Vol.2010-DBS-151 No.31
2010/11/13

妥当でないXPath式に対するK最適修正候補発見アルゴリズム

池田光雪[†1]　鈴木伸崇[†1]

DTDが存在する状況において，XPath式を用いて検索を行う場合を考える．DTDが複雑であったりユーザがXPathに不慣れな場合，DTDに関して妥当なXPath式を記述するのは必ずしも容易でない．本稿では，軸としてchild, descendant-or-self, following-sibling, preceding-siblingのみを許すXPath式を対象とし，XPath式p，DTD D，および正整数Kに対して，Dに関して妥当なXPath式をpに類似したものからK個列挙するアルゴリズムを提案する．XPath式間の類似性は編集距離に基づいて判定する．また，提案アルゴリズムに関する評価実験について述べる．

An Algorithm for Finding K Optimum Corrections to Invalid XPath Expression

Kosetsu Ikeda[†1] and Nobutaka Suzuki[†1]

In this paper, we consider using XPath expressions in the presence of DTDs. If a DTD is complex or a user is unfamiliar with XPath, it is not always easy to write XPath expressions conforming to the DTD. In this paper, we consider XPath expressions containing only child, descendant-or-self, following-sibling, and preceding-sibling axes. We first propose an algorithm that finds, for a DTD D, an XPath expression p, and a positive integer K, K optimum valid candidates of p such that the expressions conform to D, where the similarity between two XPath expressions are measured by the edit distance between XPath expressions. Then we show an experimentation of the algorithm.

†1 筑波大学
University of Tsukuba

1. はじめに

XMLデータをデータベース等で蓄積・管理する場合，DTD等のスキーマを用いて格納すべきデータの構造を予め定義しておき，それに関して妥当なXMLデータを格納するのが一般的である．ここで，DTDが存在する状況において，XPath式で検索を行うことを考える．DTDが複雑であったりユーザがDTDやXPathに不慣れな場合，DTDの規則を満たす(妥当な)XPath式を記述するのは必ずしも容易でない．また，DTDとその下にあるXMLデータが管理者によって更新され，検索対象のデータの構造が変化することもあり得る．この場合，妥当であったXPath式の妥当性が失われる可能性がある．

上記のような場合，DTD Dに関して妥当でないXPath式pに対して，Dに関して妥当かつpとの類似度が高い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．ただし，Dに関して妥当かつpとの類似度が高いXPath式は一般に複数存在するため，複数の候補からユーザが所望の式を選択できることが望ましい．そこで，本稿では，XPath式p，DTD D，および正整数Kに対して，Dに関して妥当なXPath式をpに類似したものからK個列挙するアルゴリズムを提案する．ただし，本稿では，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を許すという制限されたXPath式を対象とする．また，XPath式間の類似度の判定のため，本稿ではXPathに関する編集距離を導入する．XPath式p_1とp_2間の編集距離を，p_1をp_2に更新するために必要な編集操作(要素名の置換，軸の置換など)のコストの総和と考える．

本アルゴリズムの特徴は，DTDの全体構造を把握していなくても，目的の要素名が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 html (div)*>
<!ELEMENT div  (div|p)+>
<!ELEMENT p    (#PCDATA|span)*>
<!ELEMENT span (#PCDATA)>
```

ここで，ユーザが`/spen`というXPath式を入力したとする．本稿のアルゴリズムを用いれば，このXPath式に類似した妥当な式の列挙が得られる(以下は$K = 4$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. //span                    (1.25)
```

```
2. /html//span              (2.25)
3. //p/span                 (2.25)
4. /html/div/p/span         (3.25)
```

このように，spen の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変更することができるので，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」等が可能である．

本稿と最も関連が強いのは文献[4] 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 に関して妥当で 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 DTD ではなく有向非循環グラフ (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．文献[7),11)] は，ユーザへの質問を通して所望の XQuery 式を学習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，それと類似した複数の正解候補を出力する．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[2),3)] 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．XMLSpy[1)] などの XML エディタには，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，修正候補の列挙は行わない．これまで，文字列に対する編集距離やアライメントに関する研究は数多くなされている (文献[8),9)] など)．本研究は，この考え方を XPath や DTD に応用したものであるといえる．

2. 諸 定 義

Σ をラベル (要素名) の集合とする．DTD を組 (d, s) と表す．ここで，d は Σ から Σ 上の正規表現への写像であり，$s \in \Sigma$ は文書要素である．ラベル $a \in \Sigma$ に対して，$d(a)$ を a の内容モデルという．例えば，前章の DTD は組 (d, html) と表すことができ，ここで $d(\mathrm{html}) = \mathrm{div}^*$，$d(\mathrm{div}) = (\mathrm{div}|\mathrm{p})^+$，$d(\mathrm{p}) = (\epsilon|\mathrm{span})^*$，$d(\mathrm{span}) = \epsilon$ である．内容モデル $d(a)$ の言語を $L(d(a))$ と表す．ラベル b, c に対して，もし b が c より後に出現するような文字列 $s \in L(d(a))$ が存在するならば，$d(a)$ において b は c の右側に出現可能であるという．例えば，$d(a) = c(f|e)^*$ の場合，$d(a)$ において e は c の右側に出現可能である．同様に，$d(a)$ において b が c の左側に出現可能であることも定義できる．DTD $D = (d, s)$ 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，D において a から b へ到達可能であるという．

