構造を持つような文書系の XML の場合は難しい問題である．2) ついては，例えば，図 1 において，著者は **authors** タグで表現されているが，図 2 ではより詳細な表記になっている．また，図 1 では，出版年を **year** 属性で表現しているが，図 2 では要素となっている．マークアップ語彙の違いだけでなく，要素の出現順や XML データの木構造が異なっている場合もある．このように，内容および構造に異種性があっても，類似した情報であれば適切に結合処理を行うことができなければならない．

本研究では，XML データの類似結合を目的として，木直列化を用いた手法を提案する．木直列化とは，XML 等の木構造をある決められた順序で巡回することによって，ノードの系列に変換することである．これによって，木構造から意味的，構造的にまとまった部分木を切り出す操作は，部分木に相当する部分系列の抽出として実現できる．すると，類似結合演算は部分系列同士の類似性判定問題となる．部分系列同士の類似度を，まずテキスト情報と構造情報に分けて考え，テキスト情報が類似していると判定されたものに対してのみ構造情報の類似性判定を行なう．テキスト情報の類似性判定には，確率的な集合要素判定法である Bloom フィルタを利用する．構造情報の比較は，編集距離にもとづいて計算される編集類似度を利用する．

本稿の構成は以下の通りである．第 2 章では関連研究について述べ，第 3 章では提案手法について詳しく説明する．第 4 章では評価実験について説明し，最後に第 5 章において，まとめと今後の課題について述べる．

2. 関連研究

Liang 等 [4] は，XML データの類似結合手法を提案している．まず，XML データを意味的にまとまりのある部分木に分割する．各部分木毎に完全に一致するテキストノードを調べ，その割合が一定以上のペアを結果候補とする．さらに，各テキストノードについて，その根からのパス式を調べ，共通するタグ名の割合がある閾値以上であるものを結合する．本手法では，テキストノードのペアを完全一致によって求めている．また，XML データの構造はパス式のみを考えている点が本研究とは異なる．

Weis 等 [5] は，XML データのデータクリーニングを目的と

```
<textbook>
  <title>Database Systems: The Complete Book</title>
  <authors>H. Garcia-Morina, J. Ullman, J. Widom</author>
  <bibinfo>
    <publisher>Prentice Hall</publisher>
    <year>2001</year>
    <isbn>123</isbn>
  </bibinfo>
</textbook>
```

図 1 文献 XML データ (1)
Fig. 1 Bibliographic data in XML (1)

```
<book year="2001">
  <title>Database Systems: The Complete Book</title>
  <authors>
    <author>
      <fname>Hector</fname>
      <lname>Garcia-Morina</lname>
    </author>
    <author>
      <fname>Jeffrey</fname>
      <lname>Ullman</lname>
    </author>
    <author>
      <fname>Jennifer</fname>
      <lname>Widom</lname>
    </author>
  <authors>
  <url>http://infolab.stanford.edu/~ullman/dscb.html</url>
</book>
```

図 2 文献 XML データ (2)
Fig. 2 Bibliographic data in XML (2)

して，あるXMLデータ内で重複しているXMLオブジェクトを見つける手法を提案している．ここでいうXMLオブジェクトとは，意味的にまとまりを持った部分木のことである．XMLデータの木構造を辿りながら各要素単位で比較を行うことで，重複オブジェクトを探索する．このとき，ある要素が類似しているとは，要素名が一致し，かつその親ノード，子ノードの構造が類似することが求められる．ある要素ノードにおいて，その子要素の類似度が低い場合は，それらの，算は行わない．

三木等[6]は，XMLデータのキーワード検索に，スーパーインポーズドコーディングを利用している．これは，XML木のリーフノード毎に割り当てたビット列を，階層構造に従って足し込んで行く．これにより，検索キーワードの包含判定をビット演算で行うことが可能となる．本研究では，部分木に含まれるキーワードの包含判定に，Bloomフィルタを用いている点が類似している．

3. 提案手法

本稿で提案するXMLデータの類似結合手法の概要は以下の通りである．二つのXMLデータが与えられるとする．

（1） XMLデータを直列化し，ノード系列に変換する．直列化にはNoK (Next-of-Kin) パタン[8]をベースに，ポストオーダ（後置順）でXMLデータを巡回して得られるノード系列を用いる．

