社団法人　情報処理学会　研究報告
IPSJ SIG Technical Report

2005－DD－50（1）
2005／5／27

全文・構造検索両立型索引方式

稲葉光昭　里雄二　玉利公一　松浦俊　菅野祐司

松下電器産業（株）　ネットワーク開発センター

構造化文書に対して単一の索引で，構造を問わない全文検索と，構造を指定した検索の両方が可能な新しい索引検索方式について提案する．既存の全文検索用の転置索引を拡張し，単語の出現位置情報に文書構造を表す情報項目を追加するとともに，要素や属性，祖先パスなどの文書構造そのものを要素名や属性名，祖先パス名を表す特別な単語の出現として索引に記録する．人工的に生成した１ＧＢ程度の大規模なＸＭＬ文書を用いて評価実験を行った結果，従来の全文検索用索引の約１．７倍程度の索引サイズで，「*」や「//」などを含む XPath 式での検索も極端に性能が劣化することなく実用上十分な検索性能が得られ，本方式の有効性を確認することができた．

A Single Index Supporting Full-text Search and Structure-aware Search

Mitsuaki INABA　Yuji SATO　Koichi TAMARI　Shun MATSUURA　Yuji KANNO

Network Development Center, Matsushita Electric Industrial Co., Ltd.

We propose a new indexing method of structured documents supporting full-text search and structure-aware search by a single index. We expand an inverted index for full-text search, add structure information items to a word occurrence information, and register the occurrences of elements, properties and ancestor-paths as those of a special word. Our experiments show that the size of index is about 1.7 times larger than that of former full-text index, and the search speed for queries of XPath expressions including “*” and “//” is enough fast for practical use. We confirmed the effectiveness of our method.

1. イントロダクション

インターネットにおける情報表現および情報交換の標準形式としてＸＭＬはますます重要視されており，ＸＭＬ文書に対する検索の必要性は高まる一方である．

ＸＭＬ文書はデータ主体のものとテキスト主体のものに大別されるが，実際には両方の性質を併せ持つ場合が多い．そのような文書を扱うためには，構造検索と全文検索の両方の要求を満たす必要があるが，多くの挑戦的な課題が存在しており[1]，近年，研究が盛んに行われるようになってきた．

現時点では全文検索と構造検索がともに可能である主なＸＭＬ検索方式は以下のように大別される．

１）全文索引アドオン型

構造検索用の索引をベースにし，必要に応じて全文検索用の索引を併用するタイプで，eXist[2]がある．

効率の良い構造検索が実現できる半面，両方の索引を持つため索引サイズや索引作成時間が増大する問題があり，また構造を指定した全文検索の際には，構造検索条件，全文検索条件でそれぞれの索引を検索し，両者の結果の共通部分を求めるため，中間結果が増え検索時間が長くなるといった課題がある．

２）構造索引アドオン型

全文検索用の索引をベースにし，構造検索条件を全文索引用の条件式に変換するための構造索引を追加したタイプで，[3]がある．特に日本語においては全文検索の必要性が高く，数多くの全文検索方式が発表されている[4,5,6,7,8,9]ため，このタイプの検索方式が実用的であるとすればその重要性は高い．

全文検索用の索引構造を変更しなくて済むため，全文検索の機能・性能は維持できる半面，ある種の構造検索条件が膨大な数のＯＲ条件に展開され，検索速度が非常に遅くなってしまう場合があるという問題がある．

３）走査型

索引を持たず，検索実行時にＸＭＬ文書を読み込み１パスで照合を行う検索方式で，[10,11]がある．

この方式には，索引が不要で更新時間が短いという特長があるが，文書サイズに比例した検索時間がかかるため大規模文書に対する検索の高速化が難しいという課題がある．

また近年，検索条件とＸＭＬ文書の両方を構造符号化列に変換し，両者のマッチングをとることで検索を行う部分列照合(subsequence matching)方式が研究され

ている[12,13].

この方式では構造情報とテキスト情報とを単一の索引に統合しているために，原理的には１）や２）の方式の持つ課題を解決できる可能性がある．ただし，木構造を符号化列に変換して索引化する方式のため，出現順を問わない構造検索条件やワイルドカード（例えば XPath の「//」）を含むような検索条件に対しては検索時間が長くなるという課題がある．さらに，研究の歴史が浅く，従来の全文検索方式と同等の機能・性能は現時点では実現できていない．

そこで，本稿では全文検索用に一般的に用いられている転置索引（日本語全文検索用の n-gram 索引や極大単語索引[5]を含む）を拡張することにより構造検索を可能にする新たな索引検索方式を提案する．

本稿は以下のような構成になっている．2 章で我々がどのような点に着目しアプローチしたかを，3 章および 4 章で索引の構造とその検索方法について，5 章で索引サイズと索引検索速度に関する評価実験の結果について述べ，6 章で結論と将来課題について述べる．

