XML検索におけるデバッグ支援システム

内田　隆彦[†]　田島　敬史[†,††]

† 北陸先端科学技術大学院大学情報科学研究科 〒923-1292 石川県能見市旭台 1-1
†† 京都大学情報学研究科 〒606-8501 京都市左京区吉田本町
E-mail: tk-uchi@jaist.ac.jp, tajima@i.kyoto-u.ac.jp

あらまし　XPath による XML データからの検索を行う場合，ユーザが記述した問合せ式になんらかの誤りがあったために，意図したデータが取り出せないということが，しばしば生じる．原因としては，タイプミスや，対象データの構造に関するユーザの勘違いなどが考えられる．そこで，本研究では，そのようなユーザによる問合せ式のデバッグ作業を支援するシステムを開発する．本研究で開発するシステムでは，ユーザが実行した問合せの結果が当初の意図に反しており，問合せ式に何か誤りがあったと思われる場合は，結果のどのような点が意図に反しているのかを指定することができる．システムはその情報を用いて，ユーザが当初に記述した問合せ式に近く，かつ，結果に関するユーザが指定した問題点を解消するような問合せ式を全て見つけ，それらを，データの統計情報などを用いた確率論的な手法で，ユーザが本来書きたかった問合せである可能性が高い順にランキングして表示する．
キーワード　XPath，デバッグ，問合せ式，検索式

A XML Query Debugging System

Takahiko UCHIDA[†] and Keishi TAJIMA[†,††]

† Japan Advanced Institute of Science and Technology, Aasahidai 1-1 Nomi, Ishikawa 923-1292 Japan
†† Kyoto University, Yoshida-Honmachi, Sakyo, Kyoto 606-8501 Japan
E-mail: tk-uchi@jaist.ac.jp, tajima@i.kyoto-u.ac.jp

Abstract　When users issue XPath queries against XML data, it often happens that they cannot get what they wanted because of some errors in the queries. Such errors include typos and misunderstanding of the structure of the target XML data. In this paper, we propose a system to help users debugging their XPath queries. In our system, when a user gets a query answer that is not what they wanted, the user tells the system how that answer disagrees with the users intention. Then, the system finds queries that are similar to the original queries and that solves the disagreement with the users intention, and the system shows those queries in the order of the probability of being the query the user was intending, which is computed based on the statistics information.
Key words　XPath, debugging, query

1. まえがき

XML (eXtensible Markup Language) は，タグの入れ子構造を用いてデータを記述する汎用のデータ記述形式であり，XML で記述されたデータは，ノードにタグ名のラベルがついた木構造として表すことができる．そのような XML 形式のデータを処理する場合，木構造中の特定のノードを指定する方法が必要となることが多い．そこで，木構造の根ノードからどのような経路をたどって到達するノードであるかを指定することによって，木構造中の特定のノードを指定する言語である，XPath (XML Path Language) [1] が規格化されている．XPath は，他の規格の様々な言語の一部として用いられているが，XPath 単独で，XML データからその部分木を取り出すための簡単な検索言語としても広く用いられている．

XPath を用いて XML データからの検索を行う場合，ユーザが記述した検索式になんらかの誤りがあったために，意図したデータが取り出せないということが，しばしば起こりうる．誤りの原因としては，タイプミスや，対象としている XML データの構造に関するユーザの勘違い，また，ユーザの XPath に関する知識が不十分だったための構文の誤りなどが考えられる．このうち，最後の構文の誤りについては，XPath 処理系がユーザに，どこが誤っていたのかを知らせることがある程度可能と考えられるが，その他の場合については，問合せ式としては正しい式になっている限り，処理系は，問合せ式のどこがユーザ

の意図に反していたかは判断できない．よって，そのような場合は，ユーザ自身がデバッグ作業，すなわち，問合せ式を見直したり，データの一部を閲覧したりしながら，どこが間違っていたのかを見つける作業を行う必要がある．しかし，問合せ式が複雑であったり，データが巨大であるような場合，ユーザが直感を便りにそのような作業を行っていくのは，非常に労力の高い作業となる．