（2） ノード系列を先頭から走査し，テキスト情報および構造について意味的にまとまりのある部分系列を抽出する．

（3） 部分系列の各組合せについて，テキスト情報の類似度を計算し，ある閾値以上のペア候補として残す．

（4） 上で残ったペアについて，次は木構造の類似度を計算し，ある閾値以上のものを最終的な結果とする．

3.1 XMLデータの直列化

XMLデータは木構造を有するため，各種処理（検索や類似度計算）が複雑になってしまうという問題がある．この問題に対応する一つの方法が，木構造をノードの系列に直列化するアプローチである[7]〜[9]．基本的なアイデアは次の通りである．XMLデータを前置順 (preorder)，後置順 (postorer)，あるいはその他の順序で巡回し，木構造をノードの系列に直列化する．ViSTでは前置順，PRIXではPrüfer系列を用いている．さらに，問合せパタン（マッチしたいXMLデータ）も同様の方法で直列化する．すると，問合せ（構造マッチング）の問題は，XMLノード系列の部分一致問題となる．

本研究では直列化手法として，NoKパタンをベースにした方法を用いる．オリジナルが前置順を用いているのに対して後置順を用いる．これは，後置順を用いると，リーフノードに近いノードが系列上で近い位置に配置されるからである．以下の議論では，後置順に基くNoKパタンを**後置*NoK*パタン**と呼ぶことにする．

一般に，木構造を単にノード系列に直列化したたけでは，構造に関する情報が失われてしまう．(注1) 例えば，「<a><b /><c /></c>」なるXML木を考える．ここで後置順は「bca」であるが，「<a><c><b /></c>」の後置順も同様に「bca」となることから，構造情報が失われてしまっていることが分かる．ここで，「a」，「b」，「c」，はそれぞれ，タグ名に対応する識別子である．後置NoKパタンでは，木構造を表現するため，補助記号（カッコ）を導入する．例えば，上述のXMLデータはそれぞれ「((b)(c)a)」，「(((b)c)a)」と表現できる．ここで，後置順においては「)」を省略しても構造の一意性は保たれるので，「((b(ca」，「(((bca」を後置NoKパタンとする．大量のXMLデータの直列化において，閉じ括弧を省略することは，記憶領域の節約に繋がる．

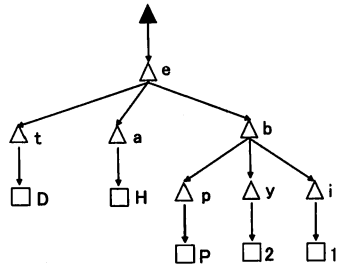

図3 XMLデータ
Fig. 3 An XML data

図4 後置NoKパタン
Fig. 4 Post-order NoK pattern

図3の木構造を，Nokパータンをベースにノード系列に変換すると，図4のようになる．図の上部にある折れ線は，XMLデータ上のレベルを表している．後置NoKパタンは次のような性質を持つ．1) テキストノードは，その性質から，必ず折れ線の最下部に出現する．2) 親ノードは，自分から一段上の右側にある点である．例えば，「y」の親ノードはその段から一段の右側にある点であるが，その点は「(」になっている．その段をまた右に行くと「b」の点があるので，「y」の親は「b」であることが分かる．

3.2 XMLデータの分割

直列化されたXMLデータは，一本の長大なノード系列となるが，ノード系列同士の類似結合を行なうために，ノード系列から，XML部分木に相当する部分系列を抽出する．このとき，部分系列はXMLデータが表現する意味や構造的にまとまりのある部分を抽出できることが望ましい．

基本的な考え方は次の通りである．ある閾値τとhを与える．τは，部分木が最低限含まなければならないテキストノードの個数を表わし，hは，部分木が最低限持たなければならない根からの深さである．後置NoKパタンにおいて，「(」をカウントすることによって現在注目しているノードの深さを計算

