例示操作に基づくXQuery問合せの学習機構

A Mechanism to Learn XQuery Queries Based on Example Operations

森嶋 厚行♥ 松本 明♦ 北川 博之♠
Atsuyuki MORISHIMA Akira MATSUMOTO
Hiroyuki KITAGAWA

既存のXML問合せ言語や操作系では，ユーザはテキストエディタや視覚的操作系を用いて問合せを直接記述する．本稿ではこれらとは全く異なるアプローチのXML操作系であるXLearnerについて述べる．XLearnerは，サンプルのXML要素に対するユーザの例示操作からシステムがXQuery問合せを学習する．XQuery問合せ学習という自明でない問題に対し，XLearnerはシステマティックな解を提供する．

Existing XML query languages are textual or graphical languages in which we can specify queries for XML manipulation. This paper explains XLearner, a different kind of manipulation framework for XML. XLearner learns XQuery queries based on operations of sample XML elements. XLearner provides a systematic solution to learn XQuery queries, although learning XQuery queries is a non-trivial task.

1. はじめに

XMLはデータ交換のためのデファクトスタンダードとなり，XMLを対象とした問合せ言語や操作系が数多く提案されてきた．これらでは一般に，ユーザはテキストエディタや視覚的操作系を用いて問合せを記述する．我々はこれらとは全く異なるアプローチのXML操作系XLearnerの研究開発を行っている．これは，サンプルのXML要素に対するユーザの例示操作からシステムがXQuery問合せを学習するものである．

XQueryの学習は次の理由により自明ではない．(1)XMLは半構造データである．RDBで問合せ対象となるリレーションは完全に規則的な構造を持つが，半構造データはそうとは限らない．したがって，単純な例示操作だけでは学習が出来ない．(2)XQuery問合せは様々な構成要素(パス式，結合/選択条件，入れ子構造など)の複合体である．またこれらは複雑に絡み合っている．

この学習は自明でないだけでなく，計算量的に困難である．XML問合せ言語では，半構造性に対応するため一般にパス正規表現（もしくは相当物）が使われる．ここで，XML要素のパスを文字列と見なすと，パス正規表現を学習することは，パスの集合が表す言語を受理する有限オートマトンを求めることと言い換えることが出来る．ある言語が与えられた時，それを受理する最小の有限オートマトンを見つけることはNP-completeである[2]．一方，能動学習(Active Learning)を利用すれば，多項式時間で発見する事が可能であることが知られている[1]．ここで能動学習とは，ユーザがシステムに例を与えるだけでなく，システム側からユーザに質問可能な枠組みである．本システムは，能動学習の枠組みを利用して，パス正規表現(以下パス式)の学習を行う．

本稿では，XLearnerシステムの概要および学習機構のアルゴリズムについて説明する．本アルゴリズムの特徴は，特定のクラスのXQuery問合せの学習に関して完全な事である．

2. XLearner

まずXmark[4]のデータを利用した例を示す．このデータは，インターネットオークションサイトのデータのXML表現である．図1にDTDの一部を示す．item要素はオークションの対象となるitemを表す．categoryは，itemが属するカテゴリを示す．それぞれのitemが実際にどのカテゴリに属するかは，itemの子要素であるincategoryのcategory属性に，category要素への参照（IDREF）を持つことにより表す．この時，図2 の問合せを行いたいとする．これは，各カテゴリごとに，AfricaもしくはEuropeのitemの名前および説明を示すものである．

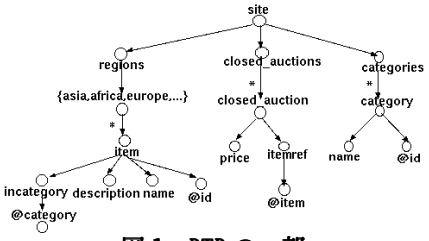

図 1 DTD の一部
Fig.1 A Fragment of DTD

```
<i_list> {
  FOR $c IN /site/categories/category
  RETURN <category>
    <cname>$c</cname> {
      FOR $i IN /site/regions/(europe|africa)/item
      FOR $o IN /site/closed_auctions/closed_auction
      WHERE $o/itemref/@item = $i/@id
      RETURN <item>
        <iname>$i/name</iname>
        <desc>$i/description</desc>
      </item>
  }</category>
}</i_list>
```

