

木直列化に基づく XML データの類似結合における木構造の統合

Integrating XML Trees in Similarity Join based on Tree Serialization

寺島 慎太郎[♥] 天笠 俊之[♦]
北川 博之[♦]

Shintaro TERAJIMA Toshiyuki AMAGASA
Hiroyuki KITAGAWA

XML の普及に伴い XML 形式で記述されたデータの情報統合に対する要求が高まっている。このため、我々の研究グループでは、XML の木構造をノード系列に直列化した上で類似結合処理を行う手法を提案した[2]。この手法は、XML 木同士の類似判定を直接木構造で行うアルゴリズムより高速であるが、実際の統合後の木構造を求める処理までは考慮していなかった。そこで本論文では、[2]に基づきノード系列として直列化された XML データで類似していると判断されたペアに対して、木構造の統合を行う手法を提案する。直列化された二つの XML データに対し、双方で一致している部分木の情報をノード系列上で判定し、その情報を元に類似結合を行う。実験によって、提案手法の有効性を検証する。

The need of integration techniques for XML data is increasing due to the rapid diffusion of XML. For this reason, our research group has proposed a scheme for similarity join, in which XML trees are serialized into sequences[2]. Although this approach is computationally inexpensive than existing methods, which are based on tree structures, it does not consider the generation of a new tree structure from a couple of XML trees being integrated. In this paper we propose a method for integrating XML node sequences, obtained by [2], by taking into account their (original) tree structures. We detect common subsequences, and merge them into a new sequence. We experimentally show that the proposed method is feasible enough to perform the integration process.

1. はじめに

XMLがW3Cによって勧告されて以来、XML形式のデータが様々な分野で使用されるようになった。その急速な普及に伴い、内容は類似しているにも関わらず、異なる構造を持つ XML データ群が多く存在するようになった。図1はWeb上に

```

<article>
  <author>
    Karl Aberer
  </author>
  <title>
    Call for Book Reviews.
  </title>
  <pages>81</pages>
  <year>2003</year>
  <volume>32</volume>
  <journal>
    SIGMOD Record
  </journal>
  <number>1</number>
</ee>
http://www.acm.org...
</cc>
<url>
  db/journals/sigmod...
</url>
</article>

```

ACM SIGMOD DBLP Bibliography

図1 「ACM SIGMOD」「DBLP Bibliography」のXML。

Fig.1 XML data from “ACM SIGMOD” and “DBLP Bibliography”.

公開されているXMLデータ「DBLP Bibliography」(左)と「ACM SIGMOD」(右)の一部である。この二つのXMLは同じ論文を表しているが、著者とタイトルの順番が逆になっていたり、後者はURLの情報を含むなど、表現する情報が異なっている。

このような異なる情報元を活用するためには、類似したXMLデータの統合が重要である。このような処理を類似結合と呼ぶ。XMLは内容に加えて、マークアップにより記述される階層構造を有するので、通常の文書とは異なり、その文書の構造を考慮した処理が必要である。

木構造を考慮したXMLデータの類似結合の手法として、比較する部分木同士の木編集距離を利用する方法が研究されている[1]。木編集距離とは、一方の木構造をもう一方の木構造に変換するのに要する、最少の編集操作の回数である。しかし木編集距離の計算には、木のノード数を n とすると、 $O(n^4)$ の計算量を必要とすることが知られている。

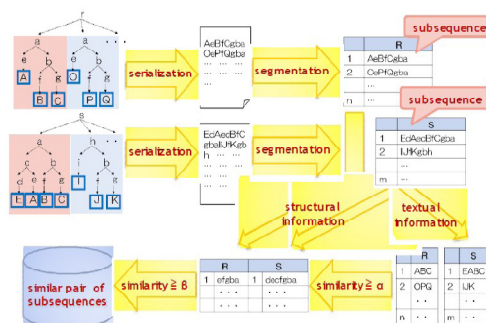


図2 木直列化を用いた類似結合手法の概要。

Fig.2 An overview of similarity join based on tree serialization.

このため、我々の研究グループでは、XMLデータをノード系列に直列化し、ノード系列同士の編集距離を類似度として類似結合を行う手法を提案している[2]。図2はその概要を表したものである。まず統合する部分木をNoKパターン (2節で後述) に直列化する。その際、テキストノードのリストも作成する。次にテキストノードに対する類似度を計算する。その類似度が閾値を超えた部分木ペアに対し、直列化したノード系列に対する類似度を計算し、最後にその類似度が閾値