(注1)：Prüfer系列は，構造情報を保持したまま直列化できる．

```
分割アルゴリズム　Div(e)
Input : {str,τ, h}
  nok : NoK パタン;
  τ : テキストノード数の下限;
  h : 深さの下限;
Output : S(s1, s2, ... sn) 部分系列の集合;
Procedure  Div(e)
1 :  n = 1; p = 1; d = 1;
2 :  start = 1; countText = 0; countDrpth = 0;
3 :  while(nok){
4 :      node = nok の p 番目の要素;
5 :      if(node = textNode){
6 :          countText + +;
7 :          if((countText ≧ τ)&&(countDepth ≧ h)){
8 :              sn = start から node までの部分系列;
9 :              Output(sn);
10 :             p = snの最後の要素;
11 :             n + +;
12 :             start = p;
13 :             countText = 0;
14 :         }
15 :     }else  if(node = "(''){
16 :         countDepth + +;
17 :     }else  countDepth − −;
18 :     p + +;
19 : }
```

することができるので，これによって部分木の切れ目を判定する．図 3.2 にアルゴリズムを示す．

3.3 部分系列同士の類似度計算

切り出された（XML 部分木に相当する）部分系列同士の類似度を定義する．これに基づき，類似度が一定以上となる部分系列のペアを探すことによって，最終的な目標である類似結合を行なう．XML データは，テキスト情報と構造情報を持つので，その両方を考慮した類似度を与える必要がある．本研究では，以下の方針で行なう．

- 部分木に現れるテキストノードおよび属性に含まれる全ての単語のうち，比較しようとする部分木に共通して出現する単語の割合が一定以上のものを，テキスト情報が類似していると判定する．この計算は素朴な方法で計算してしまうと高コストになってしまうので，本研究では，確率的な集合要素判定法である Bloom フィルタ [10] を用いる．
- テキスト情報が類似していると判定された部分系列について，次は構造が類似しているかどうかを判定する．構造情報は後置 NoK パタンとして系列表現されているので，構造の類似性判定も，系列同士の類似性判定を行えばよい．従って本研究では，編集距離に基づいて算出される編集類似度によって構造の類似度を測る．

3.3.1 テキスト情報の比較

Bloom フィルタは，空間効率の良い確率的データ構造であり，集合の要素判定をビット演算で実行することができる．本手法では，ある部分木に含まれる全てのテキスト（属性）ノードに含まれる単語について，Bloom フィルタによるビット列への変換を行なう．このようにして，ビット列同士の演算でテキストノード同士の類似性判定を行う．

Bloom フィルタでは，ある要素を k 個のハッシュ関数を使って，長さ m の固定長ビット列にマッピングする．各々の単語から生成されたビット列は論理和 OR を取って，一つのビット列にする．例えば，部分木に含まれている単語 "database"，"system"，…，"book" を 2 個のハッシュ関数を使って以下のように一つのビット列に変換できる．

$h_1(database)$	001000000000 … 00000000000
$h_2(database)$	000000000100 … 00000000000
$h_1(system)$	001000000000 … 00000000000
$h_2(system)$	000000000000 … 00100000000
$h_1(book)$	000000000000 … 00000000100
$h_2(book)$	000000000000 … 00000000000
ビット列 (OR)	001000000100 … 00100000100

このように得られたビット列を用いると，二つのテキスト情報同士の類似度は，両者が共通に持っている単語数の比として，ビット演算で近似的に計算できる．A.sig と B.sig をそれぞれ部分文書 A と B のビット列表現とすると，

$$Sim(A.sig, B.sig) = \frac{count(A.sig\ AND\ B.sig)}{count(A.sig\ OR\ B.sig)} \qquad (1)$$

