正規木文法を用いたXPath式修正候補発見アルゴリズム

池田　光雪†　　鈴木　伸崇 ††

† 筑波大学大学院 図書館情報メディア研究科　〒 305–8550 茨城県つくば市春日 1–2
†† 筑波大学 図書館情報メディア系　〒 305–8550 茨城県つくば市春日 1–2
E-mail: †{lumely,nsuzuki}@slis.tsukuba.ac.jp

あらまし　スキーマが存在する状況において，XPath 式を用いて問合せを行う場合を考える．近年，XML はますます巨大・複雑化しており，ユーザがその構造を把握すること，さらには XPath 式を記述・修正することは必ずしも容易でない．これまで筆者らは DTD が存在する状況において XPath 式間の編集距離を定義し，K 最短経路問題に帰着させるという手法で入力 XPath 式に構文的に類似し，かつ DTD に対して妥当な K 個の XPath 式を出力する手法を提案してきた．本稿では，スキーマとして正木木文法を用いた XPath 式修正候補発見アルゴリズムを提案する．

キーワード　XML, XPath, 正木木文法

1. はじめに

スキーマの規則を満たす（妥当な）XML 文書に対し XPath 式を記述する場合を考える．このとき，ユーザが対象となる XML の構造をよく理解していない場合，スキーマに関して妥当な，すなわち何らかの解が得られるであろう XPath 式を記述することは必ずしも容易でない．とりわけ，近年において XML やスキーマはますます巨大化・複雑化してきており，その構造を理解することはより困難になってきている．さらに，スキーマ進化によって XML やスキーマの構造が変化し，元々スキーマに対し妥当であった XPath 式が妥当でなくなる場合がある．また，XML のキーワード検索ではユーザはそれらの構造を意識すること無く問合せを行うことが可能であるが，XPath に比べ詳細な検索を行うことは難しい．そこで，スキーマに関して妥当でない XPath 式 p に対し，スキーマに関して妥当かつ p との類似性が高い XPath 式をユーザに提示できれば XPath 式記述の支援に有用であると考えられる．

これまで著者らはこの問題に対し，ユーザが真に欲しい XPath 式はユーザが記述した式 p に構造や要素名などが似ているという仮定のもと，スキーマが DTD D の場合に XPath 式間の編集距離を定義し D と p を合成した XD グラフを作成して，その上で K 最短経路問題を解くという手法で p に構文的に類似し，かつ D に対して妥当な K 個の XPath 式を出力する手法を提案してきた [11].

このアルゴリズムの特徴は DTD の全体構造を把握していなくとも，目的の要素名さえある程度特定できれば妥当な XPath 式が得られることである．例として次の DTD を考える．

```
<!ELEMENT site   (people)>
<!ELEMENT people (person)*>
<!ELEMENT person (name)>
<!ELEMENT name   (#PCDATA)>
<!ATTLIST person id ID #REQUIRED>
```

ここで，ID が 2013 である person の name 要素を取得したいと考えたが，誤って $q =$ `/person[@id = "2013"]/naem` という XPath 式を記述した場合を考える．本アルゴリズムを用いれば，以下のようにこの XPath 式に類似した妥当な式の列挙が得られる ($K = 3$ の場合の例). 右端の数値は，要素名を置換する際の重みを文字列間の正規化編集距離の値 [15] とし，それ以外の編集操作の重みを 1 とした場合の編集距離を表している．

```
1. //person[@id = "2013"]/name              (0.75)
2. //people/person[@id = "2013"]/name       (1.75)
3. /site/people/person[@id = "2013"]/name  (2.25)
```

このように `naem` というタイプミスが修正された上で，その要素を検索する妥当な XPath 式が入力式に近いものから得られる．なお，各編集操作に付加するコストは自由に変更することができる．例えば,「/より//の使用を優先してより簡潔な式を優先的に列挙する」,「//より/の使用を優先し，途中の経路を詳細に指定した式を優先的に列挙する」等の調整が可能である．

本稿では，DTD のみに対応していた既存アルゴリズムを正木木文法まで対応可能なように拡張したアルゴリズムを提案する．XML と併用される代表的なスキーマ言語である RELAX NG, W3C XML Schema, DTD は正規木文法で表現可能なことが知られている [13] ため，本拡張により上記三つのスキーマ言語全てにアルゴリズムが対応可能となる．具体的には，まず入力スキーマを正規木文法 $G = (N, T, S, P)$ と考え，G から要素間の構造を表すグラフ $G(P)$ を構築する．さらに，XPath 式 p と $G(P)$ から XG グラフ $G(p, G(P))$ を作成し，その上で K 最短経路問題を解くことで K 個の p に類似した XPath 式を得る．

XML に対する問合せに関しては様々な研究が行われている [9] が，本稿と最も関連が強いのは文献 [7] である．同文献では，与えられた問合せ式 p とスキーマ D に対して，D に関して妥当かつ p との類似度が高いものを列挙するアルゴリズムを提案している．本稿との主な違いは，同文献のスキーマは有向非循環グラフ (DAG) であり再帰を扱うのが事実上困難なこと，ノード間の順序を考慮しないデータモデルとなっており，問合せ式も無順序木として表されていること，編集距離でなく独自に定義した類似度を用いている (編集操作に対するコストの

変更が困難) ことである．一方本稿では，再帰を許しかつノード間の順序を前提としたデータモデルを扱い，編集操作に対するコストは任意の値に設定可能である．また DTD における再帰について，実際に使用されている DTD を対象とした調査が行われており，60 個の DTD のうち 35 個が再帰を含んでいた [6]．文献 [2], [16] では，ユーザへの質問を通して対話的に所望の XQuery 式を学習し，提示するシステムを提案している．一方，本稿では 1 つの XPath 式から，類似した複数の正解候補を出力する．また，スキーマの存在を陽に仮定しない場合において，XML への問合せに対する近似的な問合せ結果を列挙する手法が提案されている (文献 [4], [5] など)．これらはインスタンスに対する問合せを扱っており，本稿のような問合せ式の修正候補の発見とは異なる．XHTML など，問合せの対象において使用されている要素が非常に少なく，かつ複雑に組み合わせられている場合，一般には本研究のようなスキーマを用いた修正候補の発見手法に比べ，インスタンスを用いたこれらの手法では有用な修正候補を得ることができる．しかし，XMark [18] が生成するインスタンスのようにスキーマで文書構造を詳しく定義している場合は，本稿のような問合せ式の修正候補発見手法でも高速かつ有用な修正候補の提示が可能である．

