修 士 論 文

XML 検索におけるデバッグ支援システム

北陸先端科学技術大学院大学
情報科学研究科情報処理学専攻

内田 隆彦

2006 年 3 月

修 士 論 文

XML 検索におけるデバッグ支援システム

指導教官　田島敬史 助教授

審査委員主査　田島敬史 助教授
審査委員　日比野靖 教授
審査委員　落水浩一郎 教授

北陸先端科学技術大学院大学
情報科学研究科情報処理学専攻

410017 内田 隆彦

提出年月: 2006 年 2 月

概 要

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そうな順にランキングして表示する．

目 次

第1章　はじめに

1.1　研究の背景と目的

XML (eXtensible Markup Language) [1] 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) [2] 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や，取り出したいデータを絞り込むための条件式の論理に関する勘違い，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．

本論文は，以下の構成からなる．

- 第 2 章　本研究においての基礎的事項について述べる
- 第 3 章　システムの概要について述べる
- 第 4 章　ランキング計算の考え方について述べる
- 第 5 章　実装手法について述べる
- 第 6 章　まとめと今後の課題について述べる

第2章　基礎的な事項

2.1　XML

XML は，本文以外のデータ (ファイルそのものなど) の情報を付加することができる言語である，属性や要素の名前を自由に決めることができ，階層構造でデータを表現できる．例として，図 2.1 は XML データと，XML データを階層構造へ表現したものである．図 2.1 の矩形のノードは XML の要素ノードであり，楕円のノードはテキスト要素ノードである．

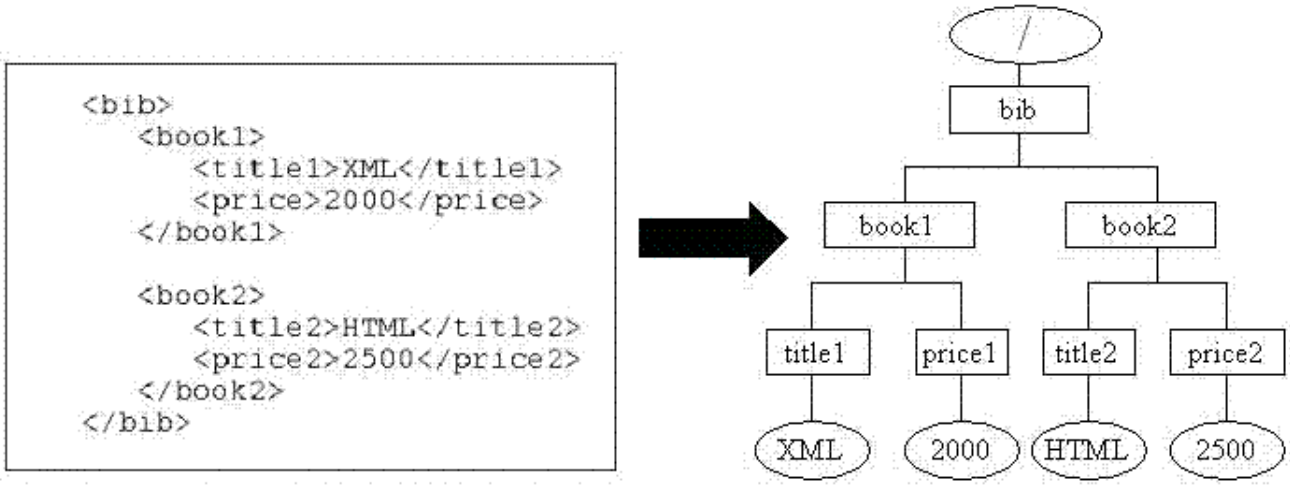

図 2.1: XML のデータと階層構造

2.2　XPath

XPath とは，XML 文書の特定の部分を指し示すために使用される言語の規格のことである．XPath では，XML 文書の要素，属性を含む文書の一部のことをノードと呼ぶ．ノードは階層構造になっており，必ず単一のルートノードから始まる．ルートノードとは XML 文書のルートである文書要素の仮想上のルートである．ルートノードは/(スラッシュ) で指定する．

2.2.1 ロケーションパス

ロケーションパスは，XML 文書の階層構造の中から特定のノードを指定するための記述方法である．ロケーションパスは 1 つ以上のロケーションステップから構成され，左から右へステップを順に解釈していく．ここで例として，図 2.2 のような階層構造を持つ XML データに対して book1 要素のロケーションパスは/bib/book1 となり，title1 要素のロケーションパスは/bib/book1/title1 となる．述語を使うことで book2 要素の子ノードである title2 要素の値が HTML である book2 のロケーションパスは/bib/book2[title2=”HTML”] と指定できる．