[♥] 学生会員 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博士前期課程

shintaro@kde.cs.tsukuba.ac.jp

[♦] 正会員 筑波大学大学院システム情報工学研究科/計算科学研究センター

{amagasa, kitagawa}@kde.cs.tsukuba.ac.jp

を超えた部分木ペアを抽出する。木構造を文字列に変換する事で、木構造をそのまま扱うより効率的に類似度の計算を行うことが可能となる。

[2]の手法では、類似する部分木同士を判定することが可能である。しかし類似する部分木の統合処理までは検討していない。そこで本研究では、直列化によって得られたノード系列に対し、木構造を考慮して統合を行う手法を提案する。ノード系列中で共通する部分木を判定し、それらを考慮しながら統合後の木構造を生成する。実験により手法の有用性を検証する。

本論文の構成は以下の通りである。2節でNoKパターンについて述べ、3節でXMLの差分の表現について説明する。4節で本手法について述べる。5節で評価実験とその考察について説明し、最後に6節で本論文をまとめる。

2. XML 木の直列化

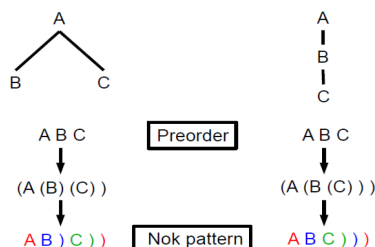


図3 NoK パターンの例.
Fig.3 An example of NoK pattern.

Wen らの手法[2]では、XML の木構造を保持しつつノード系列に直列化する手法として、NoK パターン[3]を利用している。図3は、NoK パターンの例である。

まずXML木を前置順に走査していき、ラベルを取得していく。この際、部分木を「()」で囲う。「(」は部分木の始まりを表し、「)」は部分木の終わりを意味する。木全体を走査した後、出来た文字列から「(」を取り除く。各々のノード「(」以外の文字が部分木の始まりの意味を含むため、「(」を除いても木構造の表現には影響が無く、各々のノードと「)」は一对一対応するようになる。あるノードに対応する「)」は、左にあって高さが等しい一番近い「)」となる。例えば図3の左側のNoKパターンの場合、ノード"A"と最後の「)」、ノード"B"と最初の「)」、ノード"C"と2番目の「)」が対応する。

またNoKパターンを利用すると、文字列上の各文字と木の高さを容易に対応付けることができる。「)」が連続すると木の高さが上がり、ノードが連続すると木の高さが下がる。「)」とノードが交互に来る場合は、木の高さは変化しない。

3. XML の差分の表現

次に、直列化された二つのノード系列を、木構造を考慮しながら統合することを考える。この際基本的な方針として、双方で一致する情報はそのまま残し、一致しない情報だけを他方に複製することで統合木を作ることとする。そのためには、まず比較するXML木の一致部分(不一致部分)を判定する必要がある。この目的のため、我々はLindholm らによって提案されたマッチリストを用いる[4]。

比較する二つのXML木を直列化する。直列化したXML

木をシーケンスと呼ぶ。シーケンス s は n 個の要素(トークンと呼ぶ)からなるリスト、 $s = \langle s^0, \dots, s^{n-1} \rangle$ として表される。また、シーケンスの一部 $\langle s^i, \dots, s^j \rangle$ ($i < n \leq j$) を $s^{i..j}$ と表し、サブシーケンスと呼ぶ。

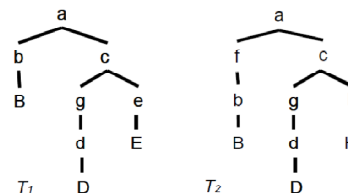


図4 直列化するXML木。

Fig.4 An example of serializing XML.

Lindholm らの手法では、直列化する際にXAS[5]という、SAXに似たJAVA APIを用いている。トークンはXMLの開始タグや終了タグ等のXMLの要素に相当し、シーケンスはその集合となる。例えば、図4のXML木 T_1, T_2 をXASを用いて直列化した場合、それぞれのシーケンス s_1, s_2 は以下ようになる。

$s_1 = \langle SE(a), SE(b), SE(B), EE(B), EE(b), SE(c), SE(g), SE(d), SE(D), EE(D), EE(d), EE(g), SE(e), SE(E), EE(E), EE(e), EE(c), EE(a) \rangle$

$s_2 = \langle SE(a), SE(f), SE(b), SE(B), EE(B), EE(b), EE(f), SE(c), SE(g), SE(d), SE(D), EE(D), EE(d), EE(g), SE(h), SE(H), EE(H), EE(h), EE(c), EE(a) \rangle$

SEは開始タグ、EEは終了タグを指す。

ここで、二つのサブシーケンス $s_1^{i..j}, s_2^{k..l}$ が一致している場合、 $cpy(i, j)$ と表す。双方のシーケンス内で一致しているサブシーケンスを、全て $cpy()$ で置き換えたものをマッチリストと呼ぶ。例えば、図4のXML木 T_1, T_2 のマッチリスト u, v は以下になる。