さらに，XMLSpy [3] や文献 [1] で提案されているエディタなどには，XPath 式を記述する際に要素名などを補完する機能があるが，本稿のように修正候補の列挙は行わない．XML のキーワード検索に関しても多くの研究がなされている (例えば文献 [20], [12], [17])．これらは，XML の問合せ言語に詳しくないユーザにとって特に有用である．

提案アルゴリズムでは軸として child, descendant-or-self, following-sibling, preceding-sibling のみを許し，ノードテストは要素名のみという制約を設けた XPath 式を対象としている．これらの制約のうち，軸については parent 軸など上方向の軸も対象とした方が望ましいが，実際に使用されている XPath 式のほとんどは下方向の軸のみを使用している [10] ため，有用性は損なわれないと考えられる．

2. 諸 定 義

正規木文法を 4 次組 $G = (N, T, S, P)$ と表す．ここで，N は非終端記号の有限集合，T は終端記号の有限集合，S を開始記号，P を $X \to ar$ の形をした遷移規則の集合とする．ただし，$X \in N, a \in T$ であり，r は N 上の正規表現である．X を規則の左辺，ar は右辺，a，r をそれぞれこの規則のラベル，内容モデルという．

正規木文法 $G = (N, T, S, P)$ とラベル a, b に対して，もし次の条件のいずれか一方が成立するならば，G において a から b に到達可能であるという．

- $a = b$，または P 内に遷移規則 $X \to ar$ と $X' \to br'$ が存在し，X' が r に出現する．
- あるラベル a' に対して，a から a' に到達可能，かつ遷移規則 $X \to a'r$ と $X' \to br'$ が存在し，X' が r に出現する．

正木木文法 G に対する木 t の解釈 I は，t 中の各ノード e から状態 $I(e)$ への以下を満たす写像である．

表 1　XP の構文定義

XP	::=	“/” RelativePath
RelativePath	::=	LocationStep | LocationStep “/” RelativePath
LocationStep	::=	Axis “::” Label | Axis “::” Label Predicate
Axis	::=	“↓” | “↓*” | “$\to^+$” | “$\leftarrow^+$”
Label	::=	(any label in T)
Predicate	::=	“[” Exp “]”
Exp	::=	PredPath | PredPath Op Value
PredPath	::=	RelativePath
Op	::=	“=” | “<” | “>” | “=<”| “=>”
Value	::=	“’ (any string other than “’) “’

- e が t の根であるとき，$I(e)$ は初期状態である．
- 各ノード e とそれらの子ノード $e_0, e_1, ..., e_m$ について，G 中の遷移規則 $X \to \mathbf{a}r$ が存在し，以下を満たす．
  - $I(e) = X$,
  - e の終端記号は a,
  - $I(e_0)I(e_1)...I(e_m)$ が r にマッチする．

正木木文法 G によって受理される集合を $L(G)$ と表す．正木木文法 G が木 t を受理するとは，G に対する t の解釈が存在するということである．

例えば，前章の DTD は正規木文法 $G = (N, T, S, P)$ と表すことができ，ここで

$$
\begin{aligned}
N &= \{Site, People, Person, Name, Pcdata\} \\
T &= \{site, people, person, name, pcdata\} \\
S &= \{Site\} \\
P &= \{Site \to site(People), People \to people(Person^*), \\
&\quad Person \to person(Name), Name \to Pcdata(Pcdata), \\
&\quad Pcdata \to pcdata(\epsilon)\}
\end{aligned}
$$

である．

本稿で用いる XPath 式の集合を XP と表記する．XP は child 軸 (↓)，descendant-or-self 軸 ($\downarrow^*$)，following-sibling 軸 ($\to^+$)，preceding-sibling 軸 ($\leftarrow^+$) のいずれかを用いた，ノードテストとしてラベルのみを許すロケーションパスの集合であり，形式的には表 1 で定義される．定義から，XP における XPath 式は次のように表せる．

$$p = /ax[1] :: l[1][exp[1]]/ \cdots /ax[m] :: l[m][exp[m]]$$

ここで，$1 \leqq i \leqq m$ に対して $ax[i] \in$ Axis，$l[i] \in T$，$exp[i] \in$ Exp である．

$G = (N, T, S, P)$ を正規木文法，$p = /ax[1] :: l[1][exp[1]]/ \cdots /ax[m] :: l[m][exp[m]]$ を XPath 式とする．p のロケーションステップ数を $|p|$ と表す．もし以下の条件が成立するならば，p は G に関して妥当であるという．

- $ax[1] = \downarrow$ かつ $S \to l[1]r$ となるような遷移規則が P に存在，または，$ax[1] = \downarrow^*$ かつ $l[1]$ は T 中に出現する
- $2 \leqq i \leqq m$ に対して以下が成り立つ
  - $ax[i] = \downarrow$ かつ遷移規則 $X \to \mathbf{l}[i-1]r$ と $X' \to \mathbf{l}[i]r'$ が存在し，X' が r に出現する
  - $ax[i] = \downarrow^*$ かつ G において $\mathbf{l}[i-1]$ から $\mathbf{l}[i]$ へ到達可能

である

- $exp[i] \neq \epsilon$ の場合，右辺が $\mathbf{l}[i]r$ である遷移規則が P に存在し，XPath 式 $/\downarrow::l[i]/exp[i]$ が正規木文法 $(G=(N,T,r,P))$ に関して妥当である $(1 \leqq i \leqq m)$

XPath 式 $p \in$ XP に対して, もし p が述語を含まない場合，p は単純であるという．

3. XPath 式に対する編集操作

本章では，XPath 式に対する編集操作を定義する．XPath 式に対する編集操作は以下の 4 種類である．

- 軸の置換：軸 ax を ax' に置換する．$ax \to ax'$ と表す．例えば，$/\downarrow::a$ に更新操作 $\downarrow \to \downarrow^*$ を適用すると $/\downarrow^*::a$ を得る．
- ラベルの置換：ラベル l を l' に置換する．$l \to l'$ と表す．例えば，$/\downarrow::a$ に更新操作 $a \to b$ を適用すると $/\downarrow::b$ を得る．
- ロケーションステップの追加：ロケーションステップ $ax::l$ を追加する．$\epsilon \to ax::l$ と表す．例えば，$/\downarrow::a$ の末尾に更新操作 $\epsilon \to \downarrow::b$ を適用すると $/\downarrow::a/\downarrow::b$ を得る．
- ロケーションステップの削除：ロケーションステップ $ax::l$ を削除する．$ax \to \epsilon$ と表す．例えば，$/\downarrow::a/\downarrow::b$ の最初のロケーションステップに更新操作 $/\downarrow::a \to \epsilon$ を適用すると $/\downarrow::b$ を得る．
- ロケーションステップの交換：隣接する 2 つのロケーションステップを入れ替える．$ax::l/ax'::l' \to ax'::l'/ax::l$ と表す．例えば，$/\downarrow::a/\downarrow::b$ に対し更新操作 $\downarrow::a/\downarrow::b \to \downarrow::b/\downarrow::a$ を適用すると $/\downarrow::b/\downarrow::a$ を得る．