図 2 XQuery (text()は省略)
Fig.2 XQuery

XLearnerのアーキテクチャを図3に示す．まず，結果として欲しいXMLのDTDをテンプレート生成器に入力する．次に，ユーザは操作対象となるXML文書の中から，いくつか要素もしくは値（ここでは**XMLノード**と呼ぶ）を選び，それらをテンプレートにドラッグアンドドロップし，結果として欲しいXML文書の一部分を示す．テンプレートにドラッグアンドドロップされたXMLノードを**例示ノード**と呼ぶ．

与えられたXML問合せ結果の断片に従って，学習エンジンは，ユーザが意図する問合せを学習するための質問をユーザに行う．最後に，システムはXQuery問合せを出力し，XQuery処理系に渡す．

具体例として上記の操作を説明する．まず，結果のDTDをテンプレート生成器に入力すると，XLearnerは図4(b)のテンプレートを表示する．ユーザは，XMLブラウザ中のXMLの要素をいくつか選び，ドラッグアンドドロップし，テンプ

♥ 正会員 芝浦工業大学工学部情報工学科
amori@sic.shibaura-it.ac.jp
♦学生会員 筑波大学大学院システム情報工学研究科
akey@kde.is.tsukuba.ac.jp
♠ 正会員 筑波大学電子・情報工学系
kitagawa@is.tsukuba.ac.jp

レートを埋める. 出力したいXML文書と矛盾していなければ何でも良い．ここでは，図4のように埋めるとする．矢印はドラッグアンドドロップ操作を表している. PotterはEuropeのitemの例である.

次にXLearnerは，ユーザの意図したXQuery問合せ（図2）を求めるために，ユーザに対して質問を行う．より具体的には，各例示ノードの**extent**を決定するための質問である．例示ノードe のextent（EXT_e）とは，eの**context**において，例示ノードが代表しているノードの集合である．直感的には，例示ノードのcontextとは, 既にextentが決定された例示ノードの集合である．例えば，extentが“book,”“Potter,”“Best Seller”の順番で決まるとする．この場合，EXT_{book}のcontextは空であり，全てのcategoryの名前の値の集合がEXT_{book}となる．二番目の例示ノードのextent（EXT_{Potter}）は, そのcontext（すなわち, bookカテゴリ）において, Africaもしくは Europeのitemの名前の集合となる．extentを求める順番の決定については後述する.

質問には，**Membership Query**（以下MQ）と**Equivalence Query**（以下EQ）の二種類がある.

MQ: 例示ノード eに対するMQとは，XLearnerがあるXMLノードnを選び，ユーザに対してnがEXT_eに含まれるかどうかを聞くことである．ユーザはYもしくはNで答える.

EQ: XLearnerは，XMLブラウザで，eの推測したextentである $E\hat{X}T_e$ のXMLノードをハイライト表示し， $E\hat{X}T_e = EXT_e$ であるか聞く．正しい場合，ユーザは[OK]ボタンを押すことで，XLearnerに正しいことを伝える．正しくない場合，ユーザは反例nをXLearnerに与える．反例は，EXT_e と $E\hat{X}T_e$ の対称差に属するノードである． $n \in EXT_e - E\hat{X}T_e$ $(n \in E\hat{X}T_e - EXT_e)$ の時，nは正（負）の反例という.

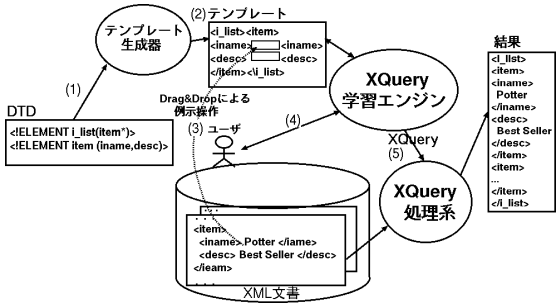

図 3 XLearner のアーキテクチャ
Fig.3 XLearner's Architecture

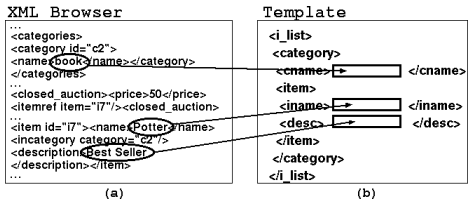

図 4 XML ブラウザとテンプレート
Fig.4 XML Browser and Template

3. XQ-Tree

XQ-TreeはXQuery問合せを表す木構造であり，XLearnerが内部的に利用する．図2の問合せに対応するXQ-Tree t_1を図5に示す．XQ-Treeは次のように説明できる．まず，本質的にXMLインスタンスはXMLノードの入れ子列である．XQ-Tree t の各ノードは，それぞれの列の計算方法を表している．入れ子関係は木構造で表される．tの各ノードn は $N_i : -q_t(N_i)$ の形式をしている．ここで， N_i はノード識別子であり， $q_t(N_i)$ は，XMLノードの各列を計算するための問合せ断片（flwr式の形式を持つ）である．以下ではtが文脈から明らかな場合には添え字tを省略することがある.