2. ねらいとアプローチ

実用上重要な以下の２条件を満たすよう，2.1.～2.3. のアプローチで転置索引の拡張を行った．

- 辞書と転置索引という基本構成は踏襲すること
- 幅広い全文検索用の索引方式に適用できること

2.1. 全文検索用索引への構造情報の追加

[3]では「普通の文書をＸＭＬ化しても検索要求の中心は従来の項目検索」であり「項目検索に対する性能低下がなくかつ構造検索に対する性能劣化が少ないことが望ましい」という主張がなされている（項目検索は全文検索の一機能）．特に日本語文書に対する全文検索が比較的重い処理である点を考慮すると，この主張は妥当であると考えられる．しかしながら，ギガバイト級の大規模なＸＭＬ文書や複雑な構造をもつＸＭＬ文書に対しては，全文検索用の索引構造を全く変更しないアプローチでは性能上の問題が深刻になるため，扱える文書の量や質に限界がある．

本方式では，全文検索の内部処理で構造条件による候補の絞込み等が効率良く行えるように，構造検索用の索引をアドオンするのではなく，全文検索用の索引内部に構造検索用の豊富な情報を追加し，解決を図った．

2.2. 構造情報の転置

しかしながら，2.1.の追加情報だけでは全文検索条件を伴わない構造のみの条件での索引検索が困難である．転置索引の形式を保ったままで構造情報を扱うために，要素や属性などの文書構造を表す情報を要素名や属性名を鍵として転置し，索引に記録する方針を採用した．

2.3. 単語の出現位置情報の有効利用

構造検索用の各種情報を追加することは，索引サイズの増加につながる．

しかし，[4,5,6,7,8,9]等で用いられている転置索引には高速な検索を可能にするために出現レコード番号の他に出現する文字位置等の豊富な情報が含まれている．そこで，転置索引に含まれている文字位置等の情報を構造検索にも活用し，構造検索用の各種情報のうち転置索引には本質的に含まれない情報のみを複数の情報項目の組として追加することで索引サイズの増加を最小限に抑えることを狙った．