ロケーションステップ ls の位置を $pos(ls)$ と表し，次のように定義する．$q = /ax[1]::l[1][exp[1]]/\cdots/ax[m]::l[m][exp[m]] \in$ XP を XPath 式とする．まず，$pos(ax[i]::l[i]) = i$ $(1 \leqq i \leqq m)$ と定義する．述語におけるロケーションステップの位置について，$exp[j] = ax'[1]::l'[1][exp'[1]]/\cdots/ax'[n]::l'[n][exp'[n]]$ とする．このとき，$pos(ax'[k]::l'[k]) = j.k$ $(1 \leqq k \leqq n)$ と定義する．$exp'[h]$ における位置も同様に定義できる．例えば，$q = /\downarrow::a/\downarrow::b[\downarrow::f[\downarrow::g]]/\rightarrow^+::c$ を考える．このとき，$pos(\downarrow::b) = 2$, $pos(\downarrow::f) = 2.1$, $pos(\downarrow::g) = 2.1.1$ である．位置 pos のロケーションステップに適用する編集操作 op を $[op]_{pos}$ と表す．もし op がロケーションステップ ls の追加である場合，$[op]_{pos}$ は ls を pos に位置するロケーションステップの直後に追加する．

XPath 式 p に対し，位置をもつ編集操作の系列を編集操作列と呼ぶ．編集操作列 sc を p に適用して得られる XPath 式を $sc(p)$ と表す．例えば，$sc = [\epsilon \to \downarrow::b]_1\,[c \to f]_3$，$p = /\downarrow^*::a/\downarrow::d/\downarrow::c$ の場合，$sc(p) = /\downarrow^*::a/\downarrow::b/\downarrow::d/\downarrow::f$ である．

本稿では以下の仮定を置く．$A_1 = \{\downarrow, \downarrow^*\}$，$A_2 = \{\rightarrow^+, \leftarrow^+\}$ とする．

- 同種の軸への置換のみ許す．$ax \in A_1$ は A_1 に属する軸へのみ置換可能とする．A_2 に関しても同様である．
- 軸が A_1 に属するロケーションステップのみ追加可能とする．

編集操作に対してコストを付与する関数をコスト関数という．編集操作 op のコストを $\gamma(op)$ と表し，このとき γ がコスト関数である．以下，$\gamma(op) \geqq 0$ と仮定する．コスト関数として，定数だけでなく関数も使用できる．例えば，$op = l \to l'$ のとき，$\gamma(op)$ を l と l' の (文字列間の) 編集距離等に設定することが可能である．また，軸名に関しても同様の設定が可能である．編集操作列 $sc = op_1 op_2 \cdots op_n$ のコストを $\gamma(sc) = \sum_{1 \leqq i \leqq n} \gamma(op_i)$ と定義する．正木木文法 G，XPath 式 p，正整数 K に対して，G の下での p に対する K 最適編集操作列とは，以下のすべての条件を満たす編集操作列の系列 $sc_1, \cdots, sc_K$ である．

- $sc_1(p), \cdots, sc_K(p)$ はすべて G に関して妥当である
- $\gamma(sc_1) \leqq \cdots \leqq \gamma(sc_K)$
- $sc_1, \cdots, sc_K$ は最適，すなわち，p に対する任意の編集操作列 sc に対して以下が成り立つ
  - もし $sc(p)$ が G に関して妥当ならば，$sc(p) \in \{sc_1(p), \cdots, sc_K(p)\}$ または $\gamma(sc) \geqq \gamma(sc_K)$

$sc_1(p), \cdots, sc_K(p)$ を G の下での p に対する K 最適修正候補という．

4. XPath 式修正候補発見問題の NP 困難性

XPath 式の修正候補発見問題を，正規木文法 $G = (N,T,S,P)$，XPath 式 p，および正整数 K が与えられた時に，p がコスト K 以下の修正候補をもつか否かを決定する問題と定義する．この問題が NP 困難であることを示す．

［定理 1］ ロケーションパスの交換を許した場合，XPath 式の修正候補発見問題は NP 困難である．

証明： XPath 式の修正候補発見問題が NP 困難であることを，有向ハミルトン経路問題から XPath 式の修正候補発見問題への帰着により示す．有向ハミルトン経路問題とは以下のような問題である．

入力: 有向グラフ $H = (V,E)$ および頂点 $u, v \in V$．

問題: H が u から v に至る有向ハミルトン経路 (V のすべての点をちょうど 1 つづつ通る経路) を含むか否か決定せよ．

有向ハミルトン経路問題のインスタンス，有向グラフ $H = (V,E)$ および頂点 $u, v \in V$ に対して，XPath 式修正候補発見問題のインスタンス，正規木文法 $G = (N,T,S,P)$，XPath 式 p，および正整数 K を次のようにして構成する．まず，$G = (N,T,S,P)$ は次のように定義される．

$$
\begin{aligned}
N &= \{V_1, V_2, \cdots, V_k\} \cup \{D\} \\
T &= \{v_1, v_2, \cdots, v_k\} \cup \{d\} \\
S &= \{V_j \mid u = v_j\} \\
P &= \{V_i \to v_i(V_j) \mid v_i \to v_j \in E\} \cup \\
&\qquad \{V_h \to v_h(D) \mid v_h \in V\} \cup \{D \to d(\epsilon)\}
\end{aligned}
$$

次に，XPath 式 p を次のように定義する．

$$p = /v_1/v_2/\cdots/v_k/d$$

また，編集操作のコストについて，ロケーションステップの交換操作のコストを 1，それ以外のコストを ∞ とする．また，$K = k(k+1)/2$ とする．

以下，H が u から v への有向ハミルトン経路をもつことと，p がコスト K 以下の修正候補をもつことが同値であることを示す．

($\Rightarrow$) H が u から v への有向ハミルトン経路 $u = v_{i_1} \to v_{i_2} \to \cdots \to v_{i_k} = v$ をもつとする．G の定義から，XPath 式 $p' = /v_{i_1}/v_{i_2}/\cdots/v_{i_k}/d$ は G に関して妥当である．更に，p から p' へ $k(k+1)/2$ 回以下のロケーションステップの交換で修正可能である．したがって，p はコスト $k(k+1)/2$ 以下の修正候補 p' をもつ．

