1D1-2

制約論理型言語に基づくXML要素の検索について

Search of XML Elements Based on Constraint Logic Programming

駅 昌弥[*1] 浅見 昌平[*1] 大囿 忠親[*1] 新谷 虎松[*1]
Masaya Eki Shohei Asami Tadachika Ozono Toramatsu Shintani

名古屋工業大学大学院 工学研究科[*1]
Dept. of Intelligence and Computer Science, Nagoya Institute of Technology

This paper describes a restrictions dissolution system based on constraint logic programming. This system realized searching in XML and reasoning system. This system generates knowledge about XML tags automatically. Users are inquiring knowledge and the knowledge returns view which is a set of data needed. Constraint logic programming generated at this time is based on meaning and pattern in tag structures, and becomes an ontology in the database stored the tag structures. By using constraint logic programming, the numerical computation about tags etc. is made possible. We realized flexible searching XML system.

1. まえがき

XMLは，その柔軟性や豊かな表情力およびプラットフォーム中立という特徴から，データ交換の標準に役立つ．広範囲でXMLが利用されるようになり，コモンツールの開発や，アプリケーションを構築するインフラストラクチャが提供されてきた．これらにより，次世代のビジネスコンピューティングインフラストラクチャの重要なコンポーネントであるSOAPやXMLベースのwebサービスなどの普及に拍車がかかった．しかしながら，XMLが次世代のビジネスコンピューティングインフラストラクチャとして用いられるが，広く用いられているXMLの解析技術は，XMLに基づいたコンピューティング・インフラストラクチャの実行要求を満たすことができない．

XMLデータベースは，XML形式の情報を格納できるデータベースであり，例えばIBM社のDB2やOracle社のOracle XML DB，Microsoft社のSQL Server XMLなどがある．XMLのスキーマ言語の1つとしてDTDが使われているが，データ型が定義できない点や，文法がXMLと無関係である点など様々な欠点が指摘されている．W3CではXML Schemaの開発を進めているが，仕様が複雑で標準化は難航している．他の主なスキーマ言語として，XDR，SOX，Schematron，DSD，RELAX NGなどがある．これらのスキーマ言語を，使いやすさの観点で比較すると，DTDが独自構文であるにもかかわらず，スキーマ言語習得が最も簡単であるとされている．Schematronの言語仕様は非常に単純で記述性が高いが，XPathを知っていることが前提になる．RelaxingはXML Schema並みの記述性を兼ね備えていながら，シンプルな構造の言語である．スキーマ言語の比較について，詳しくは文献[Dongwon InternetDocument]に記されている．

XMLデータベースへの問い合わせ言語は，いくつか提案されているが，広く普及されていて機能性の高いものに，W3Cから勧告を受けているXQuery 1.0: An XML Query Language[*1]がある．XQueryは，表現式という単位から構成され，表現式の組み合わせによって複雑な問い合わせを記述できる関数型言語である．XQuery 1.0はXPath 2.0をサブセットとして含んでおり，XMLの各要素を指定するにはXPathを使用する．

一般的にXMLの検索の用いられる問い合わせ言語のXQueryは，検索に関する問い合わせしか行えず，更新や削除などに対してパスの指定を行えるのみである．W3CによりXQuery Update Facility[*2]が考案されているが，勧告が難航しており，XQuery Update Facilityを実装しているシステムがほとんど無いのが現状である．また，検索の問い合わせなどを行う際に，ユーザは詳しい内部構造を知っている必要があり，既存の検索は，タグ構造のみに着目した検索である．

本論文では，制約論理プログラミングに基づくXML要素の検索について説明する．本論文で提案するXML検索システムは，XML文書に出現するタグの意味的な情報，例えばタグの構造パターンや要素のデータの種類のような情報からXMLデータを格納や更新などを行う際に学習し，制約論理プログラミングの制約として格納するシステムである．制約論理プログラミングのスキーマ構造をXMLデータから生成し，そのスキーマ構造がXMLデータの意味的な定義となっているため，異なるXML構造の文書に対して同じ意味のデータを認識することができる．また，タグやタグ要素に対する数値計算の制約を処理することも可能とする．

2. 制約論理プログラミング

ある問題を解決しようとするとき，その問題自身を厳密に記述する必要がある．そのために，問題の領域とその領域において問題を構成する対象及びこれら対象間の関係を明確にしなければならない．制約とは，このような対象間の関係を宣言的に記述したものである．制約論理プログラミングは，論理プログラミングに制約の考えを付加したものである．