XQuery問合せQが与えられた時, それに対応するXQ-Tree $XQT(Q)$は次の手順で計算される: (1)let式を除去し，変数をそれに割り当てられた式で置き換える．(2)全てのXMLノード列を返す式（例えば，$c/name）を，等価なflwr式（for $cn in $c/name return $cn）に書き換える．これにより入れ子のflwr式ができる. (3)以上で作成したflwr式間の入れ子関係を木構造で表す.

実際に各ノード列を計算するにはnの**完全問合せ** $cq(n)$ が必要である．ここで $cq(n) = \oplus_{\forall m \in depends(n)} q(m)$ であり，$depends(n)$ は，nのノード列を計算するのに必要な問合せ断片を持つXQ-Treeノードの集合（nを含む）である．一般に，nのノード列の計算にはnの祖先の問合せ群が必要である. また，もしnのreturn節以外に参照（ノード識別子）が現れるならば，被参照ノードの式の計算も必要である． $q \oplus q'$ は問合せqとq' の合成を表す．合成は，式中のXQ-Treeノード識別子N_i を$q(N_i)$で置き換える事である．例えば，$cq_{t_1}(N1.1.1)$ = for $c in /site/categories/category, $cn in $c/name return $cn[1]である.

$q(n)$のreturn節の式と$q(n)$のfor節で変数に束縛される値が1対1対応し，かつ同じ値であるとき，nは**simple**であるという．形式的には，$q(n)$ = for v in p where c return v の形をしている時（where節は省略可．return節のタグは無視）である．nがsimpleである時$simple(n)$と表記する.

XQ-Treeの辺は，子要素のmultiplicityが1である時，1でラベル付けされる．この情報は学習アルゴリズムにおける問合せ断片の依存関係の判定で重要な役割を果たす．これは与えられたDTDに基づいて決定される．すなわち．要素定義がA=(B,C)の時，辺n_A-n_Bとn_A-n_Cにはそれぞれ‘1’がラベル付けされる.

XQuery問合せQをXMLインスタンスIに対して適用した結果を$Q(I)$と表記する．同様に，XQ-Tree tに関しては$t(I)$を用いる.

XQ-Treeの問合せ断片はlet節を持たないため，変数vは常にfor v in p もしくは，some v in p によって定義される．この時，“v in p”を$expr_t(v)$で表記する．$Expr_t^*(v)$ はv を計算するために必要なパス式の集合である．次のように定義する．(1) $expr_t(v) \in Expr_t^*(v)$. (2)もし変数wが存在し，$"x \text{ in } w/q" \in Expr_t^*(v)$ ならば， $expr_t(w) \in Expr_t^*(v)$. $Expr_t^*(v)$ 中の式を一つの式にまとめたものを $expr^*(v)$ と書く. 例えば， $Expr_{t_1}^*(\$cn)$ = {$c in /site/categories/category/, $cn in $c/name}であり， $expr_{t_1}^*(\$cn)$ $cn in /site/categories/category/nameである.

$associatedVar(v)$を $Expr^*(v)$ 中に現れる変数の集合とす

[1] 説明を簡単にするため, 式を同等のより簡単な式で表している. また return 節のタグを省略する.