ここで count() は，ビット文字列上の "1" の数を数える関数である．この類似度が閾値 α 以上であれば，テキストが類似すると判定する．

3.3.2 構造情報の比較

テキスト情報が類似していると判定された部分系列について，次は構造が類似しているかどうかを比較する．NoK パタンは，構造情報も保持しているので，構造的に類似していれば NoK パタンも類似することになる．このことから，NoK パタンの系列同士の類似度を，編集距離によって評価する．まず，編集距離 (Edit Distance) は二つの文字列がどの程度異なっているかを示す数値である．具体的には，文字の挿入，削除，置換によって，一つの文字列を別の文字列に変形するのに必要な手順の最小回数として与えられる．編集類似度 (Edit Similarity) [11] は，編集距離から求められる類似度である．二つの文字列 σ_1 と σ_2 の編集距離を $ED(\sigma_1,\sigma 2)$ としたとき，二つの文字列の編集類似度 $ES(\sigma_1,\sigma_2)$ は，以下のように計算できる．

$$ES(\sigma_1,\sigma_2) = 1.0 - \frac{ED(\sigma_1,\sigma_2)}{max(|\sigma_1,\sigma_2|)} \qquad (2)$$

二つの文字列が類似するほど編集類似度も高い値となる．例えば，

```
d1: <b>          d2: <b>
      <y />            <t />
      <t />            <a />
      <a />            <u />
    </b>             </b>
```

の d1, d2 の二つの（部分）XML データを考える．後置 NoK パタンはそれぞれ，d1: ((y(t(ab, d2: ((t(a(ub となる．この編集類似度は以下のようになる．

d1	(	(	y	(	t	(	a		b
d2	(	(			t	(	a	u	b
コスト	0	0	1	1	0	0	0	1	0

ノード系列 d1 と d2 の編集距離は 3 となり，編集類似度は $ES(d1, d2) = 1.0 - \frac{3}{8} = 0.625$ になる．計算されたノード系列同士の編集類似度が，ある閾値 β 以上であれば類似していると判定する．

最後に，二つの部分ノード系列のテキスト情報とノード系列（構造）の類似度がそれぞれのある閾値以上であれば，類似するペアと判別する．

4. 予備実験

提案手法の妥当性と処理時間を検証するため，実験を行った．

4.1 実験環境

用いた計算機環境を表 1 に示す．

表 1 実験環境

CPU	2-way Dual Core AMD Opteron(tm) processor (2.4GHz)
OS	Sun OS 5.10.
メモリ	16GB
JAVA	J2SE 1.5.0 09
DB	PostgreSQL 8.1.4

実験データとして，SIGMODRecord.xml (464KB) [2], DBLP.xml (357,284KB) [3] を用いた．

本実験では，まず二つの XML データをノード系列に直列化し，構造的，内容的にまとまりのある部分系列を抽出した．その結果，SIGMODRecord.xml から 1,541 個，DBLP.xml から 828,276 個の部分系列を得た．そこで，予備実験として，SIGMODRecord.xml の 1,541 個と類似していると分かっているものを含む DBLP.xml の部分系列 1,000 個を選択し，提案手法を適用した．

4.2 実験結果

まず，部分木のテキストを Bloom フィルタによって 256 ビットのビット列に変換し，類似度が閾値以上となるペアを検索する実験を行った．その実行時間と検索されたペア数を表 2 のようになる．なお，Bloom フィルタで仕様するハッシュ関数の個数は 2〜5 の間で変化させた．

表 2 実験結果

	ハッシュ関数=2個		ハッシュ関数=3個		ハッシュ関数=4個		ハッシュ関数=5個	
	pair	time[ms]	pair	time[ms]	pair	time[ms]	pair	time[ms]
SIM(>0.3	1799	1983						
SIM(>0.4	199	1789						
SIM(>0.5	53	1794	257	1797	2968	2230		
SIM(>0.6	5	1774	18	1773	305	1800	2375	2095
SIM(>0.7	0	1776	1	1775	1	1772	182	1791
SIM(>0.8			0	1772	0	1776	1	1771
SIM(>0.9							0	1772

Bloom フィルタにおいて，ハッシュ関数をより多く使うと計算コストが増える代わりに精度が上がる傾向を示す．本実験では，ビット長を 256 ビットにしたが，実験データでは 5 個以上のハッシュ関数を使うと，精度が落ちることが分かった．これは，テキストノードの傾向にもよるが，多数の単語を含むような場合，1 の立つビット数が増えすぎてしまうことに起因すると思われる．なお，ハッシュ関数 2 個の場合で類似していると判定されたものについて目視で確認した結果，類似度が閾値 0.4 以上であるペアは概ね類似していた．

