# 索引サイズと検索語数を考慮したVLCA用の索引構成と検索処理

小林　径† 横田　治夫†

† 東京工業大学 大学院 情報理工学研究科 計算工学専攻
E-mail: kei@de.cs.titech.ac.jp,yokota@cs.titech.ac.jp

**あらまし** XML 文書キーワード検索では，全検索語を含む適切な部分木を抽出することが重要である．そのための方法の 1 つに，VLCA(Valuable LCA) を根とする部分木を抽出する方法が提案されているが，それを抽出する手法は性能に問題がある．この問題に対し我々は効率的な VLCA 探索のために，シグネチャを用いた索引構成を提案し効果を示してきたが，生成する索引サイズと検索語数の増加への対応に課題が残っていた．本稿では，索引サイズを削減する方法として，低頻出語にはシグネチャ値を置かない方法を試行する．また，検索語数増加に伴う検索時間の増加を軽減する手法の検討を行い，既存手法 VLCAStack との比較実験を行うことで，その効果を示す．

**キーワード** XML キーワード検索,LCA,VLCA, シグネチャ索引

# Index Construction Methods for Effective VLCA Detection Improving Index Sizes and Performance for Multiple Keywords

Kei KOBAYASHI† and Haruo YOKOTA†

† Department of Computer Science, Graduate School of
Information Science and Engineering, Tokyo Institute of Technology
E-mail: kei@de.cs.titech.ac.jp,yokota@cs.titech.ac.jp

## 1. はじめに

近年，XML(Extensible Markup Language) 文書の増加に伴い，XML 文書から必要な部分のみを漏れなく検索したい，というユーザの要求も増大している．この要求に対し，我々は XML 文書に対しキーワード検索によって必要な部分を取得する手法に着目する．キーワード検索を用いるメリットとしては，その扱いやすさ，使用の簡単さにある．ユーザは検索ワードを入力するだけで，結果としてキーワードを含む部分を得ることができる．従って，専用の問い合わせ言語 (XPath [3]，XQuery [4] 等) の事前学習が不要であること，また，検索する文書の構造情報を事前に知っておくことが不要である点が利点である．

XML は，対応するタグの入れ子構造から木構造とみなすことができる．このことから，XML 文書に対するキーワード検索は，与えられた検索語からそれに関連する部分木を抽出する問題とみなすことができる．検索語のみからそれに関連する部分木を得る 1 つの方法として，全ての検索語を含む部分木である，木構造中の各キーワードを含むノードの最も近い共通祖先である LCA(Lowest Common Ancestor) を根とする部分木を抽出する手法 [11] [7] がよくとられてきた．しかし，得た LCA を全て提示することは，全ての検索語を必ず含む点では有効であるが，その適合率に課題がある．

この問題に対する提案の 1 つに，中間ノードのタグ名情報を考慮した LCA である VLCA(Valuable LCA) [8] がある．VLCA は，適合率と F 値の面で，LCA および他のいくつかの手法より優れることを文献 [8] で示している．しかし，その VLCA を抽出する手法である VLCAStack は，高頻出語での検索性能が十分でないという問題があった．

この問題に対して，我々はこれまでに，効率的に VLCA を抽出するための手法である BitMapKBSI 手法の提案を行ってきた．BitMapKBSI 手法では，LCA が VLCA かどうかを判定する際必要な，葉ノードからそのノードに至るまでに出現したタグ名に対応するビット列の論理和を，あらかじめ索引に格納する．このような構成の索引を用いることで，VLCA の判定を VLCAStack より効率的に行うことが可能になり，検索速度が向上する．実際に BitMapKBSI 手法と，既存手法 VLCAStack との比較実験を行いその効果を示した [14]．また，その際に作成する索引データサイズが大きくなる課題があるため，文献 [15] ではその削減手法を提案した．

[15] で提案した各手法は，データサイズは削減したが検索速度が BitMapKBSI 手法に比べて劣る．そこで本稿では，性能同程度に維持しつつ，索引サイズを削減する新たな手法として，

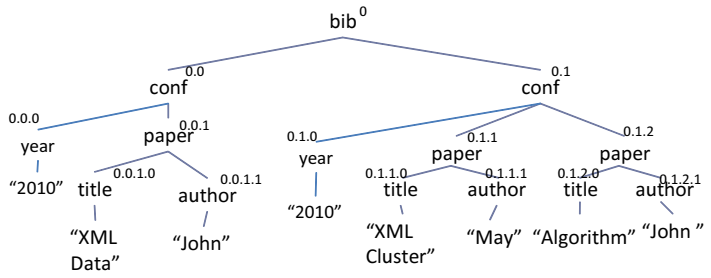


図 1　XML 文書の例

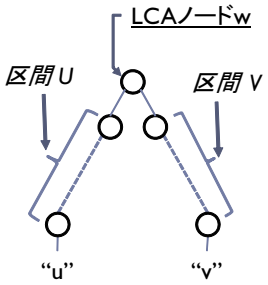


図 2　区間 U,V の木構造中の位置

低頻出語にはシグネチャの値を置かない構成方法を試行する．

また, [14] で提案した検索方法では，検索語数 3 以上の場合の適用に課題があった．なぜなら，タグの重複出現と，VLCA 判定を行うノードへ至る手前でのパスの合流によって，同一ノードのタグ名を含む場合とを区別できないためである．これらを区別する処理を行うことで検索語数 3 以上の場合の VLCA 抽出を行うことが可能になる．しかし，区別する処理を全ての場合について行うと性能が著しく劣化してしまう課題がある．

