Vol.2010-DBS-151 No.29
2010/11/13

XMLデータに対するキーワード検索結果の理解支援

本村 徹太郎[†1] 清水 敏之[†1] 吉川 正俊[†1]

近年，多くの XML データが蓄積されるにつれ，XML キーワード検索が重要になっている．有用な結果を得るためには，効果的な検索結果を効率的に求めることだけでなく，検索結果を理解できるような情報を付加することが望ましい．我々は，検索結果の理解支援のために，検索結果と同じ結果を返す代替問合せを取得する方法を提案する．代替問合せは検索結果の異なる側面を表しているため，利用者は検索結果の隠れた関係を推測することで検索結果を深く理解する，あるいは利用者自身の問合せを洗練することなどが可能となる．

Understading Keyword Search Results over XML Data

Tetsutaro Motomura,[†1] Toshiyuki Shimizu[†1]
and Masatoshi Yoshikawa[†1]

Recently, the more XML data are stored, the more important XML keyword search is. To get useful results, it is desirable for users not only to calculate effective results efficiently but also to suggest some information that help users understand search results. We propose a method calculating alternative queries which can get the same results queried by the original keyword set. Since each alternative query shows diverse aspects of the results, users are able to understand the results deeply by inferring hidden relationship in the results and to refine their query to get more effective results.

1. はじめに

近年，XML はデータ記述フォーマットとして急速に普及し，標準的なデータ表現の一つとして利用されている．記述されるデータの量が膨大である場合は，その中からユーザが有用なデータを探し出すために，検索する機構が必要である．XPath などの XML 特有の問合せ言語に関する知識，および検索対象の XML データの構造に関する知識を持たない多くの一般ユーザにとって，Web 検索におけるキーワード検索のような，単語を用いることによる直感的な XML 検索が可能であれば，キーワードのみを用いて情報を探し出すことが可能となり，現在までに XML キーワード検索に関する多くの研究が行われてきた[7)8)9)]．

しかしながら，一度の検索でユーザが有用な結果を得ることは困難である場合が多い．その場合，ユーザは有用な結果を得るために，問合せを洗練して再度問い合わせる，という操作を何度も行う必要がある．ユーザが有用な結果を得るためには，検索システムが効率よく効果的な検索結果を返すのみでなく，ユーザ自身の問合せを洗練することが可能な情報や，検索結果を深く理解することを可能にする有用な情報を付加して理解支援を行うことが望ましい．

XML の分野における検索結果の理解支援としては，XML 部分木の要約情報を抽出する研究[1)] や，部分木を特徴づける情報を抽出する研究[2)] がなされている．また，関係データベースの分野においては，データベースの関係表間の構造的な関係を発見するために，データベースの構造をマイニングする研究[3),4)]，あるいは，問合せに答えるのみではなく，追加の情報を付すことにより，ユーザに対する検索結果の理解支援を行う研究[5)] がなされている．

上記の検索結果の中から有用な情報を抽出するというアプローチに対して，Tran ら[6)] は，関係データベースにおいて，検索結果である関係表と同じ関係表を得る，元の問合せとは異なる問合せ *IEQs (Instance-Equivalent Queries)* を得る方法を提案している．6) の目的は，SQL 問合せ Q および関係データベース D，問合せの結果 $Q(D)$ に対して，$Q(D) \equiv Q'(D)$ となる SQL 問合せ Q' を求めることである．具体的には，問合せ結果に関連する関係表を結合した巨大な関係表を用意し，その関係表の中で検索結果に出現すると考えられる行の集合 (positive) とそうでない行の集合 (negative) に分ける．Q' は，positive に出現する行を導出するための決定木を構築し，その決定木を SQL 問合せに変換して求める．

検索結果の理解支援として，ユーザに IEQs を提示することは有用である．例えば，昨今のデータベーススキーマは非常に巨大かつ複雑であり，ユーザの問合せにより参照されているデータベーススキーマの一部分と，IEQs により参照されているデータベーススキーマの一部分が異なる場合がある．このようなスキーマにおける異なる経路の発見は，データベーススキーマを深く理解することや，ユーザの問合せを洗練することに役立つと，Tran らは主張している．