る．また，*associatableVar*(v)を次のように定義する．まず，for節でvを定義するノードをn_vとする．この時，*associatableVar*(v)はn_vの祖先ノード(n_v自身を含む）に現れる変数の集合である．これらの関係として，*associatedVar*(v)⊆*associatableVar*(v)が成立する．また，もしv'∉ *associatableVar*(v)ならば，XQueryの仕様により，$q(n_v)$ではv'を参照できない．

```
N1:-       return <i_list>N1.1</i_list>
N1.1:-    for $c in /site/categories/category
           return <category>N1.1.1 N1.1.2</category>
N1.1.1:-   for $cn in $c/name return <cname>$cn</cname>
N1.1.2:-   for $i in /site/regions/(europe|africa)/item
          where some $ic in $i/incategory/@category satisfies
           (some for $ci in $c/@id satisfies ($ic=$ci))
           return <item>N1.1.2.1 N1.1.2.2</item>
N1.1.2.1:-for $in in $i/name return <iname>$in</iname>
N1.1.2.2:-for $id in $i/description return <desc>$id</desc>
```

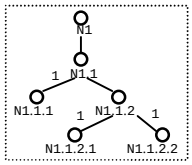

図5　図2の問合せのXQ-Tree表現 t_1
Fig.5 XQ-Tree t_1 **for the query in Figure2**

クラスXQuery0*(I)*. まず，XQ-Treeのクラス**XQT0**$^-$を定義する．説明を簡単にするため，一般性を損なうことなく，議論するXQ-Treeの形式に制限を付け，「根から葉へのいかなるパスも1でラベル付けされた辺（1-edge）を連続して持たない」XQ-Treeについてのみ議論する．図5のXQ-Treeはこの制限を満たす．

XQT0$^-$を次のように定義する．XQ-Tree tの全てのノードnについて*Learnable*(n)が成立するとき，t∈ **XQT0**$^-$である．*Learnable*(n)は$q(n)$がMQとEQによって学習可能であることを示す．具体的には，$q(n)$(= `for` e_f `where` e_w `return` e_r)が次の6つの条件を満たす場合に成立する．

- **(C1)** 全ての変数vに対して，$expr_t^*(v)$は，`document`関数から始まるパス正規表現である．
- **(C2)** $n \in N_t^{simple}$
- **(C3)** `if then else`式を条件として持たない．
- **(C4)** e_rは変数，タグ，XQ-Treeノード識別子以外を含まない．
- **(C5)** `sortby`節を持たない．
- **(C6)** vをXQ-Treeノードnで定義される変数とする．この時，e_wは$\wedge_{v' \in (associatableVar(v) - associatedVar(v))} RS(\{v,v'\})$の形式である．ただし，$RS(\{v_1,v_2\})$は*True*もしくは$RS'(\{v_1,v_2\})$である．$RS'(\{v_1,v_2\})$は次の式のいずれかである．
  - $data(v_1) = data(v_2)$
  - some w in v_1/p satisfies $RS'(\{w,v_2\})$
  - some w in document()/p satisfies $RS'(\{v_1,w\}) \wedge RS'(\{w,v_2\})$

ここで，pはchild軸だけからなるパス式(a/b/cなど)である．

次に，*Learnable*(n)の条件を次のように緩めたクラスを考える．「あるノードがsimpleノードでない(条件**(C2)**を満たさない)ときには，間接的に$q(n)$が一意に決まる．」具体的には，条件**(C2)** を，$r \notin N_t^{simple} \Rightarrow Learnable'(n)$ とした XQ-Treeのクラスを**XQT0**とする．直感的には，*Learnable*'(n)は，$q(n)$が他の1対1対応した要素の問合せ$q(n')$から間接的に求まるか，そうでなければ$q(n)$ = ()であることを表す．これは$q(n)$が`return`節に変数を含まず，かつ次の2つが共に成立するとき成立する．

(A1) $clue(n) \neq \phi \Rightarrow \forall n' \in clue(n)(simple(collapse(n,n')))$

(A2) $clue(n) = \phi \Rightarrow q(n) = ()$

ここで$clue(n) \equiv 1\text{-}children(n) \cap \{m \mid simple(m)\}$である．1-*children(n)*は1-edgeで接続された子ノードの集合である．例えば，$clue(N1)$={*N1.1*}，$clue(N1.1.2)$={*N1.1.2.1*, *N1.1.2.2*}である．$collapse(n,n')$はnとn'を合成したノード（すなわち，$q(collapse(n,n')) = q(n) \oplus q(n')$）である．

XQT0は`return`節以外にノード識別子を含まないので*depends(n)*はnの祖先の集合である．

XQueryのクラス**XQUERY0*(I)***を次のように定義する．あるXQuery問合せQが与えられた時，Q∈ **XQUERY0*(I)***である必要十分条件は，$\exists Q'(Q'(\boldsymbol{I}) = Q(\boldsymbol{I}) \wedge XQT(Q') \in$ **XQT0**)である．図2の問合せは**XQUERY0*(I)***に属する．**XQUERY0*(I)***がXMLインスタンス$\boldsymbol{I}$でパラメタライズされている理由は，XMLインスタンスを用いた例のみを用いるため，学習可能性が$\boldsymbol{I}$に依存するからである．

