情報処理学会論文誌 Vol. 49 No. 9 3280–3290 (Sep. 2008)

推薦論文

XML 文書に対する構造を指定した全文検索方式の提案

兵藤正樹[†1] 榎本俊文[†1]
江田毅晴[†1] 山室雅司[†1]

近年，蓄積された XML 文書に対して柔軟に構造を指定できる全文検索システムが必要とされている．本論文は，XML 文書に対する構造を指定した全文検索方式を提案する．提案手法は，指定された構造の部分文書検索にはストロング DATAGUIDE および範囲ラベルを用い，全文検索にはテキストインデックスを用いる．提案手法では，テキストインデックスを拡張して dg ノード ID と範囲ラベルを付与することで不必要な構造ジョインを回避している．これにより，XML 文書から検索ワードを含む XML 部分文書を柔軟かつ高速に検索することができる．本論文では，提案手法を，柔軟な検索を犠牲にした高速な部分インデックス手法と，検索単語とマッチした全テキストノードをそのまま構造ジョインする単純組合せ手法と比較して評価を行った．評価実験では，部分インデックス手法より 9%以下の速度低下で柔軟な検索を実現し，単純組合せ手法より最大 1 桁以上の高速化を確認できた．

A Proposal of a Full-text Search Method for XML Documents Using Their Structures

Masaki Hyodo,[†1] Toshifumi Enomoto,[†1]
Tsuneharu Eda[†1] and Masashi Yamamuro[†1]

Recently, a full text search system for stored XML documents using their structures flexibly is needed. In this paper, we propose a full text search method for XML documents using their structures. Our approach uses strong DATAGUIDE and range label for searching partial XML documents, and text index for a full text search. Our approach avoids structural join by giving a dg node and a range label to the text index. Our approach can search the partial XML documents including the search word flexibly and at high speed. In this paper, we compare our method with a partial index method and a naive method. Our experimental evaluation shows that the proposed technique has flexibility and efficiency.

1. はじめに

XML（Extensible Markup Language）は，1998 年に World Wide Web Consortium でデータ交換のためのフォーマットとして標準化されて以来，様々なアプリケーションで用いられるようになってきている．そのため，計算機で用いられる様々なデータが XML で記述されるようになってきた．このような背景から，XML で記述され，蓄積されたデータを効率良く検索する技術の研究は急務といえる．現在，XML データの検索には，XPath や XQuery などの構造を指定可能な問合せ言語が広く用いられており，さかんに研究が行われている．

XML データには文書指向（document-centric）のものと，データ指向（data-centric）のものがある．文書指向の XML データは，データ指向の XML データに比べ，各テキストノードに多くの平文で記述されたテキストを含む傾向にある．ここでは文書指向の XML データを XML 文書と呼ぶ．XML 文書に対して検索を行う場合，全文検索の手法を適用することが有効であると考えられる．全文検索とは，本論文では，単語による文書サーチのことで，単語を入力することでその単語を含む文書群を出力することであるとする．これまでに提案されている手法は，基本的に XML 文書のすべてのテキストノードを対象にし，文書単位の全文検索を行うというものだった[1),2)]．

しかし，文書指向といえども XML は構造化データである．そのため，構造を利用し，検索する対象の文書の範囲を限定して全文検索を行う手法が求められている．そのように文書が限定される範囲を本論文では部分文書と呼ぶ．文献 3) では利用者が目的とする部分文書を自動的に選択する研究がなされており，文献 4) では検索結果の部分文書群からユーザが選択するインタフェースが提案されている．

本論文では，ユーザが構造を指定して，部分文書単位での全文検索を行う問題に取り組んでいく．たとえば，本のデータを構造化し，章ごと，節ごとにテキストをマークアップする．これにより，ユーザがある単語で文書の検索をするとき，必要とする部分だけを検索することができる．つまり，検索単語を含む文書をより局所的に検索する場合には節単位で，ある程

†1 日本電信電話株式会社 NTT サイバースペース研究所
NTT Cyber Space Laboratories, NTT Corporation
本論文の内容は 2007 年 7 月のマルチメディア，分散，協調とモバイル（DICOMO2007）シンポジウムにて報告され，DPS 研究会主査により情報処理学会論文誌ジャーナルへの掲載が推薦された論文である．

度広く探索する場合には章単位で，といった検索が考えられる．この場合，節は章に含まれるため，章単位で構造を指定すると，章ノード以下の節のテキストノードもすべて検索される．

部分文書単位での検索を実現する場合，ナイーブな手法として 2 つの手法が考えられる．まず 1 つは，部分文書に対して全文検索のテキストインデックスを作成することが考えられる．ただしその場合，部分文書の総数が非常に大きいことおよび多数のテキストノードが重複することにより，非常に膨大なテキストインデックス群の作成を必要とし，巨大なディスク容量を必要としてしまう．2 つ目に，文書全体に対して単一のテキストインデックスを作成することが考えられる．ただしその場合，検索ワードを含む多数のテキストノードから部分文書に含まれるテキストノードを別途判別する必要があり，検索速度の低下が懸念される．

本論文では，XML 文書に対して構造検索を効率良く実現する手法であるストロング DATAGUIDE および範囲ラベルを利用した全文検索方式を提案する．提案手法では，テキストインデックスに XML 構造の要約情報（ストロング DATAGUIDE）およびノード情報（範囲ラベル）を持たせることでテキストインデックスを拡張し，あらかじめ構築をしておく．検索時は，拡張インデックスに含まれるストロング DATAGUIDE の情報を用いて，検索ワードを含みかつ部分文書に含まれるテキストノードを取り出す．