3. 索引構造

3.1. 全体構造

本方式によるシステムは，従来と同様，辞書と転置索引から構成されている（図１）．

ただし，辞書には予め登録されている通常の単語に加えて，対象文書から抽出された要素名，属性名，祖先パス名（後述）を表す特別な単語を登録する．

転置索引には以下に述べる(a)～(d)の 4 種の出現情報を登録する．このうち，本方式で新たに追加されるのは(a)の情報項目の一部と(b)～(d)である．

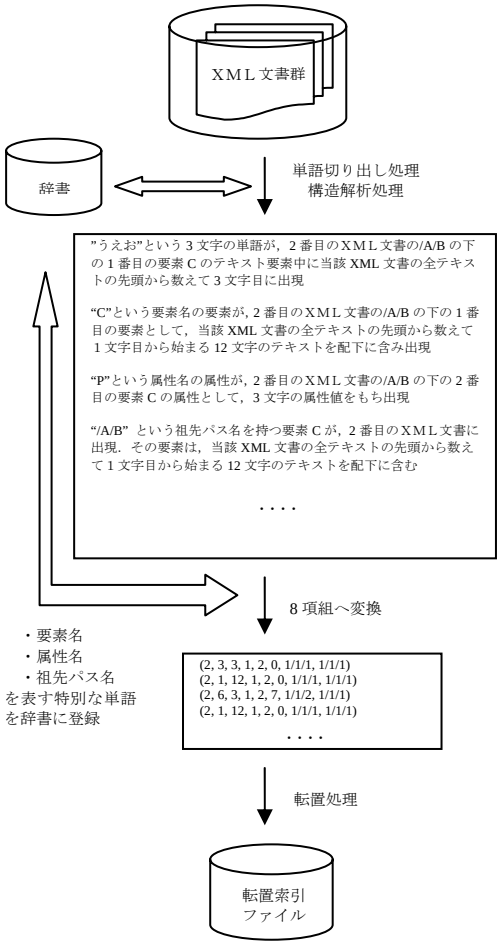

図１：索引作成処理の流れ

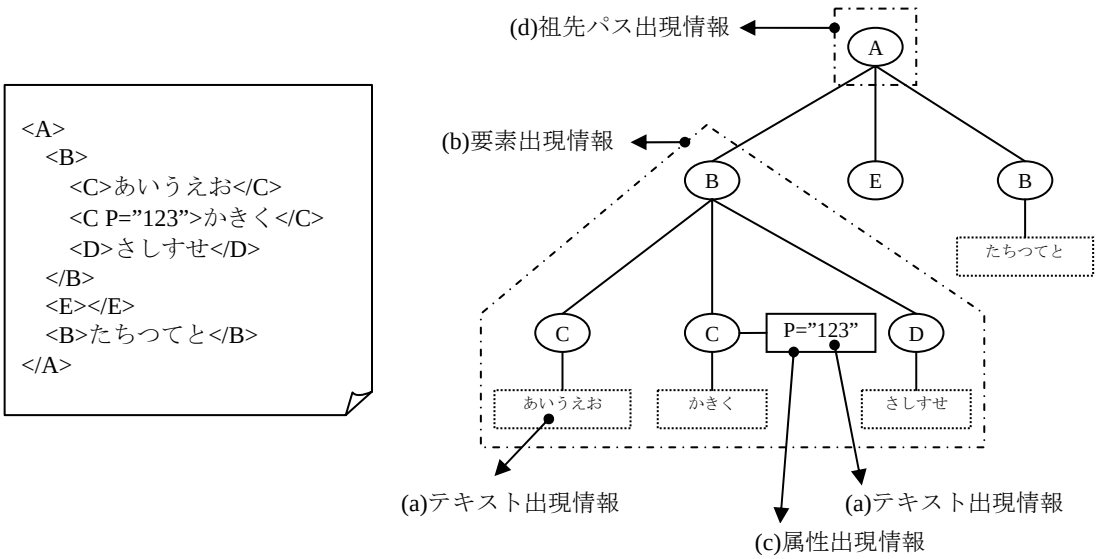

図２：四種類の出現情報

(a) テキスト出現情報

テキスト要素，および属性値中の文字列から切り出された単語の出現情報として，文書番号，出現文字位置，文字数，属する要素の要素名ＩＤ，祖先パス名ＩＤ，属性名ＩＤ，分岐順，空要素順の８つの情報項目を索引に記録する．

８つの情報項目のうち，文書番号と出現文字位置は従来の転置索引と共通であり，それ以外の６つの情報項目が本方式で新たに追加される項目である．

出現文字位置は，タグを除いたテキスト文字列中における当該単語の文書内オフセットとする．なお，属性値から切り出された単語の場合は，属性値文字列が仮想的に当該要素の先頭に出現しているとみなして算出した値を記録する．

テキスト要素から切り出された単語の場合には，属性名ＩＤには特別な値である０を格納する．

テキスト要素と属性値からの単語切り出しの方式は従来通りである．テキスト部分の出現情報はテキスト出現情報によって完全に転置することができ，従来の全文検索機能を損なうことがない．

テキスト出現情報を用いて次のような検索条件での結果を効率良く求めることができる．

・構造を問わない従来の単純な全文検索条件
・以下のような XPath 式
　/A/B/C[contains(., ”あいう”)]
　//C[contains(., ”あいう”)]
　/A/B/C[contains(@P, ”23”)]
　/A/B/*[contains(., ”あいう”)]

以下の(b)～(d)は従来の転置索引には含まれない情報であり，特別な単語の出現情報として転置索引に格納する．

(b) 要素出現情報

ある要素名を持つ要素の出現情報を，要素名を表す特別な単語の出現として，文書番号，出現文字位置，文字数，要素名ＩＤ，祖先パス名ＩＤ，属性名ＩＤ，分岐順，空要素順の８つの情報項目を索引に記録する．

ただし，出現文字位置としては着目している要素配下のテキストの先頭文字位置を，文字数としては要素配下にあるテキスト全体の文字数を記録する．

また，属性名ＩＤには常に０を記録する．

要素出現情報を用いて次のような検索条件（XPath 式）での結果を効率良く求めることができる．

　/A/B/C
　//C
　/A/*/C

(c) 属性出現情報

ある属性名を持つ属性の出現情報を，属性名を表す特別な単語の出現として，文書番号，出現文字位置，文字数，要素名ＩＤ，祖先パス名ＩＤ，属性名ＩＤ，分岐順，空要素順の８つの情報項目を索引に記録する．

ただし，出現文字位置としては着目している属性が属している要素配下のテキストの先頭文字位置を，文字数には属性値の文字数を記録する．

同じ要素に属する複数の属性間の出現順序に関する情報は特に記録しない．

属性出現情報を用いて次のような検索条件

（XPath 式）での結果を効率良く求めることができる．

/A/B/C/@P
//*/@P
/A/*/C/@P

また，テキスト出現情報と併せて用いることにより，次のような検索条件での検索も可能である．

/A/B/C[@P=”123”]

(d) 祖先パス出現情報

ある要素の親要素のパス名を当該要素（およびその属性）の「祖先パス名」と呼ぶことにする．たとえば, パス名が “/A/B/C” である要素の祖先パス名は “/A/B” であり, パス名が “/A/B” である要素の祖先パス名は ”/A” である．なお，ルート要素およびその属性の祖先パス名は空列とする．