本稿では，検索語数の増加に伴う検索時間の増加を軽減するための手法を提案する．その方法は，タグ名のビット列の AND 演算を行い，その結果から VLCA にならない候補を特定し，タグ名の重複出現の有無を区別する処理を行う回数を減少させるための枝刈り処理を行うことで，検索時間の増加を抑制する．

これらの手法について，データサイズについては削減前後での比較，性能については既存手法 VLCAStack との比較を行うことで本稿の提案手法の有効性を検証する．

最後に本稿の構成を述べる．2 節では，本稿の提案手法の抽出対象である VLCA について説明し，既存手法 VLCAStack の課題について述べる．3 節では，VLCA 抽出効率化のために我々が提案してきた BitMapKBSI 手法の概要を述べる．4 節では，前半では本稿で試行するデータ削減手法の説明を行い，後半で枝刈り処理手法の提案を行う．5 節では実装実験を行い本稿の提案手法の評価を行う．6 節では本稿のまとめと今後の課題について述べる．

## 2. 研究の前提

本節では，本稿の抽出対象である VLCA [8] の説明，および抽出手法 VLCAStack とその課題についての説明を行う．

### 2.1 VLCA(Valuable LCA)

ここでは，本研究で着目した，提案手法の抽出対象である VLCA [8] についてその概要を説明する．最初に，LCA の概念を導入した後に，VLCA についての説明を行う．

XML 文書は，タグ付けされた部分文書に包含関係を持つことから，図 1 のように，木構造で表現することができる．なお，木の各ノードには DeweyOrder [10] によるラベルを付す．

m 個の葉ノードの LCA とは，それらの最低の共通祖先のノードである．XML キーワード検索では，LCA を根とする部分木を抽出することで，必ず全検索語を含む部分文書をユーザに提示することができる．ただし，LCA を根とする部分木を全て抽出し提示することは，その適合率に課題がある．この課題に対する提案の 1 つとして VLCA がある．

次に，VLCA についての説明を行う．LCA が VLCA であるとは，2 つのノード u,v とそれらの LCA ノードを w とした時に，w-u 間，w-v 間 (図 2 の区間 U,V に相当) に出現するノードの中に，同名タグを持つノードが存在しないことである．

次に，LCA と VLCA の例を示す．

例 1　LCA と VLCA の例

図 1 の文書に対して，2 つの検索語 XML, John で検索を行う場合を考える．最初に，LCA の例を与える．XML の出現するラベル 0.0.1.0 のノードと，John の出現するラベル 0.0.1.1 のノードの LCA は，ラベル 0.0.1 のノードである．同様にして得られる全ての LCA を列挙すると，ラベル 0.0.1, 0.1, 0 のノードである．

次に，VLCA の例を与える．前述の 3 つの LCA のうち，ラベル 0.0.1 の LCA については，タグ名の重複出現がないため VLCA である．ラベル 0.1 の LCA は，検索語を含むノードから LCA 間に同一タグ名 paper が出現 (ラベル 0.1.1 と 0.1.2) するため，VLCA でない．同様にして得られる全ての VLCA を列挙すると，ラベル 0.0.1 のノードのみである．

LCA の内で，ラベル 0.1, 0 を根とする部分木は，各検索語が別々の paper に属すため，検索語に対する結果としてふさわしくない．一方 VLCA であるラベル 0.0.1 を根とする部分木は，同一の paper に全検索語を含むことから，検索結果としふさわしい．この例からも見受けられるように，VLCA を根とする部分木を抽出することで，検索結果としてふさわしい部分を含みやすい．

#### 2.1.1 VLCAStack の問題点

VLCA を求める手法である VLCAStack が，文献 [8] で提案されている．この手法は，キーワードが出現する葉ノードラベルを，文書出現順に 1 回だけ Stack に格納することで，VLCA か否かを調べる組数を総当たりで行うより減らすことによって効率化した手法である．ただし，調べる組合せ数を減らしたことに伴い，全組合せについての VLCA を根とする部分木を抽出する場合に比べて再現率は低下する．また，高頻出語での検索時に性能が不十分である問題がある．その原因としては，特にあるノードの下に検索語を含むノードが集中する場合に，VLCA

表 1　VLCA 抽出手順と提案手法が用いる索引

| VLCA 抽出手順 | 用いる索引 |
|---|---|
| 1.　キーワードを含むノード取得 | B+tree 索引 |
| 2.　LCA の特定 | シグネチャ索引 (3.2.1 節で説明) |
| 3.　LCA が VLCA か判定 | タグ名のビット列 (3.2.2 節で説明) |

表 2　ワードシグネチャ割り当て例

| キーワード | ワードシグネチャ |
|---|---|
| Algorithm | 10001000 |
| Cluster | 01000100 |
| Data | 00001100 |
| John | 10010000 |
| May | 01100000 |
| XML | 00100010 |
| 2010 | 00010001 |

か否かを調べる組み合わせ回数が著しく増大するからである．

## 3.　BitMapKBSI 手法

2.1.1 で示した VLCA 抽出手法の性能の課題に対し，我々は BitMapKBSI 手法を提案し，VLCAStack との比較実験によりその性能向上効果を示してきた [14]．本節では，その手法の概要を述べる．