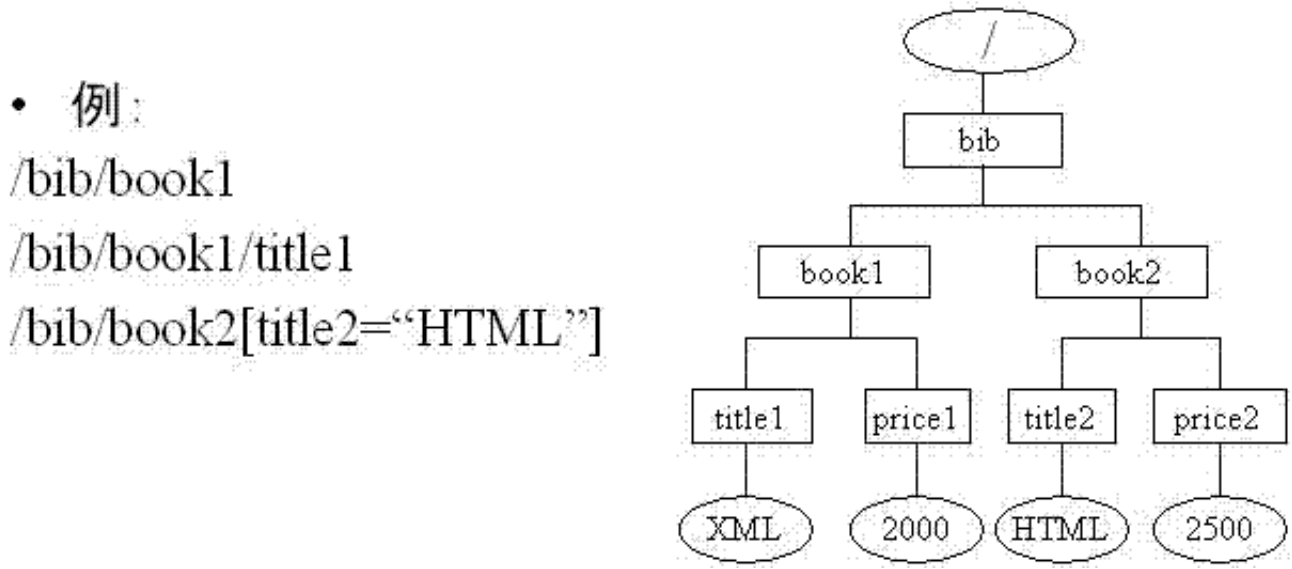

図 2.2: XPath 式の表記

2.3 XML パーサ

この節では XML 文書を処理するためのモジュールとして代表的なものである DOM(Document Object Model) と SAX(Simple API for XML) について説明する．

2.3.1 DOM

DOM [5] は XML 文書をメモリー上に階層構造で管理し，プログラムからアクセスできるようにした API である．DOM での XML 文書への操作は階層構造をたどる操作，データ (要素，テキスト，属性) の追加，削除，取り出し操作が基本となる．以下に DOM を用

いて図 2.2 の階層構造の要素名を全て取り出す例を示す．また XML 文書の操作で使うメソッドの説明を表 2.1 に示す．

1. Document doc = db.parse(new FileInputStream(“Sample.xml”));
   既存ファイル (ここでは Sample.xml とする) を読み込む
2. Element root = doc.getDocmentElement();
   root ノードを取得
3. for(Node ch = root.getFirstChild();ch!=null;ch = ch.getNextSibling())
   ノードの子を 1 階層たどる，多層をたどるには 1 階層分の処理を再帰で呼ぶ必要がある
4. if(ch.getNodeType() = Node.ELEMENT_NODE)
   ノード種類が要素である場合にだけ要素名を取得

表 2.1: XML 文書の操作

メソッド	機能
Element getDocumentElement()	文書のルート要素を取得
getFirstChild()	ノードの最初の子を取得
getNextSibling()	ノードの次の兄弟を取得
getNodeType()	ノードの種類を取得
getNodeName()	ノードのノード名を取得
getNodeValue()	ノードのノード値を取得

2.3.2 SAX

SAX は W3C で開発されたものではなく，XML-DEV メーリングリストで議論を重ねて考案した API である．SAX は，XML 文書を先頭から読み込み，そこに登場してくる要素やテキストに応じて処理をするしくみを持っている，XML 文書中の要素の開始や終了を見つけたり，テキストが出現した時に，ハンドラと呼ばれるしくみにその旨を通知する．

2.4 Xalan

Xalan [3] は，Apache XML Project の一部として開発されている XSLT プロセッサである．W3C の XSLT と XPath を実装している．XPath プロセッサは単体でも使用できる．以下に Xalan を用いての XPath 処理の例を試す．図 2.3 は対象となる XML データ (Sample.xml) であり，図 2.4 は，Xalan を用いて XML データに XPath 式 (/cars/car/name)