そこで，本研究では，そのようなユーザによる問合せ式のデバッグ作業を支援するシステムを開発する．本研究で開発するシステムでは，ユーザが実行した問合せの結果が当初の意図に反しており，問合せ式に何か誤りがあったと思われる場合は，結果のどのような点が意図に反しているのかを指定することができ，システムはその情報を使って，ユーザが最初に記述した問合せ式に近く，かつ，結果に関するユーザが指定した問題点を解消するような問合せ式を全て見つけ，それらを，ユーザが本来書きたかった問合せである可能性が高そうな順にランキングして表示する．

2. システムの概要

前章で述べたようなシステムを開発する場合，以下の三つの点について考慮する必要がある．

（1） 問合せ結果のどのような点が意図に反しているのかをユーザにどのように指定させるか．

（2） 当初の問合せ式に近くて，かつ，ユーザの指定した問合わせ結果中の問題点を解消するような問合せ式群をどのように見つけるか．

（3） それらの問合せ式をランキングするために必要な，「各問合せ式が，ユーザが本来書きたかった問合せ式である可能性」をどうやって求めるか．

以下，これら三点について，順に詳しく説明する．

2.1 意図に反する点の指定方法

ここでは，問合せ結果のどこが意図に反しているのかを，ユーザは以下のいずれかの形式で指定することにする．

- このデータは問合せ結果に含まれるべきなのに含まれていないというデータを指定する．
- このデータは問合せ結果に含まれないべきなのに含まれているというデータを指定する．
- 問合せ結果が空集合であったが空集合となるはずがないと指定する．

2.2 候補となる問合せ式の発見

ここでは，ユーザが犯す可能性のある間違いとして，以下の物を考え，以下の間違い (または，それら複数の組み合わせ) によって，ユーザが書いた問合せ式へとなりうる物をまず考え，それらの中で，ユーザが指定した条件を満たす物を候補解とする．

（1） **タグ名の誤り**

(例) 正: /book/name
誤: /book/names

（2） **パスの誤り**

(例) 正: /site/regions/africa/name/location
誤: /site/regions/africa/location

（3） **述語中の定数値の誤り**

(例) 正: /book/chapter[title="introduction"]
誤: /book/chapter[title="imtroduction"]

ただし，タグ名の誤りとしては，ここでは二文字以内の誤りまでのみを考える．理由としては，三文字以上の誤りを考えた場合，

- 実装において，三文字以上の誤りまで考えると，ランキングを計算すべき問合せ式の候補が多くなり過ぎて，システムの反応時間が低下し，使い勝手が悪くなることが予想される．
- 三文字以上の誤りを含む候補問合せ式まで考えても，結局は，後述のランキングでランキング低位になる場合が多いと予想される．
- 問合せ式が三文字以上の誤りを含む場合には，システムの力を借りなくても，ユーザが自分の書いた問合せ式を目で確認して間違いを発見できる場合が多いと思われる．このシステムでサポートしたいのは，ユーザが目で確認して発見しにくいような間違いの発見の支援である．

一方，定数値の誤りとしては，長い文字列定数が使用される場合などもあり，そのような場合まで含めて考えると，単純に何文字までと字数を制限するのは難しい．そこで，現時点では，定数値中の誤り文字数に関しては特に制限しないこととする．しかし，その場合，実行効率が悪くなることが予想され，定数値中の適切な誤り文字数の制限については，今後の課題である．

また，タグ名や定数値中で誤って用いる可能性がある文字としては，

- タグ名中では，XML でタグ名中に使うことが許されている任意の文字
- 定数値中では，元の定数値がアルファベットを含む場合は，XML で PC データ中に許される任意の文字，元の定数値が 0〜9 の数字のみからなる場合は，0〜9 の数字のみとする．

より実用的なシステムの実装の段階においては，上の点について，さらなる詳細な検討が必要と思われる．例えば，浮動少数点少数を表していると考えられる文字列等についても考慮する必要があるであろう．

また，パスの誤りについても，ここでは，二階層の追加，削除までを考える．ユーザが間違えて追加する階層のタグ名として許す範囲については，特に考慮する必要はない．なぜなら，ユーザが誤って余計な階層を追加している問合せ式を与えられて，ユーザが本来書きたかったのであろう問合せ式の候補を求める際には，単純に，どの階層とどの階層を取り除けばユーザが指摘した意図に合った問合わせになるかを調べればよいからである．