本方式を用いることで巨大なテキストインデックスを作成することなく柔軟に構造を指定した全文検索を高速に行うことができる．

本論文の構成は以下のとおりである．2 章では部分文書を対象とした全文検索について述べる．3 章では 2 章で述べたインデックスの持つ問題を解決する手法を提案する．4 章では提案手法を評価するために行った実験とその結果について述べる．5 章でまとめと今後の課題を述べる．

2. 部分文書を対象とした全文検索

2.1 検索対象となる部分文書

XML 文書は構造化データであり，文書の記述内容の分析とは別に，構造を利用して文書の内容を分析することができる．そのため，パスを指定することで大きな 1 つの XML 文書から，特定のノードを対象に検索することができる．これらの XML 文書の一部分に対して全文検索を行う場合，指定されたパスの該当ノード以下のすべてのテキストノードを対象に検索することが考えられ，このような検索が今後必要となる．

具体的な XML 文書例を用いて説明する．**図 1** に XML 文書と XML 構造をそれぞれ示す．図 1 の XML 文書 001 は本のデータを表している．XML 文書 001 は/book/chapter 配下に章の記述内容が各章別に格納されている．章の中にはさらに小さな区分である節がある．たとえば，「地球」について章ごとに調査したい場合，/book/chapter ノードの下位にあるすべてのテキストノードを対象として「地球」が含まれているかどうか検索を行う．また，節ごとに調査したい場合，/book/chapter/section ノードの下位にあるすべてのテキストノードを対象に検索を行う．/book/chapter で指定される部分文書と/book/chapter/section で指定される部分文書の違いを**図 2** に示す．

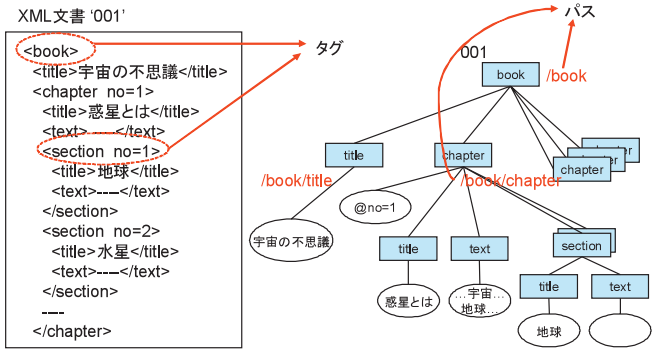

図 1　XML 文書と XML 文書の構造
Fig. 1　An XML document and its structure.

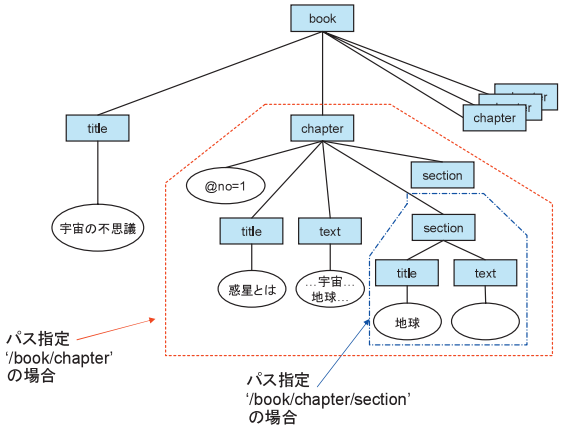

図 2　パスによる部分文書の指定
Fig. 2　Specifying partial documents with paths.

2.2　全文検索の実現と課題

蓄積された XML 文書群から，部分文書を全文検索したい場合，検索パスと検索ワードの 2 つの条件が必要になる．検索パスが検索対象となる部分文書を決定し，検索ワードが各部分文書のテキストノード群に出現しているかを調べる．たとえば，前述の例でいえば，検索パスは/book/chapter や/book/chapter/section であり，検索ワードは「地球」である．

検索ワードを探すには，全文検索手法を用いて行うことが有効である．今回は，大量の蓄積文書を検索対象とすることから，あらかじめテキストインデックスを構築しておいて検索パフォーマンスを向上させる手法を用いることとする．ただし，その対象が部分文書となるため，単純に適用した場合，たとえば/book/chapter 以下のノードすべて，/book/chapter/section 以下のノードすべて，……といったパスで指定された部分文書群に対してテキストインデックスを構築することになる．しかし，部分文書間には，ノードの包含関係があるものがあり，インデックス構築対象となるテキストノードに重複が起こる．そのため，すべての部分文書に対してテキストインデックスを構築した場合，インデックスの容量が巨大になり，またその構築・更新にかかるコストが膨大になるという問題がある．この問題はインデックスを構築する部分文書群をあらかじめ指定されたパス配下に限定することで軽減することは可能だが，インデックスを張った部分文書しか高速な検索が行えず，任意の XML 部分文書に対する全文検索に対応できないという新たな問題が発生する．

3. 提 案 手 法

前述した問題から，全文検索のためのテキストインデックスを拡張して構築し，任意の部分文書に対して柔軟に全文検索を行うことができる方法を検討する．

ここで課題となるのは，従来のテキストインデックスを用いた全文検索では，文書に単語が含まれているかどうか判定できれば十分であったのに対し，構造を指定した部分文書の検索においては，検索クエリの検索パスによって決定される部分文書に単語が含まれているかどうかを判定できなくてはならないことである．検索パスは検索クエリによって任意に指定されるため，単語の出現するテキストノードを文書単位ではなく検索対象の最小単位であるノード単位で把握しなくてはならない．

さらに，ノード単位で単語の出現位置を把握したとして，該当する部分文書に含まれるかどうかのチェックを検索実行時に行う必要がある．この処理のコストを低減させることも課題となる．