を用いて結果を表示したものである．

```
<?xml version="1.0" encoding="Shift_JIS" ?>
- <cars>
  - <car>
      <name>ABC</name>
      <price>150</price>
    </car>
  </cars>
```

図 2.3: XML データ (Sample.xml)

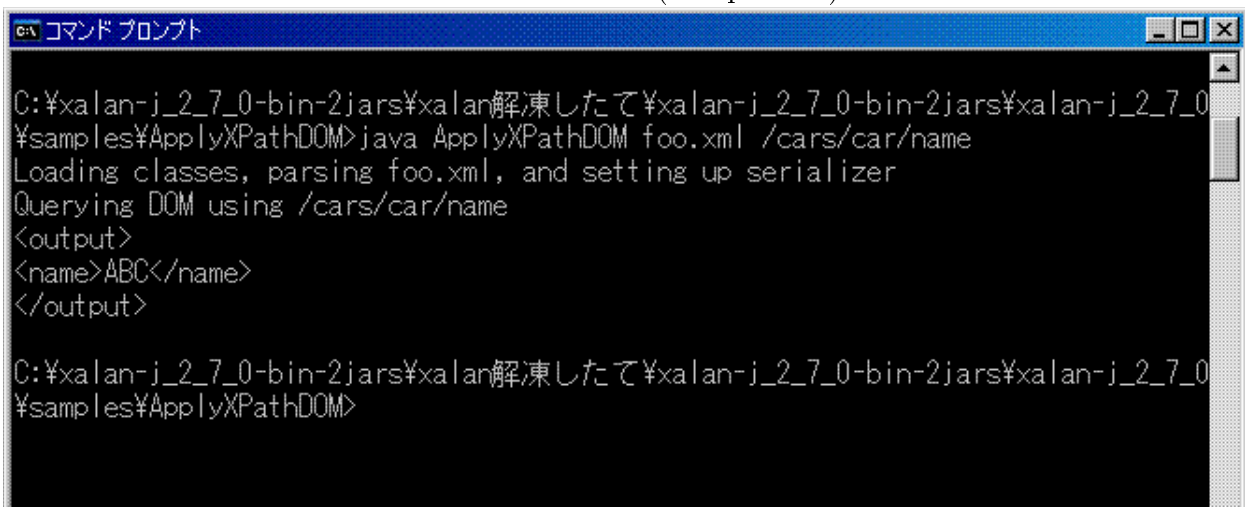

図 2.4: Xalan を用いての XPath 処理結果

2.5　条件付き確率 (ベイズの定理)

確率とは，ある現象が起こる度合い，ある試行が行われたあとある事象が現れる割合のことをいう．偶然性を含まないひとつに定まった数値であり，発生の度合いを示す指標として使われる．

$P(A)$ = 事象 A が発生する確率
$P(A|B)$ = 事象 B が既に発生している場合に，事象 A が発生する確率

事象 B_i が発生した時，事象 A が発生する確率は $P(A|B_i)$ であるが，これを逆に見て，事象 A が発生したときに，B_i が発生していた確率 $P(B_i|A)$ を知りたい場合，ベイズの定理が使われる．つまりベイズの定理とはある結果が得られた時，その結果に対する原因の確率を求める定理である．

$B_1, B_2, \cdots, B_n$ が互いに背反事象であり,
$B_1 \cup B_2 \cup \cdots B_n = \Omega$

ここで Ω は全事象を表すとする．
であるならば,

$$P(A) = \sum_{i=1}^{n} P(A \cap B_i) = \sum_{i=1}^{n} P(A \mid B_i)P(B_i) \cdots \text{(全確率の定理)}$$

から

$$P(B_i \cap A) = \frac{P(A \cap B_i)}{P(A)} = \frac{P(A|B_i)P(B_i)}{P(A)} = \frac{P(A|B_i)P(B_i)}{\sum P(A|B_i)P(B_i)}$$

が得られる．これをベイズの定理と呼ぶ．

例えば，赤い袋が 2 個，青い袋が 1 個ある．赤い袋の中には白球 1 個と黒球 4 個，青の袋の中には白球 4 個と黒球 1 個が入っている．いま無作為に選ばれた袋から，無作為に球 1 個を取り出したところ黒球であったという．この黒球が青の袋から取り出されたものである確率を求める．

赤の袋が選ばれる事象を A_1，青の袋が選ばれる事象を A_2，黒球が選ばれる事象を B とする．事前確率は $P(A_1) = \frac{2}{3}, P(A_2) = \frac{1}{3}$, 黒球を選ぶ条件つき確率は $P(B \mid A_1) = \frac{4}{5}, P(B \mid A_2) = \frac{1}{5}$ である．取り出された黒球が，青の袋から選ばれる事後確率 $P(A_2 \mid B)$ はベイズの定理により

$$P(A_2 \mid B) = \frac{P(A_2)P(B|A_2)}{P(A_1)P(B|A_1)+P(A_2)P(B|A_2)} = \frac{1}{9}$$

ここで用いた $P(A_i)$ を事前確率, $P(A_i \mid B)$ を事後確率という.