### 3.1　BitMapKBSI 手法の概観

VLCA を求めるためには，表 1 で示す 3 つの手順を経る．本手法では，表で示す通り，手順 1 で各検索語のエントリに含まれる検索語を含むノードを取得する．次に，2 番目の手順で LCA ノードを得る．その際，ノードに対応するシグネチャ索引を用いることで，LCA ノードを得る処理の効率化を行う．最後に，3 番目の手順で，前手順で得た LCA ノードが VLCA であるかを判定する．その際，LCA が VLCA か否か調べるために必要なノードタグ名に対応するビット列の索引を用いることで処理の効率化を行う．以上の 3 つの事項のうち，シグネチャとタグ名のビット列の索引構成法を次節以降で説明し，その後に構成した索引を用いた VLCA 探索手順を述べる．

### 3.2　BitMapKBSI 手法の索引構成法

#### 3.2.1　ノードシグネチャの構成方法

BitMapKBSI 手法は，表 1 の手順 2 においてシグネチャ索引を用いて LCA を得る．その構成方法の概要は以下である．

最初に，文書に出現する全ノードに対し，ノードに対応するシグネチャ値を計算する．そして，計算したシグネチャ値を B+tree 索引中に格納する．

XML ノードに対応するシグネチャ値の計算法，および B+tree 索引へ格納する方法について，我々が提案した XML 文書にスーパーインポーズドコード [5][6] とキーワード B+tree 索引を組み合わせた手法である KBSI 手法 [9][12] を用いる．

以下では順を追ってその方法について説明する．

1.　各ノードに対応するシグネチャ値の計算

　最初に，葉ノードのテキストに出現する全ての語について，長さの等しいビット列を割り当てる (表 2 は，図 1 の葉ノードに出現する全索引語についてシグネチャ値を与えた例である)．

　次に，文書中の各ノードに対応するノードシグネチャを計算する．葉ノードのノードシグネチャは，そのノードのテキストが含む全語のビット列の論理和である．中間ノードは，そのノードの持つ全ての子ノードのビット列の論理和である．これを，木の下位階層から根に向かって再帰的に計算を行うことで，全ノードに対応するノードシグネチャの値が計算できる．

2.　キーワード B+tree 索引への格納

　キーワード B+tree 索引の各語のエントリ中の，ラベルに対応するノードのノードシグネチャの値を，B+tree 索引のノードシグネチャ列に格納する．なお，キーワード B+tree は，各索引語をエントリとし，そのキーワードを含む葉ノード，およびそれらの先祖ノードラベルの値を，ラベル列に持つ．

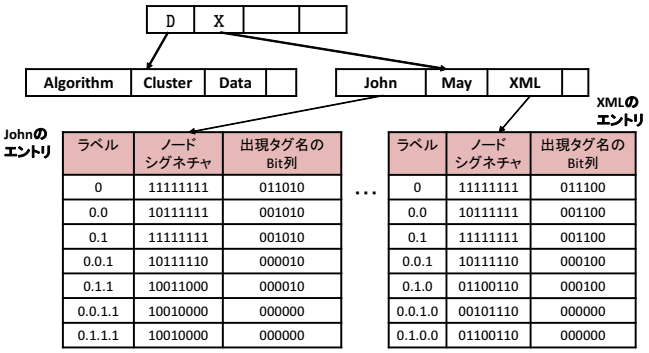


図 3　BitMapKBSI 手法の索引構成例

図 3 は，文書 1 に対して索引付けを行った例である．上記の説明に対応する部分は，図中のノードシグネチャの列である．例えば，ラベル 0.0 に対応するノードシグネチャの値は 10111111 であるが，この時ラベル 0.0 を持つタプルのノードシグネチャ列にこの値を格納している．

#### 3.2.2　タグ名のビット列の構成方法

表 1 の手順 2 で得た LCA ノードの VLCA 判定を効率化するためには，LCA ノードと，各検索語を含む葉ノードとの間に出現するノードのノードタグ名があればよい．以下では，ビットコード化したタグ名のビット演算による VLCA 判定方法，およびタグ名のビット列を索引へ格納する方法について説明する．

まず最初に，LCA ノード w が VLCA か否かを判定する処理をビットの AND 演算によって行うことで，VLCA 判定処理を効率化する方法について説明する [13]．それは以下の手順に従う．

1.　あらかじめ，XML 文書に出現する全てのタグ名に対して，ビットの 1 の位置が重ならないようにビット列を割り当てる (表 3 は，図 1 に出現する全ての XML タグ名にビット列を割り当てた例である)．

表 3　タグ名のビット列割り当て例

| タグ名 | ビットコード |
|---|---|
| bib | 100000 |
| conf | 010000 |
| paper | 001000 |
| title | 000100 |
| author | 000010 |
| year | 000001 |

2. ノード u と v の LCA ノード w が VLCA か否かを検証するために，u-w 間，v-w 間に出現する全ノードのタグ名が必要である．それに対応するのは，それらの区間のノードタグのビット列の論理和である．その値を PU,PV と呼ぶ．

3. PU と PV の AND 演算を行い，その結果から，区間 U,V におけるタグ名の重複出現の有無を判定できる．PU ∧ PV=0 ならば重複はないし，0 でなければ重複がある．

次に，タグ名のビット列を索引へ格納する方法について説明する．

先程の PU の値を，キーワード B+木索引に一緒に格納する．基本的には，索引語を含む葉ノードから索引付けするノード (図 2 の w とする) に至る区間 (図 2 の区間 U に相当) に存在するノードタグ名のビット論理和である (詳細については, [14] の 3.2 節を参照).