($\Leftarrow$) H が u から v への有向ハミルトン経路をもたないとする．このとき，H における u から v への長さ k の任意の経路は，ある頂点を 2 個以上含む．このことと G の定義から，p を妥当な XPath 式に修正する場合，必ずロケーションステップの削除および追加が必要となり，その修正コストは ∞ となる．したがって，p がコスト k 以下の修正候補をもつことはない．

以上の結果から，ロケーションステップの交換を許した場合，K = 1 の場合でも修正候補発見問題を効率よく解くのは困難である．以下，ロケーションステップの交換を用いない場合に，修正候補発見問題を効率よく解くアルゴリズムを考える． □

5. XG グラフ

提案アルゴリズムの中心をなす XG グラフの構成法について述べる．本章では，XPath 式は単純であると仮定する．一般の場合については **6.2** にて考察する．

本稿では，$G = (N, T, S, P)$ の下での p に対する K 最適編集操作列を次のようにして求める．

（1） G の生成規則 P から遷移規則グラフ $G(P)$ を構成する．

（2） p と $G(P)$ から，XG グラフ $G(p, G(P))$ を構成する．これは S に関して妥当な，p に編集操作列を適用して得られる XPath 式を表すすべての経路から構成される．

（3） $G(p, G(P))$ 上で K 最短経路問題を解く．その結果が G の下での p に対する K 最適編集操作列を表す．この詳細は **6.** で述べる．

5.1 遷移規則グラフ

XG グラフを構成するには，正木木文法の生成規則をグラフとして表す必要がある．そこで，まず遷移規則グラフを定義する．$G = (N, T, S, P)$ を正木木文法とする．このとき，遷移規則グラフ $G(P) = (V, E)$ は次のように定義される有向グラフである．

$$
\begin{aligned}
V &= \{(X, a) \mid X \to ar \in P\} \\
E &= \{(X, a) \to (X', a') \mid X \to ar \in P, \\
&\qquad X' \text{ が } r \text{ に出現，} X' \neq \mathrm{Pcdata}\text{，ある } a' \in T \text{ と} \\
&\qquad N \text{ 上の正規表現 } r' \text{ に対して } X' \to a'r' \in P\}
\end{aligned}
$$

例えば，正木木文法 $G = (N, T, S, P)$ において，

$$
\begin{aligned}
N &= \{S, A, B, C, D, Pcdata\} \\
T &= \{s, a, b, c, d, pcdata\} \\
S &= \{S\}
\end{aligned}
$$

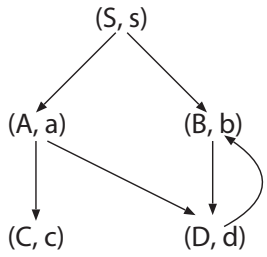

図 1 遷移規則グラフ

$$
\begin{aligned}
P = \{&S \to s(A, B), A \to a(C, D), B \to b(D) \\
&C \to c(Pcdata), D \to d(B|Pcdata), \\
&Pcdata \to pcdata(\epsilon)\}
\end{aligned}
$$

であるとき，G の遷移規則グラフは図 1 になる．

5.2 XG グラフ

XPath 式 p と遷移規則グラフ $G(P)$ から XG グラフを構成する．直観的には，XG グラフにおける各辺は **3.** で定義した編集操作に対応している．そのため，扱う軸ごとに異なる辺が存在する．以下，まず **5.2.1** で下記 (1) の場合の XG グラフについて説明を行う（紙面の都合上 (2) と (3) は省略する）．次に，**5.2.2** で XG グラフの形式的定義を示し，辺に対するコストの付与の仕方について述べる．

(1) $\downarrow$ 軸のみを用いる場合

(2) $\downarrow$ 軸と $\downarrow^*$ 軸を用いる場合

(3) $\rightarrow^+$ 軸と $\leftarrow^+$ 軸を用いる場合

5.2.1 $\downarrow$ 軸のみを用いた XG グラフ

まず，XPath 式 $p = /\downarrow::a/\downarrow::d$ と図 1 の生成規則グラフ $G(P)$ に対して，どのような XG グラフが構成されるのかを述べる．軸は $\downarrow$ のみを用いるので，編集操作としてロケーションステップの削除・追加, ラベルの置換のみを考えればよい．p と $G(P)$ の合成グラフ $G(p, G(P))$ を図 2 に示す．この図に示すように，XG グラフは $G(P)$ を $|p|+1$ 個並べてそれらのノードを辺で接続したものである．ただし，$(N, n)_0, (N, n)_1, (N, n)_2$ は新たに追加したノードであり，XPath のデータモデルにおけるルートノードに対応する．遷移規則グラフのノードに添字を付加してノードを区別する．例えば，$G(P)$ におけるノード (S, s) は，$G(p, G(P))$ の一番上の遷移規則グラフでは $(S, s)_0$，その下の遷移規則グラフでは $(S, s)_1$ と表記される．

図 2 に示すように，遷移規則グラフのノード間に追加される辺は以下の 3 種である．

- $\to$: ロケーションステップの追加 $(\epsilon \to \downarrow::l)$ に対応
- $\dashrightarrow$: ロケーションステップの削除 $(\downarrow::l \to \epsilon)$ に対応
- $\dashrightarrow$: ラベルの置換 $(l \to l')$ に対応

直感的には，右方向の辺はロケーションステップの追加，下方向の辺はロケーションステップの削除，斜め方向の辺はラベルの置換をそれぞれ表している．これら 3 種の辺についてより具体的に説明する．まず，図 2 の辺 $(N, n)_0 \to (S, s)_0$ について考える．この辺は「ルートノードから子ノード s へ，p のロケーションステップを用いずに移動する」ことを表している．この移動は子ノード s への移動であり，(p には元々存在しない) ロ

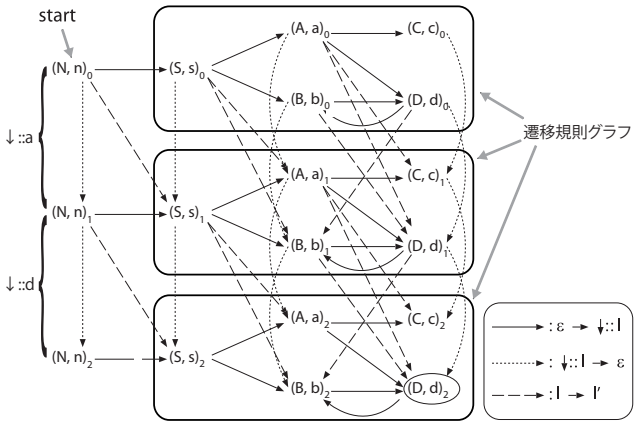

図 2　XG グラフ $G(p, G(P))$

ケーションステップ $\downarrow::s$ によって行われるとみなす．したがって，辺 $(N,n)_0 \to (S,s)_0$ は新たなロケーションステップ $\downarrow::s$ の追加，すなわち編集操作 $[\epsilon \to \downarrow::s]_0$ を表す．