そこで，我々は提案済みの XPath 処理方式[5] を拡張し，全文検索条件の制約つきシングルパスの処理方法を提案することで，上記の課題解決を行う．

本手法では XPath で記述された検索クエリを対象とする．ロケーションパスを用いて検索対象の構造を指定し，述語を用いてマッチングの条件を指定することで，条件にあった蓄積された部分文書を検索する．また，すべての XML 文書を，共通のルートを持つ大きな 1 つの XML 文書として扱い，XPath で記述された検索クエリは共通ルートでまとめられた全文書群を対象にマッチングを行う．

部分文書の全文検索に関しては以下のような関数を新設し XPath に組み込む．

/book/chapter/section[ftscontains (. , '地球')]

新設した「ftscontains()」関数は，第 1 引数に指定された検索パスが決定する部分文書群に対し，第 2 引数に指定された検索ワードが含まれる場合 *true* を返却する関数と定義する．つまり，この場合第 1 引数が「.」（カレントノード）であるので，節（section）単位の部分文書で検索ワード「地球」を含むものを探索することになる．第 1 引数に相対パスを指定することで全文検索対象を部分文書内の特定のノードを指定するように検索クエリを記述することが可能である．たとえば，

/book/chapter[ftscontains (./section/title, '地球')]

上記のクエリを用いることで「節見出しに '地球' を含む章単位の部分文書」のような複雑な条件の部分文書を検索することもできる．また，第 2 引数には複数の検索単語を AND や OR を用いて記述することが可能である[⋆1]．

本章では，3.1 節で提案方式の基礎技術となるストロング DATAGUIDE の概要を，3.2 節で同じく基礎技術である範囲ラベルについて紹介する．3.3 節では提案済みの XPath 処理方式の概要と本論文で提案する全文検索を適用するための拡張方法について説明し，3.4 節で新たな方式である部分文書に対する全文検索方式を提案する．

3.1　ストロング DATAGUIDE

ストロング DATAGUIDE[6] とは，XML 木に出現するすべてのパスから，共通のパスを集約し，木構造表現したものである．**図 3** に XML 木と XML 木から構築されたストロング DATAGUIDE を示す．一般に，XML 木に対し，ストロング DATAGUIDE は非常に小さくなることが分かっており，ストロング DATAGUIDE 中のノード（以降，dg ノードと呼ぶ）は，XML 木ノードに対し，1 対多の対応関係にある．

⋆1 2 つ以上の検索単語を条件として記述した場合，内部では単一検索単語で全文検索条件指定されたシングルパス（後述）群に分解される．

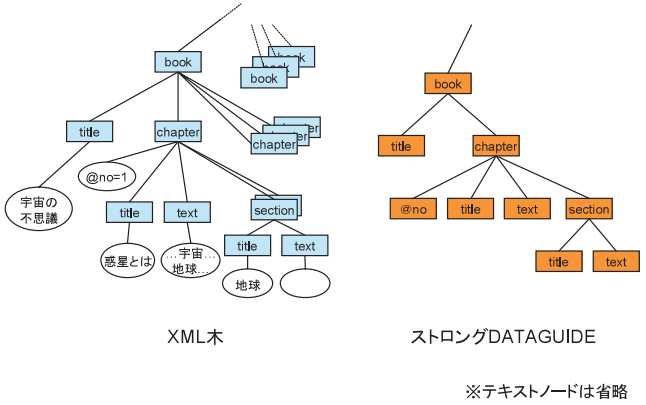

図 3　XML 木とストロング DATAGUIDE
Fig. 3　An XML tree and its strong DATAGUIDE.

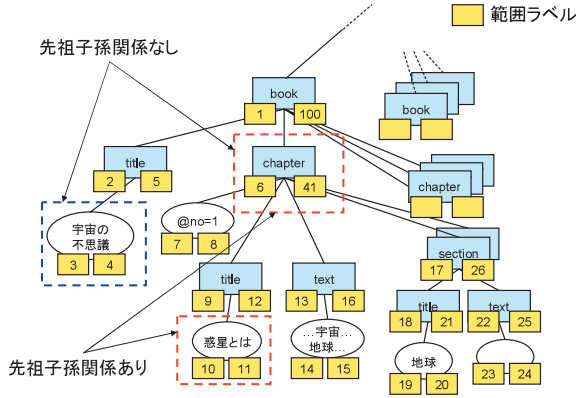

図 4　範囲ラベル
Fig. 4　A Range Label.

3.2　範囲ラベル

範囲ラベルは XML データにおけるノード間の先祖子孫関係を保持する．範囲ラベルは preorder と postorder の 2 値からなり，子ノードの preorder 値と postorder 値は親ノードの preorder 値と postorder 値の範囲に含まれるようにラベル付けされる．範囲ラベルの例を**図 4** に示す．範囲ラベル $(10, 11)$ のテキストノードは，範囲ラベル $(6, 41)$ の ‘chapter’ の中間ノードの範囲に含まれるのでこれらのノードは先祖子孫関係にあり，‘chapter’ の中間ノードが先祖，テキストノードが子孫であることが分かる．このように，範囲ラベルを用いることで離れたノードの先祖子孫関係を高速に判別することができる．

3.3　XPath 処理方式の概要と拡張方式

文献 5) の XPath 処理方式は，ストロング DATAGUIDE を利用して構造ジョインを削減し，指定された構造の XML ノード列（ノードの集合）を高速に検索するものである．XPath によって指定された構造で絞り込み，指定された部分文書の範囲に含まれる XML ノード群を特定することができる．

図 5 に XPath 処理方式のアルゴリズムを示す．これは文献 5) の XPath 処理方式アルゴリズムを部分文書の全文検索が可能となるよう拡張したものである．拡張部分は 6 行目の条件分岐と 9 行目の処理を追加したところである．

処理の流れを説明する．まずクエリをシングルパスの列に分解する（4 行目）．たとえば XPath

/book/chapter/section[ftscontains (. , ’地球’)]