- $a = b$，または，b が $d(a)$ に出現する
- あるラベル a' に対して，a から a' へ到達可能，かつ，b が $d(a')$ に出現する

以下，DTD に出現する任意のラベルは文書要素から到達可能であると仮定する．

$ax :: l$ という形の式をロケーションステップといい，ここで (i) ax は $\downarrow$ (child 軸)，$\downarrow^*$ (descendant-or-self 軸)，$\rightarrow^+$ (following-sibling 軸)，$\leftarrow^+$ (preceding-sibling 軸) のいずれか，(ii) l はラベルである．$/ax[1] :: l[1]/\cdots/ax[m] :: l[m]$ の形をした式のことを XPath 式といい，ここで $ax[i]$ は軸，$ls[i]$ はラベルである．XPath 式 $/ax[1] :: l[1]/\cdots/ax[m] :: l[m]$，ラベル l, および $i < j$ に対して，もし次のいずれかの条件が成り立つならば l を $l[j]$ の親ラベルという．

- $ax[i] = \downarrow$，$l = l[i]$ かつ $ax[i+1], \cdots, ax[j] \in \{\rightarrow^+, \leftarrow^+\}$
- $ax[i] = \downarrow^*$，l は $l[i]$ から到達可能，$l[i+1]$ が $d(l)$ に出現，かつ，$ax[i+1], \cdots, ax[j] \in \{\rightarrow^+, \leftarrow^+\}$

例えば，XPath 式 $/ \downarrow :: a/ \downarrow :: b/ \rightarrow^+ :: c/ \leftarrow^+ :: d$ において，d の親ラベルは b である．XPath 式 p のロケーションステップ数を $|p|$ と表す．DTD $D = (d, s)$ と XPath 式 $p = /ax[1] :: l[1]/\cdots/ax[m] :: l[m]$ に対して，以下の (1) と (2) が成立するならば p は D に関して妥当であるという．

(1) $ax[1] = \downarrow$ かつ $l[1] = s$，または，$ax[1] = \downarrow^*$ かつ $l[1]$ は D 中に出現する

(2) $2 \le i \le m$ に対して以下が成り立つ
- $ax[i] = \downarrow$ かつ $l[i]$ が $d(l[i-1])$ に出現する
- $ax[i] = \downarrow^*$ かつ D において $l[i-1]$ から $l[i]$ へ到達可能である
- $ax[i] = \rightarrow^+$ かつ $d(l)$ において $l[i]$ が $l[i-1]$ の右側に出現可能である．ここで，l は $l[i]$ の親ラベルである．
- $ax[i] = \leftarrow^+$ かつ $d(l)$ において $l[i]$ が $l[i-1]$ の左側に出現可能である

3. K 最適修正候補発見アルゴリズム

本章では，DTD D，XPath 式 p，正整数 K が与えられた時に，D に妥当な XPath 式を p に近いものから K 個出力するアルゴリズムを示す．

Vol.2010-DBS-151 No.31
2010/11/13

3.1 編集操作

まず，準備として編集操作等の定義を行う．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．
- ラベルの置換：ラベル l を l' に置換する．$l \to l'$ と表す．
- 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．
- 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．

編集操作 op を XPath 式の i 番目のロケーションステップに適用する場合，op に位置 i を付加して $[op]_i$ と表す．op がロケーションステップの追加であった場合，$[op]_i$ は i 番目のロケーションステップの直後に追加操作を行う．p の先頭にロケーションステップを追加する場合，位置として 0 を指定する．

s を (位置付き) 編集操作の系列とする．s を p に適用して得られる XPath 式を $s(p)$ と表す．例えば，$s = [\epsilon \to \downarrow :: b]_1 [c \to f]_3$，$p = /\downarrow^* :: a/\downarrow :: d/\downarrow :: c$ の場合，$s(p) = /\downarrow^* :: a/\downarrow :: b/\downarrow :: d/\downarrow :: f$ である．以下，同じロケーションステップに対して軸の置換とラベルの置換はそれぞれ高々1 回しか適用されないと仮定する．例えば，XPath 式に $[a \to b]_i [b \to c]_i$ を適用した結果と $[a \to c]_i$ を適用した結果は同一である．

$/ax[1] :: l[1]/\cdots/ax[m] :: l[m]$ を XPath 式とする．アルゴリズムの定義を簡潔にするため，本稿では以下の仮定を置く (アルゴリズムを拡張し，仮定を置かない場合に対応することも容易に可能である)．

- $ax[i] \in \{\downarrow, \downarrow^*\}$ のとき，$ax[i]$ から $\downarrow$ と $\downarrow^*$ への置換のみ可
- $ax[i] \in \{\leftarrow^+, \rightarrow^+\}$ のとき，$ax[i]$ から $\leftarrow^+$ と $\rightarrow^+$ への置換のみ可
- $\rightarrow^+$ 軸や $\leftarrow^+$ 軸を含むロケーションステップの追加は行わない