第3章　システムの概要

1 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．

3.1　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．

3.2　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．

3.3 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 [6] 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．

第4章　ランキング計算

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考え方について説明する．まず，例を用いて，ランキング計算の基本的な考え方を示し，その後で，一般的な場合に対するランキングの定義を示す．

4.1　ランキング計算の基本的な考え方

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．

$$P(i(Q_1) \wedge w(Q_0)) = P(i(Q_1)) \times P(w(Q_0)|i(Q_1)) \tag{4.1}$$

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．

$$P(i(Q_2) \wedge w(Q_0)) = P(i(Q_2)) \times P(w(Q_0)|i(Q_2)) \tag{4.2}$$

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

$$P(i(Q_1)|w(Q_0)) = \frac{(4.1)}{(4.1)+(4.2)}$$

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

$$P(i(Q_1)|w(Q_0)) = \frac{(4.2)}{(4.1)+(4.2)}$$

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，(4.1) と (4.2) の大小関係，あるいは (4.1) と (4.2) の比である (4.1)/(4.2) が一より大きいかどうかさえわかればよい．

では，(4.1) と (4.2) 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，

$$\frac{P(i(Q_1))}{P(i(Q_2))} = \frac{1}{200} \tag{4.3}$$

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，

$$\frac{P(w(Q_0)|i(Q_1))}{P(w(Q_0)|i(Q_2))} = 50 \tag{4.4}$$

となる．(4.3) と (4.4) を総合すると，

$$\frac{(4.1)}{(4.2)} = \frac{1}{4}$$

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，$(4.1)/(4.2)$ 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，

$$\frac{(4.1)}{(4.2)} = \frac{1}{2}$$

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，

$$\begin{aligned} & P(w(Q_0)|i(Q_1)) \\ &= P(\text{'Saito' 一文字削除し } y \text{ と記述} \wedge \text{上式が } y \text{ で解有り}) \\ &= P(\text{'Saito' 一文字削除し } y \text{ と記述}) \times \\ &\quad P(\text{上式が } y \text{ で解有り} \,|\, \text{'Saito' 一文字削除し } y) \end{aligned} \tag{4.5}$$

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，

$$\begin{aligned} & P(w(Q_0)|i(Q_2)) \\ = \;& P(\text{'298288' 一文字変更し } x \text{ と記述} \wedge \text{上式が } x \text{ で解有り}) \\ = \;& P(\text{'298288' 一文字変更し } x \text{ と記述}) \times \\ & P(\text{上式が } x \text{ で解有り} \,|\, \text{'298288' 一文字変更し } x) \end{aligned} \tag{4.6}$$

となる．すると，両者の比は，

$$\begin{aligned} \frac{(4.1)}{(4.2)} &= \frac{P(i(Q_1)) \times (4.5)}{P(i(Q_2)) \times (4.6)} = \frac{(4.5)}{(4.6)} \\ &= \frac{P(\text{一文字削除})}{P(\text{一文字変更})} \times \\ & \quad \frac{P(\text{上式が } y \text{ で解有り} \,|\, \text{'Saito' 一文字削除し } y)}{P(\text{上式が } x \text{ で解有り} \,|\, \text{'298288' 一文字変更し } x)} \\ &= \frac{1}{2} \times \frac{P(\text{上式が } y \text{ で解有り} \,|\, \text{'Saito' 一文字削除し } y)}{P(\text{上式が } x \text{ で解有り} \,|\, \text{'298288' 一文字変更し } x)} \end{aligned}$$

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，

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

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，

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

となる．よって，

$$\frac{(4.1)}{(4.2)} = \frac{1}{2} \times \frac{1/5}{1/54} = 27$$

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．

4.2 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．

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

$$
\begin{aligned}
Q_1: \quad & /L_1^1[L_1^{1,1} = C_1^{1,1}][L_1^{1,2} = C_1^{1,2}] \ldots [L_1^{1,m_1^1} = C_1^{1,m_1^1}]/ \\
& L_1^2[L_1^{2,1} = C_1^{2,1}][L_1^{2,2} = C_1^{2,2}] \ldots [L_1^{2,m_1^2} = C_1^{2,m_1^2}]/ \\
& \vdots \\
& L_1^{l_1}[L_1^{l_1,1} = C_1^{l_1,1}][L_1^{l_1,2} = C_1^{l_1,2}] \ldots [L_1^{l_1,m_1^{l_1}} = C_1^{l_1,m_1^{l_1}}]
\end{aligned}
$$