†1 京都大学大学院 情報学研究科
Graduate School of Informatics, Kyoto University

Vol.2010-DBS-151 No.29
2010/11/13

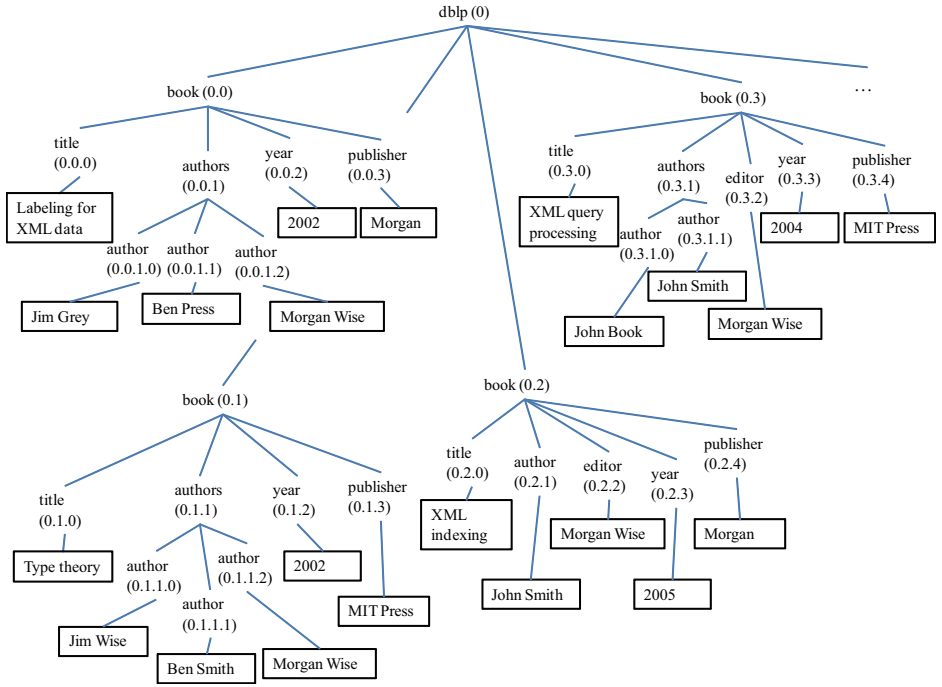


図 1 文献データベースの XML 木.

IEQs は，検索結果を異なる視点から眺めた情報と見なすことができる．それゆえ，XML 検索においても，IEQs をユーザに提示することにより，ユーザは検索結果の隠れた関係を推測することによって検索結果を深く理解することが可能となると考えられる．例えば，図 1 に示した文献データベースに対する，キーワード “book John” の問合せ結果と，キーワード “book Morgan Wise editor” の問合せ結果が等しい場合，Morgan および Wise は必ず John が執筆した本に関わっている．そのため，John と Morgan および John と Wise は何らかの関係があるという推測を行うことが可能となる．

XML キーワード検索結果に対して IEQs を抽出する場合，XML を関係表に落とし込むことができれば Tran らの手法を適用することが可能であると思われるが，XML は多様な構造を持つためそれは困難である．そのため，我々は，XML キーワード検索における IEQs 獲得のためのアルゴリズムを提案する．

一方，XML キーワード検索における問題として，入力として与えたキーワードがユーザの予期せぬテキスト値や要素名とマッチし，検索意図とは異なる結果が得られる場合がある．これは，(1) キーワードが要素名にもテキスト値にもマッチするという曖昧性，(2) キーワードが出現する文脈によって意味が異なるという曖昧性，に起因していると考えられる．例えば，図 1 において，book という単語は要素名としてもテキスト値としても出現していることが (1) に該当し，morgan という単語が author，editor，publisher のテキスト値としてそれぞれ出現していることが (2) に該当する．一般に，検索システムが入力キーワード集合を解釈する際，上記の曖昧性のために複数の解釈が存在するため，検索結果には複数の解釈に基づき検索された結果が混在する．キーワード検索により求まった検索結果に対して代替問合せを求めることは，異なる解釈に基づき実行された検索結果集合に対して代替問合せを求めることとなり，有意な情報は抽出できないと考えらえる．そのため，代替問合せを求める前に，検索結果を解釈ごとに分類する必要がある．

入力キーワードの解釈は，各入力キーワードがどの要素名あるいはどの要素のテキスト値として出現するかによって定まると考えられる．そのため，我々は入力キーワードがどのような形で検索結果に貢献しているかという情報を用いて，検索結果を解釈ごとに分類する．解釈に基づく検索結果の分類に関しては 4 節で詳しく述べる．

本稿の構成は以下の通りである．2 節では関連研究について述べる．3 節では，本稿で用いる概念および定義を説明する．4 節では検索結果を解釈ごとに分類する方法を示す．5 節では代替問合せの取得方法について述べる．6 節ではまとめと今後の課題について述べる．

2. 関 連 研 究

XML キーワード検索の分野では，検索結果として，子孫に入力キーワードを全て含むノードである LCA (Lowest Common Ancestor) を求める手法，LCA の中から特定のノードのみを取得する ELCA (Exclusive LCA)[7]，SLCA (Smallest LCA)[8]，MLCA (Meaningful LCA)[9] といった手法，入力キーワードから返すべきノードを推測する手法[10),12)]，事前に検索結果のオブジェクトとなり得る要素を決めておく方法[11)] などが提案されている．

上述した既存研究においては，キーワードが要素名にもテキスト値にもマッチするという曖昧性，キーワードが出現する文脈によって意味が異なるという曖昧性は考えられていな

Vol.2010-DBS-151 No.29
2010/11/13

い．XReal[15] は，これら二つの曖昧性を考慮した上でユーザが所望していると考えられる SLCA を推定し，ユーザに提示する．しかしながら，ユーザの検索意図を一意に特定できない限り，どの解釈が正しいかを一概に決定することはできない．そのため，我々は，検索結果を解釈ごとに分類し，ユーザに提示した方が望ましいと考えている．

検索結果の理解支援として，Huang ら[1] は，XML 木の要約情報として，XML のスニペットを抽出する方法を提案した．また，Liu ら[2] は，複数のリポジトリから得られ XML 木に対し，Differentiation Feature Sets (DFS) と呼ばれる，検索結果を特徴づけるデータを関連情報として付加する方法を提案した．