$u = \langle cpy(0, 0), cpy(2, 5), cpy(7, 7), cpy(8, 13), SE(e), SE(E), EE(E), EE(e), cpy(18, 18), cpy(19, 19) \rangle$

$v = \langle cpy(0, 0), SE(f), cpy(1, 4), EE(f), cpy(5, 5), cpy(6, 11), SE(h), SE(H), EE(H), EE(h), cpy(16, 16), cpy(17, 17) \rangle$

ノード a, c 、及び部分木 $b \sim B, g \sim d \sim D$ が一致しているので、 s_1, s_2 内で該当するサブシーケンスを全て置き換えている。

4. 提案手法

本論文で提案する木構造の統合手法は、大きく二つの手順に分けられる。

1. 直列化したXML木に対しマッチリストを作成する
2. そのマッチリストを元に、木構造を考慮して文字列の統合を行う

手順1の説明を4.1節、手順2の説明を4.2節で行う。

4.1 マッチリストの作成

オリジナルのマッチリストはXASを元に作成されるが、本研究ではNoKパターンであらわされたノード系列を扱っているため、そのまま適用することができない。このため本研究では、NoKパターンで直列化した文字列に対するマッチリストの作成手法を三つ提案する。なお、統合するシーケンスを $s_1 = \langle s_1^0, \dots, s_1^{n-1} \rangle$, $s_2 = \langle s_2^0, \dots, s_2^{m-1} \rangle$, そのマッチリストを $u = \langle u^0, \dots, u^{n-1} \rangle$, $v = \langle v^0, \dots, v^{m-1} \rangle$ と表す。さら

に「)」のトークンを「括弧」、それ以外の文字のトークンを「文字」と呼ぶ。

4.1.1 提案手法 1: XAS

NoK パターンで直列化した文字列を、2 節で説明した XAS で直列化した文字列に置き換えた後、[4]で提案されている手法でマッチリストを作成する。手法の詳細を図 5 に示す。

1. For $i = 0, 1, \dots, n(m) - 1$, if $s_{1(2)}^i = \text{文字}$:
 $s_{1(2)}^i = SE(s_{1(2)}^i)$, $s_{1(2)}^i$ に対応する括弧 = $EE(s_{1(2)}^i)$
2. $s = 32$, $u = s_1$, $v = s_2$
while $s > 0$ do:
For each $u^{ps\dots(p+1)s}$ ($p = 0, 1, \dots$) do:
For $i \rightarrow 0, 1, \dots, m - 1$, if $u^{ps\dots(p+1)s} = v^{i\dots i+s}$:
while $u^{(p+1)s+j} = v^{i+j+1}$, $j = j + 1$
 $u^{ps\dots(p+1)s+j}(v^{i\dots i+s+j})$ を
 $cpy(i, i + s + j)$ ($cpy(ps, (p + 1)s + j)$) に置き換え
 $s = s/2$
3. For $i \rightarrow 0, 1, \dots, n(m) - 1$, if $u(v)^i = SE(EE)$ かつ $SE(EE)$ と対応する $EE(SE)$ が同じ $cpy()$ 内に無い:
if $EE(SE)$ が $cpy()$ 内に無い: $u(v)^i$ を $cpy()$ からははず
else: $u(v)^i$ を $cpy(i, i)$ に置き換え

図5 手法1: XAS.

Fig.5 Method 1: XAS.

1.でNoKパターンをXASに置き換える。2.では一致しているサブシーケンスを探す。その時 u は s 個のトークンで構成されたサブシーケンス毎に比較する(s は2のべき乗で、32, 16, 8, ...と段々小さくしていく)。一致しているサブシーケンスが見つかったら、双方のサブシーケンスの隣のトークンを比較する。一致していたらそのトークンをサブシーケンスに入れて拡大し、これを繰り返す。3.では2.で作られたマッチリストを木構造に対応した形にする。 $cpy()$ 内には一つのノードまたは部分木だけが入るようにし、半端なものは分断する。この手法の計算量は高々 $O(n^2 + m^2)$ となる。



図6 手法1: XAS の例.

Fig.6 An example of method 1: XAS.