$$\vdots$$

$$
\begin{aligned}
Q_n: \quad & /L_n^1[L_n^{1,1} = C_n^{1,1}][L_n^{1,2} = C_n^{1,2}] \ldots [L_n^{1,m_n^1} = C_n^{1,m_n^1}]/ \\
& L_n^2[L_n^{2,1} = C_n^{2,1}][L_n^{2,2} = C_n^{2,2}] \ldots [L_n^{2,m_n^2} = C_n^{2,m_n^2}]/ \\
& \vdots \\
& L_n^{l_n}[L_n^{l_n,1} = C_n^{l_n,1}][L_n^{l_n,2} = C_n^{l_n,2}] \ldots [L_n^{l_n,m_n^{l_n}} = C_n^{l_n,m_n^{l_n}}]
\end{aligned}
$$

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，

$$P(i(Q_i) \wedge w(Q_0)) = P(i(Q_i)) \times P(w(Q_0)|i(Q_i)) \tag{4.7}$$

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 $(0 \leq n \leq 1)$
- 語を n 個削除する $(0 \leq n \leq 1)$
- 語を u 文字変更する $(0 \leq u \leq 2)$

第5章　実装手法

この章では，システムを実装するにあたって使っている手法について説明する．

5.1　エディットグラフ

ユーザがタイプした文字列になんらかの誤りがあり，意図したデータが取りだせない場合，ユーザが欲しいであろう解の候補を求めるために，ユーザがタイプした文字列と対象となる XML データのパス式との文書比較を行う．

文書比較を行う問題は，2 つの文書 A，B の最長共通部分 (LCS : Longuest Common Subsequence)，または最小エディット距離 (SED : Shotest Edit Distance) を求める問題と等価である．本システムでのアルゴリズムでは，これらの問題をエディットグラフ上の最短距離取得問題に還元し，解を求める．
エディットグラフとは文書 A，B の各要素 (A1 A2 ... AM， B1 B2 ... BN) を X 軸 Y 軸上に並べ，それらの交点を縦横の辺で結合し，A の x 番目の要素と B の y 番目の要素が等しい場合のみ (x-1, y-1) から (x, y) を結合したものである．ここで例を一つあげる．A が”chart” で，B が”chapter” の場合エディットグラフは図 5.1 の様になる．

図 5.1 のようなエディットグラフを考える時，LCS ／ SED を求める問題は，エディットグラフ上で，(x, y) から (x + 1, y) または (x, y + 1) への移動 (縦，または横方向への移動) が距離 1，(x, y) から (x + 1, y + 1) への移動 (対角線方向への移動) が距離 0 とする時に，(0, 0) から (M, N) までの経路で距離が最小のものを求める問題に還元される．
上記の例では，図 5.2 の経路が最小コストを与えるものである．