の場合，以下の 2 つのシングルパスに分解される．

a)　/book/chapter/section
b)　/book/chapter/section (制約: 全文検索 ’地球’)

そして，それぞれのシングルパスに対して処理を行う（5〜11 行目）．その際，シングルパスが制約を持つかどうか判定し（6 行目），全文検索の制約を持たない場合は従来の XPath 処理手法[5] である SingleProcess を（7 行目），そうでない場合は提案手法の全文検索処理手法である FtsProcess で処理を行う（9 行目）．最後に，それぞれのシングルパスの処理で得たノード列を，範囲ラベルを用いた構造ジョイン[7] を行うことで最終結果となるノード列を取得する（12 行目）．

従来のストロング DATAGUIDE を利用した構造検索処理である SingleProcess を**図 6** に示し，以下に処理の概要を述べる．

ストロング DATAGUIDE はラベル付き木であるので，それを XML 木と見なして XPath 処理を行うことができる．この処理を dgEval と呼ぶ．与えられたシングルパスを用い dgEval を行うことで，ストロング DATAGUIDE からマッチする部分木すべてを探し出す（1 行目）．次に，マッチする部分木に対し，葉ノードとなる dg ノードを取り出す（3 行目 getOut）．そのうえで，dg ノードに対応する XML 木のノード列を取得し（3 行目 select），それらを

```
input: XML 木 T，問合せ q
output: 結果 XML ノード列{vi}
1 if (XML 木が再帰構造を持つ)
2    return normalProcess(T, q)
3 else
4    シングルパス列{ (p1,p2,...,pm) = decompose(q)
5      for(i = 1; i < m+1; i++)
6        if (pi が全文検索の制約を持たない)
7          xml ノード列 vi= SingleProcess(T, pi)
8        else
9          xml ノード列 vi= FtsProcess(T, pi, Index)
10       endif
11     endfor
12   return getOutSeq(brunchJoin(q, (v1, v2,...vm)))
13 endif
method
    normalProcess(XML 木 T, 問合せ q)
      T に対して問合せ q をシングルパス列に分解する．
    decompose(問合せ q)
      問合せ q をシングルパス列に分解する．
    branchJoin(問合せ q, xml ノード列{vi})
      問い合わせ中の分岐ノードと葉ノード間で構造ジョイ
      ンを行い xml マッチ木列を返す．
    getOutSeq(xml マッチ木列 t)
      xml マッチ木列 t のアウトプット xml ノード列を返す．
```

図 5　XPath 処理アルゴリズム
Fig. 5　XPath processing algorithm.

合わせたものが結果となる（5 行目）．つまり，XML 木と比較し非常に小さいストロング DATAGUIDE に対し XPath 処理を行っておくことで，処理の効率化を図っている．

3.4　全文検索方法

本節では部分文書の全文検索を行う方法としてテキストインデックスの拡張方法と拡張したインデックスを利用した検索方法について述べる．

```
Algorithm: SingleProcess
Input: XML 木　T，問い合わせ q
Output: 結果 XML ノード列{vi}
1 (t1, t2,…, tm) = dgEval(T, q)
2 for(i = 0; i<m+1; i++)
3    xml ノード列 xi = select(T, getOut(ti))
4 endfor
5 return merge((x1, x2,…, xm))
method:
    getOut（マッチ木 ti）
      マッチ木 t のアウトプット dg ノードを返す
    select（XML 木 T,　dg ノード o）
      根から dg ノード o までのタグの連なったパスを持つ
      XML ノード列を返す．
    merge(xml ノードリスト列(x1, x2,…, xm))
      すべての xml ノードリストをマージした結果を返す．
```

図 6　シングルパス処理アルゴリズム
Fig. 6　Single path processing algorithm.

全文検索を高速に処理するためにはテキストインデックスを用いることが一般的である．部分文書の全文検索を行うことを考えた場合，検索単語出現ノードの特定が必要となる．そこで，ノード情報（範囲ラベル）をインデックスに格納する．通常テキストインデックスは単語と，単語の出現位置として文書情報を持っているが，これを拡張し，出現するノード情報を格納することで単語の出現する XML 文書の詳細な位置（ノード）が分かる．

しかし，上記のテキストインデックスを用いて全文検索を行うと，テキストインデックスから求められる中間結果ノードに検索パスに適合しないノードが大量に含まれる場合があり，検索パスとのマッチング処理のコストが大きくなってしまうことが懸念される．そこで，テキストインデックスをさらに拡張し，単語出現ノードに対応する dg ノード情報を持たせ，全文検索においても，ストロング DATAGUIDE を用い，全文書の出現位置から検索パスに適合するノードだけに絞り込むことで，任意の部分文書に対する検索を効率的に行う．

以下，具体的な処理の流れを説明する．ノード情報と dg ノード情報を格納できるよう拡張構築したテキストインデックスに対し，**図 7** のアルゴリズム（FtsProcess）により処理

```
Algorithm: FtsProcess
Input: XML 木　T,　全文検索制約付きシングルパス　q
Output:　結果 XML ノード列{vi}
1 検索ワード　w = getWord(q)
2 検索対象パス　f = getPath(q)
3 (y1,y2,...ym) =select( dgEval(T, f))
4 ((x1, x2, ..., xm), (p1, p2, ..., pm)) = matchingWord(w)
5 for(i = 1,　i < m+1;　i++)
6　　if(pi が(y1,y2,...ym)のいずれかにマッチしていたら)
7　　{ vi}　add　xi
8　　endif
9 endfor
10 return merge({vi})
method:
　getWord(問合せ q)
　　検索ワードを返す
　getPath(問合せ q)
　　全文検索対象パスを返す.具体的には, 検索パス下位にある
　　全てのテキストノードを対象とするため,　検索パスに
　　'//text()' を追加したパスを返却する.
　matchingWord(検索ワード w)
　　テキストインデックスを用い, 検索ワードを含むテキスト
　　ノードと対応する dg ノードを返す
```

図 7　Fts 処理アルゴリズム
Fig. 7　Fts processing algorithm.