関係データベースの分野では，多くの関係表を含む巨大な，かつ複雑なデータベースに対して，構造的な関係を発見する database structure mining[3),4)]，問合せに対してデータベース内のデータのみを答えるのではなく，情報を付加することによって問合せの答えを増補する，intensional query answering[5] が研究されている．Tran ら[6] は，検索結果と同じ結果を返す，別の問合せを発見する手法を提案している．

3. 準　　備

今回議論の対象とする XML 木のモデルは，兄弟間に順序のある，かつラベル付けされた XML 木である．XML 木の各ノード v は XML の要素に一致し，要素名を $\lambda(v)$ と表現する．また，簡単化のため属性ノードは要素ノードとして見なす．XML 木の各ノードは，一意に定まる先行順の ID を持つ．先行研究では，ノード ID として，祖先となるノード ID は子孫となるノード ID の接頭となり，かつ先行順であるという性質を持つ Dewey 数を用いている[7),8)]．それらの研究では，スタックに基づくアルゴリズムと Dewey 数は親和性が高いことが示されているため，我々は，ノード ID として Dewey 数を用いる．

我々は，検索結果を解釈ごとに分類する際に転置リストを利用する．ある単語 k の転置リストは，k を直接含むノードのリストである．Dewey ID によりラベリングされた XML 木の転置リストは，*Dewey Inverted List (DIL)* と呼ばれる．我々は，転置リストの各エントリが，ノードの Dewey ID だけでなく，その親の要素名を含む，拡張した DIL を作成する．図 1 に示す XML 木に対する，拡張した DIL を図 2 に示す．ただし，ここでは単語が要素名として出現した場合は親の要素名として $ 記号を用いている．また，すべての転置リストは Dewey ID によって昇順にソートされている．

二つのノード u，v に対して，$v \prec u$ は v が u の先祖であることを示し，$v \preceq u$ は，v が u の先祖あるいは $v = u$ であることを示す．キーワード集合 $K = \{k_1, ..., k_n\}$ および

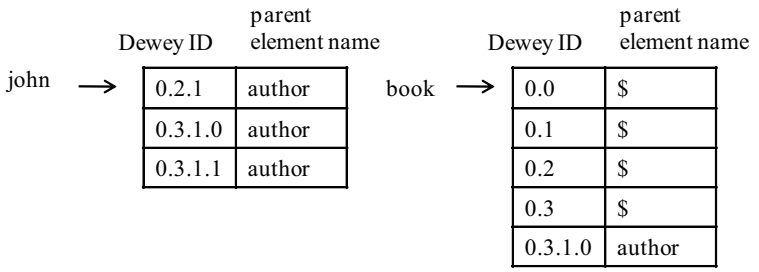


図 2　拡張した Dewey 転置リスト．

XML 木 T が与えられたとき，T における K の *LCA* は，T のノードの中で子孫に入力キーワードの各キーワード k_i が少なくとも一回以上出現するノードの集合である．関数 $lca(v_1, ..., v_k, T)$ は，k 個のノード v_1,…，v_k の LCA を求める．文脈から T が明らかな場合は，T を省略して $lca(v_1, ..., v_k)$ と表す．今，キーワード k_i の転置リストを IL_i とする．このとき，転置リスト IL_1,…，IL_n の LCA は，IL_1 から IL_n までのノードの各組合せの LCA の集合である．

$$lca(IL_1, ..., IL_n) = \{lca(v_1, ..., v_n) | v_1 \in IL_1, ..., v_n \in IL_n\}$$

キーワード集合 $K = \{k_1, ..., k_n\}$ が与えらたとき，ELCA は子孫の中ですでに各キーワード k_i が少なくとも一回以上出現している部分木を除いた後に，子孫に k_i が少なくとも一回以上出現するノードである．T における K の ELCA は，以下で与えられる．

$$elca(k_1, ..., k_n) = elca(IL_1, ..., IL_n) =$$
$$\{v | \exists w_1 \in IL_1, ..., w_n \in IL_n (v = lca(w_1, ..., w_n) \wedge$$
$$\forall i \in [1, n] \not\exists x (x \in lca(IL_1, ..., IL_n) \wedge child(v, w_i) \preceq x))\}$$

ここで，$child(v, w_i)$ は，v から w_i への経路における v の子ノードを表している．また，w_i を v における k_i の *witness* と呼ぶ．ある ELCA に対して，k_i の witness は複数存在する場合がある．そのため，v における k_i の witness の集合を W_i とするとき，v のすべての witness を $v.witness = (W_1, ..., W_n)$ と表現することにする．

例えば，図 1 に示す XML 木に対して，キーワード “book john” の問合せ結果は，LCA

Vol.2010-DBS-151 No.29
2010/11/13

の概念を採用する場合，{0, 0.2, 0.3, 0.3.1, 0.3.1.0} が，ELCA の概念を採用する場合，{0.2, 0.3, 0.3.1.0} が得られる．このとき，ELCA である三つのノード 0.2，0.3，0.3.1.0 のそれぞれにおける book，john の witness は (0.2, 0.2.1)，(0.3, 0.3.1.1)，(0.3.1.0, 0.3.1.0) である．ここで，0.3.1.0 の witness である二つのノードは，それぞれ 0.3 の witness としては出現していないことがわかる．