この操作を，全索引語の，全ラベルについて同様に行う．実際にこの操作を行い作成した例が図 3 のタグ名のビット列の項目である．このような索引の構成にすることで，従来はその都度逐一木を手繰って求める必要があったタグ名に関する情報を得る計算コストを削減し，表 1 の手順 3 を効率化する．

### 3.3　構成した索引による VLCA 探索手順

3.2 で構成した索引を用いて，検索語に対する VLCA を抽出する方法は以下の通りである．

1. 全検索語について，各検索語のエントリ中に含む情報を取得する．

2. 全検索語に対応するワードシグネチャの論理和であるクエリシグネチャQ を作成する．そして，検索語のエントリ中のノードシグネチャの値 NS と Q について，式 $Q = Q \wedge NS$ を満たす行のノードラベルを抽出する．全ての検索語のエントリで抽出されたラベルのノードが LCA ノードである．

3. 2. で得た LCA ノードラベルについて，同ラベル同士で「タグ名のビット列」項の値の AND 演算を行い，結果が 0 となるノードラベルが VLCA である．

次に例を述べる．図 3 の索引を用いて，2 つの検索キーワード XML，John の VLCA を求める場合の例は以下である．

例 2　図 3 の索引を用いた VLCA 探索例

1. 各検索語 XML，John のエントリ中にあるデータを得る．

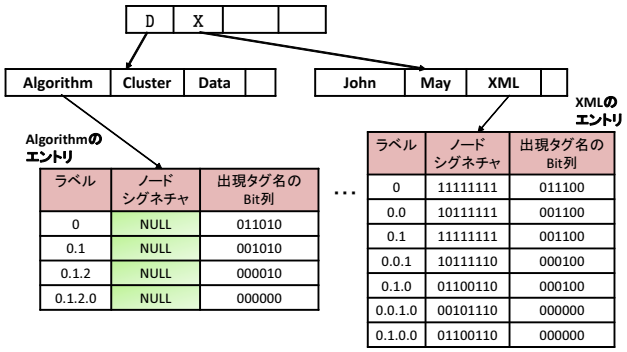


図 4　提案手法の索引構成

2. 表 2 のワードシグネチャを用いて Q を計算する．XML と John の値の論理和をとると，Q=10110010 となる．次に，Q と，図 3 の索引のエントリが XML および John についてノードシグネチャとの論理積の結果が Q になるラベルを全て抽出し，両語で出現するラベルを探索すると，ラベル 0, 0.0, 0.1, 0.0.1 である．

3. 前で求めた候補ラベル 0, 0.0, 0.1, 0.0.1 について，ラベルの等しいタグ名のビット列の項同士で AND 演算を行うと，0.0.1 のみが演算結果が 0 となる．従って，VLCA は 0.0.0 のみとなる．

## 4.　本稿の提案

本節では，本稿で試行する索引サイズの削減手法の説明，および検索語数の増加に伴う検索時間の増加を軽減する手法の提案を行う．

### 4.1　低頻出語にシグネチャ値を配置しないサイズ削減法

図 3 からもわかるように，作成する索引のデータサイズが大きくなるという課題があり，この解決のために，文献 [15] では，削減する部分を分析し，データサイズを削減した方法をいくつか考案した．本稿でも，新たなデータサイズ削減法として低頻出語での検索処理ではシグネチャを用いない方法を考案し試行する．

文献 [15] の実験において，シグネチャを用いない手法の性能が，検索語が低頻出語の場合にはシグネチャを用いた場合と同程度であった．なぜなら，文書中の検索語の出現回数が少ない場合は，シグネチャの有無に関らずごく短時間で表 1 の手順 2 の LCA の特定を完了するからである．従って，低頻出語での場合はシグネチャ値を置かない索引構成をとることで，性能は同程度のまま，索引データサイズを削減することが可能になる．図 4 は，出現数=1 の語 (Algorithm) についてシグネチャ値を置き換えた例である．本稿では，図 4 のように，ある語の出現ノード数が閾値 n 以下の語のエントリのシグネチャ列について，Null 値を設定する，という構成をとった．

#### 4.1.1　VLCA 探索手順

VLCA 探索は，基本的に 3.3 節で述べた 3 手順に沿って行う．変更点は，3.3 の探索手順 2. において，シグネチャ列の値が Null ならば，検索語のエントリに含まれるカラムを全て取

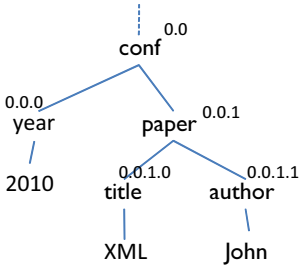


図 5　3 語の VLCA の例

得する操作をする点のみである．

### 4.2　検索語数の増加への対応

3 節では，効率的な VLCA 抽出手法を説明した．しかし，この手法を検索語数 3 以上の場合に適用する場合，単純な方法では語数の増加に対する検索時間の大幅な増加を伴う課題がある．この課題に対して，本節では語数の増加に対する枝刈り処理を提案する．