を行う．FtsProcess は，クエリを分解して得られたシングルパスが全文検索の制約を持つ場合に用いられ，前述した SingleProcess とは異なり，検索ワードを含みかつ検索パス配下にあるテキストノード列を返す．

まず，全文検索の制約から検索ワードを得て（1 行目），検索パスからその下位のテキストノードすべてを表す検索対象パスを得る（2 行目）．前述の XPath 例のシングルパス b) の場合，

検索ワード w = 地球

検索対象パス f = /book/chapter/section//text()

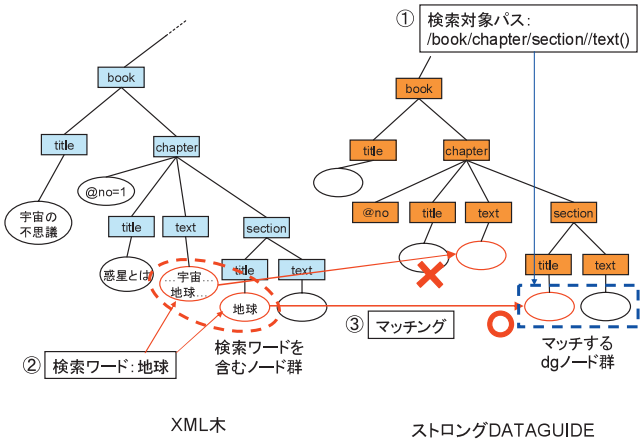

図 8　FtsProcess の流れ
Fig. 8　A flow of FtsProcess.

となる．次に，検索対象パスからはマッチする dg ノード列を取得（3 行目）し，検索ワードからはテキストインデックスを用いてその出現位置であるノード列と dg ノード列を取得する（4 行目）．そして，取得したそれぞれの出現位置の dg ノードを検索対象パスにマッチした dg ノード列を比較し（6 行目），含まれていれば返却ノード列へ追加する（7 行目）．そうして最終的に返却ノード列を得る（10 行目）．

例として，図 3 に示される XML 文書に対して前述の XPath 例で検索をかける場合を説明する．まず，検索対象パスからマッチする dg ノードをすべて探す．その結果，**図 8** に示される「マッチする dg ノード群」が得られる．次に，検索ワード「地球」を含むすべてのテキストノードを探す．その結果，図 8 に示される「検索ワードを含むノード群」が得られる．「マッチする dg ノード群」と「検索ワードを含むノード群」から/book/chapter/section/title/text() のパスで示されるノードが結果 XML ノード列として返される．

図 7 の 6 行目の比較処理が低速化が懸念された検索パスとのマッチング処理に相当するが，ストロング DATAGUIDE 自体が小さいことと，ハッシュテーブルによる実装などを行うことで，高速処理が実現できる．

4. 評 価 実 験

前述のアルゴリズムについて実装を行い，評価実験を行った．

表 1　実験環境
Table 1　Experimental setup.

マシン スペック	Dell Power Edge 2800 CPU: Intel Xeon 3.6GHz(HT on) ×2 RAM: 8GB
OS	RedHat Enterprise Linux ES 4.0 (64bit)
その他	PostgreSQL 8.2.4

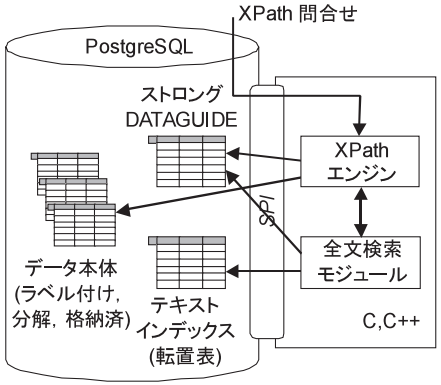

図 9　システム構成
Fig. 9　Experiment system.

実験環境を**表 1** に，システム構成を**図 9** に示す．基本的には文献 5) のシステム構成をベースにしているが，本論文ではストロング DATAGUIDE についても，PostgreSQL[8] の拡張機能として実装した．具体的には，すべての出現パスを ID 付けし，テーブルに格納し，パスには ltree[⋆1]を改造したインデックスを構築した．この出現パスの ID がストロング DATAGUIDE における dg ノードの ID にあたる．テキストインデックスの実装については後述する．

4.1　テキストインデックスの実装

転置表方式を採用し，PostgreSQL のテーブルとして出現単語と出現位置を格納し，サーバプログラミングインタフェースを通して検索処理を行うモジュールを C 言語により実装した．単語の抽出には文献 9) に示される N-gram 方式を採用した．

提案手法においては，単語の出現位置として，出現文書の ID に加え，出現ノードの ID（範囲ラベル）および dg ノードの ID も合わせて記録する．**図 10** に XML データおよびストロング DATAGUIDE の例を，**図 11** に図 10 の例から作成されるテキストインデックスを示す．図 10 の※ A のノードで例を説明する．※ A のテキストノードは「惑星とは」と