次に，図 2 の辺 $(S,s)_0 \dashrightarrow (B,b)_1$ について考える．この辺は「p のロケーションステップ $\downarrow::a$ を用いてノード s から子ノード b へ移動する」ことを表しているが，移動先ノードが b であるため $\downarrow::a$ のラベルを b へ置換する必要がある．すなわち，辺 $(S,s)_0 \dashrightarrow (B,b)_1$ は編集操作 $[a \to b]_1$ を表している．

最後に，図 2 の辺 $(B,b)_1 \dashrightarrow (B,b)_2$ を考える．これはノード b からノード b への移動，すなわち「p のロケーションステップ $\downarrow::d$ を無視 (削除) して同じノード b に留まる」ことを表している．すなわち，辺 $(B,b)_1 \dashrightarrow (B,b)_2$ は編集操作 $[\downarrow::d \to \epsilon]_2$ を表す．

図 2 において，$(N,n)_0$ を開始ノード，$(D,d)_2$ を受理ノードという．開始ノード及び受理ノードの決定方法は **6.** にて述べる．開始ノードから受理ノードへの経路は，p を修正して得られる G に関して妥当な XPath 式を表す．例えば，図 2 における $p' = (N,n)_0 \to (S,s)_0 \dashrightarrow (A,a)_1 \dashrightarrow (D,d)_2$ という経路を考える．上記の経路のうち，最初の辺 $(N,n)_0 \to (S,s)_0$ はロケーションステップ $\downarrow::s$ の追加 $[\epsilon \to \downarrow::s]_0$ を表す．次の辺 $(S,s)_0 \dashrightarrow (A,a)_1$ は，$\downarrow::a$ におけるラベルの置換 $[a \to a]_1$ を表すが，同一ラベルへの置換でありロケーションステップは変化しない．同様に，最後の辺 $(A,a)_1 \dashrightarrow (D,d)_2$ においても p のロケーションステップ $\downarrow::d$ はそのままである．したがって，p' は XPath 式 $/\downarrow::s/\downarrow::a/\downarrow::d$ を表しており，これは G に関して妥当である．

5. 2. 2　XG グラフの形式的定義と辺に付与されるコスト

前節で示した XG グラフを形式的に定義し，その辺に付与されるコストを示す．ただし，軸の置換を表す辺の形式定義，及びそのコストについては紙面の都合上省略する．$G = (N,T,S,P)$ を正規木文法，$G(P) = (V,E)$ を遷移規則グラフ，$p = /ax[1]::l[1]/\cdots/ax[m]::l[m]$ を単純 XPath 式とする．$G_i(P) = (V_i,E_i)$ $(0 \leqq i \leqq m)$ を $G(P)$ の各ノードに添え字 i を付加したものと定義する．すなわち，$V_i = \{(X,a)_i \mid X \to ar \in P\}$ かつ $E_i = \{(X,a)_i \to (X',a')_i \mid X \to ar \in P\}$ である．XG グラフ $G(p, G(P)) = (V', E')$ は次のように定義される有向グラフである ($(N,n)_0, \cdots, (N,n)_m$ の n は $n \notin V$ なるラベルとする)．

$$V' = \{(N,n)_0, \cdots, (N,n)_m\} \cup V_0 \cup \cdots \cup V_m$$
$$E' = E_{insc} \cup (F_1 \cup \cdots \cup F_m) \quad (1)$$

式 (1) の E_{insc} は $\downarrow::l$ の追加を表す辺 (図 2 の "$\epsilon \to \downarrow::l$" の辺に該当) であり，次のように定義される．下記のうち，E_i は $G_i(P)$ の辺である．

$$E_{insc} = \{(N,n)_0 \to (S,s)_0, \cdots, (N,n)_m \to (S,s)_m\} \cup (E_0 \cup \cdots \cup E_m)$$

また，式 (1) の F_i $(1 \leqq i \leqq m)$ は，$G_{i-1}(P)$ と $G_i(P)$ の間に張られる辺であり，p の i 番目の軸 $ax[i]$ の種類に応じて定義が異なる．本稿では簡単のため $ax[i] \in \{\downarrow\}$ とし，$F_i = D_i \cup C_i$ とする．ここで，D_i，C_i は次のように定義される．また，D_i は図 2 の "$\downarrow::l \to \epsilon$" の辺，$C_i$ は図 2 の "$l \to l'$" の辺にそれぞれ対応する．

$$D_i = \{(N,n)_{i-1} \to (N,n)_i\} \cup \{l_{i-1} \to l_i \mid l \in V\} \quad (2)$$
$$C_i = \{(N,n)_{i-1} \to (S,s)_i\} \cup \{l_{i-1} \to l'_i \mid l \to l' \in E\} \quad (3)$$

最後に，XG グラフの各辺に付与するコストを考える．**3.** で定義した編集操作の種類に応じて，以下のコストが定義されているとする (実際の値は現状ユーザ等で指定するが，将来的には使用状況に応じていくつかのコスト群を提供する予定である)．

- $\gamma(l \to l')$: ラベル l を l' に置換するコスト
- $\gamma(\epsilon \to ax::l)$: ロケーションステップ $ax::l$ を追加するコスト
- $\gamma(ax::l \to \epsilon)$: ロケーションステップ $ax::l$ を削除するコスト

上記のコストに基づいて，XG グラフの辺 $e \in E'$ のコスト $\gamma(e)$ を次のように定義する．

- $e \in E_{inssc}$ のとき : $e = l_i \to l'_i$ と表す．この辺は $\downarrow::l'$ の追加を表すので，$\gamma(e) = \gamma(\epsilon \to \downarrow::l')$ と定義する．
- $e \in D_i$ のとき:$e = l_{i-1} \to l_i$ と表す．この辺は $ax[i]::l[i]$ の削除を表すので，$\gamma(e) = \gamma(ax[i]::l[i] \to \epsilon)$ と定義する．
- $e \in C_i$ のとき : $e = l_{i-1} \to l'_i$ と表す．この辺は $ax[i]$ の $\downarrow$ への置換と $l[i]$ の l' への置換を表すので，$\gamma(e) = \gamma(ax[i] \to \downarrow) + \gamma(l[i] \to l')$ と定義する．