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，ここで γ がコスト関数である．以下，$\gamma(op) \geq 0$ と仮定する．コスト関数として，定数だけでなく関数を指定できる．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 $s = op_1 op_2 \cdots op_n$ のコストを $\gamma(s) = \sum_{1 \leq i \leq n} \gamma(op_i)$ と定義する．DTD D，XPath 式 p，正整数 K に対して，D 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 $s_1, \cdots, s_K$ である．

- $s_1(p), \cdots, s_K(p)$ はすべて D に関して妥当である
- $\gamma(s_1) \leq \cdots \leq \gamma(s_K)$
- $s_1, \cdots, s_K$ は最適，すなわち，p に対する任意の編集操作列 s に対して以下が成り立つ
  - もし $s(p)$ が D に関して妥当ならば，$s(p) \in \{s_1(p), \cdots, s_K(p)\}$ または $\gamma(s) \geq \gamma(s_K)$

$s_1(p), \cdots, s_K(p)$ を D の下での p に対する K 最適修正候補という．

3.2 アルゴリズム

本稿で提案するアルゴリズムの概要を示す．

- 入力：DTD D，XPath 式 p，正整数 K
- 出力：D の下での p に対する K 最適修正候補 $p_1, \cdots, p_K$
- 処理内容：
  - (1) D から DTD グラフ $G(D)$ を構成する．
  - (2) p と $G(D)$ から合成グラフ $G(p, G(D))$ を構成する．
  - (3) $G(p, G(D))$ 上で K 最短経路問題を解く．その出力から K 最適修正候補 $p_1, \cdots, p_K$ を得る．

以下，上記 (1)〜(3) について説明する．

(1) DTD グラフ

$D = (d, s)$ を DTD とする．このとき，D の DTD グラフ $G(D) = (V, E)$ は次のように定義される有向グラフである．

$V = \{l \mid l$ は D に出現するラベル $\}$

$E = \{l \to l' \mid l'$ は $d(l)$ に出現するラベル $\}$

例えば，DTD $D = (d, s)$ において，$d(s) = ba^*$, $d(a) = c|d$, $d(b) = d$, $d(c) = \epsilon$, $d(d) = b|\epsilon$ であるとする．このとき，D の DTD グラフは図 1 のようになる．

(2) 合成グラフ

XPath 式 p と DTD グラフ $G(D)$ から合成グラフを構成する．まず，以下の 3 つの場合に分けて例を用いて説明する．

(2a) $\downarrow$ 軸のみを用いる場合

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

(2c) $\rightarrow^+$ 軸と $\leftarrow^+$ 軸を用いる場合

次に，合成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．

Vol.2010-DBS-151 No.31
2010/11/13

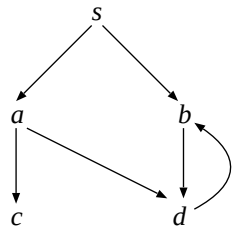

図 1 DTD グラフ

(2a) ↓ 軸のみを用いる場合

まず，XPath 式 $p = /\downarrow::a/\downarrow::d$ と図 1 の DTD グラフ $G(D)$ に対して，どのような合成グラフが構成されるのかを述べる．軸は ↓ 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 $G(D)$ の合成グラフ $G(p, G(D))$ を図 2 に示す．この図に示すように，合成グラフは $G(D)$ 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_0, n_1, 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．DTD 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，$G(D)$ におけるノード s は，$G(p, G(D))$ の一番上の DTD グラフでは s_0，その下の DTD グラフでは s_1 と表記される．

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

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

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 $n_0 \rightarrow s_0$ 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 $n_0 \rightarrow s_0$ 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 $[\epsilon \rightarrow \downarrow::s]_0$ を表す．次に，図 2 の辺 $s_0 \dashrightarrow b_1$ 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．

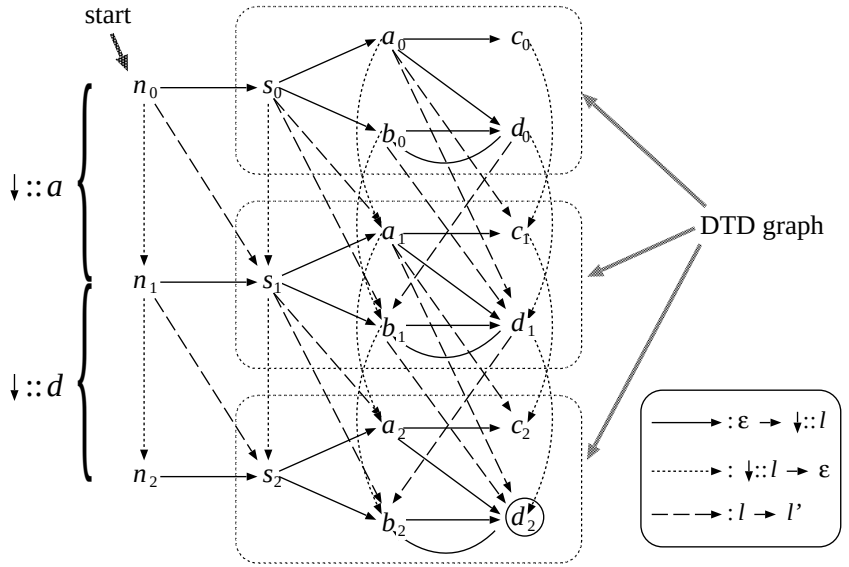

図 2 合成グラフ $G(p, G(D))$

すなわち，辺 $s_0 \dashrightarrow b_1$ は編集操作 $[a \rightarrow b]_1$ を表している．最後に，図 2 の辺 $b_1 \dashrightarrow b_2$ 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 $b_1 \dashrightarrow b_2$ は編集操作 $[\downarrow::d \rightarrow \epsilon]_2$ を表す．