⋆1 PostgreSQL の contrib モジュールの 1 つ．擬似ツリー構造用のインデックス

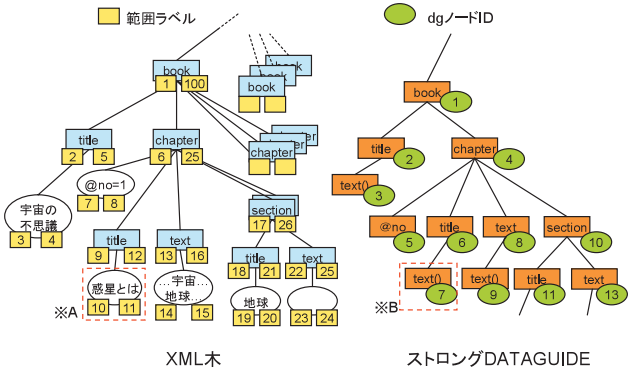

図 10　XML データとストロング DATAGUIDE の例
Fig. 10　An example of XML data and Strong DATAGUIDE.

	term	文書ID	範囲ラベルの preorder	dgノードID	出現位置
	宇宙	(1) (1)	(3) (13)	(3) (9)	(1) (145)
	宙の	(1)	(3)	(3)	(2)
	の不	(1) (2)	(3) (126)	(3) (21)	(3) (91)
	…	…	…	…	…
	惑星	(1)	(9)	(7)	(1)
	星と	(1)	(9)	(7)	(2)
	…	…	…	…	…

termカラムにB-treeインデックスを構築

図 11　テキストインデックス
Fig. 11　A Text Index.

いう値を持ち，「9」という範囲ラベルの preodrer 値を持つ．また，※ A のノードはストロング DATAGUIDE の※ B の dg ノードに対応し，dg ノード ID は「7」である．したがって，図 11 に示すように，term が「惑星」「星と」のレコードの範囲ラベルの preorder カラムには「9」，dg ノード ID のカラムには「7」が格納される．また，テキストインデックスの格納される各データは term を除いてすべて integer 型（4 バイト）とした．

そのうえで，検索処理として，テキストインデックスに対し，term を条件とした SQL を発行し，結果を取得した直後に，dg ノードの ID を用いて求める部分文書に適合しない出現位置をすべて排除する．可能な限り早い段階で対象外の出現位置を減らすことで，その後の処理の効率化を図っている．

4.2　データセット

公開特許公報 2006 年 6 月 1 日発行（2006_021）および 6 月 8 日発行（2006_022）から，1 万件を使用した．サイズは約 400 MB，総ノード数は約 923 万ノード（うちテキストノードは 458 万ノード），テキスト量は約 1.67 億文字である．

実験に用いた特許データはルートノードとして jp-official-gazette ノードを持ち，その直下に，下位に発明の名称や当事者，国際特許分類などの書誌情報を持つ bibliographic-data ノードや下位に技術分野や実施例などの発明の詳細な説明データを持つ description ノード，請求範囲情報を持つ claims ノード，要約情報を持つ abstract ノード，図情報を持つ drawings ノードなどを持つ．特許データのスキーマの主要なノードを**図 12** に示す．

実験に用いた特許データの詳細については文献 10) を参照されたい．

4.3　ク　エ　リ

今回の実験では特許データから規模の異なる 3 種類の部分文書を指定して全文検索を行った．以下に XPath クエリを示す．

A)「詳細な説明（description）」のクエリ
　/jp-officail-gazette/description[ftscontains(. , 'word')]
B)「請求の範囲（claims）」のクエリ
　/jp-officail-gazette/claims[ftscontains(. , 'word')]
C)「要約（abstract）」のクエリ
　/jp-officail-gazette/abstract[ftscontains(. , 'word')]

それぞれ，全文検索の対象となる部分文書の規模が異なり，**表 2** に示すとおりである．

A) のクエリは「詳細な説明」の部分文書単位での全文検索を行う．図 12 の※ 1 以下の部分文書を対象に，全文検索を行う．XML 文書全体の中の 40%以上のノードが部分文書に

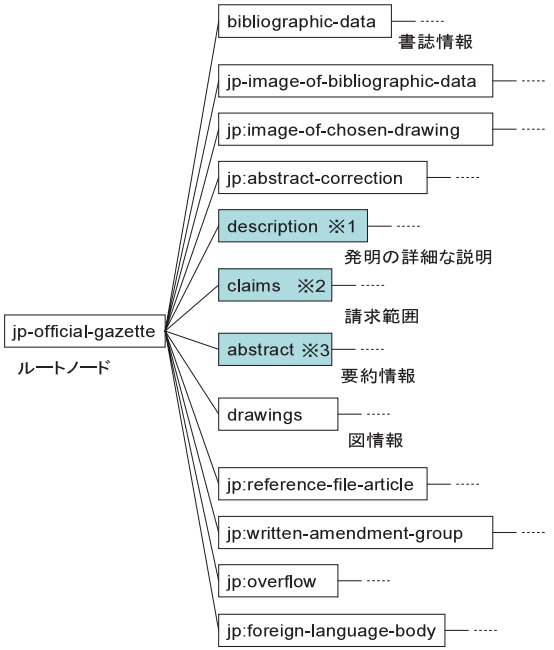

図 12　特許データスキーマの一部
Fig. 12　A part of patent data schema.

表 2　部分文書の規模
Table 2　Scale of the partial documents.