#### 4.2.1　m 語の検索語に対する VLCA

最初に，m 語の検索語を含む m 個の葉ノードの VLCA について説明する．各検索語をそれぞれ含む m 個のノード $n_1,n_2,\cdots,n_m$ に関して，それらの LCA をノード w と呼ぶ．ノード w が VLCA であるとは，全ての 2 つのノード $(n_i,n_j)$ の組と，それらの LCA ノード u に対して，ノード u - $n_i$ 間，u - $n_j$ 間に同一のタグ名を持つノードがないという条件を満たす場合である．つまり，LCA ノード u が $n_i$ と $n_j$ の VLCA である場合である．次に例を示す．

例 3　m 語の VLCA の例

図 1 の文書に 3 語の検索語 John, XML, 2010 を与えた場合の LCA の 1 つであるラベル 0.0 のノードが VLCA であるかを検証する場合を考える．次に示す全 3 組のペア {0.0.0 , 0.0.1.0}，{0.0.1.0 , 0.0.1.1}，{0.0.1.1 , 0.0.0} の LCA が全て VLCA であれば，それらの LCA ノードであるラベル 0.0 のノードが VLCA である．最初に，{0.0.0 , 0.0.1.0} の組について，これらの LCA はラベル 0.0 のノードで，かつ 0.0〜0.0.0 間，0.0〜0.0.1.0 間に同一タグ名を持つノードが無いため，これらの LCA は VLCA である．同様の検証を，他 2 組について行い，それぞれ VLCA であることより，3 ノードの LCA であるラベル 0.0 のノードは VLCA であると判る．

#### 4.2.2　BitMapKBSI 手法の索引を単純に使用した検索と課題

最初に，3 節で提案した手法を 3 語以上へ単純に適応した検索手法を述べ，単純な方法の課題を提示する．なお，m 語の検索語を $kw_1,kw_2,...,kw_m$ とする．

1,2　3.3 節で述べた手順 1., 2. は同一の処理を行う．

3.　3.3 節で述べた手順 3. において，手順 2. で求めた LCA ノード w に対して，m 語のエントリから，ノード w のラベルに対応する m 個のタグ名のビット列を得る．それを P[$kw_1$],P[$kw_2$],⋯P[$kw_m$] とする．

全ての P[$kw_i$]，P[$kw_j$] の組に対して P[$kw_i$] ∧ P[$kw_j$]=0，となる w を VLCA として抽出する．

次に例を示し，発生する問題を示す．

例 4　例 3 の VLCA の検証を AND 演算で行う例

各検索語のエントリ中から，LCA ノード 0.0 に対するタグ名のビット列の値を得る．それぞれ，P[XML]=001100, P[John]=001010，P[2010]=000001 である．これら 3 つから 2 つを選んだ全組に対して AND を行うと，P[XML] ∧ P[John] ≠ 0 であるため，VLCA でないと判定される

ラベル 0.0 のノードは，例 4 では VLCA でないと判定されるが，定義では VLCA である．この問題が発生する原因は，XML を含むラベル 0.0.1.0，および John を含む 0.0.1.1 から，0.0 に至るパス上に同一のノード 0.0.1 が存在するから，タグ名の重複有りと誤認されてしまうからである．

この false negative の問題を回避するために，P[$kw_i$] ∧ P[$kw_j$] ≠ 0 の場合にさらなる検証が必要になる．

4.2.2 節で示した手順 3. で，P[$kw_i$] ∧ P[$kw_j$]≠0 ならば，次の検証を行うようにする．

検証 1:　最初に，P[$kw_i$] ∧ P[$kw_j$] の bit の 1 の数を調べる．次に，語 $kw_i$,$kw_j$ を含む葉ノードラベルと，検証対象の LCA ノードラベルを用いて，パス上に共有するノードの数を調べる．それらが等しい場合に，タグの重複出現が無い，と判定する．

例えば，上記 P[XML] と P[John] の場合だと，AND の結果の 1 の数=1 で，葉ノードラベルと候補ノード 0.0 より，パス上の共有ノードが 1 つとわかる．これらが等しいから，LCA までのタグ名の重複が無いことが判る．この処理を行うことで，正しい結果が得られるが，計算コストが大きくなる．なぜなら，実際には大半の場合で AND 演算が ≠0 であるため，ほぼ毎回葉ノードまで見る必要があるからである．従って，この検証を行う回数を減らす必要がある．

#### 4.2.3　枝刈り手法の提案

ネックとなる 4.2.2 節の検証 1 を行う回数を減らすために，VLCA に成り得ないものを除外したい．そのために，m 語の VLCA が満たす十分条件を用い，これを満たさない場合は検証 1 を行わないことで，全ての場合で検証 1 を行うより効率化できる．

最初に m ノードの VLCA が満たす条件を示す．

条件 1:　ある m 個のノード $n_1,\cdots n_m$ と，それらの LCA ノード w があるとする．w が VLCA であるならば，全てのノード $n_i$ について，ノード w と 2 ノードの VLCA を構成するあるノード $n_j$ が 1 つは存在する．