また，ユーザが誤って一部の階層を抜いてしまっている問合せ式を与えられて，ユーザが本来書きたかったであろう問合せ式の候補を求める場合は，追加する階層のタグ名としては，そもそも，データ中に実際に現れるタグ名しか考慮する必要がない．

2.3 候補となる問合せ式のランキングの方法

上述の方法で候補となる問合せ式を求めた結果，複数の問合せ式が候補となった場合，これらの候補を，ユーザが書きたかった問合せである可能性が高いものから順にランキングして表示することが望ましい．本研究では，そのような確率を求め，それに基づいて候補解をランキングする手法を開発する．

本研究の手法では，ユーザが書いた問合せ式が Q_0 であった時に，本当に書きたかった問合せ式は Q_1 であった確率を，ユーザが問合せ Q_1 を実行したい確率と，問合わせ式 Q_1 を書きたい時に誤って Q_0 と書いてしまう確率から，ベイズの定理 [2] を用いて求める．また，問合わせ式 Q_1 を書きたい時に誤って Q_0 と書いてしまう確率を求める際には，単純に Q_1 と Q_0 の近さだけでなく，Q_0 が単なる誤りによって書く確率が高そうな問合せ式であるか，それとも，単なる誤りで，うまくそのような問合せ式を書くような可能性は低いと思われるような特徴を持った問合せ式であるかという点についても考慮する．

例えば，Q_0 中の Q_1 とは異なる部分が，対象データには全く存在しないようなタグ名であったり定数値であったりする場合には，Q_0 は単なる Q_1 の書き間違いで書かれた問合せ式である可能性は高いであろう．しかし，Q_0 中の Q_1 とは異なる部分が，Q_1 には全く存在しないようなタグ名や定数値であるにも関わらず，それが対象データ中には実際に現れるようなものであった場合，Q_1 を書こうとして誤りで，ちょうどそのようなものを書く可能性は低く，むしろ，その部分は誤りで書かれたのではなくて，ユーザがデータに対する知識のもとに意図して書かれた可能性が高いと考えられる．よって，ユーザが当初に書いた Q_0 中においてユーザが間違えていた部分とはどこか他の場所であり，ユーザが本来書きたかったのは Q_1 ではなくて他の候補であるという可能性が高くなる．詳細については次章で例を用いて説明する．

3. ランキング計算

3.1 ランキング計算の基本的な考え方

本章では，本研究で提案するシステムの核である候補となる問合せのランキング計算の部分の基本的な考え方について，例を用いながら説明していく．まず最初の例として，ユーザが問合せ

Q_0: /bib/book[title='XSL']

を実行して結果が空集合となり，結果が空集合となるのはおかしいと指摘した場合を考える．今，前述の誤りのパターンのいずれかによって Q_0 となりうる問合せで，解が空集合でないものとしては以下の二つがあったとする．

Q_1: /bib/book[title='XML']
Q_2: /bib/book[title='XSL Manual']

今，以下の 4 つのことがわかっている．

（1） Q_0 は解がない

（2） ユーザが書きたかった検索には解があるはず (とユーザが言っている)

（3） Q_1 には解がある

（4） Q_2 には解がある

また，以下で用いるために，下のような記法を定義する．

$i(Q)$: ユーザが Q を実行したいと思う事象
$w(Q)$: ユーザが Q を記述するという事象

すると，ユーザが Q_1 を実行したいと思っていて，実際には Q_0 と書いてしまうという確率，すなわち $P(i(Q_1) \wedge w(Q_0))$ について，以下が成り立つ．

$$P(i(Q_1) \wedge w(Q_0)) = P(i(Q_1)) \times P(w(Q_0)|i(Q_1)) \qquad (1)$$

ただし，$P(w(Q')|i(Q))$ は，ユーザが Q を実行したいと思っている時に，Q' を記述する条件付き確率である．同様に，以下が成り立つ．

$$P(i(Q_2) \wedge w(Q_0)) = P(i(Q_2)) \times P(w(Q_0)|i(Q_2)) \qquad (2)$$

今，Q_1，Q_2 以外の検索をしたくて，間違って Q_0 と書く確率は 0 だとする．すると，ユーザが書いた問合せ式が Q_0 である時に，実際に実行したかった問合せ式が Q_1 である条件付き確率 $P(i(Q_1)|w(Q_0))$ は，ベイズの定理より