本システムの制約論理プログラミングは，大囿らが提案しているリフレクティブな制約論理プログラミングの提案を参考に実装されている [?]．

論理プログラミングにおけるユニフィケーションは，等式の左辺はすべて変数であり，左辺に現れる変数は互いに異なり，左辺に現れる変数は右辺に現れないという制約を持つとき，すべての充足可能な制約に対してまったく同じ解を持つ標準形の制約がただ1つ存在することを保証している．本システムでは，線形方程式，線形不等式を扱うために制約解消系としてシンプレックス法を用いており，制約に対して制約充足可能かどうかを判定し，充足可能であればすべての解を見つけ出す．ま

連絡先: 名古屋工業大学大学院 工学研究科 情報工学専攻
466-8555 愛知県名古屋市昭和区御器所町
{eki, asami, ozono, tora}@ics.nitech.ac.jp

*1 http://www.w3.org/TR/xquery/

*2 http://www.w3.org/TR/2006/WD-xqupdate-20060508/

た，解が有限個の場合には，生成手続きが必ず停止するという条件も持つ．

```
<?xml version="1.0" encoding="Shift_JIS"?>
<xml>
  <school>
    <college>
      <meta id="0" />
      <NIT>
        <student>
          <name>MasayaEki</name>
          <number>12345678</number>
          <address>Japan</address>
        </student>
      </NIT>
    </college>
    <college>
      <meta id="1" />
      <NIT>
        <student>
          <name>AsamiShohei</name>
          <number>98765432</number>
          <address>Japan</address>
        </student>
      </NIT>
    </college>
    <college name="NIT">
      <meta id="2" />
      <student>
        <name>MasayaEki</name>
        <number>12345678</number>
        <age>22</age>
      </student>
    </college>
    <daigaku>
      <Nitech>
      <meta id="3" />
        <gakusei>
          <name>MasayaEki</name>
          <number>12345678</number>
          <address>Japan</address>
        </gakusei>
      </Nitech>
    </daigaku>
    <junior name="nagoya">
      <meta id="4" />
      <student>
        <name>MasayaEki</name>
        <number>12345678</number>
        <address>Japan</address>
      </student>
    </junior>
  </school>
  <shop>
    <nop />
  </shop>
</xml>
```

図 1: XML 文書

3. XML 文書の格納

XML データから知的情報を取得する試みとして，浅井らが木のグラフのマイニングに関する研究を行っている [Asai 04]．著者らは，半構造データストリームマイニングの手法を提案している．また，姚らが Suffix Array を用いた XML データの検索に関する研究を行っている [姚 04]．本システムは，Suffix Array にて XML データを格納し，パターンなどの発見を行っている．

格納される XML データは，あらかじめスキーマが定義されている必要は無く，XML 文書を格納してから制約によるスキーマを生成する．このとき生成されるスキーマは，タグとタグの関係をタイトに表現するものではなく，意味的なタグの集合を制約で表している．

本システムは，最初に XML データのタグの親子関係やタグの要素，属性などの制約を格納する．次に，部分的なタグ構造に関する制約を格納し，最後にそれらの制約の重複を削除したり，意味的な対応関係などを判別する．

また，本システムは Java 言語で実装されており，論文 [Margaret 06] を参考に実装されている．また，Java 言語で実装されているので，本システムはマルチプラットフォームに対応している．

3.1 本システムにおける制約論理プログラミング

以下は，本システムであらかじめ設定されている制約の一部である．本システムの制約論理プログラミングは，[大囿 96] を参考にしている．紙面の都合上，制約の一部分は記述は省略する．変数は，タグに大文字が許されているので，Prolog の変数と異なって '_ (アンダースコア)' で始まる英数字の文字列である．例えば，_X, _hohe, _12345 などは全て変数である．制約に記述されている変数は，college のようなタグの名前を意味するもしくは，一番上の college タグのようなある特定されたタグを意味するかのどちらかである．