多くの LCA に基づく概念の中で，ELCA は witness として検索結果に貢献しているノードも含めて取得することが可能であるため，多くユーザにとって望ましい検索結果を取得することができる概念の一つであると考えられている．そのため，本稿では ELCA をルートノードとする部分木を検索結果とする．つまり，XML 木 T および キーワード集合 $K = \{k_1, ..., k_n\}$ が与えられたとき，キーワード検索の結果として，$R(K,T) = \{(subtree(v), v.witness) | v \in elca(k_1, ..., k_n)\}$ を取得する．ただし，$subtree(v)$ は，v をルートノードとする部分木である．

XML 木 T および入力キーワード集合 $K = \{k_1, ..., k_n\}$ が与えられたとき，検索結果 $R(K,T)$ およびある XML 部分木 r とそのすべての witness のペア $(r, (W_1, ..., W_n)) \in R(K,T))$ が得られる．このとき，単語の witness の組合せ $wc = (w_1, ..., w_n)$ $(w_i \in W_i)$ に対して，r における枝抜き部分木 (*pruned subtree*) を，$ps(r, wc) = (N_{ps}, E_{ps}, r_{ps})$ と定義する．ただし，r_{ps} は，r のルートノードであり，$N_{ps} = \{descendant(r_{ps}, w_1) \cup ... \cup descendant(r_{ps}, w_n)\}$，$E_{ps} = \{path(r_{ps}, w_1) \cup ... \cup path(r_{ps}, w_n)\}$ である．ここで，$descendant(r_{ps}, w_i)$，$path(r_{ps}, w_1)$ は，それぞれ r_{ps} から w_i までの経路上に存在するすべてのノードおよびエッジである．図 1 の XML 木に対する，キーワード “smith morgan” の問合せ結果として，図 3 (a)，(d) が得られる．(a) において，morgan の witness として，0.2.2 および 0.2.4 の二つのノードが存在する．それ故，0.2.2，0.2.4 を それぞれ morgan の witness とする枝抜き部分木（図 3(b)，(c)）が得られる．

枝抜き部分木構造は，枝抜き部分木からテキスト値を取り除いた木構造である．図 1 の XML 木に対する，キーワード “smith morgan” の問合せ結果として，図 3 (a)，(d) が得られる．このとき，(d) の枝抜き部分木および枝抜き部分木構造は，それぞれ図 3 (e)，(f) である．

XML 部分木 T_1, T_2 およびそれぞれの枝抜き部分木 $t_1 = (N_1, E_1, r_1)$，$t_2 = (N_2, E_2, r_2)$ が与えられたとき，$\lambda(r_1) = \lambda(r_2)$ かつすべての入力キーワード k に対して，t_1 における k の witness と t_2 における k の witness の要素名が等しいノードが存在するとき，t_1 と t_2 は等しいといい，$t_1 \equiv t_2$ と表す．$t_1 \equiv t_2$ ならば T_1 と T_2 を同一のクラスタに分類する．一

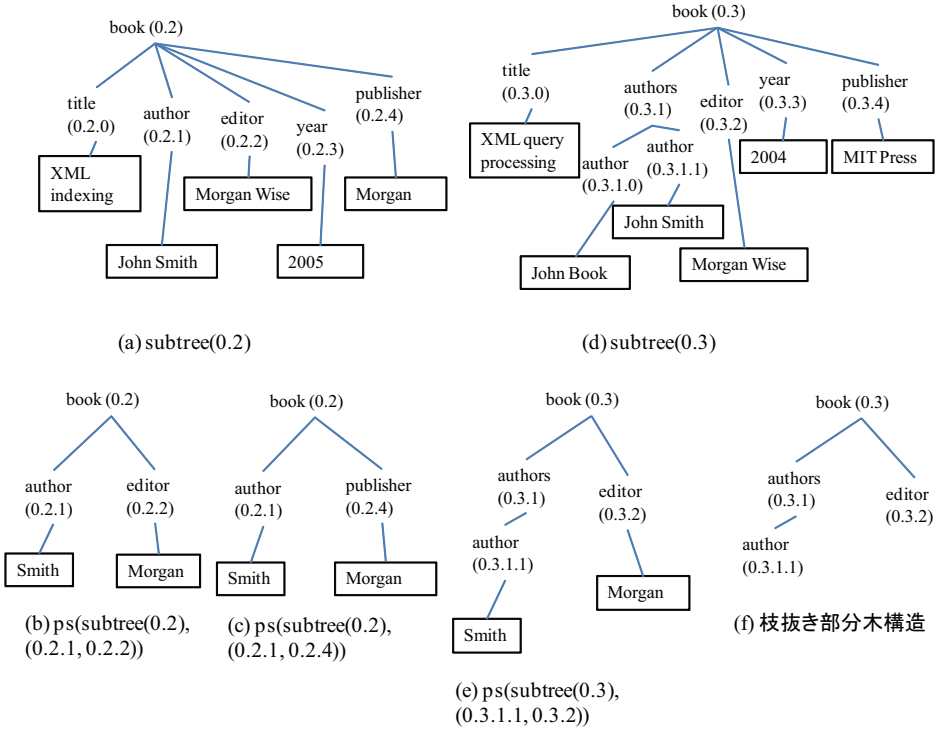


図 3 枝抜き部分木と枝抜き部分木構造.