この条件を用いて，4.2.2 節で示した処理の手順 3. を次のように変更する．

3-1　手順 2. で求めた LCA ノード w に対して，m 語のエントリ

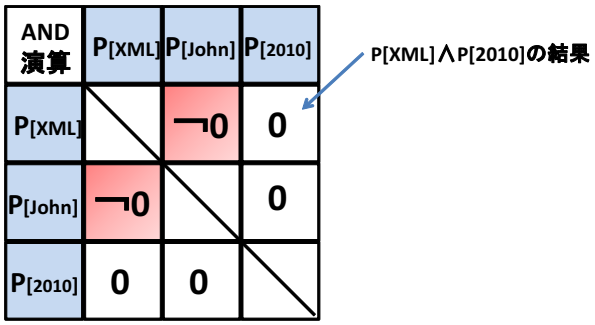


図 6　VLCA 検証用マトリクスの例

から，それに対応する m 個のタグ名のビット列を得る．それを P[$kw_1$],P[$kw_2$],…P[$kw_m$] とする．全ての P[$kw_i$]，P[$kw_j$] の組に対して P[$kw_i$] ∧ P[$kw_j$] を計算する．(丁度，図 6 のようなマトリクスに対し，クロスする P[$kw$] の値どうしの AND 演算を行い，その結果が 0 か非 0 かを順次記入する操作に相当する．)

3-2　マトリクスが全て埋まった後に，それを用いて次のような判断を行う．

Case1: 結果が全て 0 ならば，VLCA である．

Case2: 全ての行について，どこか 1 つでも 0 が存在する場合に，結果が ≠0 であった組に対してのみ，4.2.2 節の検証 1 を行う．この操作は，マトリクスの全ての横列に 0 が少なくとも 1 つあることを探す操作に相当する．

3-2 の Case1 を満たす場合は，w が m ノードの VLCA になる可能性がある．なぜなら，w で VLCA を構成するノードが全ての語について存在するからである (条件 1 に相当)．

逆に，case1 を満さない場合は，VLCA に成りえない．そのため，これを満さない LCA の組についての 4.2.2 節の検証 1 の処理を行わなくてすむため，全ての LCA の組に対しそれを行う必要があった単純な方法に比べて効率的である．

最後に，枝刈り手法を用いた検証の例を示す．

例 5　VLCA 検証の例

検索語 XML, John, 2010 でのラベル 0.0 が VLCA か否かを検証する．最初に，例 4 中の各 P の値を用いて，図 6 で示す値が入る．図 6 の表の全ての横列について，1 つは 0 が存在するかを検証すると，1 行目: (P[XML], P[2010]), 2 行目: (P[John], P[2010]), 3 行目: (P[2010], P[XML]) と全て存在する．従って，この場合は，≠0 の組合せ (図 6 の赤く塗ったマス) に対し 4.2.2 節の検証 1 を行ない，重複がないと判定される．結果として 0.0 が VLCA と判断される．

## 5.　評 価 実 験

本稿で提案した手法の有効性を調べるために，4 節で述べた手法，および対抗手法である VLCAStack を実装し，その性能と，作成する索引データサイズについての比較実験を行う．

表 4　実 験 環 境

| | |
|---|---|
| CPU | Quad Core Intel Xeon processor 5570 2.93G *4 |
| OS | CentOS 5.4 |
| Memory | 4GB*4 |
| Java | 1.6.0_02 |
| Database | PostgreSQL 8.4.2 |

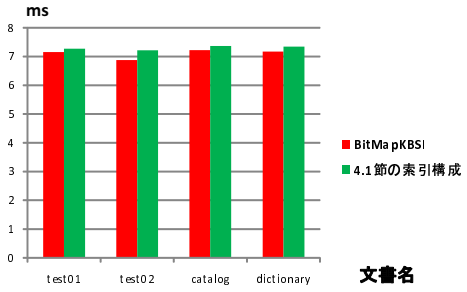


図 7　各文書での検索時間

| 文書名 | 削減前のサイズ | 削減後のサイズ |
|---|---|---|
| test01.xml | 988.8MB | 918.6MB |
| test02.xml | 1972.9MB | 1898.1MB |
| catalog.xml | 372.5MB | 317.5MB |
| dictionary.xml | 710.4MB | 630.0MB |

図 8　各文書でのデータサイズ比較

### 5.1　実験環境と実験に使用する文書

実験に用いた計算機，および環境を表 4 に示す．ただし，全ての手法は single thread で実装し，シグネチャ長には 512bit を採用した．

実験で使用する文書は Xmark の XML 文書生成器 xmlgen [2] で生成した文書 test01(size=11.6MB)・test02(size=23.4MB) の 2 種類，および XBench の XML 文書生成器 ToXgene [1] で生成した文書 catalog.xml(size=10.6MB)・dictionary.xml(共に 10.6MB) である．

また，検索語の出現頻度に応じた検索時間への影響を調べるために，出現する語をその出現回数に応じ低頻度 (1-50)，中頻度 (51-500 回)，高頻度 (501 以上) の 3 つに分類した．

### 5.2　低頻出語ではシグネチャを配置しない手法の実験

最初に，4.1 で述べた索引構成に対し評価実験を行う．その目的は，削減前の BitMapKBSI 手法と同程度の性能を維持しているかを確認することにある．

今回は，出現数 50 以下の語について，シグネチャ値に Null 値を代入し実験を行い，上記 2 文書について性能を測定した．比較対象は，通常の BitMapKBSI 手法，および 4.1 の手法である．低頻度語のみ，語数=2 の条件で 100 回検索を行い，その時間の平均値を測定した．