一般的なタグ構造の格納に関して，通常はシステムであらかじめ設定されている制約及び数値演算を用いて問い合わせ行う．本システムの制約解消系は，制約論理プログラミングの制約解消系としてユニフィケーションが存在するが，本システムはシンプレックス法を用いている．ユニフィケーションは，等式の左辺はすべて変数であり，左辺に現れる変数は互いに異なり，左辺に現れる変数は右辺に現れないという制約を持つとき，すべての充足可能な制約に対してまったく同じ解を持つ標準形の制約がただ 1 つ存在することを保証している．本システムでは，線形方程式，線形不等式を扱うためにシンプレックス法を用いており，制約に対して制約充足可能かどうかを判定し，充足可能であればすべての解を見つけ出す．また，解が有限個の場合には，生成手続きが必ず停止するという条件も持つ．

```
parent(_X, _Y)    :- _X が_Y の親タグである．
child(_X, _Y)     :- _X が_Y の子供タグである．
parallel(_X, _Y) :- _X と_Y が同じ親タグを持つ．
member(_X, _Y)    :- _Y が_X に含まれる．
count(_X)         :- _X のタグを数える．
element(_X, _Y)   :- _X の要素を取得する．
attribute(_X, _Y):- _X の属性_Y を取得する．
node(_X)          :- _X 以下の木構造を取得する．
root(_X)          :- _X がルートタグである．
leaf(_X)          :- _X が葉である．
```

3.2 タグに関する情報の格納

ここでは，タグに関する情報の格納の手順について説明する．本システムではまず，タグに関する情報の格納を行い，規則性を持った部分的な構造には次に説明する部分的なタグ構造の格納を行う．

図 1 の XML 文書を格納したときに生成される論理節について説明する．生成される論理節は，root(_X)，parent(_X, _Y)，leaf(_X)，element(_X, _Y)，attribute(_X, _Y) の 5 つである．ここでは，最初に root(xml)．及び root(xml タグの id) が生成される．

次に，parent(xml, school)． 及び parent(xml タグの id, school タグの id)．が生成される．parent(xml, school)．は，xml タグと school タグが親子関係であることを示し，xml タグの部分集合が，school タグの部分集合を子に持つという意味である．parent(xml タグの id, school タグの id)．は，xml タグと school タグの id の親子関係を示し，特定の xml タグが特定の school タグを子に持つという意味である．以降すべての構造たどるように parent を生成していく．

parent を生成している間に属性を持つタグ，ここでは meta タグを見つけたとき，attribute(meta, [id, 0])． 及び attribute(meta タグの id, [id, 0])．を生成する．また，葉タグを見つけたとき，ここでは name タグを見つけたとき，leaf(name)．及び leaf(name タグの id)．を生成する．さらにその葉タグ

が要素を持つとき，element(name, MasayaEki). 及び element(name タグの id, MasayaEki). を生成する．

最後に，タグから id を求めるインデックスを生成する．図 1 のインデックスは以下のようになる．このインデックスを利用することで，どのタグがどこに存在するかわかる．

3.3 部分的なタグ構造の格納

ここでは，部分的なタグ構造の格納の手順について説明する．部分的なタグ構造の格納は，タグに関する情報の格納の後に行う．ここでのタグ構造とは，同じ構造が繰り返し出現するという意味であるが，同じ意味の構造が繰り返し出現するときも規則性があるといえる．

例えば，図 1 では NIT という college の student に関するタグ構造が，若干異なったタグ構造で表現されているものがある．一般に，ユーザやアプリケーションによって，同じ種類の意味だがタグ構造が若干異なる構造が作られる場合がある．シングルユーザのみで用いられるシステムなら問題はあまり無いが，一般にはデータベースシステムは多人数が作ったデータを多人数で操作するシステムであることが多々あるので，このとき自分で作ったタグ構造を操作することは可能だが，他のユーザが作ったタグ構造に対して操作を行うことができなくなる．

本システムでは，このようなタグ構造を規則的なタグ構造の制約として生成する．この制約に問い合わせを行うことで，実構造は変わらないが，システムがビューを作ることで，見かけ上同じ構造とみなし，必要なデータを返すことができる．

図 1 の XML 文書における規則的なタグ構造の制約をつくる手順について説明する．

葉にあたるタグは，タグ構造に関する制約は持たないので，葉にあたるタグの親のタグの規則性を探す．さらに親のタグへとたどっていき，ルートタグまで繰り返す．

図 1 では student タグが 4 個存在し，それらの子には name, number, address, age タグが存在する．この中で name, number タグはどの student タグも子として持ち，address, age タグはどちらか片方だけを子として持っている．このとき，student に関する制約は以下のようになる．制約の名前は，システムの内部的に使用する値だが，ここでは説明のため student() という名前にした．

```
student() :-
  child(name, student), leaf(name),
  child(number, student), leaf(number),
  (child(address, student);
  child(age, student)),
  (leaf(address); leaf(age)).
```