般に，ある XML 部分木は複数のクラスタに分類され得る．例えば，図 3 において，(b) と (e) は枝抜き部分木が等しいが，(c) と (e) は等しくない．そのため，(a) および (d) を含むクラスタと，(a) のみを含むクラスタが生成される．言い換えると，(a) の部分木は，二つのクラスタに分類されている．

Vol.2010-DBS-151 No.29
2010/11/13

4. 検索結果の取得および曖昧性を考慮した分類

4.1 ELCA を取得するアルゴリズム

ELCA を取得するアルゴリズムとして，XRANK[7]，Indexed Stack (IS) アルゴリズム[13]，Hash Count (HC) アルゴリズム[14] などがある．XRANK は，スタックを用いて処理中の LCA ノードを記憶し，入力キーワードの転置リストを走査して ELCA を求める．IS アルゴリズム，HC アルゴリズムは，転置リストをすべて走査せずに，最初に ELCA の候補ノードを求める．IS アルゴリズムはその後，各候補ノードが witness を保持しているかを検証する．HC アルゴリズムは，予め計算しておいた各単語の出現回数を利用して ELCA であるかを検証する．HC アルゴリズムは他の二手法と比較して高速であるが，witness を求めることができない．我々は，解釈ごとに分類する際に witness の情報を用いるため，XRANK の手法を拡張して解釈ごとに検索結果を分類するが，IS アルゴリズムを採用した場合も，同様にして拡張することが可能である．

4.2 枝抜き部分木を用いた検索結果の分類

検索結果を解釈ごとに分類しようと考えた際，解釈は入力キーワードが出現した位置によって決まると考えられる．例えば，図 1 に示す文献データベースの XML 木に対して，“book john”というキーワードを用いて検索する場合，book というキーワードは，book という要素名，title 要素のテキスト値に出現する単語，author 要素のテキスト値に出現する単語などとマッチしうる．同様に，john というキーワードは，author 要素のテキスト値に出現する単語，editor 要素のテキスト値に出現する単語などとマッチしうる．結果として，先ほどの入力は，book および john という単語を含む auhtor を検索しているという解釈，john が author に含まれる book を検索しているという解釈などが可能である．検索結果を分類する際は，book が要素名として出現し，john が author の名前として出現した結果集合，book および john が author の名前として出現した結果集合，などと分類できる．これは，各入力キーワードがどのような形で検索に寄与しているかによって分類していることと同義である．これらの情報は，ELCA における witness のノードに該当する．それ故，我々は ELCA の witness が出現している構造を抜き出し，その構造を用いて検索結果を分類する．

解釈ごとによる検索結果の分類は，検索結果をクラスタに分類することで行う．クラスタは共通の枝抜き部分木を持つ XML 部分木の集合からなる．また，クラスタは自身に含まれる XML 部分木が共通して保持する枝抜き部分木の情報を保持している．

検索結果を分類する際は，最初に ELCA および witness から枝抜き部分木を抽出する．クラスタ集合の中に抽出した枝抜き部分木と等しい枝抜き部分木を持つクラスタが存在すれば，そのクラスタに抽出した ELCA をルートノードとする部分木を分類し，そうでなければ新たにクラスタを生成し，そのクラスタに抽出した ELCA をルートノードとする部分木を分類する．ELCA および witness が求まることで，検索結果を分類することが可能である．そのため，ELCA および witness を取得する既存手法を拡張することにより，検索結果を解釈ごとに分類することが可能である．

ELCA を求め，その結果をクラスタに分類する方法を例を用いて説明する．今，図 1 に示す XML 木に対して，キーワード集合 {“*book*”, “*john*”} により問い合わせたとする．最初に，ELCA として 0.2 が，book の witness として 0.2 が，john の witness として 0.2.1 が，それぞれ得られる．0.2 をルートノードとする部分木（$subtree(0.2)$）から 0.2 および 0.2.1 に関する枝抜き部分木を抽出すると，図 4(a) に示す枝抜き部分木が得られる．現在，クラスタは作成されていないため，新しくクラスタ C_1 を生成し，$subtree(0.2)$ をそのクラスタへ分類する．次に，ELCA として 0.3.1.0 が，book の witness として 0.3.1.0 が，john の witness として 0.3.1.0 が，それぞれ得られる．$subtree(0.3.1.0)$ から 0.3.1.0 および 0.3.1.0 に関する枝抜き部分木を抽出すると，図 4(b) に示す枝抜き部分木が得られる．この枝抜き部分木と同じ枝抜き部分木を持つクラスタは存在しないため，新しくクラスタ C_2 を生成し，$subtree(0.3.1.0)$ をそのクラスタへ分類する．最後に，ELCA として 0.3 が，book の witness として 0.3 が，john の witness として 0.3.1.1 が，それぞれ得られる．$subtree(0.3)$ から 0.3 および 0.3.1.1 に関する枝抜き部分木を抽出すると，図 4(c) に示す枝抜き部分木が得られる．この枝抜き部分木はクラスタ C_1 の枝抜き部分木と等しいため，$subtree(0.3)$ を C_1 へ分類する．