図6は図4のXML木に図5の手法を適用した例である。この図はマッチリスト u , v の動きを示しており、 $cpy()$ に入ったサブシーケンスに色をつけ、(最初に)探索する s 個のサブ

シーケンスに下線を引いている。

$s = 4$ の時 $u^{8\dots11}$ と $v^{10\dots13}$ が一致する。 $s = 2$ の時に $u^{2\dots3}$ と $v^{3\dots4}$ が一致し u^4 と v^5 も一致するので、 $u^{2\dots4}$ と $v^{3\dots5}$ を $cpy()$ に置換する。また $u^{5\dots6}$ と $v^{7\dots8}$ と u^7 と v^9 が一致し、次の u^8 と v^{10} が $cpy()$ 内にあって既に一致することが分かっているの、 $u^{5\dots11}$ と $v^{7\dots13}$ を $cpy()$ に置換する。以下同様置換していき、最後に u^0 と u^{17} , u^5 と u^{16} , v^0 と v^{19} , v^7 と v^{18} が別々の $cpy()$ 内にあるので分断する。

4.1.2 提案手法 2: ボトムアップ

NoKパターンで直列化したものを再びXASに変換するのは非効率的である。手法2と手法3は文字列をXASに変換せず、NoKパターンのままマッチリストを作成する。また手法2では一致している部分木を探索する際、まず一致するリーフノードを探し、そこから徐々に探索範囲を広げる方法をとる。手法の詳細を図7に示す。

1. $u = s_1$, $v = s_2$
For $i \rightarrow 0, 1, \dots, n - 2$, $j \rightarrow 0, 1, \dots, m - 2$, if $u^i = v^j = \text{文字}$ かつ $u^{i+1} = v^{j+1} = \text{括弧}$:
while $v^k = u^{i+k}$ かつ $v^{j+k+1} = u^{i+k+1}$, $k = k + 1$
(但し $cpy()$ 内のトークンは無視)
 $u^{i+k\dots i+k+1}(v^{j+k\dots j+k+1})$ を
 $cpy(i - k, i + k + 1)(cpy(j - k, j + k + 1))$ に置き換え
2. For $i \rightarrow 0, 1, \dots, n - 1$, $j \rightarrow 0, 1, \dots, m - 1$, if $u^i = v^j = cpy()$ 内に無い文字:
 $u(v)^{i(j)}$ = $u(v)^{j(i)}$ に対応する括弧
 $u^{i(a)}(v^{j(b)})$ を $cpy(j(b), i(a))(cpy(i(a), j(b)))$ に置き換え

図7 手法2: ボトムアップ.

Fig.7 Method 2: Bottom-up.

1.では一致する部分木を探索する。 u と v で一致する「連続する文字と同数の括弧」の並びからなるサブシーケンスを探索する。これは一致する部分木に相当する。例えば「bB))」がこれに該当するが、b要素の開始、Bテキストの開始と終了、b要素の終了に相当する。2.では子孫がマッチしない個別の要素ノードを探索する。まだ $cpy()$ に入っておらず、かつ一致する文字を、対応する括弧と共に $cpy()$ に入れる。この手法の計算量は高々 $O(n + nm)$ となる。

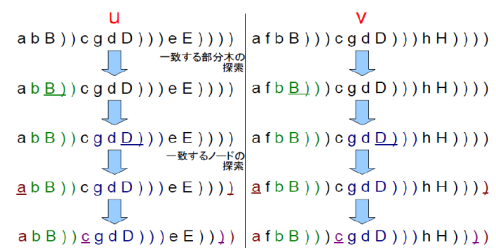


図8 手法2: ボトムアップの例.

Fig.8 An example of method 2: Bottom-up.

図8は図7の手法を適用した例である。文字 u^2 , v^3 , 括弧 u^3 , v^4 が一致し、さらに文字 u^1 , v^2 , 括弧 u^4 , v^5 が一致するので、 $u^{1\dots4}$ と $v^{2\dots5}$ を $cpy()$ に入れる。以下同様に処理し、最後に u^0 と v^0 , u^5 と v^7 が文字で一致するので、対応する括弧 u^{17} , v^{19} , u^{16} , v^{18} ごと $cpy()$ に入れる。

4.1.3 提案手法 3: トップダウン

手法2ではまず一致するリーフノードを探し、そこから

徐々に探索範囲を広げたが、手法3ではまず最大の部分木を探索し、そこから徐々に探索範囲を狭める。部分木が一致する割合大きいときに効率的であることが期待される。手法の詳細を図9に示す。

1. $u = s_1, v = s_2$
For $i \rightarrow 0, 1, \dots, n-1, j \rightarrow 0, 1, \dots, m-1$, **if** $u^i = v^j = \text{cpy}()$ 内に無い文字:
 $u(v)^{a(b)} = u(v)^{x(y)}$ に対応する括弧
if $u^{i..a} = v^{j..b} : u^{i..a}(v^{j..b})$ を $\text{cpy}(i, a)(\text{cpy}(j, b))$ に置き換え
2. 表2の2.と同じ

図9 手法3: トップダウン.
Fig.9 Method 3: Top-down.