図 2 において，n_0 を開始ノード，d_2 を受理ノードという．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 D に関して妥当な XPath 式を表す．例えば，図 2 の次の経路を考える．

$$p' = n_0 \rightarrow s_0 \dashrightarrow a_1 \dashrightarrow d_2 \tag{1}$$

最初の辺 $n_0 \rightarrow s_0$ はロケーションステップ $\downarrow::s$ の追加 $[\epsilon \rightarrow \downarrow::s]_0$ を表す．次の辺 $s_0 \dashrightarrow a_1$ は，$\downarrow::a$ におけるラベルの置換 $[a \rightarrow a]_1$ 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 $a_1 \dashrightarrow d_2$ 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 D に関して妥当である．

Vol.2010-DBS-151 No.31
2010/11/13

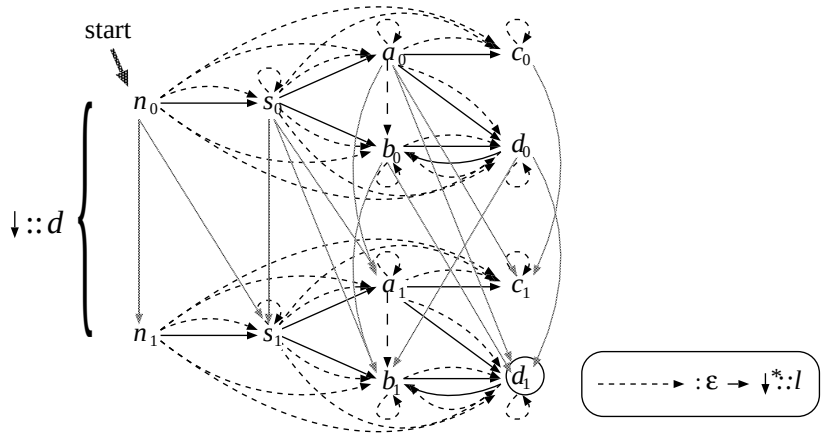

図 3 $\downarrow^* :: l$ の追加

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

まず，ロケーションステップ $\downarrow^* :: l$ の追加について説明する．図 3 は，DTD グラフは前述の (2a) と同様かつ XPath 式を $p = /\downarrow :: d$ とした場合に，ロケーションステップ $\downarrow^* :: l$ を追加する場合の合成グラフを表したものである．この図において，破線の辺がロケーションステップ $\downarrow^* :: l$ の追加を表している．例えば，辺 $s_0 \dashrightarrow d_0$ は「p のロケーションステップは用いずに，ノード s から $\downarrow^*$ 軸でノード d へ移動する」ことを表しており，ロケーションステップ $\downarrow^* :: d$ の追加 $[\epsilon \rightarrow \downarrow^* :: d]_0$ を表す．前述の例と同様，開始ノードから受理ノードへの経路が p を修正して得られる妥当な XPath 式を表す．例えば，経路 $n_0 \rightarrow s_1 \dashrightarrow d_1$ は，p のロケーションステップ $\downarrow :: d$ のラベルを s に置換してロケーションステップ $\downarrow^* :: d$ を追加して得られる XPath 式，すなわち $/\downarrow :: s/\downarrow^* :: d$ を表す．

次に，$\downarrow$ 軸と $\downarrow^*$ 軸の置換について考える．ここでは $\downarrow$ 軸を $\downarrow^*$ 軸へ置換する場合について説明する．図 4 は，DTD グラフと XPath 式は前述のものと同様で，ロケーションステップ $\downarrow :: a$ を $\downarrow^* :: l$ に置換する (軸を $\downarrow$ から $\downarrow^*$ に置換し，ラベルを a から l に置換する) 場合の合成グラフを表したものである．ただし，図が煩雑になるのを避けるため，ロケーションステップ $\downarrow^* :: l$ の追加，ロケーションステップ $\downarrow :: a$ の削除，ロケーションステップ $\downarrow :: a$ に