$$P(i(Q_1)|w(Q_0)) = \frac{(1)}{(1)+(2)}$$

となる．同様に，ユーザが書いたのが Q_0 である時に，ユーザが実際に実行したかった問合せ式が Q_2 である条件付き確率 $P(i(Q_2)|w(Q_0))$ は

$$P(i(Q_1)|w(Q_0)) = \frac{(2)}{(1)+(2)}$$

となる．やりたいのは，Q_1 と Q_2 をランキングすることなので，結局，(1) と (2) の大小関係，あるいは (1) と (2) の比である (1)/(2) が一より大きいかどうかさえわかればよい．

では，(1) と (2) の中に現れる各確率の値について具体的に考えてみる．$P(i(Q_1))$ や $P(i(Q_2))$ としては，例えば，これらの問合せが過去の一定期間中に実行された頻度を用いることが考えられる．今，過去の一定期間中に Q_1 が実行された頻度は Q_2 が実行された頻度の $1/200$ であったとする．すると，

$$\frac{P(i(Q_1))}{P(i(Q_2))} = \frac{1}{200} \qquad (3)$$

となる．ただし，過去の頻度のみで $P(i(Q))$ を決定すると，過去に実行されていない問合せについては $P(i(Q)) = 0$ となってしまい，ランキングが必ず最下位になってしまうので，過去の頻度が 0 である場合は，ある適当な値 ϵ を使う等の考慮が必要であろう．このあたりの，ファクターとなる各確率の求め方の詳細については，今後，さらにつめていく必要がある．

次に，$P(w(Q_0)|i(Q_1))$ と $P(w(Q_0)|i(Q_2))$ について考える．Q_1 を書こうと思って Q_0 と書いてしまったとすると，誤りの内容は，三文字中の真ん中の一文字の変更である．一方，Q_2 を書こうと思って Q_0 と書いてしまったとすると，誤りの内容は，十文字中の末尾七文字の削除である．これらの間違いをおかす確率とはどのような値になるかについては，関連する研究の調

査や実験等を用いて，今後，詳細を決定する必要があるが，ここでは仮に，例えば，一般的な統計に基づいて求めた結果，前者の方が後者の 50 倍の確率で起きやすかったとする．すると，

$$\frac{P(w(Q_0)|i(Q_1))}{P(w(Q_0)|i(Q_2))} = 50 \tag{4}$$

となる．(3) と (4) を総合すると，

$$\frac{(1)}{(2)} = \frac{1}{4}$$

となり，ユーザへの表示としては Q_2，Q_1 の順にランキングして表示することになる．これは，直感的に一言で言えば，Q_2 の方がユーザが書いた問合せ式 Q_0 からの距離は遠いが，このユーザは過去に頻繁に Q_2 を実行していることを考慮して，Q_2 の方が，ユーザが本来書きたかった問合せである可能性が高いと判定したということになる．

次に，以下のような例を考える．今，ユーザが

Q_0: /db/emp[id='298285'][name='Sato']

を実行して結果が空集合となり，結果が空集合となるのはおかしいと指摘した場合を考える．今，前述の誤りのパターンのいずれかによって Q_0 となりうる問合せで，解が空集合でないものとしては以下の二つがあったとする．

Q_1: /db/emp[id='298285'][name='Saito']
Q_2: /db/emp[id='298288'][name='Sato']

この場合も，上の例と同様，(1)/(2) の比を求めていけばよい．今，どちらの問合せも，過去の一定期間中には実行されたことがなく，よって，

$$\frac{P(i(Q_1))}{P(i(Q_2))} = 1$$

だったとする．次に，$P(w(Q_0)|i(Q_1))/P(w(Q_0)|i(Q_2))$ を求める．今，Q_1 を書こうと思って Q_0 を書いてしまったとすると，誤りの内容は一文字の削除である．一方，Q_2 を書こうと思って Q_0 を書いてしまったとすると，誤りの内容は一文字の変更である．今，このような誤りを犯す確率を統計的に求めた結果，両者の比としては，