以下に，類似すると判定された 3 つのペアとその類似度を示す．

```
S.id = 258

<article>
 <title articleCode="202264">Starburst II:
       the extender strikes back!</title>
 <authors>
  <author authorPosition="01">Guy M. Lohman</author>
  <author authorPosition="02">George Lapis</author>
  <author authorPosition="03">Tobin Lehman</author>
  <author authorPosition="04">Rakesh Agrawal</author>
  <author authorPosition="05">Roberta Cochrane</author>
  <author authorPosition="06">John McPherson</author>
  <author authorPosition="07">C. Mohan</author>
  <author authorPosition="08">Hamid Pirahesh</author>
  <author authorPosition="09">Jennifer Widom</author>
 </authors>
<article>

 D.id = 407

<inproceedings mdate="2006-03-31" key="conf/sigmod/LohmanLLACMMPW91">
  <crossref>conf/sigmod/91</crossref>
  <author>Guy M. Lohman</author>
  <author>George Lapis</author>
  <author>Tobin J. Lehman</author>
  <author>Rakesh Agrawal</author>
  <author>Roberta Cochrane</author>
  <author>John McPherson</author>
  <author>C. Mohan</author>
  <author>Hamid Pirahesh</author>
  <author>Jennifer Widom</author>
  <title>Starburst II: The Extender Strikes Back!</title>
  <pages>447</pages>
  <cdrom>SIGMOD91/P447.PDF</cdrom>
  <year>1991</year>
  <booktitle>SIGMOD Conference</booktitle>
  <url>db/conf/sigmod/sigmod91.html#LohmanLLACMMPW91</url>
  <ee>http://doi.acm.org/10.1145/115790.115864</ee>
  <ee>db/conf/sigmod/LohmanLLACMMPW91.html</ee>
</inproceedings>
```

図 5 類似するペア (1)

Fig. 5 similar pair (1)

表 3 類似度

S.id	D.id	テキスト類似度	ノード系列の類似度
258	407	0.52	0.26
792	933	0.50	0.19
1372	670	0.41	0.18

5. まとめ

本稿では，XML データの類似結合のために，後置 NoK パタンによる木直列化を利用する手法を提案した．構造を持つ XML データをノード系列に直列化し，内容的にまとまりのある部分文字列に分割する．テキスト情報と構造の両面から類似度を計算し，類似するペアを見つけることで，類似結合を行なうことができる．テキスト情報の類似度を計算するにあたり，Bloom フィルタを使用し，ノード系列の類似度判定には，編集類似度を利用した．また，予備実験を行い，手法の妥当性を証

```
S.id = 792

<article>
 <title articleCode="272046">Java and Relational Databases:
    SQLJ (Tutorial)</title>
 <authors>
  <author authorPosition="00">Gray Clossman</author>
  <author authorPosition="01">Phil Shaw</author>
  <author authorPosition="02">Mark Hapner</author>
  <author authorPosition="03">Johannes Klein</author>
  <author authorPosition="04">Richard Pledereder</author>
  <author authorPosition="05">Brian Becker</author>
 </authors>
</article>

 D.id = 933

<inproceedings mdate="2006-03-31" key="conf/sigmod/ClossmanSHKPB98">
  <crossref>conf/sigmod/98</crossref>
  <author>Gray Clossman</author>
  <author>Phil Shaw</author>
  <author>Mark Hapner</author>
  <author>Johannes Klein</author>
  <author>Richard Pledereder</author>
  <author>Brian Becker</author>
  <title>Java and Relational Databases: SQLJ (Tutorial).</title>
  <pages>500</pages>
  <year>1998</year>
  <booktitle>SIGMOD Conference</booktitle>
  <url>db/conf/sigmod/sigmod98.html#ClossmanSHKPB98</url>
  <ee>http://doi.acm.org/10.1145/276304.276352</ee>
  <ee>db/conf/sigmod/ClossmanSHKPB98.html</ee>
  <cdrom>SIGMOD98/P500.PDF</cdrom>
  <cite>...</cite>
  <cite>...</cite>
</inproceedings>
```