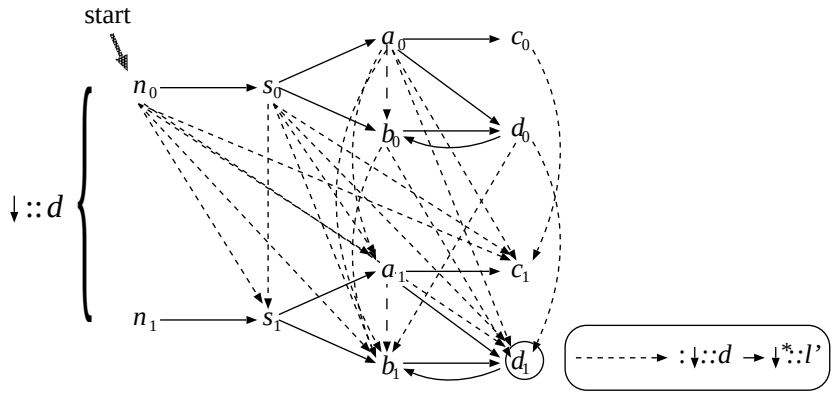

図 4 $\downarrow$ 軸から $\downarrow^*$ 軸への置換

おけるラベルの置換に関する辺は省略している．図 4 において，破線の辺が $\downarrow :: a$ から $\downarrow^* :: l$ への置換を表している．これは，DTD グラフの各ノード l に対して，l の 1 つ下の DTD グラフに含まれる l の子孫ノードへ辺を追加したものである．例えば，辺 $n_0 \dashrightarrow a_1$ は「ルートノードから $\downarrow^*$ 軸でノード a へ移動する」こと，すなわち $\downarrow :: d$ を $\downarrow^* :: a$ に置換することを表す．ここで，次の経路について考える．

$$p' = n_0 \rightarrow s_0 \dashrightarrow d_1 \tag{2}$$

この経路は，p にロケーションステップ $\downarrow :: s$ を追加し $\downarrow :: d$ を $\downarrow^* :: d$ に置換して得られる XPath 式 $/\downarrow :: s/\downarrow^* :: d$ を表している．

最後に，$\downarrow^*$ 軸から $\downarrow$ 軸への置換は，図 2 におけるラベルの置換 $(l \rightarrow l')$ と同様の辺を用いることで表現できる．また，$\downarrow^*$ 軸を含むロケーションステップの削除は (2a) のロケーションステップの削除と同様である．詳細は紙面の都合で省略する．

(2c) $\rightarrow^+$ 軸と $\leftarrow^+$ 軸を用いる場合

$\rightarrow^+$ 軸と $\leftarrow^+$ 軸の扱いについて考える．図 5 は，DTD グラフは前述のものと同様かつ XPath 式を $p = /\rightarrow^+ :: d$ とした場合の合成グラフである．まず，$s_0 \rightarrow s_1$ や $a_0 \rightarrow a_1$ などの添え字のみ異なるラベルを結ぶ辺を考える．これはノードの位置が変わらない (p の $\rightarrow^+ :: d$ は無視される) ことを意味し，p から $\rightarrow^+ :: d$ を削除することを表す．次に，「兄弟のラベル」を結ぶ 2 種の破線の辺を考える．例えば，$d(s) = ba^*$ において a と b が兄弟であり，$a_0 \dashrightarrow b_1$ や $b_0 \dashrightarrow a_1$ など a_0, b_0, a_1, b_1 間に計 4 個の辺が追加されている．これは，

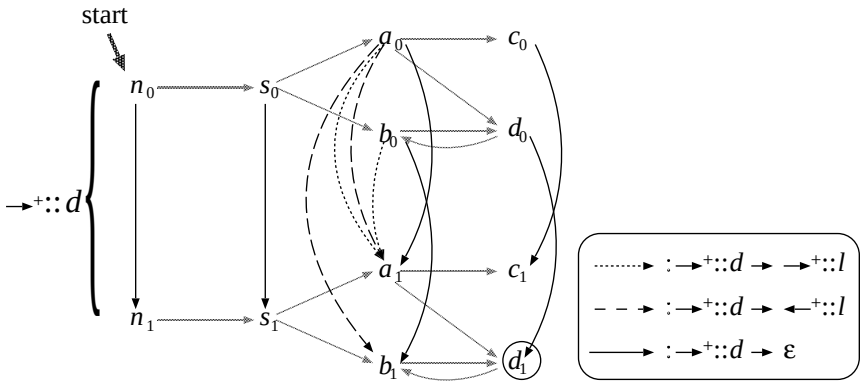

図 5 $\rightarrow^+$ 軸と $\leftarrow^+$ 軸の扱い

⇢ が $\rightarrow^+$ への置換，⇢ が $\leftarrow^+$ への置換に対応する．例えば，辺 a_0 ⇢ b_1 は，$\leftarrow^+$ 軸を用いてノード a から b へ移動することを意味し，p のロケーションステップ $\rightarrow^+::d$ を $\leftarrow^+::b$ に置換することを表す．一方，b_0 ⇢ a_1 は，p のロケーションステップ $\rightarrow^+::d$ を $\rightarrow^+::a$ に置換することを表す．なお，兄弟間の移動が不可能な場合は辺は設けられない．例えば，$d(s) = ba^*$ において，a は b の左側に出現可能でないので，辺 b_0 ⇢ a_1 は設けられていない．