例として，次のコスト関数を考える．

$$\gamma(l \to l') = \begin{cases} 0 & l = l' \text{ のとき} \\ 1 & \text{それ以外} \end{cases}$$
$$\gamma(\epsilon \to ax::l) = 1$$
$$\gamma(ax::l \to \epsilon) = 1$$

このとき，経路 $(N,n)_0 \to (S,s)_0 \dashrightarrow (A,a)_1 \dashrightarrow (D,d)_2$ のコストは $\gamma(\epsilon \to \downarrow::s) + (\gamma(\downarrow \to \downarrow) + \gamma(a \to a)) + (\gamma(\downarrow \to \downarrow) + \gamma(d \to d)) = 1 + 0 + 0 = 1$ となる．

6.　XPath 式に対する K 最適修正候補発見手法

本章では，正規木文法 G と XPath 式 p に対して，G の下で

の p に対する K 最適修正候補を発見するためのアルゴリズムについて述べる．まず，p が単純 XPath 式の場合について述べ，次に一般の場合について説明する．また，グラフに対する枝刈りについても考察する．

6.1 単純 XPath 式に対するアルゴリズム

$G(P)$ を正規規則グラフ，$p = /ax[1]::l[1]/\cdots/ax[m]::l[m]$ を XPath 式，$G(p, G(P)) = (V, E)$ を XG グラフとする．$(N, n)_0 \in V$ を開始ノード，$(X, l[m])_m \in V$ を受理ノードと定める．ただし，$l[m]$ が (タイプミス等の要因で) $G(P)$ に出現しないラベルであった場合，$l[m]$ に最も類似したラベル l を持つノードを V から選択し，添字 m を付加したラベル l_m をもつノードを受理ノードとする．現状，$l[m]$ と (文字列間の) 編集距離が最も小さいものを選択している．上記のコスト定義から，p に対する K 最適修正候補を求めるには，合成グラフ $G(p, G(P))$ の開始ノードから受理ノードまでの K 最短経路問題を解き，得られた経路 $p'_1, \cdots, p'_K$ が表す K 個の XPath 式を出力すればよい．次の定理が成り立つ (証明は省略する)．

［定理 2］ $G = (N, T, S, P)$ を正規木文法，p を XPath 式，K を正整数とする．上記アルゴリズムは P の下での p に対する K 最適修正候補を出力する． □

提案アルゴリズムの時間計算量について考える．まず，正規木文法 G と XPath 式 p に対する XG グラフ $G(p, G(P))$ のサイズを考える．P を G に出現する遷移規則の集合とする．$G(p, G(P))$ の各ノード n に対して，n を始点とする辺の数は $O(|P|)$ である．$G(p, G(P))$ のノード数は $O(|p| \cdot |P|)$ なので，$G(p, G(P))$ の辺の総数は $O(|p| \cdot |P|^2)$ である．次に，この XG グラフ上で K 最短経路問題を解くことを考える．K 最短経路問題を解くアルゴリズムはいくつか提案されているが [8], [14]，本稿では Dijkstra の最短経路アルゴリズムの拡張を用いるとする．そのアルゴリズムの時間計算量は $O(K \cdot |E| \cdot \log |V|)$ (E が辺集合，V はノード集合) である．XG グラフの辺数は $O(|p| \cdot |P|^2)$，ノード数は $O(|p| \cdot |P|)$ なので，XG グラフ上で K 最短経路問題を解く時間計算量は $O(K \cdot |p| \cdot |P|^2 \cdot \log(|p| \cdot |P|))$ であり，これが提案アルゴリズムの時間計算量である．

6.2 一般の場合

$p = /ax[1]::l[1][exp[1]]/\cdots/ax[m]::l[m][exp[m]]$ を XPath 式とする．p から述語を用いたロケーションステップを削除した XPath 式を $sp(p)$ と定義する．すなわち，

$$sp(p) = /ax[1]::l[1]/\cdots/ax[m]::l[m]$$

である．

以下にアルゴリズムを示す．P の下での p に対する K 最適編集操作列を求めるため，今回も XG グラフ $G(sp(p), G(P))$ を構成し，その上で K 最短経路問題を解く．しかし，今回は p が単純とは限らないため，K 最短経路問題を解く前に $G(sp(p), G(P))$ を以下のように修正する．(注1)

- $exp[i] \neq \epsilon$ とする．ロケーションステップ $ax[i]::l[i][exp[i]]$ の削除コストを $\gamma(ax[i]::l[i] \to \epsilon) + \gamma(exp[i] \to \epsilon)$ とする．ここで，"$exp[i] \to \epsilon$" は $exp[i]$ における全てのロケーションステップを削除する編集操作列を表す (下記の 3-a 行目)．
- $exp[i] \neq \epsilon$ の場合，$exp[i]$ の修正コストも必要である．これを求めるため，XPath 式 $/l[i]/exp[i]$ と正規木文法 $G = (N, T, l[i], P)$ に対してアルゴリズムを再帰的に呼び出す．その結果を図 3 の変換を用いて $G(sp(p), G(P))$ に組み込む．ここで，再帰呼び出しで得られた K 個の編集操作列を図 3 右の K 個の辺 $e_1, \cdots, e_K$ に割り当てる (下記の 3-b 行目)．

3-a 行目と 3-b 行目において，D_i と C_i は式 (3) と式 (2) で定義される辺集合である．

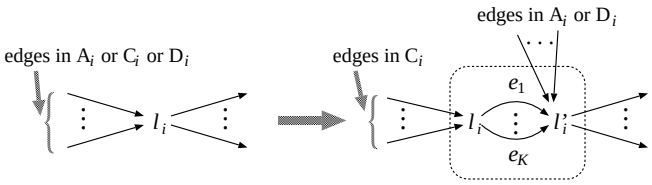

図 3 ノード l_i に対する変換

FindKPaths(G, p, K)

入力: 正規木文法 $G = (N, T, S, P)$，XPath 式 $p = /ax[1]::l[1][exp[1]]/\cdots/ax[m]::l[m][exp[m]]$，正整数 K．
出力: D の下での p に対する K 最適編集操作列 $s_1, \cdots, s_K$．