上記のように定義された祖先パス名に関して，ある祖先パス名を持つ要素および属性の出現情報を，祖先パス名を表す特別な単語の出現として，文書番号，出現文字位置，文字数，要素名ＩＤ，祖先パス名ＩＤ，属性名ＩＤ，分岐順，空要素順の８つの情報項目を索引に記録する．

ただし，出現文字位置，文字数は着目要素の要素出現情報，もしくは着目属性の属性出現情報に記録する値と同じである．

祖先パス出現情報は，要素出現情報，属性出現情報と重複した内容であり，索引サイズの増大を招くものの，次のような検索条件（XPath 式）に対する検索速度向上に大きな効果が期待できるため，採用することにした．

/A/B/*
/A/B/*/@P

3.2. 分岐順

「分岐順」とは，ルート要素から着目している要素に至るパスの各階層において，同じ親要素を持つ同一要素名の要素の中で何番目に出現したかの順を示す番号を並べたものである．図３は分岐順の例を示したもので，各要素の右肩に記した “1/1/2” 等が分岐順を表している．

分岐順の情報項目は，要素名，祖先パス名に含まれない情報で，次のような検索条件（XPath 式）での結果を効率良く求めるために利用する．

/A/B/C[2]

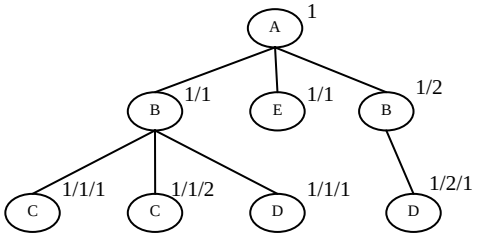

図３：分岐順の例

3.3. 空要素順

「空要素順」とは，ルート要素から着目している要素に至るパスの各階層において，同じ親要素を持つ要素（兄弟要素）のうちで，先頭の要素であるかもしくは直前の兄弟要素が空要素（子孫要素を含めて要素内容にテキストを全く含まない要素）でない要素の場合には１，それ以外の場合（すなわち，直前の兄弟要素が空要素である場合）には，直前の兄弟要素の空要素順の番号に１を加えた番号を順に並べたものである．

図４は空要素順の例を示したもので，灰色の楕円は空要素を表し，各要素の右肩に記した “1/1/2” 等が空要素順を表している．

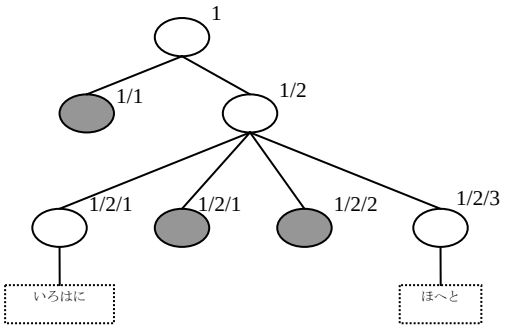

図４：空要素順の例

空要素順は，次のような検索条件に対して，空要素 E の直後の兄弟要素 B を結果として求めるために利用する．

/A/E/following-sibling::B

なお，要素 E が空要素でない場合には上記式の評価の際に空要素順を使用せずに出現文字位置の情報で要素 B が条件を満たすかどうかを判定することができる．また，空要素が含まれていなければ空要素順はすべて同じになるので記述情報量は抑えることができる．

3.4. 索引データ形式

以上で述べたような内容の索引情報のデータは図５に示す８項組の形式で記述することができる．

索引情報は単語辞書 D と出現位置索引 I から構成される．

D［単語辞書］ :文字列wから id=GetWid(w)によってユニークな単語ＩＤを求める機能を持つ．
ただし，wが辞書に登録されていない場合にはwが D に新規登録される．
索引化の際，辞書 D には以下の４種の単語が登録される．

（1）テキスト要素および属性値中から切り出された（狭義の）単語
（2）要素名（登録時には先頭に $ を付加し，他の単語と区別する）
（3）属性名（登録時には先頭に@を付加し，他の単語と区別する）
（4）祖先パス名（登録時には先頭に%を付加し，他の単語と区別する）

I［出現位置索引］: 以下の４種類の出現位置情報を単語ＩＤを鍵にして転置した索引．

（1）テキスト出現情報
テキスト要素または属性値から切り出された単語 w に対して，その出現を表す８項組の情報 P．索引 I には（GetWid(w), P）を登録する．

P = (rno, pos, len, aid, nid, pid, bno, eno)