1.では一致する部分木を探索する。uの対応する文字から括弧までのサブシーケンスがvのサブシーケンスと一致していたらcpy()に入れる。2.では一致する個別ノードを探索する。手法2と同じ過程なので説明は省略する。この手法の計算量は高々 $O(n^2 + nm)$ となる。

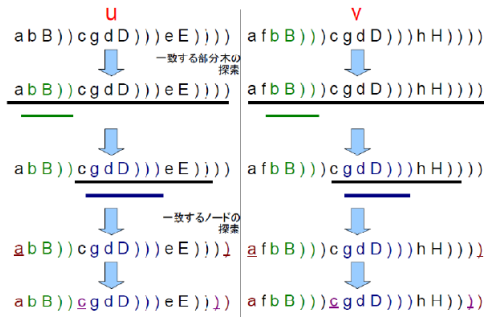


図10 手法3: トップダウンの例.
Fig.10 An example of method 3: Top-down.

図10は図9の手法を適用した例である。まず、文字 u^0 から対応する括弧 u^{17} までのサブシーケンス $u^0...^{17}$ を探索する。しかし v に一致するサブシーケンスが存在しないので、次の文字 u^1 を含むサブシーケンス $u^1...^4$ を探索する。これは $v^{2...5}$ に一致するので、双方ともcpy()に入れる。以下同様に処理し、最後に図8と同様、 u^0, v^0, u^5, v^7 および $u^{17}, v^{19}, u^{16}, v^{18}$ をcpy()に入れる。

4.2 マッチリストを使った木構造の統合

4.1節で作成したマッチリストを使って、直列化した文字列上で木構造の統合として行う。木構造の統合を行った木(統合木)の構造は、統合に用いる手法に応じて特徴が出てくる。本研究の手法で作成する統合木は、以下の条件を満たす。

1. 統合前の木の親子関係・兄弟関係は、統合木でも維持される。但し双方の木で親子・兄弟関係が異なる場合、挿入される方の木の構造が優先される。
2. 統合木内で同じ構造を持つ部分木があった場合、その先祖ノードの集合は包含関係に無い。

4.2.1 提案手法

4.2節の条件を満たした統合木を作成する手法を提案する。手法の詳細を図11, 図12で示す。なお、統合に用いるマッチリストを $u = \langle u^0, u^1, \dots, u^{r1} \rangle$, $v = \langle v^0, v^1, \dots, v^{r1} \rangle$, 統合木のシーケンスを $T = \langle T^0, T^1, \dots, T^p \rangle$ と表す。また $u(v)(T)$

1. $T = u$
For $j \rightarrow 0, 1, \dots, m-1$, **if** $v^j = \text{文字}$:
 $v^j = v^j$ に対応する括弧
For $i \rightarrow 0, 1, \dots, n-1$, **if** $u^i = v^j$ かつ $h_{u^i} = h_{v^j}$:
 $u^i = u^i$ に対応する括弧
if $u^i = v^j = \text{一文字分のみのcpy}()$: 入力を $u^{i..a}, v^{j..b}$ にして2行目に戻る
else if $u^i = v^j = \text{二文字分以上のcpy}()$ の一部かつ $u^{i..a} = v^{j..b}$: 何もせずに2行目に戻る
 $T = \text{サブシーケンスの挿入}(j, b, u, v, T)$
2. **For** $i, j \rightarrow 0, 1, \dots, o-1$, **if** $T^i = T^j = \text{文字}$:
 $T^{a(b)} = T^{x(y)}$ に対応する括弧
if $T^{i..a} = T^{j..b}$:
 $H = h_{T^{a(b)}}, \text{ANC}_{A(b)} = T^{i..a}(T^{j..b})$ の先祖ノードの集合
For $k \rightarrow i(j), i(j)-1, \dots, 0$, **if** $T_k = \text{文字}$ かつ $h_{T^k} < H$:
 $\text{ANC}_{A(b)}$ に T_k を追加
 $H = h_{T^k}$
if $\text{ANC}_{A(b)} \subseteq \text{ANC}_{A(b)} : T^{i..a}(T^{j..b})$ を削除

図11 木構造の統合手法.

Fig.11 Method of integration XML tree.

入力: 挿入するサブシーケンスの始点と終点 S, E マッチリスト u, v
 挿入前の統合木のNoK パターンシーケンス T
 出力: 挿入後の統合木のNoK パターンシーケンス T

1. **For** $i \rightarrow 0, 1, \dots, n-1, j = S, S-1, \dots, 0$, **if** $u^i = v^j = \text{文字}$ かつ $h_{u^i} = h_{v^j} = h_{v^j}$:
 $u(v)^{a(b)} = u(v)^{x(y)}$ に対応する括弧
if $u^i = v^j = \text{一文字分のみのcpy}()$ または ($u^i = v^j = \text{二文字分以上のcpy}()$ の一部かつ $u^{i..a} = v^{j..b}$):
 $v^{S..E}$ を T の u^i に相当する位置に挿入して T を返す
 $v^{S..E}$ を T の u^1 に相当する位置に挿入して T を返す

図12 関数: サブシーケンスの挿入.