クエリ	規模	比率
A) (詳細な説明 description)	404 万ノード	43.8%
	208 万テキストノード	45.4%
	1.24 億文字	74.0%
B) (請求の範囲 claims)	81.4 万ノード	8.82%
	45.3 万テキストノード	9.87%
	1220 万文字	7.32%
C) (要約 abstract)	9.60 万ノード	1.04%
	4.73 万テキストノード	1.03%
	263 万文字	1.58%

含まれており，3 つのクエリの中で最も大きな部分文書を対象としている．B) のクエリは「請求の範囲」の部分文書単位での全文検索を行う．図 12 の※ 2 以下の部分文書を対象に，全文検索を行う．XML 文書全体の 10%弱のノードが部分文書に含まれている．C) のクエリは「要約」の部分文書単位での全文検索を行う．図 12 の※ 3 以下の部分文書を対象に，全文検索を行う．XML 文書全体の 1%程度のノードが部分文書に含まれている．

また，word には検索ワードが入り，データセット約 200 種類の名詞句をランダムに抽出し，使用した．

4.4　測定結果および考察

実験は検索速度評価とインデックスの評価の 2 つの観点で行った．検索速度評価は提案手法，部分インデックス手法，単純組合せ手法の 3 種類に対して行い，それぞれを比較した．各手法の詳細は後に詳述する．インデックスの評価に関しては提案手法と単純組合せ手法を比較した．部分インデックス手法においてすべてのインデックスを構築しようとすると，インデックス構築箇所の組合せ爆発からインデックス容量が現実的に不可能な大きさになる．そのため，今回の実験ではクエリが発行される箇所のみにインデックスを構築し，インデックスの評価の比較は行わない．

部分インデックス手法とは，今回の実験で用いる 3 種類の検索クエリが指定する部分文書単位でテキストインデックスを構築し，それを用いて部分文書の全文検索を行う手法である．3 種類の手法の中で最も高速に検索を行うことができる手法だが，2.2 節で述べたとおり問題があるため，今回の実験のようにあらかじめクエリの指定する部分文書が分かっている場合でないと適用しにくい手法である．部分インデックス手法を用いて，XML 文書に対して任意の XML 部分文書に対する全文検索を実現することは事実上不可能である．

単純組合せ手法とは，文書全体にテキストインデックスを構築し，転置表と検索ワードから検索ワードを含むテキストノードを検索するところまでは提案手法と同じ処理を行う．その後ストロング DATAGUIDE による検索ワードを含むテキストノードの絞り込みを行わずに，それらテキストノードと検索パスとの構造ジョインを行って部分文書への適合処理を行ったものである．3 種類の手法の中では最も処理に時間がかかる手法である．また，単純に文書全体にインデックスを構築するだけのためインデックスの容量は比較的小さい．

速度測定範囲は，クエリに適合するノード ID を取得するところまでとし，結果（部分文書）の返却は対象外とした．

また，速度測定用検索クライアントは Ruby 1.8.4 + PostgreSQL 拡張モジュール（ruby-postgres-0.7.1）により作成した．

表 3　検索速度測定結果
Table 3　Retrieval execution time.

クエリ	手法	平均検索時間(sec)	提案手法との比率
A)	提案手法	1.5749	100%
	部分インデックス	1.5726	99.85%
	単純組合せ	2.4258	154.03%
B)	提案手法	0.4091	100%
	部分インデックス	0.4002	97.82%
	単純組合せ	2.3692	579.16%
C)	提案手法	0.0763	100%
	部分インデックス	0.0686	91.20%
	単純組合せ	2.3502	3078.49%

表 4　インデックスの評価
Table 4　Index construction time.

手法	インデックスデータ容量(MB)	インデックス構築時間(min)
提案手法	3,391	31.17
単純組み合わせ	2,868	30.28
(提案手法との比率)	(84.59%)	(97.15%)

速度測定結果を**表 3** に，インデックスの評価を**表 4** に示す．

検索速度の点で提案手法と部分インデックス手法を比較すると，提案手法との比率で 90%以上の性能が出ている．対象となる部分文書の規模が小さくなるにつれ，その差は開いているが，クエリパターン C) においても，その差は 9%未満と，部分文書の比率を考えると非常に小さな差といえる．

また，提案手法と単純組合せを比較すると，クエリパターン A) でも 50%以上の差があり，C) においては 1 桁以上の差となっている．これは提案手法のストロング DATAGUIDE を用いた絞り込みの効果であると考えられる．本実験はクエリに適合するノード ID を取得するところまでで時間を計測しているため，実サービスを考えると，処理時間の絶対量として 2.4 秒前後の処理時間は大きい．クエリパターン A) は理想的な部分インデックス手法で

も 1.5 秒の処理時間を要するが，クエリパターン B) や C) においても 2.4 秒前後の処理時間を要する単純組合せ手法のコストは高いと考えられる．

一方インデックスサイズでは，単純に文書全体にテキストインデックスを構築する単純組合せ手法と比較して dg ノード ID を格納する容量 523 MB（テキストノードの文字数 × 4 バイト）分の増加量は予想の範囲内であり，その差は全体量から見ると増加量は小さい．また，部分インデックス手法で任意の XML 部分文書に対する全文検索に対応することが事実上不可能な点から見ても効率の良いインデックスの構築ができたといえる．

また，インデックスの構築時間も単純組合せ手法と比較してもほぼ同程度の時間で構築が可能であることが分かる．

以上より，提案手法は，任意の XML 部分文書に対する全文検索への対応を実現しつつ，部分インデックス方式に匹敵する高速性も達成し，単純組合せ手法と比べても遜色ない構築時間のインデックスで実現できる，非常に有用な手法であると考えられる．

5. 関 連 研 究

これまで，XML 文書に対する検索は文書単位で行われてきた．文献 1) では構造化データに強い XML 問合せ言語である XQuery を全文検索機能拡張させた TeXQuery という技術を提案し，テキストとデータが混在する XML を柔軟に検索することができる手法を提案した．また，文献 2) では XML データに対する参照の手法としての XPath や XQuery などの XML 問合せ言語と XML 文書のテキスト部に対する全文検索手法を効率良く統合するための手法を論じている．これらの文献では，全文検索を用いた効率の良い検索手法について論じているが，構造を利用して文書の一部分を検索することは論じられていなかった．