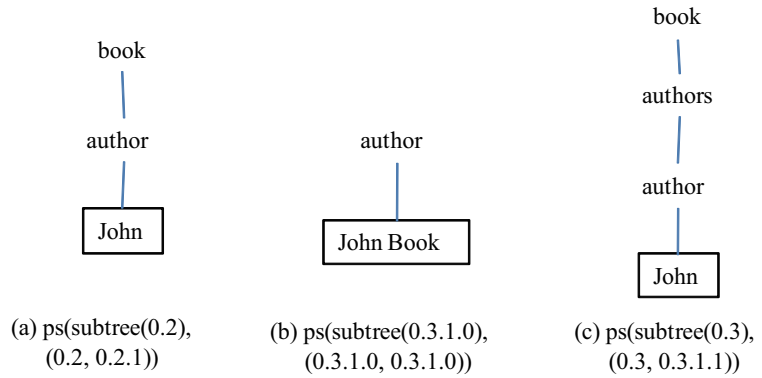
5. 代替問合せの取得

5.1 代替問合せ

我々は，XML キーワード検索において，ある単語集合を用いて問い合わせた結果を解釈ごとに分類したときに得られるクラスタの一つと，入力キーワードを用いて問い合わせた検索結果を分類して得られるクラスタの一つが等しいとき，この単語集合を代替問合せと呼ぶ．代替問合せの定義を以下に示す．

定義 1 (代替問合せ (*Alternative Query*))**.** 入力キーワード集合 K，*XML* 木 T が与えられたとき，検索結果 $R(K,T)$ を $\{c(K,T)_1, ..., c(K,T)_n\}$ と n 個のクラスタに分類する．ク

Vol.2010-DBS-151 No.29
2010/11/13



(a) ps(subtree(0.2), (0.2, 0.2.1))　(b) ps(subtree(0.3.1.0), (0.3.1.0, 0.3.1.0))　(c) ps(subtree(0.3), (0.3, 0.3.1.1))

図 4　三つの検索結果における枝抜き部分木.

ラスタ集合の中から一つのクラスタ $c(K,T)_i$ を選択したとき，そのクラスタに共通して出現する単語集合を K_p とする．K_p のある部分単語集合 S に対して，$\exists j | c(K,T)_j = c(S,T)_i$ となる $R(S,T) = \{c(S,T)_1, ..., c(S,T)_m\}$ を満たす S を代替問合せとする．

5.2　代替問合せ抽出の概要

解釈ごとに分類されたクラスタの中から，あるクラスタに含まれるインスタンスの集合に対する代替問合せを求めることを考える．今，代替問合せを求めるべきクラスタを positive と呼ぶことにする．ある単語集合が代替問合せであるか否かは，positive の各インスタンスは単語集合のすべての単語を含み，positive に含まれないインスタンスの集合（便宜上，negative と呼ぶ）には単語集合のすべての単語を含むインスタンスが存在しなければよい．negative に含まれるインスタンスの数は膨大であるが，その中で，positive に含まれるインスタンスとは同じ枝抜き部分木構造を持つが，同じ枝抜き部分木は持たないインスタンスが存在する．これらは，positive と同じ解釈の下で実行されうるが，positive には含まれないインスタンスであるので，negative としてふさわしいと考える．代替問合せを抽出する際は，positive と同一の枝抜き部分木構造を持つが，同一の枝抜き部分木を持たないインスタンスの集合を negative とし，positive に出現するインスタンスから共通して出現する単語集合を抽出し，各単語集合が negative の各インスタンスに出現しないことを検証すればよい．

しかしながら，上記の方法では，代替問合せの候補となる単語集合に対して，代替問合せであるかを確認するために，negative のすべてのインスタンスを走査する必要がある．一般に，negative となるインスタンスは非常に多いと考えられるため，単語集合ごとに代替問合せであるかを確認することは効率が悪い．一方で，negative のインスタンス中にある単語集合が出現すれば，そのインスタンスが反例となり，単語集合が代替問合せでないことを示すことが可能である．

また，negative の中には，反例となりうるインスタンスと，反例とはならないインスタンスが存在する．前者は，positive の各インスタンスから抽出した，共通して出現する単語を少なくとも一つ以上含むようなインスタンスであり，後者は一つも含まないインスタンスである．

それゆえ，我々は各単語集合が代替問合せであるかを検証するのではなく，反例となりうるインスタンスの集合から反例となるインスタンスを発見する度に，そのインスタンスに含まれる単語集合を記憶しておき，最終的に得られた単語集合の集合に含まれない単語集合を代替問合せとするアプローチを取った．

5.3　代替問合せ抽出手法

代替問合せの抽出は，処理中のノードをスタックの形で保持する．スタックの各エントリは，Dewey ID，*structures*，*commons* からなる．*structures* は positive のインスタンスに共通して出現する枝抜き部分木構造に出現する要素名の数に相当する長さの配列であり，配列の要素には ID 値 の集合を格納する．*commons* は positive のインスタンスに共通して出現する単語と，その親要素名のペアの数に相当する二値の配列である．*structures* は，witness ノードの ID を記憶する．*commons* は，単語とその親要素名のペアが直截的に，あるいは間接的に出現した場合に 1 を格納する．以降では，特に断らない限り，*structures* を s，*commons* を c と省略して表記する．アルゴリズム 1 では，s が空でなく，positive のインスタンスと同じ枝抜き部分木構造を持っていれば，negative のインスタンスと判定する．その際に，代替問合せでない単語の組合せを記録していく．

代替問合せを求めるアルゴリズムをアルゴリズム 1 に示し，それを例を用いて説明する．ただし，アルゴリズム 1 の中では可読性を考慮し，s を *structures*，c を *commons* と表記している．今，図 1 に示す XML 木に対して，“book, john” という入力キーワードにより問い合わせることを考える．4.2 節に示した分類手法を用いると，分類された検索結果として，{{(subtree(0.2), (0.2, 0.2.1)), (subtree(0.3), (0.3, 0.3.1.1))}, {(subtree(0.3.1.0),

Vol.2010-DBS-151 No.29
2010/11/13

Algorithm 1 Stack algorithm for extracting alternative queries.