ただし，各項は以下の通りとする．

rno: ＸＭＬ文書番号	（ＸＭＬ文書を索引に登録した順序）
pos: 出現文字位置	（当該ＸＭＬ文書内の全テキスト要素の値をつなげた文字列中における当該単語の出現位置）
len: テキスト文字数	（単語 w の文字数）
aid: 祖先パス名ＩＤ	(GetWid("%"<祖先パス名文字列>) によって求めたＩＤ）
nid: 要素名ＩＤ	(GetWid("＄"<要素名文字列>) によって求めたＩＤ）
pid: 属性名ＩＤ	(GetWid("@"<属性名文字列>) によって求めたＩＤ．ただし，テキスト要素中から切り出された単語の場合には 0 のような特別なＩＤ）
bno: 分岐順	（所属要素までの各要素に関する「同一要素名の兄弟要素中での順序」の列）
eno: 空要素順	（所属要素までの各要素に関する「直前に連続する兄弟空要素数＋１」の列）

（2）要素出現情報
要素に対して，その出現を表す８項組の情報Qe．索引 I には (GetWid("$"<要素名文字列>), Qe) を登録する．

Qe = (rno, pos, len, aid, nid, pid, bno, eno)

ただし，各項は以下の通りとする．

rno: ＸＭＬ文書番号	（ＸＭＬ文書を索引に登録した順序）
pos: 出現文字位置	（当該ＸＭＬ文書内の全テキスト要素の値をつなげた文字列中における，着目要素配下に含まれるテキストの先頭文字位置）
len: テキスト文字数	（着目要素配下に含まれるテキストの文字数）
aid: 祖先パス名ＩＤ	(GetWid("%"<祖先パス名文字列>) によって求めたＩＤ）
nid: 要素名ＩＤ	(GetWid("＄"<要素名文字列>) によって求めたＩＤ）
pid: 属性名ＩＤ	（常に０）
bno: 分岐順	（着目要素までの各要素に関する「同一要素名の兄弟要素中での順序」の列）
eno: 空要素順	（着目要素までの各要素に関する「直前に連続する兄弟空要素数＋１」の列）

（3）属性出現情報
属性に対して，その出現を表す８項組の情報Qp．索引 I には (GetWid("@"<属性名文字列>), Qp) を登録する．

Qp = (rno, pos, len, aid, nid, pid, bno, eno)

ただし，各項は以下の通りとする．

rno: ＸＭＬ文書番号	（ＸＭＬ文書を索引に登録した順序）
pos: 出現文字位置	（当該ＸＭＬ文書内の全テキスト要素の値をつなげた文字列中における，着目属性の属する要素配下に含まれるテキストの先頭文字位置）
len: テキスト文字数	（属性値の文字数）
aid: 祖先パス名ＩＤ	(GetWid("%"<祖先パス名文字列>) によって求めたＩＤ）
nid: 要素名ＩＤ	(GetWid("＄"<要素名文字列>) によって求めたＩＤ）
pid: 属性名ＩＤ	(GetWid("@"<属性名文字列>)によって求めたＩＤ）
bno: 分岐順	（着目属性の属する要素までの各要素に関する「同一要素名の兄弟要素中での順序」の列）
eno: 空要素順	（着目属性の属する要素までの各要素に関する「直前に連続する兄弟空要素数＋１」の列）

（4）祖先パス出現情報
当該祖先パス名を持つ要素または属性に対して，その出現を表す８項組の情報Qa．索引 I には (GetWid("%"<祖先パス名文字列>), Qa) を登録する．

要素に対する祖先パス出現情報であれば，Qa はその要素の Qe と同一内容であり
属性に対する祖先パス出現情報であれば，Qa はその属性の Qp と同一内容である．

図５：索引データ形式

3.5. 索引化の例

ＸＭＬ文書とそれに対して作成される索引情報の例を図６に示す.