図 7 は，その結果である．全ての文書で，差がとても小さい (高々0.5ms 以下) ため，性能面では同程度であると言える．

次に，データサイズを測定し，削減前後の値を比較する．図

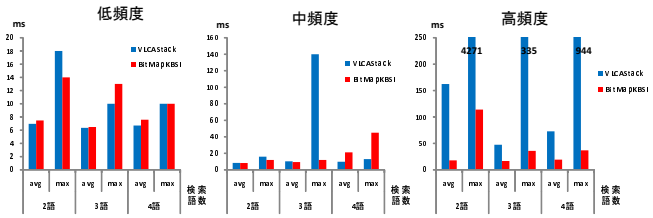


図 9　文書 test01 での検索時間比較

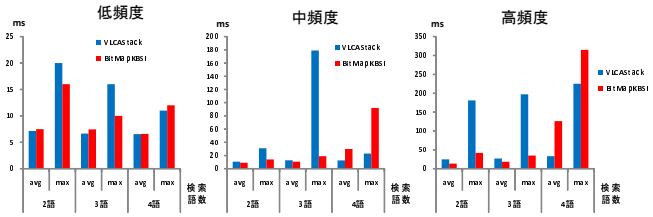


図 10　文書 test02 での検索時間比較

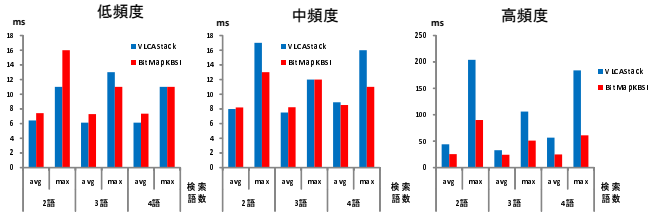


図 11　文書 catalog での検索時間比較

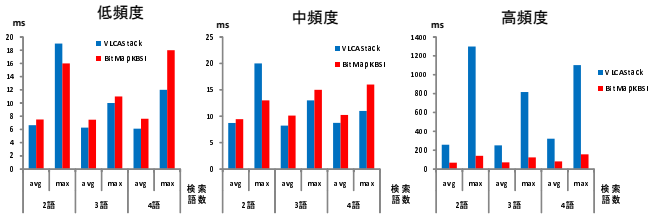


図 12　文書 dictionary での検索時間比較

8 は，その結果である．最低で 8%, 最高で 15%程度のデータサイズを削減している．

従って，この手法を適用することで，性能は同程度をキープしつつも，データ量を削減する効果が確認できる．

### 5.3　検索語数を増加した場合の実験

4.2 節で提案した語数の拡張手法に対して，検索語数の増加に対する性能の変化を分析するために，各頻度について，そのクラスに属す検索語を 2〜4 語与えた検索を 100 回行い，その平均値と最大値を比較した．なお，用いる索引はデータサイズ削減法を適用していない BitMapKBSI 手法で作成したものである．

図 9〜12 は，この実験の結果である．図中のデータラベル名 VLCAStack,BitMapKBSI は各手法の値であり，avg,max はそれぞれ平均値, 最大値に対応する．また，test01 の高頻度における VLCAStack の max のデータラベルに付した値は，その条件での VLCAStack の検索時間の最大値である．

検索語数=2,3 の場合は，全文書で平均値では低中頻度時は同程度，高頻度時に平均値，最大値とも VLCAStack に大きく勝

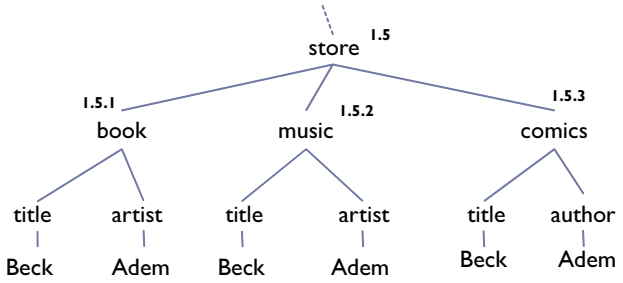


図 13　タグ名のビット列のパターン数の例

表 5　文書毎のタグ名のビット列のパターン数の最大値

| 文書名 | 種類数の最大値 |
|---|---|
| test01 | 98 |
| test02 | 150 |
| catalog | 17 |
| dictionary | 8 |

る結果を示した．4 語での検索の場合，dictionary,catalog の両文書では先程述べたのと同様の傾向を示した．

語数の増加に対する検索時間の増加率についても，次に述べる 3 例を除き，2 倍を大きく下回った．従って，適用する文書によっては枝刈り処理が効果的であったといえる．

4 語の中頻度の test01 と，4 語の中, 高頻度 test02 の場合については，VLCAStack に大きく劣る結果を示した．原因としては，図 6 で示したマトリクスを生成する回数が著しく増加したため，と考えられる．更に詳しい考察を次節で行う．

### 5.4　実験結果の検証

図 9,10 における，VLCAStack に対して検索時間で劣った場合の原因の検証を行ったところ，劣った場合とそれ以外の場合とでは，図 6 の例で示したマトリックスの生成回数が著しく異なることが分かった．

このことが起こる原因は，あるノード w に至る「タグ名のビット列のパターン数」が多いことに起因する.「タグ名のビット列のパターン数」とは，あるノード w に至るまでに出現するノードタグ名の種類数である．例えば，図 13 の場合，語 Adem を含む各ノードからラベル 1.5 のノードへ至る間に出現するタグ名のビット列のパターン数は，/book/title ,/music/title,/comics/title の 3 種類である．同様に語 Beck も 3 種類である．これらの値は，各語のエントリ下のラベル 1.5 に連結して格納されている．従って，ノード 1.5 が VLCA かを判定する際には，3*3 ＝ 9 回のマトリクス生成が必要となる．この例のように，検索語数，種類数共に小さな場合では影響は少ないが，パターン数が多い場合は，語数の増加に伴い回数が著しく増加する．例えば検索語数=4，各語の種類数=20 の場合だと $20^4$=160,000 回と多大になり，検索時間増加の要因になりうる．