また，蓄積された XML 文書に対して部分文書単位のキーワード検索を行う研究がなされている．文献 3) では，検索対象となる部分文書を分析し，その結果に基づいて解答候補となる XML 部分文書を決定する XML 文書検索システムについて論じている．文献 4) では，キーワードでの XML 文書検索の解答に大量の XML 部分文書を扱うために，検索結果提示のインタフェースについて議論されている．これらの文献では利用者が XML の構造に詳しくなく，キーワードの指定だけで部分文書の検索を適切に実行できる技術の研究であるのに対し，我々の研究では利用者がキーワードだけでなく XML 構造の検索部分を指定することでより詳細に条件指定した部分文書の検索を行うことができる技術を主眼としている．

文献 11) では文字列検索技術を応用した高速な XQuery 処理技術が提案されている．文献 11) 内で用いられているパススキーマとはストロング DATAGUIDE と同等の技術で，これを利用することで高速な検索処理を実現しており，大量のストリーミング XML データなどに対して指定されたフィルタリング処理を高速に行うことができる．本研究は蓄積された XML 文書に対して検索対象である部分文書を柔軟に設定して検索することができる技術であり，文献 11) とは利用シーンが異なる．

6. ま と め

本論文では，ストロング DATAGUIDE および範囲ラベルを用いた構造検索手法とテキストインデックスを拡張した全文検索手法を組み合わせることで，XML 文書に対して任意の指定されたパスに対応する部分文書を対象に全文検索する方式を提案した．また，PostgreSQL を用いて実装を行い，特許データを用いた評価実験を行うことで，柔軟な検索に対応しつつ，高速な部分インデックス手法に匹敵する処理速度で検索を行うことができることを確認した．

今後は，インデックスの実装方法を見直し，各レコードのヘッダ情報を縮小することでインデックスサイズの増加を抑えたい．また，検索対象となる部分文書群に対して，統計情報を用いたスコアリングを行うことによるランキング精度についても検討していきたい．

参 考 文 献

1) Amer-Yahia, S., Botev, C. and Jayavel: TeXQuery: A Full-Text Search Extension to XQuery, *WWW 2004* (2004).
2) Amer-Yahia, S., Lakshmanan, L.V.S. and Pandit, S.: FleXPath: Flexible structure and full-text querying for XML, *SIGMOD 2004*, pp.83–94 (2004).
3) 波多野賢治，絹谷弘子，吉川正俊，植村俊亮：XML 文書検索システムにおける文書内容の統計量を利用した検索対象部分文書の決定，電子情報通信学会論文誌 D，Vol.J89-D, No.3, pp.422–431 (2006).
4) 清水敏之，寺田憲正，吉田正俊：関係データベースを用いた XML 情報検索システムの開発，情報処理学会論文誌：データベース，Vol.48, No.SIG 11 (TOD 34), pp.224–234 (2007).
5) 江田毅晴，鬼塚　真，山室雅司：XML データの要約情報を用いた高速な XPath 処理方式，電子情報通信学会論文誌，Vol.J89-D, No.2, pp.139–150 (2006).
6) Goldman, R. and Widom, J.: DataGuides: Enabling query formulation and optimization in semistructured databases, *Proc. VLDB*, pp.436–445 (1997).
7) Al-Khalifa, S., Jagadish, H.V., Koudas, N., Patel, J.M., Srivastava, D. and Wu, Y.: Structural joins: A primitive for efficient XML query pattern matching, *Proc. ICDE*, p.141 (2002).

8) NPO 法人日本 PostgreSQL ユーザ会ホームページ．http://www.postgresql.jp/
9) 原田昌紀，風間一洋，佐藤進也：Unicode を用いた N-gram 索引の一実現方式とその評価，情報処理学会研究会報告，2000-NL-136-17, pp.135–142 (2000).
10) 特許庁ホームページ．http://www.jpo.go.jp/indexj.htm
11) 石野　明，竹田正幸：パスプルーニングによる決定性有限オートマトンを用いた XQuery 処理の提案，日本データベース学会 Letters，Vol.4, No.4, pp.17–20 (2006).

(平成 19 年 12 月 28 日受付)
(平成 20 年 6 月 3 日採録)

推　薦　文

文章の構造を指定した効率の良い全文検索手法として，既存方式に対するテキストインデックスの拡張方法と検索方法について提案している．全体に論理の道筋が明確な内容で新規性・有用性が認められる．特許文章を対象とした評価を通して，提案した手法の有効性も示されている．読者にとって有用な論文になることが期待でき，推薦するに値する．

（マルチメディア通信と分散処理研究会主査　櫻井紀彦）

兵藤　正樹（正会員）

2004 年北陸先端科学技術大学院大学知識科学研究科修士課程修了．同年日本電信電話株式会社入社．XMLDB における検索技術の研究に従事．

榎本　俊文（正会員）

1994 年大阪大学大学院基礎工学研究科修士課程修了．同年日本電信電話株式会社入社．エージェント通信技術，全文検索技術，XMLDB 技術に関する研究に従事．人工知能学会会員．

江田　毅晴（正会員）

2003 年奈良先端科学技術大学院大学情報科学研究科博士前期課程修了．修士（情報学）．同年 NTT サイバースペース研究所に勤務．現在，京都大学情報学研究科博士後期課程に在籍．データベースシステムおよび Web におけるデータ処理技術に関する研究開発に従事．日本データベース学会会員．

山室　雅司（正会員）

1987 年早稲田大学大学院理工学研究科数学専攻修士課程修了．同年日本電信電話株式会社入社．1990 年コロンビア大学大学院電気工学研究専攻修士課程修了．以来ネットワーク設計法，データベース設計・可視化，マルチメディア情報検索，デジタル情報流通基盤の研究開発に従事．博士（工学）．1994 年電子情報通信学会学術奨励賞．電子情報通信学会，日本ソフトウェア科学会，日本データベース学会，IEEE-CS 各会員．情報処理学会情報規格調査会理事．