図 6 類似するペア (2)

Fig. 6 similar pair(2)

```
S.id = 1372

<article>
 <title articleCode="293143">Metadata Standards for Data Warehousing:
     Open Information Model vs. Common Warehouse Metamodel</title>
 <authors>
  <author AuthorPosition="01">T. Vetterli</author>
  <author AuthorPosition="02">A. Vaduva</author>
  <author AuthorPosition="03">M. Staudt</author>
 </authors>
</article>

 D.id = 933

<article mdate="2006-04-03" key="journals/sigmod/VetterliVS00">
 <ee>http://doi.acm.org/10.1145/362084.362138</ee>
 <ee>db/journals/sigmod/VetterliVS00.html</ee>
 <author>Thomas Vetterli</author>
 <author>Anca Vaduva</author>
 <author>Martin Staudt</author>
 <title>Metadata Standards for Data Warehousing:
      Open Information Model vs. Common Warehouse Metamodel.</title>
 <pages>68-75</pages>
 <cdrom>sigmodR/29-3/P068.pdf</cdrom>
 <year>2000</year>
 <volume>29</volume>
 <journal>SIGMOD Record</journal>
 <number>3</number>
 <ee>http://www.acm.org/sigmod/record/issues/0009/standards.pdf</ee>
 <url>db/journals/sigmod/sigmod29.html#VetterliVS00</url>
 <cite>...</cite>
 <cite>...</cite>
 <cite>...</cite>
 <cite>journals/ibmsj/Zachman87</cite>
</article>
```

図 7 類似するペア (3)

Fig. 7 similar pair(3)

明した．今後の課題として，いくつかのパラメタを設定して，様々な文書を対象に提案手法で実験を行い，計算速度と結果を分析することが挙げられる．

謝　辞

本研究の一部は，科学研究費補助金特定領域研究 (#19024006)，萌芽研究 (#18650018)，若手研究 (B) (#19700083) の支援により行われた．

文　献

[1] World Wide Web consortium: Extensible Markup Language (XML) 1.0 (Fourth Edition), http://www.w3.org/TR/REC-xml. W3CRecommendation 16 August 2006, edited in place 29 September 2006.

[2] ACM SIGMOD Record in XML. Available at http://www.acm.org/sigmod/record/xml/.

[3] XML Version of DBLP. Available at http://dblp.uni-trier.de/xml/.

[4] W. Liang and H. Yokota. "A Path-sequence Based Method for Solving the One-to-multiple Matching Problem in Leaf-Clustering Based Approximate XML Join Algorithms", DEWS2006 4A-i10S.

[5] M. Weis and Felix Naumann. "Detecting duplicate objects in XML documents", Proc.Information Quality in Informational Systems(IQIS)2004, pp.10-19(2004).

[6] 三木元士，横田治夫. "スーパーインポーズドコーディングを用いた XML 文書キーワード索引". 情報学会研究会報告 DBS-140-25, 情報処理学会.

[7] H. Wang, S. Park, W. Fan and P. S. Yu, "ViST: a dynamic index method for querying XML data by tree structures", Proceedings of the 2003 ACM-SIGMOD Conference(SIGMOD), 2003.

[8] N. Zhang, V. Kacholia and M.T.Ozsu, "A succinct physical storage scheme for efficient evaluation of path queries in XML", Proceedings of the 22nd International Conference on Data Engineering (ICDE), 2004.

[9] P. Rao, B. Moon, "PRIX: indexing and querying XML using prufer sequences", Proceedings of the 22nd International Conference on Data Engineering (ICDE), 2004.

[10] X. Gong, W. Qian, Y. Yan, and A. Zhou, "Bloom Filter-based XML Packets Filtering for Millions of Path Queries", Proceedings of the 21st International Conference on Data Engineering (ICDE), 2005.

[11] S. Chaudhuri, V. Ganti and R. Kaushik, "A Primitive Operator for Similarity Joins in Data Cleaning", Proceedings of the 22nd International Conference on Data Engineering (ICDE), 2006.