Input: a cluster Cl
Input: a pruned subtree structure pss that all subtrees in Cl have
Output: non-alternative query candidate set NAQ

1: $K_p = getAllElementNames(pss)$
2: $C = getCommonWordsWithParentName(Cl)$
3: get all keyword inverted lists with regarded to K_p and C
4: $stack = empty$
5: **while** has not reached the end of all keyword inverted lists in K_p **do**
6: $v = getSmallestNode(K_p, C)$
7: $p = lca(stack, v)$
8: **while** $stack.size > p$ **do**
9: $entry = stack.pop()$
10: **if** $isSameStructure(entry.structures, pss)$ **then**
11: $NAQ.add(entry.commons())$
12: **else**
13: **if** $entry.structures[j] = true$ for $1 \leq j \leq K_p.size()$ **then**
14: $stack.top.structures[j].add(entry.structures[j])$
15: **end if**
16: **if** $entry.commons[j] = true$ for $1 \leq j \leq C.size()$ **then**
17: $stack.top.commons[j] = entry.commons[j]$
18: **end if**
19: **end if**
20: **for** $p < j \leq v.length$ **do**
21: $stack.push(v[j], [])$
22: **end for**
23: **if** $v \in K_p$ **then**
24: $stak.top.structrues[i].add(v.id)$
25: **else if** $v \in C \&\& v.parentName = C.get(v.value()).parentName()$ **then**
26: $stak.top.commons[i] = true$
27: **end if**
28: **end while**
29: **end while**
30: check entries of the stack and add NAQ to $stack.top.commons$ if any elca exists

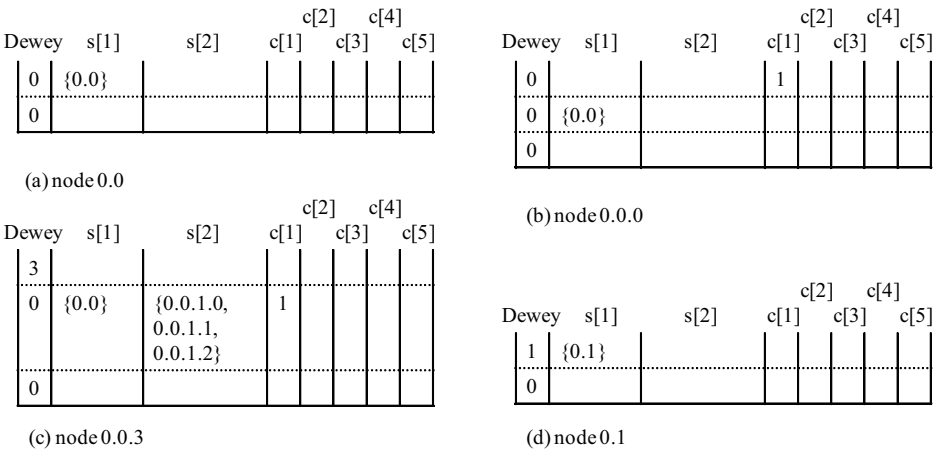


図 5 スタックの状態図.

(0.3.1.0, 0.3.1.0)}} が得られる．ここで，ユーザが自身の検索意図に合致する結果として，Cl = {(subtree(0.2), (0.2, 0.2.1)), (subtree(0.3), (0.3, 0.3.1.1))} を選択したとする．クラスタの枝抜き部分木から，要素名として出現しているすべての単語を抽出する（1 行目）と，K_p = {“*book*”, “*author*”} が得られる．また，Cl の各インスタンスのテキスト値に共通して出現する単語およびその親要素名のペアを抽出すると，C = {(xml, title), (john, author), (smith, author), (morgan, editor), (wise, editor)} が得られる（2 行目）．ここでは，s = [“book”, “author”]，c = [(xml, title), (john, author), (smith, author), (morgan, editor), (wise, editor)] である．図 5 (a) において，スタックの一番上のエントリは，book の witness が 0.0 に出現していることを示している．次いで，K_p の各単語および C の各ペアの単語に対する転置リストを取得する（3 行目）．各単語の転置リストは，図 2 に示す Dewey ID の列が該当する．全転置リストの中から ID が一番小さいノードを取得すると，0.0 が得られる（6 行目）．0.0 を処理した状態を，図 5 (a) に示す．同様にして，0.0.0 (xml) を処理した状態を図 5 (b)，0.0.1.0 - 0.0.1.2 (author)，0.0.1.2 (morgan)，0.0.1.2 (wise)，0.0.3 (morgan) を処理した状態を図 5 (c) に示す．ここで，0.0.3 を処理するとき，0.0.3 は