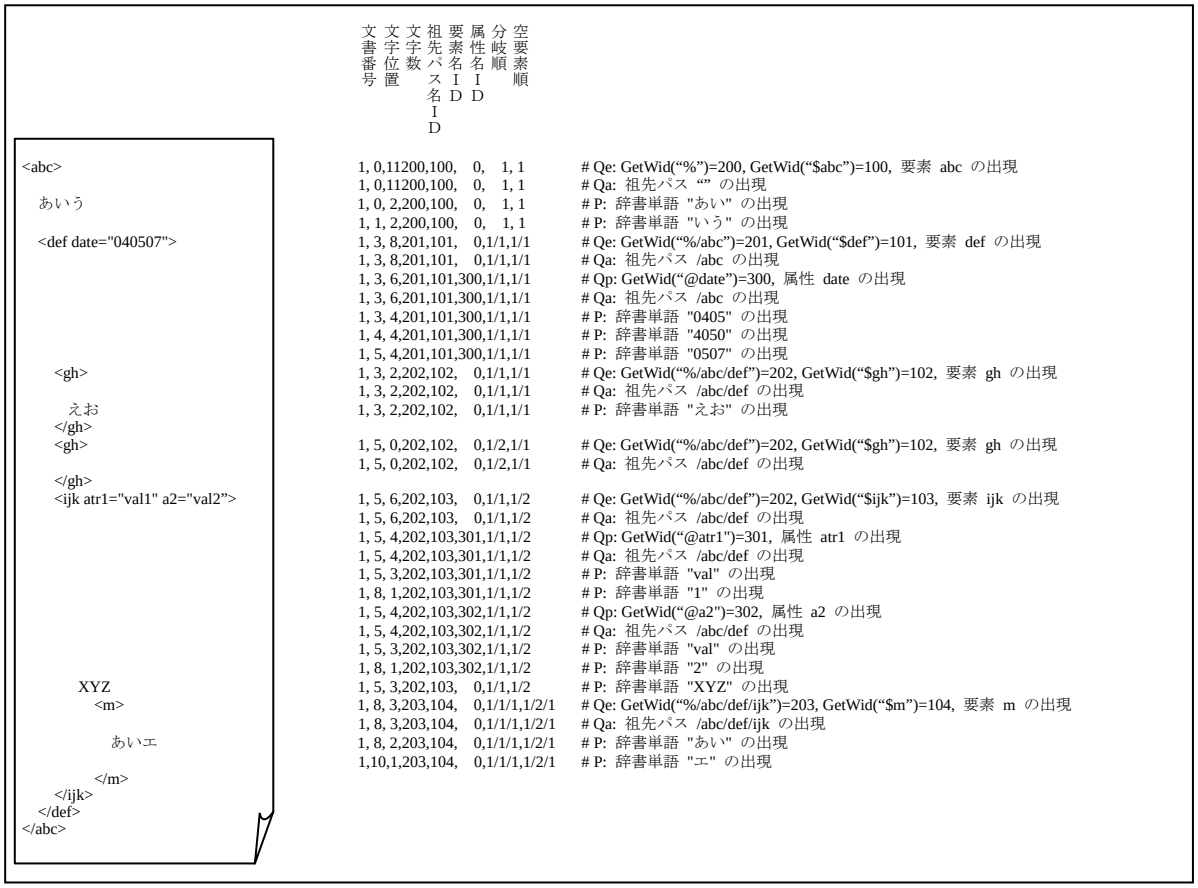

図６：ＸＭＬ文書と索引情報の例

3.6. 実装上の工夫

3.4.で述べたような８つの情報項目を素直に索引に記録してしまうと，どうしても索引サイズが大きくなる．また，検索速度の面からも得策とは言い難い．

このような問題を回避するため，図７に示すように出現情報の項目のうち文書番号と出現文字位置を除く６つの情報項目から共通項を括り出した形で複数のグループに分けて索引に記録する．これにより，索引サイズを大幅に削減するとともに，4.の例１で述べるような検索条件の場合に，条件に適合しないグループの出現情報の処理を省くことで，検索速度の向上を図ることができる．

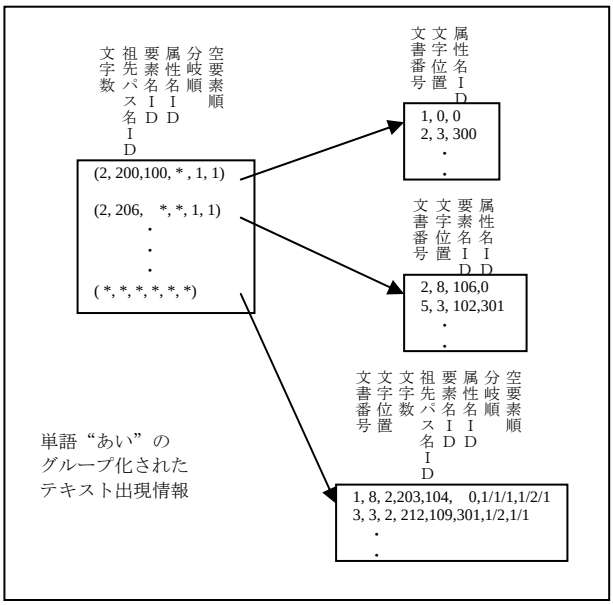

図７：出現情報のグループ化

4. 検索方式

基本的な検索方式は拡張前の転置索引と同様であるが，以下の点が異なる．

- 各出現情報が図５に示したような８項組であり，検索処理の早い段階で構造条件による絞り込みを行う
- 中間検索結果集合も８項組で表し，必要に応じて構造条件を考慮した論理演算を行う

以下，いくつかの検索条件式を例にとって，本方式の有効性を示す．

【例１】
//*[contains(.,”松下”)]
/A/B/C[contains(.,”松下”)]

従来方式１）では後者の検索式に対しては /A/B/C という構造条件と”松下”という全文条件の中間結果のＡＮＤをとる方式なので，一般に前者より後者の条件式のほうが計算量が多くなる．