(2d) 合成グラフの形式的定義と辺に付与されるコスト

以上で示した合成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．$D = (d, s)$ を DTD，$G(D) = (V, E)$ を DTD グラフ，$p = /ax[1]::l[1]/\cdots/ax[m]::l[m]$ を XPath 式とする．$G_i(D) = (V_i, E_i)$ $(0 \le i \le m)$ を $G(D)$ 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，$V_i = \{l_i \mid l \in V\}$ かつ $E_i = \{l_i \rightarrow l'_i \mid l \rightarrow l' \in E\}$ である．合成グラフ $G(p, G(D)) = (V', E')$ は次のように定義される有向グラフである ($n_0, \cdots, n_m$ の n は $n \notin V$ なるラベルとする)．

$$V' = \{n_0, \cdots, n_m\} \cup V_0 \cup \cdots \cup V_m$$
$$E' = E_{insc} \cup (E'_0 \cup \cdots \cup E'_m) \cup (F_1 \cup \cdots \cup F_m) \quad (3)$$

式 (3) の E_{insc} は $\downarrow::l$ の追加を表す辺 (図 2 の "$\epsilon \rightarrow \downarrow::l$" 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 $G_i(D)$ の辺である．

$$E_{insc} = \{n_0 \rightarrow s_0, \cdots, n_m \rightarrow s_m\} \cup (E_0 \cup \cdots \cup E_m)$$

式 (3) の E'_i $(0 \le i \le m)$ は $\downarrow^*::l$ の追加を表す辺 (図 3 の "$\epsilon \rightarrow \downarrow^*::l$" の辺に該当) であり，次のように定義される．

$$E'_i = \{n_i \rightarrow l_i \mid l_i \in V_i\} \cup \{l_i \rightarrow l'_i \mid D \text{ において } l \text{ から } l' \text{ へ到達可能 }\}$$

また，式 (3) の F_i $(1 \le i \le m)$ は，$G_{i-1}(D)$ と $G_i(D)$ の間に張られる辺であり，p の i 番目の軸 $ax[i]$ の種類に応じて次のように定義される．

- $ax[i] \in \{\downarrow, \downarrow^*\}$ の場合：$F_i = D_i \cup C_i \cup A_i$．ここで，$D_i$，$C_i$，$A_i$ は次のように定義される．D_i は図 2 の "$\downarrow::l \rightarrow \epsilon$" の辺，$C_i$ は図 2 の "$l \rightarrow l'$" の辺，A_i は図 4 の "$\downarrow::d \rightarrow \downarrow^*::l$" の辺にそれぞれ該当する．
  $$D_i = \{n_{i-1} \rightarrow n_i\} \cup \{l_{i-1} \rightarrow l_i \mid l \in V\}$$
  $$C_i = \{n_{i-1} \rightarrow s_i\} \cup \{l_{i-1} \rightarrow l'_i \mid l \rightarrow l' \in E\}$$
  $$A_i = \{n_{i-1} \rightarrow l_i \mid l_i \in V_i\} \cup \{l_{i-1} \rightarrow l'_i \mid D \text{ において } l \text{ から } l' \text{ へ到達可能 }\}$$
- $ax[i] \in \{\leftarrow^+, \rightarrow^+\}$ の場合：$F_i = D_i \cup L_i \cup R_i$．ここで，$L_i$ と R_i は次のように定義される (D_i は上で定義したものと同じ)．L_i は図 5 の "$\rightarrow^+::d \rightarrow \leftarrow^+::l$" の辺，$R_i$ は図 5 の "$\rightarrow^+::d \rightarrow \rightarrow^+::l$" の辺にそれぞれ該当する．
  $$L_i = \{l_{i-1} \rightarrow l'_i \mid d(l'') \text{ において } l' \text{ は } l \text{ の左に出現可能，} l'' \text{ は } l, l' \text{ の親ラベル }\}$$
  $$R_i = \{l_{i-1} \rightarrow l'_i \mid d(l'') \text{ において } l' \text{ は } l \text{ の右に出現可能，} l'' \text{ は } l, l' \text{ の親ラベル }\}$$

最後に，合成グラフの各辺に付与するコストを考える．**3.1** 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 (実際の値は通常ユーザ等で指定する)．

- $\gamma(l \rightarrow l')$：ラベル l を l' に置換するコスト
- $\gamma(ax \rightarrow ax')$：軸 ax を ax' に置換するコスト
- $\gamma(\epsilon \rightarrow ax::l)$：ロケーションステップ $ax::l$ を追加するコスト
- $\gamma(ax::l \rightarrow \epsilon)$：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，合成グラフの辺 $e \in E'$ のコスト $\gamma(e)$ を次のように定義する．

- $e \in E_{inssc}$ のとき：$e = l_i \rightarrow l'_i$ と表せる．この辺は $\downarrow::l'$ の追加を表すので，$\gamma(e) = \gamma(\epsilon \rightarrow \downarrow::l')$ と定義する．
- $e \in E'_i$ のとき：$e = l_i \rightarrow l'_i$ と表せる．この辺は $\downarrow^*::l'$ の追加を表すので，$\gamma(e) = \gamma(\epsilon \rightarrow \downarrow^*::l')$ と定義する．
- $e \in D_i$ のとき：$e = l_{i-1} \rightarrow l_i$ と表せる．この辺は $ax[i]::l[i]$ の削除を表すので，$\gamma(e) = \gamma(ax[i]::l[i] \rightarrow \epsilon)$ と定義する．
- $e \in C_i$ のとき：$e = l_{i-1} \rightarrow l'_i$ と表せる．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，$\gamma(e) = \gamma(ax[i] \rightarrow \downarrow) + \gamma(l[i] \rightarrow l')$ と定義する．