（1） 遷移規則グラフ $G(P)$ を構成する．
（2） XG グラフ $G(sp(p), G(P))$ を構成する．
（3） $exp[i] \neq \epsilon$ なる任意の $1 \leqq i \leqq m$ に対して，$G(sp(p), G(P))$ を次のように修正する．
（a） 各辺 $e \in D_i$ に対して，$\gamma(e) \leftarrow \gamma(e) + \gamma(exp[i] \to \epsilon)$ とする．
（b） 各ノード $l_i \in V_i$ に対して，次の (i) – (iii) を行う．
i. l_i を図 3 に沿って変換する．
ii. FindKPaths(G', p', K) を呼び出す．ここで，$G' = (N, T, X, P)$(X は $X \to ar \in P$ であるノード (X, l_i) の X) かつ $p' = /l_i/exp[i]$ である(注2)．得られた結果を $sc'_1, \cdots, sc'_K$ とする．
iii. $\gamma(e_j) \leftarrow \gamma(s'_j)$ と設定する ($1 \leqq j \leqq K$)．
（4） $G(sp(p), G(P))$ 上で K 最短経路問題を解く．結果を $sc_1, \cdots, sc_K$ とする．
（5） $sc_1(p), \cdots, sc_K(p)$ を返す．

上記アルゴリズムの時間計算量は $O(K \cdot |p| \cdot |P|^2 \cdot \log(|p| \cdot |P|))$ である．

6.3 グラフの最適化・枝刈りに関する考察

以上で述べたアルゴリズムにおいては，遷移規則グラフ，XG グラフ共に枝刈りについては考慮していなかった．しかし，遷移規則グラフにおいてノードを最適化し，XG グラフにおいて目的となる受理ノードに到達不可能なノードを削除することで計算量を削減できることが考えられる．

本章では，まず遷移規則グラフにおける最適化について述べ，その後 XG グラフにおける枝刈りについて述べる．

(注1)：述語内の式 exp の右辺や (不) 等号を正確に修正するのは困難なため，exp の左辺の修正のみ考える．

(注2)：l_i が p' の先頭に付加されているので，再帰呼び出しにおいては $\gamma(n_0 \to l) = 0 (l = (l_i)_0$ の場合)，$\gamma(n_0 \to l) = \infty (l \neq (l_i)_0$ の場合) と仮定する．ここで n_0 は開始ノードである．

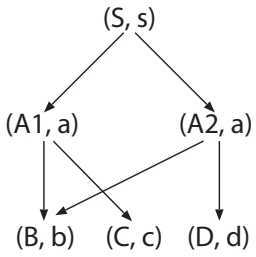

図 4　非終端記号が競合している遷移規則グラフ

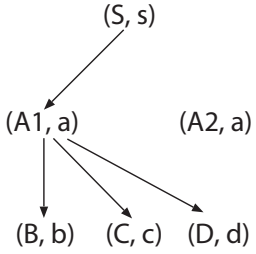

図 5　非終端記号が競合していない遷移規則グラフ

6.3.1　遷移規則グラフの最適化

正規木文法においては非終端記号の競合を許すため，同じ終端記号が遷移規則グラフの複数箇所に出現しうる．そのため，出力された K 個の最適編集操作列 $sc_1, \cdots, sc_K$ を適用して得られた XPath 式 $sc_K(p)$ の中で，文字列として同じ XPath 式が存在することがある．

例として，

$$
\begin{aligned}
N &= \{S, A1, A2, B, C, D, Pcdata\} \\
T &= \{s, a, b, c, d, pcdata\} \\
S &= \{S\} \\
P &= \{S \to s(A1, A2), A1 \to a(B, C), A2 \to a(B, D) \\
&\quad B \to b(Pcdata), C \to c(Pcdata), \\
&\quad D \to d(Pcdata), Pcdata \to pcdata(\epsilon)\}
\end{aligned}
$$

である正木木文法 $G = (N, T, S, P)$ を考える．このとき，G の遷移規則グラフは図 4 になる．ここで，経路 $(S, s) \to (A1, a) \to (B, b)$ と経路 $(S, s) \to (A2, a) \to (B, b)$ の各ノードの終端記号は同じであるため，XG グラフ上で K 最短経路問題を解いたときに $/\downarrow::a/\downarrow::b$ という XPath 式が複数個出力されうる．このような重複を回避するため，図 4 のように同じ終端記号を持つノードを縮約する．以下，このような縮約について形式的に述べる．

遷移規則グラフ $G(P) = (V, E)$ は次のように定義される有向グラフである．

$$
\begin{aligned}
V &= \{(X, a) \mid X \to ar \in P\} \\
E &= \{(X, a) \to (X', a') \mid X \to ar \in P, \\
&\quad X' \text{ が } r \text{ に出現，} X' \neq \text{Pcdata，ある } a' \in T \text{ と} \\
&\quad N \text{ 上の正規表現 } r' \text{ に対して } X' \to a'r' \in P\}
\end{aligned}
$$

$X_i \neq X_j, a_i \neq a_j$, かつノード $\{(S, a) | S \to ar \in P\}$ から到達可能なノード (X_i, a_i)，(X_j, a_j) が存在する状況で次の条件を全て満たす場合，下記 (1),(2) で遷移規則グラフの最適化が可能である．

- (X_i, a_i) から (X_j, a_j) には到達可能でない
- (X_j, a_j) から (X_i, a_i) には到達可能でない
- (X_i, a_i) の子として (X_i, a_i) を持ち得ない
- (X_j, a_j) の子として (X_j, a_j) を持ち得ない

ここで，$E_i, E_j \in E$ とする．

（1）　$E_i = \{(X_i, a_i) \to (X'_i, a'_i) \mid X_i \to a_i r \in P, X'_i$ が r に出現，$X'_i \neq$ Pcdata，ある $a'_i \in T$ と N 上の正規表現 r' に対して $X'_i \to a'_i r' \in P\}$ と $E_j = \{(X_j, a_j) \to (X'_j, a'_j) \mid X_j \to a_j r \in P, X'_j$ が r に出現，$X'_j \neq$Pcdata，ある $a'_j \in T$ と N 上の正規表現 r' に対して $X'_j \to a'_j r' \in P\}$ に対し，$a'_i \neq a'_j$ の場合，E に $(X_i, a_i) \to (X'_j, a'_j)$ を追加する．

（2）　$E_j = \{(X_{j-1}, a_{j-1}) \to (X_j, a_j) \mid X_j \to a_{j-1} r \in P, X_j$ が r に出現，$X_j \neq Pcdata$，ある $a_j \in T$ と N 上の正規表現 r' に対して $X_j \to a_j r' \in P\}$ に対し，E から E_j を削除し，E に $(X_{j-1}, a_{j-1}) \to (X_i, a_i)$ を追加する．