本方式では 3.6.で述べたように，後者の検索式に対しては，“松下” のテキスト出現情報を求める際に /A/B/C に照合しない共通項をもつグループの出現情報の処理を省くことができるため，前者より後者の条件式のほうが計算量が少なくなることが予想される．

【例２】
/A/B//C[contains(.,”松下”)]

従来方式１）２）では /A/B//C という曖昧さを含む構造条件で構造索引を検索し，直接ノード番号の集合を求めている．

本方式では辞書を検索して /A/B// に照合する祖先パス名ＩＤの範囲を求め，このＩＤ範囲を含む条件で索引検索をする．したがって従来方式のような膨大な数のノード番号に展開されるという問題がほとんど発生せず，展開処理そのものも索引サイズに直接依存しないため効率的である．

【例３】
/A/B[contains(C,” 松 下 ”)] [contains(D,” 電 器 ”)] [contains(E,”産業”)]

部分列照合方式では，３つの述語について出現順を入れ替えた６通りの式に変換して検索しそれらのＯＲを取る必要がある．

本方式では，中間結果を８項組の集合で表現し，/A/B[contains(C,”松下”)]と/A/B[contains(D,”電器”)]と/A/B[contains(E,”産業”)]の３種の条件での中間結果集合間で構造を考慮したＡＮＤをとることができるため，効率よく結果を求めることが可能である．

5. 評価実験

本方式の有効性を確認する目的で，以下の２項目について基本的な評価実験を行った．

- 追加情報による索引サイズ増加率の評価
- 4.で述べた検索処理における有効性の確認

5.1. 実験条件

評価実験には，国内公開特許の書誌・明細から人工的に生成した４種類の文書を使用し，それぞれ，通常の全文検索用索引と本方式による索引を作成した．

４種類の入力文書は，それぞれ以下のような特徴を持っており，その基本データを表１に示す．

１：構造に比較的規則性があり，
　　長いテキスト要素を含む
２：構造に比較的規則性があり，
　　長いテキスト要素を含まない
３：構造がランダム，階層が深い
４：構造がランダム，階層が浅い

表１：入力文書の基本データ

入力文書	1	2	3	4
総文書サイズ	1.27GB	1.24GB	0.98GB	0.98GB
文書数	169,992	199,991	58,312	80,588
階層の深さ(最大／平均)	7/6	7/6	9／4	7／3
子要素数(最大)	4	4	10	10
1要素あたり属性数(最大)	3	3	3	3
要素名種類数(最大)	2000	2000	16	16
属性名種類数(最大)	500	500	1000	1000
祖先パス名種類数	172	172	44,976	4,426

なお，通常の全文検索用索引，本方式の索引ともにテキストの切り出し方式としては極大単語切り出し方式[5]を用いた．また，通常の全文検索用索引の作成処理では，ＸＭＬ文書をタグも含め単なるプレーンテキストとして扱った．これにより，通常の全文検索用索引がサイズの面で不利にならないよう，辞書には予めタグ名や属性名を通常の単語として登録しておいた．

また，本方式の索引では，共通項目数が多い順にグルーピングを行い，数値は可変長符号で記録し，分岐順や空要素順も数値に変換し索引に記録した．

検索速度の評価実験は各検索条件毎にコールドスタートで行った．

評価実験に使用した計算機のスペックは以下の通りである．

・CPU: Xeon 2.2GHz ×2
・主記憶: 4GB
・HDD: SCSI RAID

5.2. 索引サイズ

5.1.の各条件下で作成した索引サイズを表２に示す．索引サイズの評価指標として１出現情報あたりのバイト数を採用した．これは，全文検索が必要となるようなテキスト部分の割合の多い構造化文書の場合，本方式で作成した索引では，４種の出現情報のうち，テキスト出現情報が８～９割を占めるため，上記指標がほぼ「本方式で新たに増えた６つの情報項目による影響」を表していると考えられるからである．

まず，本方式による索引について，対象データを変化させた場合の索引サイズの変化を見てみると，最大で１出現情報あたり１バイト程度の増加になっている

ことがわかる．

次に，本方式による索引と通常の全文検索用索引との比較を行うと，文書３のように構造がランダムな文書でグルーピングの効果が出にくい場合で約１．７倍，文書２のように構造が比較的規則的であるような文書の場合では，約１．４倍の索引サイズになっていることがわかる．

表２：索引サイズ（１出現情報あたりのバイト数）

入力文書	1	2	3	4
本方式	**3.69574**	**3.95325**	**4.73174**	**4.16598**
全文索引	2.95004	2.83153	2.86665	2.89977

5.3. 索引検索速度

続いて，文書１の索引を使って典型的な XPath 式による検索を行い，検索速度の評価を行った．

ただし，contains 関数の部分は，任意文字列検索ではなく単語検索(単語として切り出された箇所と照合)で結果を求めた．これは「転置索引の単一エントリの検索」という基本的な検索処理の効率・性質を調べるため，および，全文検索処理の計算量を減らし，構造検索部分の計算量の変化がより顕著に現れるようにするためである．