$$\frac{P(w(Q_0)|i(Q_1))}{P(w(Q_0)|i(Q_2))} = \frac{1}{2}$$

であったとする．すると，

$$\frac{(1)}{(2)} = \frac{1}{2}$$

となり，ランキングは Q_2，Q_1 の順になる．

しかし，ここで，$P(w(Q_0)|i(Q_1))$ を求める際に，Q_2 には解があるという情報も利用することにする．すなわち，ユーザが Q_1 を書こうと思って Q_0 を書いてしまうということが起きるには，ユーザが 'Saito' を記述する際に一文字削除の誤りを犯し，かつ一文字削除で生成された 'Sato' という文字列が，たまたま実際に /db/emp[id=x][name=y](ただし，x は '298285' の一字変更) という形を持つ問合わ式の y の部分に当てはめて，問合せ式全体が解を持つようなものになっていなければならないと考える．その場合，

$$\begin{aligned}
&P(w(Q_0)|i(Q_1)) \\
&= P(\text{'Saito' 一文字削除し } y \text{ と記述} \wedge \text{上式が } y \text{ で解有り}) \\
&= P(\text{'Saito' 一文字削除し } y \text{ と記述}) \times \\
&\quad P(\text{上式が } y \text{ で解有り} \mid \text{'Saito' 一文字削除し } y) \qquad (5)
\end{aligned}$$

となる．

同様に，$P(w(Q_0)|i(Q_2))$ を求める際にも，Q_1 には解があるという情報を利用する．すなわち，ユーザが Q_2 を書こうと思って Q_0 を書いてしまうということが起きるには，ユーザが '298288' を書く際に一文字変更の誤りを犯し，かつ一文字変更で生成された '298285' という文字列が，たまたま実際に /db/emp[id=x][name=y](ただし，y は 'Sato' への一字追加) の x の部分に当てはめて，問合せ式全体が解を持つようなものになっていなければならないと考える．その場合，

$$\begin{aligned}
&P(w(Q_0)|i(Q_2)) \\
&= P(\text{'298288' 一文字変更し } x \text{ と記述} \wedge \text{上式が } x \text{ で解有り}) \\
&= P(\text{'298288' 一文字変更し } x \text{ と記述}) \times \\
&\quad P(\text{上式が } x \text{ で解有り} \mid \text{'298288' 一文字変更し } x) \qquad (6)
\end{aligned}$$

となる．すると，両者の比は，

$$\begin{aligned}
&\frac{(1)}{(2)} = \frac{P(i(Q_1)) \times (5)}{P(i(Q_2)) \times (6)} = \frac{(5)}{(6)} \\
&= \frac{P(\text{一文字削除})}{P(\text{一文字変更})} \times \\
&\quad \frac{P(\text{上式が } y \text{ で解有り} \mid \text{'Saito' 一文字削除し } y)}{P(\text{上式が } x \text{ で解有り} \mid \text{'298288' 一文字変更し } x)} \\
&= \frac{1}{2} \times \frac{P(\text{上式が } y \text{ で解有り} \mid \text{'Saito' 一文字削除し } y)}{P(\text{上式が } x \text{ で解有り} \mid \text{'298288' 一文字変更し } x)}
\end{aligned}$$

となる．

'Saito' の一文字削除により生成される y には五通りの値が考えられるが，そのうち，上述の /db/emp[id=x][name=y](ただし，x は '298285' の一字変更) という問合せ式に解を与えるものは，'Sato' のみであったとする．その場合，

$$P(\text{上式が } y \text{ で解有り} \mid \text{'Saito' 一文字削除し } y) = \frac{1}{5}$$

となる．同様に，'298288' の一文字変更により生成される x には「6 個所」×「0〜9 のうち元の文字以外の 9 文字」で 54 通りの値が考えられるが，そのうち，上述の /db/emp[id=x][name=y](ただし，y は 'Sato' の一字変更) という問合せ式に解を与えるものは，'298288' を '298285' と誤った場合に y='Saito' として解がある場合のみであったとする．その場合，

$$P(\text{上式が } x \text{ で解有り} \mid \text{'298288' 一文字変更し } x) = \frac{1}{54}$$

となる．よって，

$$\frac{(1)}{(2)} = \frac{1}{2} \times \frac{1/5}{1/54} = 27$$

となり，ランキングは Q_1，Q_2 の順となる．

これは，より直感的に一言で言えば，'298288' を一文字間違

えること自体は，'Saito' を一文字削除してしまうよりも高い確率で起こるが，'298288' を '298285' という，ちょうど書こうとしていた 'Sato' と一字違いであるような 'Saito' という人物の ID に間違える確率はそれほど高くないので，'298288' の部分を間違えて '298285' と書いたのではなくて，'Sato' の方が 'Saito' の間違いであったのではないかということである．