6.3.2　XG グラフの枝刈りによる時間計算量の推移

G が局所木文法の場合は K 最短経路問題を解くにあたり，目的となる受理ノードに到達不可能なノードを削除することで計算量を削減できることが考えられる．

目的ノードに到達可能であるか否かは $\to^+$ 軸と $\leftarrow^+$ 軸の有無によって変わるため，場合分けをして考察を行う．

局所木文法 $G = (N, T, S, P)$ と XPath 式 p に対する XG グラフ $G(p, G(P))$ のサイズを考える．また簡単のため，G の遷移規則グラフ $G(P)$ を完全 n 分木と仮定する．

(1) $\to^+$ 軸及び $\leftarrow^+$ 軸が p に含まれない場合: 受理ノードに到達可能な経路は，仮定よりただ 1 つとなる．よって，$G(P)$ における有効なノード数は $\log |T|$ となり，XG グラフのサイズは $O(|p| \cdot \log |T|)$ となる．この場合の時間計算量は $O(K \cdot |p| \cdot (\log |T|)^2 \cdot \log(|p| \cdot \log |T|))$ である．

(2) $\to^+$ 軸または $\leftarrow^+$ 軸が p に含まれる場合: $\to^+$ 軸または $\leftarrow^+$ 軸が最初に出現するノードの，文書要素からの深さを l とする．仮定より，$G(P)$ において有効なノード数は $\frac{|T|}{2^l}$ となり，XG グラフのサイズは $|p| \cdot \frac{|T|}{2^l}$ となる．このとき，$l = 0$，つまり文書要素に $\to^+$ 軸または $\leftarrow^+$ 軸が出現することは起こり得ないため，XG グラフのサイズを $\frac{1}{2}$ 以下にすることが可能である．この場合の時間計算量は $O(K \cdot |p| \cdot \frac{|T|}{2^l}^2 \cdot \log(|p| \cdot \frac{|T|}{2^l}))$ である．

7. む　す　び

本稿では XPath 式 p，正規木文法 G，正整数 K に対し，p に対し最も “近い” K 個の修正候補 XPath 式を発見するアルゴリズムを提案し，XPath 式修正候補発見問題の NP 困難性を証明した．

今後の課題として，出力の質に繋がるコスト設定についての考察と，評価実験によるその実証が挙げられる．例えば，XPath 式においてルートに近いロケーションステップほど一般にはより重要であると考えられる．そこで，XG グラフにおいて遷移

規則グラフを繋ぐ辺 D_i, C_i のコストを添え字が小さくなるほど大きくし，出力される K 最適編集操作列 $sc_1, \cdots, sc_K$ がどのように変化するかを実験する必要がある．

また，動的計画法などを用いることでも，本稿で扱った最短経路問題を解くことと同様に有用な結果を得られるという見込みがあるが，今後それらの手法の違いについて詳しく検討していく必要がある．

さらに，XPath 式の修正の支援という観点からは，対象となる XPath 式のどの部分が妥当でないかを特定すること，及びそれを提示することは修正候補の提示と同じく重要であると考えられるため，それらについても検討を進めていきたい．

文　献

[1] 林 英志, 日高 隆博, 山本 晋一郎, 小林 隆志, 上原 正太, 間瀬 順一, 鈴村 延保, 阿草 清滋 “メタ情報とコンテキスト情報を用いた入力補完機能と XPath 入力への応用”, 情報処理学会研究報告, Vol.2010-SE-167, No.28, 2010.

[2] 森嶋 厚行, 松本 明, 北川 博之 “例示操作に基づく XQuery 問合せの学習機構”, 日本データベース学会 Letters, Vol.1, No.1, pp. 15-18, 2002.

[3] ALTOVA “XMLSpy”, http://www.altova.com/jp/xmlspy.html.

[4] Sihem Amer-Yahia, SungRan Cho, and Divesh Srivastava “Tree Pattern Relaxation”, Proc. EDBT, pp. 89–102, 2002.

[5] Sihem Amer-Yahia, Laks V.S. Lakshmanan, and Shashank Pandit “FleXPath: Flexible structure and full-text querying for XML”, Proc. SIGMOD, pp. 83–94, 2004.

[6] B. Choi “What Are Real DTDs Like”, Proc. Fifth Int’l Workshop Web and Databases (WebDB’02), pp. 43–48, 2002.

[7] S. Cohen and T. Brodianskiy “Correcting queries for XML”, Information Systems, Vol.34, No.8, pp. 690–710, 2009.

[8] D. Eppstein “Finding the k shortest paths”, SIAM J. Computing, Vol.28, No.2, pp. 652–673, 1998.

[9] G. Gang and C. Rada “Efficiently Querying Large XML Data Repositories: A Survey”, IEEE Transactions on Knowledge and Data Engineering, Vol.19, Issue 10, pp. 1341–1403, 2007.

[10] Z. G. Ives, A. Y. Halevy, and D. S. Weld “An XML query engine for network-bound data”, The VLDB Journal, Vol.11, No.4, pp. 380–402, 2002.

[11] K. Ikeda and N. Suzuki, “Finding top-K Correct XPath Queries of User’s Incorrect XPath Query”, The 23rd International Conference on Database and Expert Systems Applications (DEXA 2012), LNCS 7446, pp.116–130, 2012.

[12] G. Li, J. Feng, J. Wang, and L. Zhou, “Effective keyword search for valuable lcas over xml documents”, Proc. CIKM, pp.31–40, 2007.

[13] M. Makoto et.al.,“Taxonomy of XML schema languages using formal language theory”, ACM Trans. Internet Technol, Vol.5, No.4, pp. 660–704, 2005.

[14] E. Martins “K-th shortest paths problem”, http://www.mat.uc.pt/ eqvm/OPP/KSPP/KSPP.html.

[15] A. Marzal and E. Vidal, ”Computation of Normalized Edit Distance and Applications”, IEEE Transactions on Pattern Analysis and Machine Intelligence, 15(9), pp.926–932, 1993.

[16] A. Morishima, H. Kitagawa, and A. Matsumoto “A machine learning approach to rapid development of XML mapping queries”, Proc. ICDE, pp. 276–287, 2004.

[17] A. Termehchy and M. Winslett, “Using structural information in XML keyword search effectively”, ACM Trans. Database Syst., 36(1), 2011.

[18] Schmidt, A., Waas, F., Kersten, M., Carey, M., Manolescu, I., Busse, R.: Xmark: A benchmark for xml data management. In: Proc. VLDB. pp. 974–985 (2002)

[19] Y. Wu, N. Lele, R. Aroskar, S. Chinnusamy, and S.Brenes “XQGen: an algebra-based XPath query generator for micro-benchmarking”, Proc. CIKM, pp. 2109–2110, 2009.

[20] Y. Xu and Y. Papakonstantinou, “Efficient keyword search for smallest LCAs in XML databases”, Proc. SIGMOD, pp.527–538, 2005.