Fig.12 Function: insert of subsequence.

の各トークンの高さを $h_{u(v)(T)}(i=0, 1, \dots, n(m)(o)-1)$ と表す。

図11の1.で木構造の統合を行う。 u 側と v 側で文字の高さが等しいことを確認した上で、「一文字分のみのcpy() (一致する個別ノードに相当)」「二文字分以上のcpy() (一致する部分木に相当)の一部」「cpy()内に無いトークン (一致しないノードに相当)」で処理を分ける。最初の場合は探索範囲を対応する文字から括弧までに狭める。二番目の場合は何もしない。最後の場合は対応する文字から括弧までのサブシーケンスを挿入する。2.では4.2節の条件2.を満たすように、統合木の枝切りを行う。同一構造のサブシーケンスが複数存在した場合、各サブシーケンスの全先祖ノードを探索する。もし一方の全先祖が、他方の全先祖を包含していた場合、包含されている方のサブシーケンスを削除する。この手法の計算量は高々 $O(n^2 + nm)$ となる。

図12では、4.2節の条件1.を満たすようにサブシーケンスの挿入位置を決めている。挿入するサブシーケンスの同じ高さの左側に、 u 側と v 側で高さが等しいcpy()が存在する場合、挿入位置はそのcpy()の右隣となる。存在しない場合、現在の探索範囲の先頭から2番目が挿入位置となる。

図13は図11・12の手法を適用した例である。 $v^{1...6}$ がcpy()に入っていないので挿入する。 v^1 より左に同じ高さの文字が無いので、挿入位置は T^1 となる。 v^7 は一文字分のみのcpy()なので、探索範囲を $u^{5...16}, v^{7...18}$ に狭める。 $v^{7...14}$ 内の二文字分以上のcpy() $v^{8...13}$ は、 $u^{5...16}$ 内にも同じサブシーケンスが存在するので挿入しない。同じく $v^{14...17}$ はcpy()に入っていないので挿入する。 v^{14} より左に同じ高さのcpy() $v^{8...13}$ が存在し

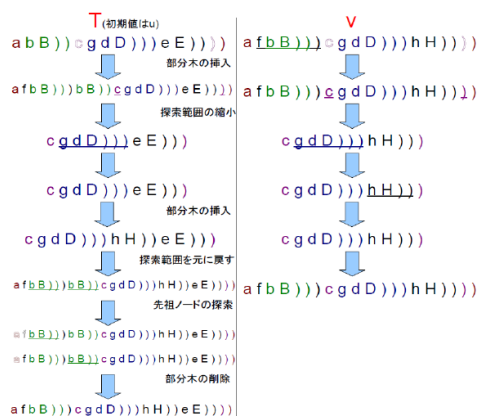


図13 木構造の統合手法の例.

Fig.13 A example of method of integration XML tree.

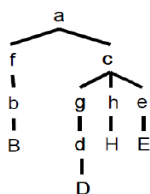


図14 最終的に出来上がる統合木.

Fig.14 A XML tree after integration.

$u^5 \dots u^{16}$ 内にも同じ高さで一致する $cpy()u^6 \dots u^{11}$ が存在するので、挿入位置はその右隣の T^8 となる。探索終了後の統合木を見ると、 $T^2 \dots T^5$ と $T^7 \dots T^{10}$ が一致しているので、双方の先祖ノードをみる。 T^2 、 T^7 よりも高さが高い左側の文字はそれぞれ " f ", " a "および" a "のみなので、 $T^7 \dots T^{10}$ を削除する。最終的に出来上がる統合木は図14の通りである。

5. 評価実験

本実験では提案手法の有用性を実データを用いて検証する。実験で使用したのはAMD Opteron(TM) 2-way Dual Core(2.4GHz)のCPUと16GBのメインメモリを持つSun OS 5.10.マシンで、Java SE 1.6.0 07で実装した。

本実験では4節で説明した各手法の比較、および従来手法に対する本手法の有用性の検証を目的として実験を行う。実験で用いるデータの概要を5.1節、前者の実験の結果と考察を5.2節、後者の実験の結果と考察を5.3節で説明する。

5.1 データセット

本実験では、1節で紹介したXML形式の論文目録データ「ACM SIGMOD」と「DBLP Bibliography」を用いる。「ACM SIGMOD」はそのまま用い、「DBLP Binliography」は一部データを抽出した、4種類のデータを作成する。抽出方法は次の通りである。まず「ACM SIGMOD」と「DBLP Binliography」の各論文単位の部分木ペアに対し、テキスト類似度 T と構造類似度 S を計算する。次に「 $T \geq 0.3$ かつ $S \geq 0.4$ (全112個)」「 $T \geq 0.3$ かつ $0.3 < S \leq 0.4$ (全282個)」「 $T \geq 0.3$ かつ $0.2 < S \leq 0.3$ (全408個)」「 $T \geq 0.3$ かつ $S < 0.2$ (全212個)」の四つの条件を満たす部分木ペアの、「DBLP Binliography」側の部分木をそれぞれ抽出する。そこにランダムに選んだ条件を満たさない部分木を加え、各データの部分木の合計が100,000個になるようにする。