表３に検索条件として与えた XPath 式と総検索時間（辞書検索時間＋索引検索時間），索引検索時間，および照合箇所数を示す．

表３：検索時間

	XPath 式	総検索時間（秒）	索引検索時間（秒）	照合箇所数
1	//*[contains(., ”,”)]	0.08	0.06	158457
2	//16[contains(., ”,”)]	0.06	0.03	24399
3	/0/1/3/6/*[contains(., ”,”)]	0.07	0.04	21322
4	//*[contains(., ”する”)]	0.18	0.09	154972
5	//7[contains(., ”する”)]	0.11	0.06	41
6	/0/1/3/6/8/9/*[contains(., ”する”)]	0.09	0.04	71
7	/0/1/3//*	0.22	0.22	801882
8	/0/1/3	0.1	0.1	166851
9	/0//3	0.09	0.08	166851
10	//30/@A392	0.04	0.04	16549
11	//*[contains(@*, ”0”)]	0.09	0.06	26025
12	//30[contains(@*, ”0”)]	0.09	0.07	2667
13	/0/1/3/7/16/16/*[contains(@*, ”0”)]	0.09	0.06	2667

表３の結果から，１ＧＢを超える大規模な文書に対し，

・構造を限定しない全文検索条件
・構造を限定した全文検索条件
・構造だけの検索条件

など「*」「//」が現れる場合を含め，極端に性能が低下することなく実用上十分な検索速度が得られていることがわかる．

さらに，構造を限定しない全文検索条件である式１や式４の検索時間よりも，それらに構造条件が加わった式２，式３，および式５，式６の方が検索時間が短くなっており，4.の例１で述べた本方式の有効性を実際に示す結果となっている．

6. 結論

日本語の全文検索を含む幅広い転置索引方式を，本来の検索機能と性能を保ちつつ，構造化検索に対応させる手法を述べた．

転置索引方式の一つである極大単語索引方式に本手法を適用し評価実験を行った結果，索引サイズに関するオーバーヘッドは１．７倍程度，検索速度についても十分実用になるレベルであることがわかった．

今後は，より広い範囲のＸＭＬ文書での評価を行うとともに，さらに効率的な索引の構造，更新手法の開発など，実用化にあたっての課題を解決していく．

参考文献

[1] 吉川正俊．XML の問い合わせをめぐる最近の話題．情報知識学会誌 Vol.10, No.3, pp59-64, 2000.

[2] http://www.exist-db.org/

[3] 難波功，井形伸之，小櫻文彦，山根康男．大規模 XML 文書の検索と格納技術の開発．情処研報，DD27-3, pp17-24 (2001)

[4] 菊池忠一．日本語全文検索用高速全文検索の一手法．電子情報通信学会論文誌，Vol.J75-D-I，No.9, pp836-846 (1992)

[5] 倉知一晃，野口直彦，菅野祐司，稲葉光昭．日本語文書に対する新しい索引検索方式－索引作成と検索の原理－．情処第 50 回全国大会予稿集(4), pp41-42 (1995)

[6] 菅谷奈津子，川口久光，畠山敦，多田勝己，加藤寛次. n-gram 型大規模全文検索方式の開発－インクリメンタル型 n-gram インデクス方式－．情処第 53 回全国大会予稿集(3), pp235-236 (1996)

[7] 川口久光，菅谷奈津子，畠山敦，多田勝己，加藤寛次. n-gram 型大規模全文検索方式の開発－文字種適応型 n-gram インデクス方式－．情処第 53 回全国大会予稿集(3), pp237-238 (1996)

[8] 福島俊一，赤峯享．高速全文検索のためのフレキシブル文字列インバージョン法(1)方式概要．情処第 53 回全国大会予稿集(3), pp239-240 (1996)

[9] 赤峯享，福島俊一．高速全文検索のためのフレキシブル文字列インバージョン法(2)実装と評価．情処第 53 回全国大会予稿集(3), pp241-242 (1996)

[10] 森川裕章，浅川達哉，有村博紀．データストリーム処理のための効率良い XPath 問い合わせ機構．信学技報，TECHNICAL REPORT OF IEICE. DE2003-35, pp19-24(2003)

[11] 竹田正幸，宮本哲，石野明，辻寿嗣．高速一方向逐次処理技術に基づく XML 文書の検索と変換．情処研報，DD41-8, pp51-58 (2003)

[12] Praveen Rao, Bongki Moon. PRIX:Indexing And Querying XML Using Prufer Sequences. The 20th ICDE(2004)

[13] Haixun Wang, Sanghyun Park, Wei Fan, Philip S. Yu. ViST:A Dynamic Index Method for Querying XML Data by Tree Structures. SIGMOD 2003