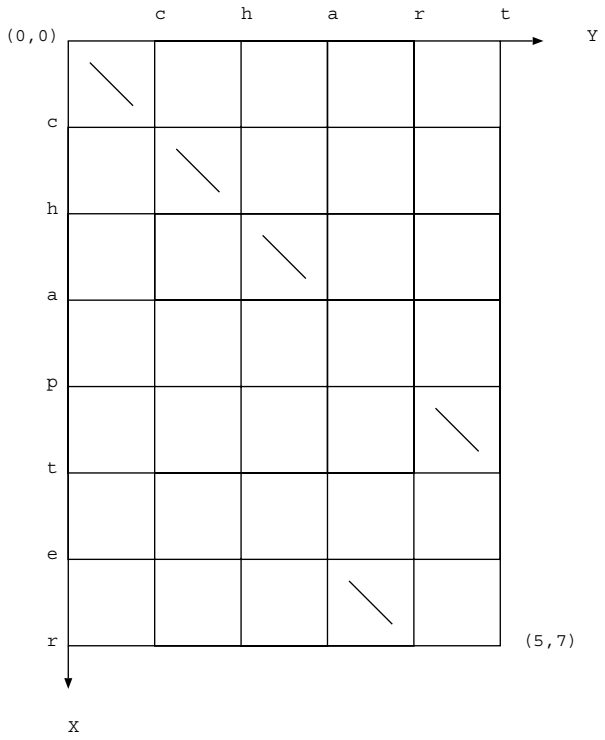

図 5.1: エディットグラフ 1

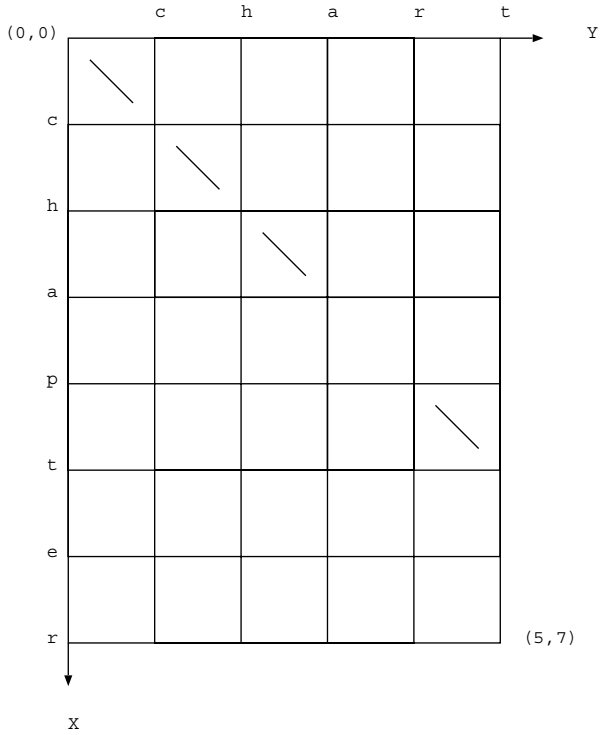

図 5.2: エディットグラフ 2

5.1.1 アルゴリズムの概要

表 4.1 のエディットグラフのアルゴリズムを用いて，文字列 1=chapter，文字列 2=chart の最小コストを求める例を以下に挙げる．

1. 文字列 1="chapter" と文字列 2="chart" をそれぞれ配列に格納

2. 配列での同じ添字数で同じ文字が入ってる場所を調べ，その場所を別の 2 次元配列 box に記憶しておく
   この場合は，文字列 1 の 1 文字目と文字列 2 の 1 文字目，文字列 1 の 2 文字目と文字列 2 の 2 文字目，文字列 1 の 3 文字目と文字列 2 の 3 文字目，文字列 1 の 5 文字目と文字列 2 の 5 文字目，文字列 1 の 4 文字目と文字列 2 の 7 文字目が該当する場所である

3. コスト数を計算するための 2 次元配列 box2 を作り，[i][0]，[0][j] の場所に該当する場所に初期値を入れておく，i=0 . . . 文字列 2 の長さ，j=0 . . . 文字列 1 の長さ
   配列 [0][0] に 0 の値を入れ，x 軸側に移動した数が，その場所に入る，[5][0] なら 5 の値が入る，同じように，y 軸側に移動した数が，その場所に入る，[0][3] なら 3 の値が入る

4. 最小コストを調べていく
   配列 box の値を 1 つずつ調べていきその場所が，同じ添字数で同じ文字が入ってる場所なら，配列 box2 の同じ添字の場所の値と x 軸側に 1 個移動した値と y 軸側に 1 個移動した値と比較，その中で一番小さい値を現在いる場所の x 軸側に 1 個，y 軸側に 1 個移動した場所に格納，同じ添字数で同じ文字が入っていない場所なら配列 box2 の同じ添字の場所から x 軸側に 1 個移動した値と y 軸側に 1 個移動した値と比較その中で一番小さい値を現在いる場所の x 軸側に 1 個，y 軸側に 1 個移動した場所に 1 足して格納

5. box2 配列の一番最後に入っている値が文字列 1 と文字列 2 の最小コスト値である 4 が入っている

```
ユーザがタイプした文字列と対象となる XML データのパス式をそれぞれ配列に格納;
エディットグラフの元となる配列 box[ユーザがタイプした文字列の
長さ][XML データのパス式の長さ] の宣言;
for(i=0; ユーザがタイプした文字列の長さ分繰り返し;i++){
  for(j=0;XML データのパス式の長さ分繰り返し;j++){
    if(ユーザがタイプした文字列 [i]==XML データのパス式の長さ [j])
    box[i][j]=1；
    else
    box[i][j]=0；
  }
} 文字列同士の最小コストを調べるための配列を宣言 box2[x+1][y+1];
cost_x=0,cost_y=0;
for(i=0;i<y+1;i++)
box2[0][i]=cost_y++;
for(i=0;i<x+1;i++)
box2[i][0]=cost_x++;
cost_x=0,cost_y=0;
for(t=0;t<x;t++){
  for(u=0;u<y;u++){
    if(box[t][u]==各文字列の要素が等しい場合){
    box2 配列の現在いる値と x 軸側に 1 個移動した値と y 軸側に 1 個移動
    した値と比較，その中で一番小さい値を現在いる場所の x 軸側に 1 個,
    y 軸側に 1 個移動した場所に格納
    }
    else if(box1[t][u]==各文字列の要素が等しくない場合){
    box2 配列の現在いる値に x 軸側に 1 個移動した値と y 軸側に 1 個移
    動した値と比較その中で一番小さい値を現在いる場所の x 軸側
    に 1 個，y 軸側に 1 個移動した場所に 1 足して格納
    }
  }
}
box2 配列の一番最後に入っている値を表示
```