- $e \in A_i$ のとき：$e = l_{i-1} \to l'_i$ と表せる．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．
- $e \in L_i$ のとき：$e = l_{i-1} \to l'_i$ と表せる．この辺は $ax[i]$ の $\leftarrow^+$ への置換と $l[i]$ の l' への置換を表すので，$\gamma(e) = \gamma(ax[i] \to \leftarrow^+) + \gamma(l[i] \to l')$ と定義する．
- $e \in R_i$ のとき：$e = l_{i-1} \to l'_i$ と表せる．この辺は $ax[i]$ の $\to^+$ への置換と $l[i]$ の l' への置換を表すので，$\gamma(e) = \gamma(ax[i] \to \to^+) + \gamma(l[i] \to l')$ と定義する．

例として，次のコスト関数を考える．

$$\gamma(l \to l') = \begin{cases} 0 & l = l' \text{ のとき} \\ 1 & \text{それ以外} \end{cases} \tag{4}$$

$$\gamma(ax \to ax') = \begin{cases} 0 & ax = ax' \text{ のとき} \\ 0.5 & \text{それ以外} \end{cases} \tag{5}$$

$$\gamma(\epsilon \to ax :: l) = 1 \tag{6}$$

$$\gamma(ax :: l \to \epsilon) = 1 \tag{7}$$

式 (1) に示した経路 $n_0 \to s_0 \dashrightarrow a_1 \dashrightarrow d_2$ 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．また，式 (2) に示した経路 $n_0 \to s_0 \dashrightarrow d_1$ のコストは $\gamma(\epsilon \to \downarrow :: s) + (\gamma(\downarrow \to \downarrow^*) + \gamma(d \to d)) = 1 + 0.5 = 1.5$ となる．

(3) K 最短経路アルゴリズム

$G(D)$ を DTD グラフ，$p = /ax[1]::l[1]/\cdots/ax[m]::l[m]$ を XPath 式，$G(p, G(D)) = (V', E')$ を合成グラフとする．$n_0 \in V'$ を開始ノード，$(l[m])_m \in V'$ 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) $G(D)$ 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 V から選択し，添字 m を付加したラベル l_m 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 $G(p, G(D))$ 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[⋆1]，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．なお，無駄な探索を省くため，K 最短経路問題を解く前に，合成グラフにおいて受理ノードへ到達可能でないノード (例えば図 2 の c_2) は削除する．次の定理が成り立つ (証明は省略する)．

⋆1 実際には，再帰対象のノードとその兄弟における $\to^+$ と $\leftarrow^+$ の移動の際には，K 最短経路アルゴリズムに若干の修正が必要となる．詳細は紙面の都合で省略する．

定理 1 D を DTD，p を XPath 式，K を正整数とする．「修正候補の最後のロケーションステップのラベルは一定 (受理ノードのラベル)」という制約の下で，**3.2** のアルゴリズムは D 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，DTD D と XPath 式 p に対する合成グラフ $G(p, G(D))$ のサイズを考える．Σ を D に出現するラベルの集合とする．$G(p, G(D))$ の各ノード n に対して，n を始点とする辺の数は $O(|\Sigma|)$ である．$G(p, G(D))$ のノード数は $O(|p| \cdot |\Sigma|)$ なので，$G(p, G(D))$ の辺の総数は $O(|p| \cdot |\Sigma|^2)$ である．次に，この合成グラフ上で 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[5),6)]，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．合成グラフの辺数は $O(|p| \cdot |\Sigma|^2)$，ノード数は $O(|p| \cdot |\Sigma|)$ なので，合成グラフ上で K 最短経路問題を解く時間計算量は $O(K \cdot |p| \cdot |\Sigma|^2 \cdot \log(|p| \cdot |\Sigma|))$ であり，これが提案アルゴリズムの時間計算量である．

4. 評価実験

提案アルゴリズムを Ruby を用いて実装し，予備的な評価実験を行った．ユーザの入力した妥当でない XPath 式に対して，それに類似した妥当な式は通常多数存在するため，提案アルゴリズムの出力に常に「正解」が含まれるとは限らない．そこで，「正解」と誤りを含む妥当でない式を用意し，後者を提案アルゴリズムに入力して得られた候補の中に「正解」が含まれる割合を算出した．具体的な実験の手順は以下の通りである．DTD には XMark[10)] の auction.dtd を使用し，コスト関数には前章の式 (4)〜(7) を用いた．