5.2 各手法の比較

4節で説明したマッチリストを作成する手法三つの実行結果及び実行時間に関して比較を行う。実験手順は次の通りである。まず5.1節で説明したデータに対し、Wenらの手法で直列化し、テキスト類似度と構造類似度が一定以上の部分XMLデータのペアを列挙する。その部分木に対し、4.1.1～4.1.3節で説明した「XAS」「ボトムアップ」「トップダウン」の3手法を用いてマッチリストを作成し、4.2.1節で説明した手法を用いて木構造の統合を行う。その後その実行結果、及び実行時間を比較する。

<pre> <article> <title> [Call for Book Reviews.] [Call for Book Reviews.] </title> <pages>81</pages> <year>2003</year> <volume>32</volume> <journal> SIGMOD Record </journal> <number>1</number> <ee> http://www.acm.org... </ee> <url> db/journals/sigmod... </url> <authors> <author>Karl Aberer</author> </authors> </article> </pre>	<pre> <article> <title> Call for Book Reviews </title> <authors> <author>Karl Aberer</author> </authors> <title> Call for Book Reviews. </title> <pages>81</pages> <year>2003</year> <volume>32</volume> <journal> SIGMOD Record </journal> <number>1</number> <ee> http://www.acm.org... </ee> <url> db/journals/sigmod... </url> </article> </pre>
--	--

提案手法

SLAX

図15 提案手法(左)と比較手法(右)の実行結果.

Fig.15 Execution result of proposal method(R) and comparison method(L).

図15(左)は図1の二つのXMLを統合した場合の統合結果である(同一タグ内にテキストノードが複数並ぶ時、それぞれを[]で括弧のようにしている)。図1の双方のXMLで一致するtitle, authorsタグを保持し、さらにDBLP側にしかないpages, years等のタグが追加されている。また、titleタグは双方に存在するが、タグ内のテキストノードが異なるため(ピリオドの有無)、titleタグ内に双方のテキストノードが入る。

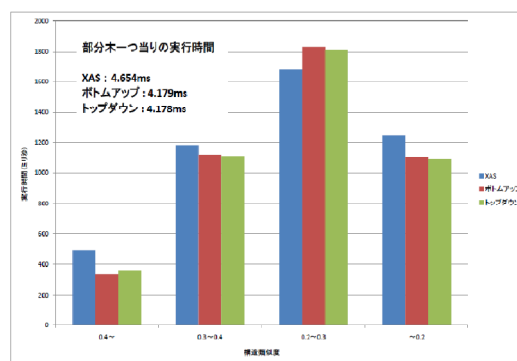


図16 マッチリスト作成における各手法の実行時間の比較.

Fig.16 Comparison of execution time for match-list generation.

図16は、マッチリスト作成の各手法の実行時間を、構造類似度毎にグラフ化したものである。また、部分木一つ辺りの実行時間も記した。結果から、手法XASは平均的に実行時間が遅いことが分かった。手法ボトムアップ、手法トップダウン

ンがNoKパターンをそのまま利用するのにに対し、手法XASはNoKパターンをXASに変換するため、その分手間がかかるためだと推測される。しかし構造類似度別に見ると、0.2～0.3のみ実行時間が早くなった。手法XASではデータを決められた長さのサブシーケンスに分断するため、そのシーケンスの構成（一致する位置や長さなど）によって実行時間が左右されやすいからだと考えられる。また、手法ボトムアップと手法トップダウンでは大きな差は見られなかった。

5.3 従来手法との比較

提案手法の有用性を検証するため、従来手法であるSLAX[6]との比較を行う。

SLAXはLiangらが提案した、XMLのテキストノードに対して類似度計算を行う手法である[6]。双方のXML木のテキストノードを比較して類似度を計算し、類似度が閾値を超えた部分木ペアを抽出する。一致しないテキストノードとその親を、ルートノードの下に挿入する。比較的容易に部分木の統合を行う事が可能だが、統合する際にその構造までは考慮していない。

実験手順は次の通りである。まず5.1節で説明したデータに対しSLAXを適用し、類似度0.11以上の部分木ペアを抽出する。「DBLP Bibliography」側は構造類似度0.4以上のデータを使用する。次に部分木ペアの数を提案手法側と同じにするため、部分木ペアをランダムに112個選択し、それに対して統合を行う。その実行結果及び実行時間を、5.2節の結果と比較する。