また，葉にあたるタグを子に持つ他の構造の制約を以下のようになる．

```
gakusei() :-
  child(name, student), leaf(name),
  child(number, student), leaf(number),
  child(age, student), leaf(age).

shop() :- child(nop, shop), leaf(nop).
```

ここで，student(). と gakuse(). は，同じ構造を持つので，同じ意味であるタグ構造と認識する．次に student タグか gakusei タグを子として持つタグ構造の制約を考えると以下のようになる．

```
NIT() :- child(student(), NIT).

college() :-
  child(student(), college),
  child(meta, college), leaf(meta),
  attribute(college, [name, NIT]).

junior() :-
  child(student(), junior),
  child(meta, junior), leaf(meta),
  attribute(junior, [name, nagoya]).

Nitech() :-
  child(gakusei(), Nitech),
  child(meta, college), leaf(meta),
```

ルートタグまでの残りすべての制約以下のようになる．

```
college2() :-
  child(NIT(), college),
  child(meta, college), leaf(meta).

school() :-
  child(college(), school);
  child(daigaku(), school);
  child(junior(), school).

xml() :-
  child(school(), xml); child(shop(), xml).

daigaku() :- child(gakusei(), daigaku).
```

ここまでの制約は，タグ構造のみに着目した制約である．これから，同じ種類の意味の構造に着目した制約を追加する．

student(). と gakuse(). は，同じ構造を持つので，同じ意味であるタグ構造と認識できる．また，college タグの中に NIT タグを持つものと，college タグの属性に NIT を持つものは同じ意味である構造と認識できる．ここでは以下の制約を追加する．

```
student() :- gakusei().
college() :- college2().
```

3.4 制約の更新

3.3 節では，意味的なタグ構造に関して制約を決めたが，ここでは，さらに XML 文書を追加したり，ユーザの問い合わせの偏りなどから制約を更新することについて説明する．

XML 文書の追加において，まずはタグ構造のみに着目した制約を 3.3 節と同様に行う．そして，タグ構造のみに着目した制約に関して，追加及び更新，削除を行う．

例えば以下のタグ構造を図 1 の school タグの下に追加したときを考える．

```
<daigaku>
  <gakusei>
    <money>10000</money>
    <sex>male</sex>
    <blood>O</blood>
  </gakusei>
</daigaku>
```

このとき，新たに以下の制約を追加する．

```
gakusei2() :-
  child(money, gakusei), leaf(money),
  child(sex, gakusei), leaf(sex),
  child(blood, gakusei), leaf(blood).
```

このとき，3.3 節で生成した gakusei に関する制約は，student と同じ意味であったが，今回新たに追加された制約によって，同じ意味という解釈ができなくなる．従って，student() :- gakusei(). を削除する．さらに，gakusei と gakusei2 の制約を合成し，以下の制約を生成する．

```
gakusei() :-
  {child(name, student), leaf(name),
  child(number, student), leaf(number),
  child(age, student), leaf(age)};
  {child(money, gakusei), leaf(money),
  child(sex, gakusei), leaf(sex),
  child(blood, gakusei), leaf(blood)}.
```

今回の例では，制約の追加及び更新，削除について説明した．このとき，データを削除したのは student() :- gakusei(). であり，gakusei2 の制約は，現状では参照されないが，今後参照される可能性があるのでデータを削除しない．このようにして，削除には制約そのものを削除するものと，現在の参照を削除するものがある．

ユーザの問い合わせに関する情報は，リアルタイムで制約に反映させるのではなく，一定量溜まったらバッチ処理を行う．問い合わせや構造には，よく用いられる問い合わせや参照される構造，ほとんど参照されない問い合わせや構造などがある．システムとして同じ意味の構造であると認識していない複数の構造に対して，ユーザが何度もそれらの構造に問い合わせを行っている場合，それらの構造が同じ意味であるか関連がある可能性が高いので，パターンの格納を条件付きで行う．この条件付きとは，同じ意味と見なす条件を緩くし，その際に生成される制約を，通常のパターンの格納において生成された制約と違うという認識をすることである．また，よく参照される構造に対しては，評価する順番の優先度を上げたり，逆に参照されない構造に対しては優先度を下げたりする．