より一般的に言えば，Q_0 中の 'A' について，もし，ユーザが 'B' を書こうと思って誤って書いた値なのに，偶然に，その 'A' を含む Q_2 が解を持っていて別の候補問合わせ式になるなどということが起こる可能性が低い場合には，この 'A' は誤って書かれたものではなくて，ユーザが 'A' がデータ中に存在することを知っていて正しく書いたものである可能性が高く，よって，Q_1 が当初書きたかった問合わせである確率を Q_2 に比べて相対的に低くするということを意味している．

上の例は，定数値に関する二つの条件節を持つ問合せの例であったが，似たような事例は，パスの間違いによる例でもありうる．例えば，ユーザが以下の Q_0 を実行して空集合が解となり，「解が空集合であるはずがない」と指摘した場合を考える．

Q_0 : /site/africa/location

そして，Q_0 に近くて，かつ解があるような候補問合わせ式としては，以下の Q_1 と Q_2 があったとする．

Q_1: /site/africa/item/location
Q_2: /site/location

最初に，$P(w(Q_0)|i(Q_1))$ を求めてみる．この場合，ユーザは Q_1 のパスを一階層抜いて Q_0 と書いてしまって解が空集合になってしまい，しかし，その際に，その誤った問合せ式からさらに一階層抜いた Q_2 も解のある問合せ式になっていたということになる．よって，Q_1 から一階層抜いた問合せのうち，さらに一階層抜いた問合せで解を持つという問合わせが存在するようなものの比率を考える．今，Q_1 から一階層抜いた問合わせは 4 通り考えられるが，そのうち，/site/africa/location がもう一階層抜いて /site/location とした時に解を持っているという場合と，/site/item/location がもう一階層抜いて /site/location とした時に解を持っているという場合の二通りのみであったとする．すると，$P(w(Q_0)|i(Q_1))$ は

$$P(\text{一階層抜いてしまう誤りを起す確率}) \times \frac{2}{5}$$

となる．

次に，$P(w(Q_0)|i(Q_2))$ を求めてみる．この場合，ユーザは Q_2 のパスに一階層余分に付け加えて Q_0 と書いてしまって解が空集合になってしまい，しかし，その際に，その誤った問合せ式にさらに一階層加えた Q_1 も解のある問合せ式になっていたということになる．よって，Q_1 に一階層加える間違え方のうち，さらに一階層加えた問合せで解を持つという問合わせが存在するようなものの比率を考える．前述のように，ここでは，パスの間違いにおいて加えてしまう可能性があるタグ名は実際にデータ中に現れるタグ名のみであると仮定している．よって，データ中に現れるタグが 30 通りあるとした場合，Q_1 に一階層追加する間違え方は，追加する個所が三通りあることを考えて，$30 * 3 = 90$ 通りである．(ここで，新しい階層 location を Q_2 の末尾に追加した場合と location の前に追加した場合のように，重複して数えられる場合を考えると，結果として生じる問合せ式は $90 - 2 = 88$ 通りであるが，そのような問合せは，二通りの間違え方で生成されうるということであり，「間違え方」は 90 通りである．) そのうち，さらに一階層追加して解のある問合わせを作れるようなものは，20 通りであったとする．すると，$P(w(Q_0)|i(Q_2))$ は

$$P(\text{一階層加えてしまう誤りを起す確率}) \times \frac{20}{90}$$

となる．

よって，仮に，一階層抜いてしまう誤りを起す確率と一階層加えてしまう誤りを起す確率が等しく，かつ，例えば $P(i(Q_1))/P(i(Q_2)) = 1$ であった場合，ランキングは，Q_1，Q_2 となる．すなわち，直感的には，本来書きたかった問合わせ/site/location に誤って一階層加えてしまう場合に，うまく偶然で，site と location の間に africa を加えるという，さらにもう一階層加えれば解がある問合わせになるような間違え方をする確率は低いので，ユーザは Q_2 を書こうと思って Q_0 を書いたのではなく，Q_1 を書こうと思って Q_0 と書いてしまったのであろうということである．