図1の二つのXMLを、SLAXを用いて統合した結果を図15(右)で表す。大きな違いは二つ目のtitleタグの存在である。titleタグの一致を考慮していないため、提案手法の結果と異なり双方のtitleタグがそのまま残る。また兄弟ノードの順番も考慮していないため、authorsタグの位置も異なっている。

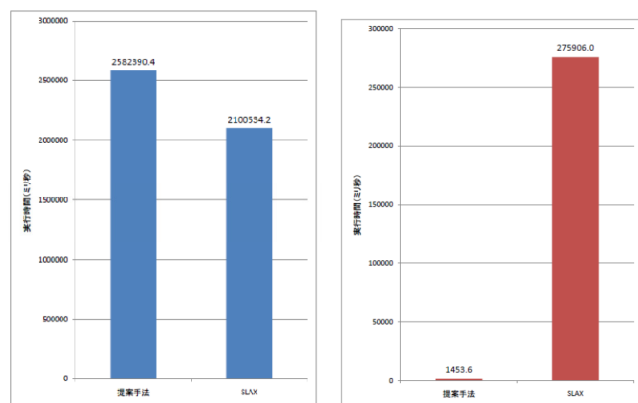


図17 類似結合全体(左)と統合部分のみ(右)の実行時間.

Fig.17 Execution time of whole(L) and integration(R).

図17はSLAXと提案手法の実行時間をグラフ化したものである。左のグラフが類似結合全体、右のグラフが部分木の統合部分のみの実行時間を表す。なお提案手法側は手法ボトムアップを用いた。実行時間全体では、提案手法の方がおよそ1.25倍遅かった。木構造の直列化など、提案手法がSLAXに比べて処理が多いからだと推測される。しかし部分木の統合部分に関しては、提案手法の方が圧倒的に早かった。統合を行う際、SLAXは木構造を直接扱うのに対し、提案手法は文字列に対し統合を行っているからだと推測される。

6. おわりに

本論文ではNoKパターンを用いて木直列化したXMLデータ群に対し、その構造を考慮して統合を行う手法を提案した。これは比較する直列化したXML木に対し、一致する部分木やノードの情報を表すマッチリストを作成する手法と、それを元に文字列の統合を行う手法で構成されている。また、実データを用いた実験によって、本手法によって木構造を考慮した部分木の統合が行えること、および従来手法より部分木の統合にかかる実行時間を短縮できることを確認した。

今後の課題としては、DTDを利用した、XML文書の整合性を考慮した統合手法の開発などが挙げられる。

【謝辞】

本研究の一部は、科学研究費補助金（#19024006，#21013004，#21700093）による。

【文献】

- [1] K.Zhang and D.Shasha. *Tree Pattern Matching. Pattern Matching Algorithm*, Aposolico and Galil Editors Oxford University Press, 1997.
- [2] Lianzi Wen, Toshiyuki Amagasa, and Hiroyuki Kitagawa. An approach for xml similarity join using tree serialization. In *13th Int'l Conf. on Database Systems for Advanced Applications (DASFAA)*, 2008.
- [3] Zhang N., Varun Kacholia, and Ozsu M.T. A succinct physical storage scheme for efficient evaluation of path queries in xml. In *20th International Conference on Data Engineering*, 2004.
- [4] Tancred Lindholm, Jaakko Kangasharju, and Sasu Tarkoma. Fast and simple xml tree differencing by sequence alignment. In *ACM symposium on Document engineering*, 2006.
- [5] T. Lindholm J. Kangasharju. A sequence-based type-aware interface for xml processing. In *European Internet and Multimedia Systems and Applications*, 2005.
- [6] W. Liang and H. Yokota. *SLAX: An improved leafclustering based approximate xml join based on clustered leaf nodes for xml data integration at subtree classes*. データベースとWeb 情報システムに関するシンポジウム, 2005.

寺島 慎太郎 Shintaro TERAJIMA

筑波大学大学院システム情報工学研究科博士前期課程修士1年在学中。日本データベース学会学生会員。

天笠 俊之 Toshiyuki AMAGASA

筑波大学大学院システム情報工学研究科、計算科学研究センター講師。データベースシステムの研究に従事。情報処理学会、電子情報通信学会、日本データベース学会各会員。

北川 博之 Hiroyuki KITAGAWA

筑波大学大学院システム情報工学研究科、計算科学研究センター教授。理学博士（東京大学）。異種情報源統合、データマイニング、文書データベース、情報検索などの研究に従事。情報処理学会および電子情報通信学会フェロー。日本データベース学会理事。日本ソフトウェア科学会、ACM、IEEE CS各会員。著書に「データベースシステム」（昭晃堂）など。