morgan という単語が出現するノードを表しているが，0.0.3 の親要素名は publisher，スタックの c[4] の親要素名は author であり異なるため（25 行目），スタックの一番上のエントリにおける，c[4] は空値のままである．次に小さい ID である 0.1 を取り出したとする（6 行目）．このとき，スタックと 0.1 に共通する最長の接頭文字は，0 である（7 行目）．そのため，スタックが表すノードが 0 になるまでポップする．0.0 をポップするとき，スタックの s はすべて空集合ではないため，0.0 をルートノードとする部分木の枝抜き部分木構造が，Cl に含まれる枝抜き部分木構造 pss と等しいか検証する（10 行目）．0.0 をルートノードとする部分木の枝抜き部分木構造 は pss の枝抜き部分木構造と等しく，かつ 0.0 は検索結果ではないため，0.0 をルートノードとする部分木中に出現する，{XML} という単語の組合せは代替問合せとはならない．最終的に，$NAQ = \{\{\text{XML}\}, \{\text{smith, wise}\}\}$ が得られる．NAQ の各要素および要素の部分集合は代替問合せとはらならい．つまり，{smith, wise} は代替問合せではなく，その部分集合である {smith} および {wise} も代替問合せではない．代替問合せでない単語集合が代替問合せとなるので，たとえば {XML, john}, {XML, morgan} といった単語集合が代替問合せとなる．

6. おわりに

我々は，XML キーワード検索の解釈の多様性から生じる，解釈の異なる検索結果に対して，枝抜き部分木を用いて検索結果を取得しつつ解釈ごとに分類する方法を示した．分類された結果は，ユーザの検索意図に合致した解を効率的に発見することを支援することが可能であると考えられる．また，解釈ごとに分類した各検索結果に対し，ユーザが入力した問合せ語とは異なるが，同じ検索結果を返す問合せ語を提示する方法を示した．

本稿では，入力キーワードにより求まる検索結果と完全に同じ結果を得る代替問合せを議論の対象とした．しかし，検索対象のデータや入力キーワードによっては，(1) 検索結果に共通して出現する単語が存在しない，(2) 代替問合せとなる単語集合が存在しない，ということが生じうる．理解支援という観点からは，結果をユーザに返すことができないのは望ましくない．それゆえ，完全に同じ結果を得る代替問合せを求めるのみではなく，ある閾値を設けて，代替問合せとして閾値以上の割合の正しい結果が得られる単語集合を代替問合せとし，これをユーザに提示することが必要になる場合もあると考えている．

今後は，提案手法の性能を評価するために，代替問合せを取得するためのオーバーヘッドを測定することが考えられる．また，類似した結果を得る代替問合せを取得する方法への拡張を考えていく．

参 考 文 献

1) Huang, Y., Liu, Z. and Chen, Y.: Query biased snippet generation in XML search, *SIGMOD Conference*, pp.315–326 (2008).
2) Liu, Z., Sun, P. and Chen, Y.: Structured Search Result Differentiation, *PVLDB*, Vol.2, No.1, pp.313–324 (2009).
3) Andritsos, P., Miller, R.J. and Tsaparas, P.: Information-Theoretic Tools for Mining Database Structure from Large Data Sets, *SIGMOD Conference*, pp.731–742 (2004).
4) Wu, W., Reinwald, B., Sismanis, Y. and Manjrekar, R.: Discovering topical structures of databases, *SIGMOD Conference*, pp.1019–1030 (2008).
5) Motro, A.: Intensional Answers to Database Queries, *IEEE Trans. Knowl. Data Eng.*, Vol.6, No.3, pp.444–454 (1994).
6) Tran, Q.T., Chan, C.-Y. and Parthasarathy, S.: Query by output, *SIGMOD Conference*, pp.535–548 (2009).
7) Guo, L., Shao, F., Botev, C. and Shanmugasundaram, J.: XRANK: Ranked Keyword Search over XML Documents, *SIGMOD Conference*, pp.16–27 (2003).
8) Xu, Y. and Papakonstantinou, Y.: Efficient Keyword Search for Smallest LCAs in XML Databases, *SIGMOD Conference*, pp.527–538 (2005).
9) Li, Y., Yu, C. and Jagadish, H.V.: Schema-Free XQuery, *VLDB*, pp.72–83 (2004).
10) Liu, Z. and Chen, Y.: Identifying meaningful return information for XML keyword search, *SIGMOD Conference*, pp.329–340 (2007).
11) Liu, Z., Walker, J. and Chen, Y.: XSeek: A Semantic XML Search Engine Using Keywords, *VLDB*, pp.1330–1333 (2007).
12) Supasitthimethee, U., Shimizu, T., Yoshikawa, M. and Porkaew, K.: XSemantic: An Extension of LCA Based XML Semantic Search, *IEICE Transactions*, Vol.92-D, No.5, pp.1079–1092 (2009).
13) Xu, Y. and Papakonstantinou, Y.: Efficient LCA based keyword search in XML data, *EDBT*, pp.535–546 (2008).
14) Zhou, R., Liu, C. and Li, J.: Fast ELCA computation for keyword queries on XML data, *EDBT*, pp.549–560 (2010).
15) Bao, Z., Ling, T.W., Chen, B. and Lu, J.: Effective XML Keyword Search with Relevance Oriented Ranking, *ICDE*, pp.517–528 (2009).