表 4.1:エディットグラフのアルゴリズム

5.2 XMLデータのパス式の取得

本節では，エディットグラフでの文書比較行うために必要な，XML データのパス式を取得する方法について述べる．アルゴリズムを表 4.2，表 4.3 に示す．

- アルゴリズムの概要 (ユーザがタイプした式がタグしか用いていない場合)
  XML 文書を SAX を使い読み込んでいき，タグの開始部分，終了部分，テキストの開始部分で root からの現在いる場所のパス式を取得する処理を行い，全てのパス式の出力を行う．図 5.3 の XML データでのパス式の処理を行った場合，

  パス式の一覧として
  /cars
  /cars/car
  /cars/car/name
  /cars/car/price
  /cars/car/color
  が出力される

- アルゴリズムの概要 (ユーザがタイプした式が述語を用いている場合)
  最初に，ユーザがタイプした式がタグしか用いていない場合の方法で全てのパス式の出力を行う．次に，ユーザがタイプした式が，どの階層で述語を使っているかを調べる，そしてユーザが述語を書いた階層の上下 2 階層の場所を覚えておき，出力されたパス式に，対応する階層に値があるか Xalan を使い調べる処理を行う．値があれば，パス式を述語を用いた XPath 式に書き換える．
  図 5.3 の XML データでのパス式の処理を行った場合，

  パス式の一覧として
  /cars/car[name="AAA"]
  /cars/car[name="BBB"]
  /cars/car[name="CCC"]
  /cars/car[price="150"]
  /cars/car[price="500"]
  /cars/car[price="200"]
  /cars/car[color="white"]
  /cars/car[color="blue"]
  /cars/car[color="yellow"]
  /cars/car[name="AAA"]/name
  /cars/car[name="BBB"]/name

```
/cars/car[name="CCC"]/name
/cars/car[price="150"]/name
/cars/car[price="500"]/name
/cars/car[price="200"]/name
/cars/car[color="white"]/name
/cars/car[color="blue"]/name
/cars/car[color="yellow"]/name
/cars/car[name="AAA"]/price
/cars/car[name="BBB"]/price
/cars/car[name="CCC"]/price
/cars/car[price="150"]/price
/cars/car[price="500"]/price
/cars/car[price="200"]/price
/cars/car[color="white"]/price
/cars/car[color="blue"]/price
/cars/car[color="yellow"]/price
/cars/car[name="AAA"]/color
/cars/car[name="BBB"]/color
/cars/car[name="CCC"]/color
/cars/car[price="150"]/color
/cars/car[price="500"]/color
/cars/car[price="200"]/color
/cars/car[color="white"]/color
/cars/car[color="blue"]/color
/cars/car[color="yellow"]/color
```
が出力される

```
<?xml version="1.0" encoding="Shift_JIS" ?>
- <cars>
  - <car>
      <name>AAA</name>
      <price>150</price>
      <color>white</color>
    </car>
  - <car>
      <name>BBB</name>
      <price>500</price>
      <color>blue</color>
    </car>
  - <car>
      <name>CCC</name>
      <price>200</price>
      <color>yellow</color>
    </car>
  </cars>
```

図 5.3: パス式取得

```
sedai;//階層の深さを表す
Path[100];//パス式を格納する配列を宣言
SAX パーサーファクトリを生成;
SAX パーサーを生成;
XML ファイルを指定されたデフォルトハンドラーで処理;
public void startDocument(){
  sedai=0;
  Path[sedai]="";
}

public void startElement(String uri,String localName,String
qName,Attributes atts){

  int i=0;
  Path[sedai]=qName;
  sedai++;
  while (Path[i]!=null){
  root から現在値までのパスを出力;
  i++;
  }
}

public void characters(char[] ch,int offset,int length){
  新しいパス式の出力に変える;
}

public void endElement(String uri,String localName,String qName){
  新しいパス式の出力に変える;
  Path[sedai]=null;
  sedai−;
}
```

表 4.2:ユーザがタイプした式がタグしか用いていない場合のパス式取得のアルゴリズム

ユーザが書いた式のどの階層が述語使われてるかチェック;
処理する階層部分を調べる;
while(処理する階層がある間繰り返し){
現在いる階層の子供のタグを全部調べる;
現在いる階層のパスに子供のタグを追加し，Xalan を用いて値を調べる;

if(Xalan を使った結果，値がなし)
現在の階層での処理は終了;
else {
　パス式を述語を扱った XPath 式対応の式に変換する;
　// 例 /doc/name → /doc[name="testdata"] に変換
　}
}

表 4.3:ユーザがタイプした式が述語を用いている場合のパス式取得のアルゴリズム

5.3　ランキング計算のアルゴリズムの概要

表4.4のランキング計算のアルゴリズムを用いて，ユーザが書いた問い合わせ式Q_0:/bib/book/chapters，本当に書きたかった問い合わせ式Q_1:/bib/books/chapters，別の解候補式Q_2:/bib/book/chapter
のランキング計算を求める例を以下に挙げる．