4. XQuery 学習アルゴリズム

本節では，**XQT0**に属する問合せを学習するアルゴリズムを示す．

(ステップ1) XQ-Treeスケルトンの生成: 与えられたDTDと例示ノードから，学習対象のXQ-Treeの基となるXQ-Treeスケルトンを生成する．図6に，2節の例に対するXQ-Treeスケルトンを示す．

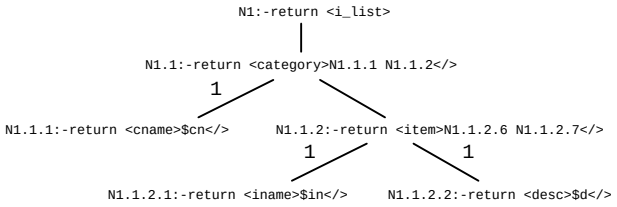

図6　XQ-Treeスケルトン
Fig.6　XQ-Tree skeleton

(ステップ2) 各問合せ断片の学習: 本稿では最上位アルゴリズムだけを示す．本アルゴリズムは，XQ-Treeスケルトンのノードを依存関係順に走査しながら，各ノードの問合せ断片$q(n)$を推論する．依存関係順にする理由は，$q(n)$学習時に*depends*(n)の問合せが常に既知となるため，それらとの関係を学習に利用可能だからである．**XQT0**$^-$の場合は，全てのXQ-Treeノードがsimpleであるため，単純に深さ優先順に各ノードの$q(n)$を決定していけばよい．**XQT0**の場合はsimpleでないノードを推論する必要があるので多少複雑となる．図7は**XQT0**のXQueryを学習するアルゴリズムの最上位レベルである．5行目のlearnQuery(n)は，$q(n)$を推論する関数である．**XQT0**の定義より，ドロップされた要素を持つかどうかでsimpleノードを判定できる．図7 の9-12行では，nがsimpleノードでない時の$q(n)$を計算している．

本アルゴリズムは1-edgeを優先した重み付き深さ優先順に学習を行なう．また，実は条件(A1)より，いずれかの$n' \in clue(n)$に対して$q(n) = q(n')$とすればユーザの意図と矛盾

しないXMLノード列を学習できるが，本アルゴリズムでは全てのn'に対して$q(n')$の合成を行なったものを$q(n)$とする（6-7行）．これらの理由は，$learnQuery(n)$は$depends_t(n)$の問合せ断片に含まれる情報を利用するため，利用する可能性のある情報は全て$depends_t(n)$に入れておくためである．詳細は次節で説明する．

```
1. XQTree learnXQTree(XQTree t) {
2.  for each node n in t { //in the weighted
3.                         //depth-first order
4.      if (simple(n)) {
5.         n.query = learnQuery(n);
6.         if (n in 1children(n.parent))
7.            replace(t,n.parent,collapse(n.parent,n));
8.      } else {
9.         for some n' in clue(n) do {
10.             replace(t,n,collapse(n, n'));
11.             n.query = learnQuery(n);
12.        }
13.     }
14.  }
15. }
```

図7　最上位アルゴリズム
Fig.7　The top-level algorithm

4.1 Simpleノードの問合せの学習

XMLノードeがドロップされたXMLノードをn_eとする．simpleノードの定義より，$q(n_e)$は`for` v_e `in` p `where` c `return` v_eの形式を持つ．したがって，$q(n)$の学習とはパス式pと条件cを学習することである．

Extentの学習．$q(n_e)$が与えられた時，n_eの完全問合せ$cq(n_e)$は次のように書ける．

$$cq(n_e) = \texttt{for } e'_f, v_e \texttt{ in } p \texttt{ where } e'_w \wedge c \texttt{ return } v_e$$