(1) 妥当な XPath 式を作成する (表 1)．これらを「正解」とする．

(2) 表 1 の各 XPath 式に対して，4 種の編集操作 (ロケーションステップの追加，ロケーションステップの削除，軸の置換，ラベルの置換) を等確率で適用し，以下の妥当でない式を生成する ($i = 1, \cdots, 5$)．すなわち，1 つの「正解」に対して Level1 から Level5 の式が 5 個づつ生成される．
   - 誤りを i 個含む，妥当でない式 5 個 (Level$i$)

(3) $K = 1, \cdots, 10$ に対して (2) で得られた XPath 式を提案アルゴリズムに入力し，その出力に「正解」(編集操作を適用する前の元の XPath 式) が含まれる割合を求める．その結果を図 6 に示す (Level 毎の平均値)．

(4) 更に，DTD の下で「正解」と常に同じ結果を出力する XPath 式も「広義の正解」と

表 1 「正解」の XPath 式

1.	/site/regions/africa
2.	//closed_auction/annotation/author
3.	/site/people/person/name
4.	/site//description
5.	/site/open_auctions/open_auction/bidder/time

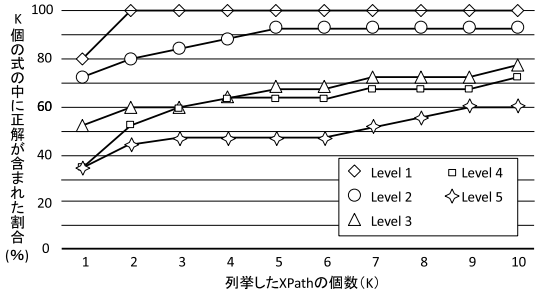

図 6 「正解」が含まれる割合

見なして (例えば，図 1 の DTD 場合 $/s/a$ と $//s/a$ は常に同じ結果を出力する)，アルゴリズムの出力に「正解」と「広義の正解」の少なくとも一方が含まれる割合を求める．その結果を図 7 に示す．

以上の結果から，妥当でない式 p が入力された場合に，ユーザが p との編集距離が大きく離れた式を敢えて必要とすることが無い限り，提案アルゴリズムを用いて「正解」または「広義の正解」が得られる可能性は高いと考えられる．ただし，今回の評価実験は妥当でない式を機械的に求めており，この点に関して今後検討の必要がある．

5. むすび

本稿では，DTD D，妥当でない XPath 式 p，正整数 K が与えられた時に，D の下で p に対する K 最適修正候補を求めるアルゴリズムを提案し，予備的な評価実験を行った．今後の課題として，XPath 式の述語や本稿で扱っていない軸 (parent 軸など) への対応などが挙げられる．また，本アルゴリズムでは，受理ノードのラベルが XPath 式の終端のラベルに限定されていた．この制限はアルゴリズム的には容易に緩和可能であるが，その場合妥当な XPath 式の総数が増加する．よって K 個の修正候補の中に目的の XPath 式が含まれなくなる確率が増加する可能性があり，今後受理ノードの指定方法について検討の必要がある．更に，被験者実験の実施など，評価実験についても検討が必要である．

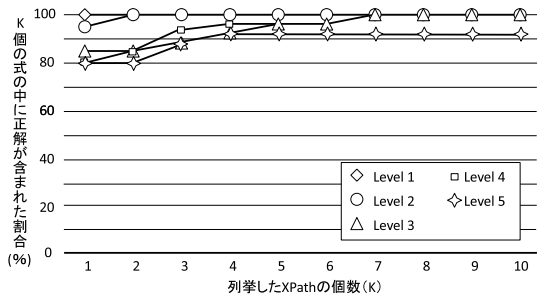

図 7 「正解」と「広義の正解」が含まれる割合

参考文献

1) ALTOVA: XMLSpy. http://www.altova.com/jp/xmlspy.html.
2) Amer-Yahia, S., Cho, S. and Srivastava, D.: Tree Pattern Relaxation, *Proc. EDBT*, pp.89–102 (2002).
3) Amer-Yahia, S., Lakshmanan, L.V. and Pandit, S.: FleXPath: Flexible structure and full-text querying for XML, *Proc. SIGMOD*, pp.83–94 (2004).
4) Cohen, S. and Brodianskiy, T.: Correcting queries for XML, *Information Systems*, Vol.34, No.8, pp.690–710 (2009).
5) Eppstein, D.: Finding the k shortest paths, *SIAM J. Computing*, Vol.28, No.2, pp. 652–673 (1998).
6) Martins, E.: K-th Shortest Paths Problem. http://www.mat.uc.pt/eqvm/OPP/KSPP/KSPP.html.
7) Morishima, A., Kitagawa, H. and Matsumoto, A.: A machine learning approach to rapid development of XML mapping queries, *Proc. ICDE*, pp.276–287 (2004).
8) Myers, E.W. and Miller, W.: Approximate matching of regular expressions, *Bulletin of Mathematical Biology*, Vol.51, No.1, pp.5–37 (1989).
9) Sankoff, D. and Kruskal, J.: *Time Warps, String Edits, and Macromolecules*, Addison-Wesley (1983).
10) Schmidt, A., Waas, F., Kersten, M., Carey, M.J., Manolescu, I. and Busse, R.: XMark: A Benchmark for XML Data Managemet, *Proc. VLDB*, pp.974–085 (2002).
11) 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).