4. 実験と評価

本節では，本システムの検索速度及び制約からの推論の実行速度の実験について論じる．

実験環境は，OS は WINDOWS XP Professional，CPU は 1.8 GHz Pentium4 Processor，メモリは 1GB(SDRAM) である．実験データは，ウェブ上に落ちていた様々な XML 文書を貼り繋いだりしてつくった XML データである．この XML データを，本システムではノードの数に実行速度が比例するので，ノード (XML タグの開きと閉じを組とし，空タグはそれ 1 つでノード 1 つと数える) のサイズ毎に実験を行った．また，そのときに XML データのドキュメントサイズも記述してある．実験では，ルートタグから木構造としてすべてのタグを辿ったときの検索速度と，意味的なタグ構造の制約に問い合わせを行った知識推論の実行時間を，それぞれ 100 回の実験を行った検索速度の平均値として記録した．表 1 は，実験の結果を表にしたものである．また，図 2 は，表 1 をグラフで表現したものである．グラフよりわかることは，通常の検索に関してはノード数 n に対して O(n) となっているが，推論にかかる時間が $O(n^2)$ で増加している．規模が中程度ならば，これらの実行速度において，支障はあまり無い．

表 1: 実験結果

Number of nodes	Document size (Kbytes)	Search time (ms)	
		element	node
10,000	1,500	78	51
50,000	7,520	164	100
100,000	16,800	251	184
150,000	25,100	420	275
200,000	33,500	662	320
250,000	41,000	990	412
300,000	52,300	1335	573

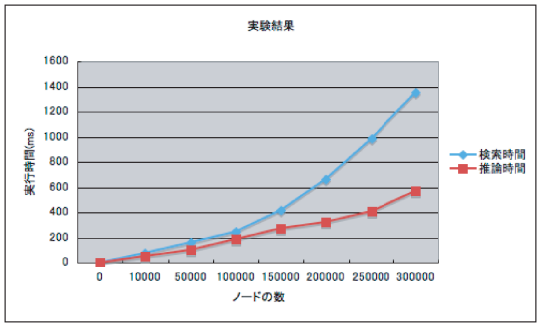

図 2: 実験結果のグラフ

5. おわりに

本論文では，制約論理プログラミングに基づく XML 要素の検索について述べた．本論文で提案した XML 検索システムは，XML 文書に出現するタグの出現頻度などを，データを格納する際に学習し，制約論理プログラミングとしてデータベースに格納するシステムである．ここでの XML 文書は，あらかじめスキーマが生成されている必要は無く，システムが XML 文書を格納してから制約によるスキーマを生成する．このとき生成されるスキーマは，タグとタグの関係をタイトに表現するものではなく，同じ意味を表す異なったタグ構造を制約で表現している．また，タグの数関係や，タグの要素に対する数値計算の制約を処理することも可能とした．

今後の課題として，Weka[*3]のような機械学習と組み合わせて，より精度の高い制約をつくることや，ユーザの問い合わせに関する偏りから制約を考えることなどが挙げられる．

参考文献

[Asai 04] 浅井達哉，有村博紀，“半構造データマイニングにおけるパターン発見技法”，電子情報通信学会論文誌，Vol.87, No.2, pp.79-96(2004).

[Dongwon InternetDocument] Dongwon Lee, Wesley W. Chu, “Comparative Analysis of Six XML Schema Languages”, Internet Document.

[Margaret 06] Margaret G. Kostoulas, Morris Matsa, Noah Mendelsohn, Eric Perkins, Abraham Heifets, “XML Screamer: An Integrated Approach to High Performance XML Parsing, Validation and Deserialization”, pp.93-102, WWW2006.

[大囿 96] 大囿忠親，新谷虎松，“マルチエージェントシステムのための制約論理型言語 RXF の実現”，情報処理学会論文誌，Vol. 37, N0. 10, pp.1765-1772(1996).

[姚 04] 姚 全珠，都司 達夫，“Suffix Array に基づく大規模 XML 文書のための高速サーチエンジン”，福井大学工学部研究報告，Vol.52, No.2, pp.153-160(2004).

∗3 http://www.cs.waikato.ac.nz/ml/weka/index.html.