ここでe'_fとe'_wは$depends(n_e)$中のノードの式である．EXT_eは$cq(n_e)$とeのcontextを用いて定義される[3]．

EXT_eの学習は次のように行われる．(1) XLearnerがEXT_eを学習するためのMQを発行する．ユーザからの答えに基づき，仮説$\hat{EXT}_e$を生成する．(2) XLearnerがEQを発行し，$\hat{EXT}_e = EXT_e$の質問をする．(3) $\hat{EXT}_e = EXT_e$ならば終了する．学習結果は$q(n_e)$=`for` v_e `in` p `where` c `return` v_e となる．そうでなければ(1)へ戻る．ただし，仮説$\hat{EXT}_e$におけるpおよびcは，与えられた反例により修正される．

パス式の学習．要素のパスを文字列とみなすと，パス正規表現を推論することは，パスの集合が表す言語を受理する有限オートマトンを求めることと言い換えることができる．XLearnerは，Angluinのオートマトン導出アルゴリズム[1]を利用して推論を行うが，このアルゴリズムをXLearnerにそのまま適用すると，ユーザとのインタラクション数が膨大になってしまう．そこで，XLearnerではXML固有の性質を利用して質問回数を削減する[3]．

where節の条件の学習．XLearnerでは，where節の条件の学習にあたり，XMLグラフの存在を仮定する．これは，XQuery and XPath data model[5]等と同様に，XMLインスタンスの構造を木構造で表したものである．さらに，我々のXMLグラフでは，v-equality辺を持つ．これは，同じ値を持つXMLノード間に存在する辺である．このとき，問合せ断片$q(n_e)$におけるcは次のように求められる．(1)XMLノード集合$\{n \mid n$はeのcontextもしくは$n_e\}$中のノード間の全てのパスを含む最小の部分グラフを求める[2]．(2)そのグラフに関して成立する制約(述語）を数え上げる．(3) 数え上げられた述語のうち，MQおよびEQを通じてユーザから与えられた全ての例に成立するものを選択する．

5. 議論

本アルゴリズムは，**XQuery0*(I)***に属する問合せの学習において完全である．ポイントは，where節の条件学習の手順(2)において，成立する可能性のある全ての述語を数え上げることである．$Learnable(n)$の条件(C6)から，$RS(\{v,v'\})$に含まれる可能性のある述語は有限である．また，条件(A1)より，問合せ断片におけるwhere節の条件で参照されるすべての変数u に対して，いずれかのreturn節に現れる変数u'が存在して，$u \in associatedVar(u')$である．したがって，uはいずれかの例示ノード間のパス上のノードであり，$RS(\{v,v'\})$に現れる述語は(1)で求めた部分グラフから求めることができる．

本機構の拡張およびインタラクション数に関する評価については[3]にある．今後の課題としては，GUI実装におけるユーザビリティーの検証などがある．

[文献]

[1] D. Angluin. Learning regular sets from queries and counterexamples. Information and Computation, 75(2):87-106, 1987.

[2] E.M. Gold. Complexity of automaton identification from given data. Information and Control, 37:302-320, 1978

[3] A. Morishima, A. Matsumoto, H. Kitagawa. XLearner: A System to Learn XML Queries through Examples. Working Paper, 2002.

[4]A. Schmidt, F. Waas, M. Kersten, D. Florescu, M. Carey, I. Manolescu, R. Busse. Why And How To Benchmark XML Databases. SIGMOD Record 30(3): 27-32 (2001)

[5]W3C. XQuery 1.0 and XPath 2.0 Data Model. http://www.w3.org/TR/query-datamodel/.

森嶋 厚行　Atsuyuki MORISHIMA
芝浦工業大学工学部情報工学科講師．1998 年筑波大学大学院工学研究科修了．博士(工学)． XML 処理・管理，異種情報源統合，情報システムのためのユーザインターフェースなどに興味を持つ．ACM, IEEE-CS，情報処理学会，電子情報通信学会各会員．

松本 明　Akira MATSUMOTO
筑波大学大学院システム情報工学研究科在学中．2001 年筑波大学第三学群情報学類卒業．XML データベースに興味を持つ．情報処理学会学生会員．

北川 博之　Hiroyuki KITAGAWA
筑波大学電子・情報工学系教授．1980 年東京大学大学院理学系研究科修了．理学博士．異種情報源統合，文書データベース，WWW の高度利用等の研究に従事．著書「データベースシステム」(昭晃堂),「The Unnormalized Relational Data Model」（共著，Springer-Verlag）等．ACM，IEEE-CS，情報処理学会, 電子情報通信学会, 日本ソフトウェア科学会，各会員．

[2] 実際の計算ではヒューリスティクスを用いて計算量を抑える[3]