最後に，これまで出てこなかったタグ名の誤りの場合の例を一つだけ簡単に示す．例えば，ユーザが

Q_0 : /bib/book/chapters

を実行して結果が空集合となり，ユーザが「解が空集合なのは意図に反している」と指摘したとする．今，候補となるような，解を持つ問合せ式は以下の二つがあったとする．

Q_1: /bib/books/chapters
Q_2: /bib/book/chapter

このような場合も，books を誤って book と書いてしまう確率と chapter を誤って chapters と書いてしまう確率だけでなく，books を書こうとして誤って書いた文字列が，実際にデータ中に現れる文字列となる確率や，chapter を書こうとして誤って書いた文字列が，実際にデータ中に現れる文字列となる確率を考える必要がある．

3.2 一般形に対するランキングの定義

今，ユーザが誤って以下の Q_0 を記述し，本来書きたかったであろう問合わせの候補として，以下の $Q_1 \ldots Q_n$ が得られたとする．

$$\begin{aligned} Q_0 : \\ &/L_0^1[L_0^{1,1} = C_0^{1,1}][L_0^{1,2} = C_0^{1,2}] \ldots [L_0^{1,m_0^1} = C_0^{1,m_0^1}]/ \\ &L_0^2[L_0^{2,1} = C_0^{2,1}][L_0^{2,2} = C_0^{2,2}] \ldots [L_0^{2,m_0^2} = C_0^{2,m_0^2}]/ \\ &\qquad\qquad \vdots \\ &L_0^{l_0}[L_0^{l_0,1} = C_0^{l_0,1}][L_0^{l_0,2} = C_0^{l_0,2}] \ldots [L_0^{l_0,m_0^{l_0}} = C_0^{l_0,m_0^{l_0}}] \end{aligned}$$

Q_1：

$$/L_1^1[L_1^{1,1}=C_1^{1,1}][L_1^{1,2}=C_1^{1,2}]\ldots[L_1^{1,m_1^1}=C_1^{1,m_1^1}]/$$
$$L_1^2[L_1^{2,1}=C_1^{2,1}][L_1^{2,2}=C_1^{2,2}]\ldots[L_1^{2,m_1^2}=C_1^{2,m_1^2}]/$$
$$\vdots$$
$$L_1^{l_1}[L_1^{l_1,1}=C_1^{l_1,1}][L_1^{l_1,2}=C_1^{l_1,2}]\ldots[L_1^{l_1,m_1^{l_1}}=C_1^{l_1,m_1^{l_1}}]$$

$$\vdots$$

Q_n：

$$/L_n^1[L_n^{1,1}=C_n^{1,1}][L_n^{1,2}=C_n^{1,2}]\ldots[L_n^{1,m_n^1}=C_n^{1,m_n^1}]/$$
$$L_n^2[L_n^{2,1}=C_n^{2,1}][L_n^{2,2}=C_n^{2,2}]\ldots[L_n^{2,m_n^2}=C_n^{2,m_n^2}]/$$
$$\vdots$$
$$L_n^{l_n}[L_n^{l_n,1}=C_n^{l_n,1}][L_n^{l_n,2}=C_n^{l_n,2}]\ldots[L_n^{l_n,m_n^{l_n}}=C_n^{l_n,m_n^{l_n}}]$$

ただし，ここで

- L_i^j は Q_i の j 番目のステップのタグ名
- $L_i^{j,k}$ は Q_i の j 番目のステップの k 番目の述語中のタグ名
- $L_i^{j,k}$ は Q_i の j 番目のステップの k 番目の述語中の定数
- m_i^j は Q_i の j 番目のステップが持つ述語の数
- l_i は Q_i の長さ (ステップの数)