1. Q_0，Q_1，Q_2 の式を単語ごとに区切る

2. Q_1 と Q_0 で対応する単語同士で比較を行う
   Q_1 で区切られた単語 books と，その単語に対応する Q_0 の単語 book と比較の結果，差が 1 なので，Q_1 の単語にその変更を行いできる文字列パターンを記憶しておく

3. Q_1 と Q_2 で 2 番目の処理が行われてない単語の中で，Q_0 と Q_2 で対応する単語同士で比較を行う
   Q_1 と Q_2 で 2 番目の処理が行われてない単語の中で，Q_1 の単語 chapters とその単語に対応する Q_2 の単語 chapter が差が 1 なのでその変更を行いできる文字列パターンを記憶しておく

4. Q_1 の式に今までの処理を行った場所は変更を加えその式全体で解を持つ数を調べる

5. ユーザが間違って書いてしまったような誤りを犯す確率を求める

6. Q_1 の一定期間中で実行された頻度*ユーザが間違って書いてしまったような誤りを犯す確率*(式全体が解を持つものの数/全パターン数) により値が求まる

Q_0,Q_1,Q_2 を単語ごとに区切る;

for(区切られた単語の数の回数分繰り返し)
{
  Q_1 で区切られた単語とその単語に対応する Q_0 の単語の長さの差を
  調べる;

  if(Q_1 で区切られた単語とその単語に対応する Q_0 の単語の比較を行
  い, その結果によりできるパターンを求める){
  今までで求められたパターン数*ミスした間違いでできるパターン数;
  }

  上記のミスした間違いでできるパターンで作られる文字列を区切られた
  各単語ごとに対応した配列に記憶しておく;

  if(Q_1 で区切られた単語とその単語に対応する Q_0 の単語の比較を行
  い，その結果が 0 以外の時だけ通過){
  Q_2 で区切られた単語とその単語に対応する Q_0 の単語の長さ
  の差を調べる;

    if(Q_2 で区切られた単語とその単語に対応する Q_0 の単語の
    比較を行い，その結果によりできるパターンを求める){
    今までで求められたパターン数*ミスした間違いでできるパターン数;
    }

  今までできた文字列のパターンに上記で書かれている間違えてで
  きるパターンを行い，作られる文字列を区切られた各単語ごとに対
  応した配列に格納;
  }
}

if(Q_1 で区切られた単語とその単語に対応する Q_0 の単語の差=0 && Q_2 で
区切られた単語と Q_1 で区切られた単語に差!=0){
対応する場所に Q_1 の対応する単語にその変化を加える;
}

if(Q_1で区切られた単語とその単語に対応するQ_0の単語の差!=0){
対応する場所に上記で求めた文字列をQ_1の対応する場所に入れる;
}

式全体が解を持つものの数を求める;

ユーザが間違って書いてしまったような誤りを犯す確率を求める;

ランキングの確率の値 $= Q_1$の一定期間中で実行された頻度*ユーザが間違って書いてしまったような誤りを犯す確率*(式全体が解を持つものの数/全パターン数);

表 4.4:ランキング計算のアルゴリズム

第6章　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．

- ランキング計算の詳細，かつ，より形式的な定義
- 本文中でも述べたが，ユーザがある問合わせを実行したいと思う確率 $i(Q)$ をどのようにして与えるか．
- 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．
- 実装手法．
- 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．

特に，実装手法については，非常に単純な実装方法として，ユーザが書いた問合せ式から与えられた誤りの種類によって到達できて，かつ，ユーザが指定した意図に合致する問合せ式を全て求め，それらについて確率を計算してランキングをするのでは，候補問合わせ式の数が多くなった場合，ランキングをユーザに提示できるまでの反応時間が非常に長くなり，使い勝手の悪いシステムになってしまうと思われる．よって，なんらかの手法を使って，最初から最終的なランキングで上位になる可能性が高いような解候補のみに絞り込んで計算を行うような，top-k 問合せの技術が必要になると思われる．よって，今後は，特に実装手法について検討を行う予定である．

謝辞

本研究に熱心に指導をして頂いた田島敬史助教授に深く感謝致します．また数々の助言を頂いた同研究室のみなさま，大堀研究室のみなさまに深く感謝いたします.

参考文献

[1] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, and Eve Maler, editors. Extenssible Markup Language(XML)1.0(Second Edition) - W3C Recommendation. http://www.w3.org/TR/2000/REC-xml-20001006,Oct.2000.

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

[3] Xalan. http://xml.apache.org/xalan-j/index.html.

[4] Simple API for XML. http:www.saxproject.org/

[5] Document Object Model(DOM) http://www.w3.org/DOM/

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