である．

ユーザが Q_i を実行したいと思っていて，実際には Q_0 と書いてしまうという確率を求めるには，

$$P(i(Q_i)\wedge w(Q_0)) = P(i(Q_i)) \times P(w(Q_0)|i(Q_i)) \qquad (7)$$

を求める．$P(i(Q_i))$ はこの問合せが過去の一定期間内に実行された頻度を調べることによって求める．過去に行われていない問合わせに対しては，そのシステムにおいて，過去に実行されたことのない新しい問合わせが実行される頻度を用いる．さらに，$P(w(Q_0)|i(Q_i))$ は，以下の式によって求める．

$$P(w(Q_0)|i(Q_i)) =$$
$$\prod_{t\neq f_0(t)\ \text{であるような}\ Q_i\text{中の語}\ t} P(t\ \text{を誤って}\ f_0(t)\ \text{と記述}) \times$$
$$\prod_{t\neq f_0(t)\ \text{かつ}\ f_0(t)=f_j(f_0(t))\ \text{となるような}\ Q_i\text{中の語}\ t\ \text{があるような}\ j} P(Q_j'\text{が解を持つ})$$

ただし，ここで Q_i 中の語とは L_i^j または $L_i^{j,k}$ または $C_i^{j,k}$ のいずれかであり，$f_j(t)$ は Q_i 中の語 t に対応する Q_j 中の語を表す．また，Q_j' とは，Q_i に以下の変更を加えて生成できる問合わせ式を表す．

- $t = f_j(f_0(t))$ となる t については，そのまま
- $t \neq f_0(t) = f_j(f_0(t))$ となる t については，t に $t \to f_0(t)$ という変更と同種の変更を加えたもの
- $t = f_0(t) \neq f_j(f_0(t))$ となる t については，t に $t \to f_j(f_0(t))$ という変更と同種の変更を加えたもの
- $t \neq f_0(t) \neq f_j(f_0(t))$ かつ $t \neq f_j(f_0(t))$ となる t については，t に $t \to f_j(f_0(t))$ という変更と同種の変更を加え，さらに $f_0(t) \to f_j(f_0(t))$ という変更を加えたもの

上の定義は,「同種の変更」という概念に基づいているが，ここでは，変更の種類として以下のものを考えるものとする．

- 語を n 個追加する $(0 \leqq n \leqq 1)$
- 語を n 個削除する $(0 \leqq n \leqq 1)$
- 語を u 文字変更する $(0 \leqq u \leqq 2)$

4. 実装手法の提案

XML データのパス式の取得についてここではすべてのパス式をあらかじめリストにしておくことにしている．

- ユーザの書いた問合せがタグ名のみで構成されている場合

タグ名だけのパス式は巨大な XML データであってもパス式自体が少ないので問題はなしとしている．

- ユーザが書いた問合せが述語を使用している場合

述語の数によりパス式が多くなるので対策の必要がある．

そこで，ユーザが書いた問合せが述語を用いている場合の解候補の絞り方だが，ユーザが書いた問合せと解候補との階層数の比較を行いパスの誤りが 2 階層以上の範囲となるパス式は除く，ユーザが書いた問合わ式の述語の値とパス式の述語の値の比較を行い定数値の誤りが一定数以上のパス式は除く，パス式の過去の頻度によりソートし下位のパス式は除くなど考えている．

5. まとめと今後の課題

本論文では，ユーザが XPath による XML 問合せを行う場合に，問合せ式のデバッグを支援するシステムの概要について提案し，特に，ユーザが誤った問合せ式を書いた時に，本来書きたかった問合わせ式であろうと思われる候補を，その確率が高そうな順にランキングする手法の基本的な考え方について示した．

しかし，現段階では，基本的なアイデアを示した段階であり，実際のシステムを実現するためには，今後，以下のような問題について検討する必要がある．

- 様々な種類の誤りについて，ユーザがそれらの誤りを犯す確率をどのようにして与えるか．
- 今回，挙げたような三種類の誤り以外のもの，たとえば,「/」と「//」の誤りや,「and」と「or」の誤り等への拡張．
- 提案システムの有効性の評価をどのように行えば良いかの検討．

文　献

[1] J. Clark and S. De Rose, eds. XML Path Language (XPath) Version1.0 – W3C Recommendation. http://www.w3.org/TR/xpath, Nov. 1999.

[2] 薩摩順吉著, 理工系の数学入門コース 7 確率・統計, pp. 27–32.