実際に今回実験で使用した 4 文書に対してパターン数の最大値を測定した結果が表 5 である．同一の構造を持つ test01,test02 のみが突出して多い結果となった．Xmark のような人工のテストデータ生成器によって生成された XML 文書では，パターン数が極端に多いケースが見受けられたが，現実に使用される

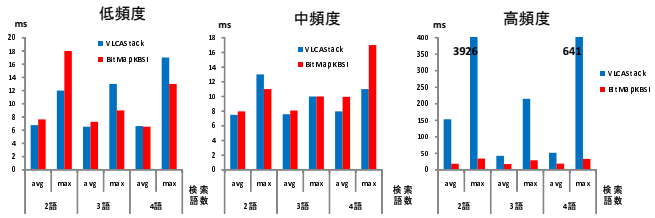


図 14　追加実験での文書 test01 の検索時間比較

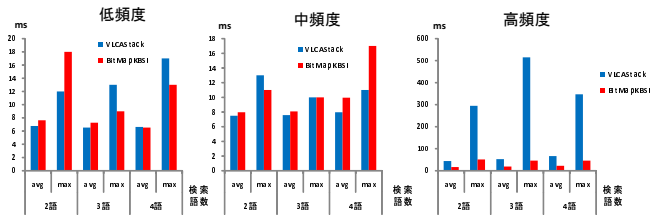


図 15　追加実験での文書 test02 の検索時間比較

XML 文書では同様のケースは発生しにくいと考えられる．なぜなら，同一の索引語は同じ意味で使う場合には同じ階層構造の下にあるため，このパターン数はそれほど多くならないからである．

次に，この仮説の正当性を検証するために，このパターン数の最大値に制限を設けた実験を行う．

### 5.5　タグ名のビット列のパターン数に制限を設けた実験

文書 test01,test02 を用いて，あるノードに至るタグ名のビット列のパターン数の最大値が 17 以下の検索語のみを使用する，という制限を設ける以外は節 5.3 と同様の環境と方法で実験を行う．なお，この 17 という値は表 5 の test01,test02 を除いた最大値から設定した．

図 14 と図 15 は，この実験の結果である．図中のデータラベル名 VLCAStack,BitMapKBSI は各手法の値であり，avg,max はそれぞれ平均値, 最大値に対応する．また，test01 の高頻度における VLCAStack の max のデータラベルに付した値は，その条件での VLCAStack の検索時間の最大値である．

この場合は，2 文書ともにで平均値では低中頻度時は同程度，高頻度時に平均値，最大値とも VLCAStack に大きく勝る結果を示した．検索語数の増加に対する検索時間の増加率についても，2 倍を大きく下回った．この傾向は 5.3 節の文書 catalog および dictionary の実験結果と同様の傾向を示した．

この結果より，語数を増加した場合の検索時間の増加率の増加には，ノードに至るタグ名のビット列のパターン数が大きく関与していることが判る．よって，本手法はこの種類数が大きくならない文書，検索語での適用が効果を発揮しやすいことがわかる．また，マトリクスの生成回数に課題があることから，これに対する改善が必要である．

## 6.　まとめと今後の課題

本稿では，VLCA 抽出の効率化のための手法である BitMapKBSI 手法について，新たな索引データ量削減手法として，低頻出語のエントリにはシグネチャを置かない方法を試行した．実験において，性能は削減前のレベルを維持しつつ，データ量を約 10%削減したことを示した．

また，検索語数の増加に対する対策として，最初に VLCA となるための条件を発見し，それを利用し VLCA になりえない LCA ノードを除外する枝刈り手法を提案した．実験において，2〜3 語の場合は VLCAStack に対し同程度または勝ることを示した．4 語の場合も文書によっては 3 語の場合と同様の傾向を示した．

今後の課題として，さらに大きなサイズの文書での実験を行い，文書サイズの増加に対する影響を観測したい．また，検索語数をさらに増加させた場合の影響も観測したい．さらに，実験において VLCAStack に劣ったケースをより詳しく分析することで，我々の提案手法である BitMapKBSI 手法を適用するのに効果的なケースを示したい．

## 謝　　辞

本研究の一部は，日本学術振興会科学研究費補助金基盤研究 (A)(#22240005) および文部科学省科学研究費補助金特定領域研究 (#21013017) の助成により行われた．

## 文　　献

[1] Xbench - a family of benchmarks for xml dbmss. http://se.uwaterloo.ca/ ddbms/projects/xbench/.

[2] The xml benchmark project. http://www.xml-benchmark.org.

[3] XML Path Language (XPath). http://www.w3.org/TR/xpath/.

[4] XQuery 1.0: An XML Query Language (Second Edition). http://www.w3.org/TR/xquery/.

[5] C.Faloutsos and S.Christodoulakis. Signature File: An Access Method for Documents and Its Analytical Performance Evaluation. In *ACM Transaction on Office Information Systems,No.4*, pp. 267–288, 1984.

[6] C.Faloutsos and S.Christodoulakis. Description and performance analysis of signature file methods for office filing. In *ACM Transaction on Office Information Systems,No.3*, pp. 237–257, 1987.

[7] Vagelis Hristidis and Nick Koudas. Keyword Proximity Search in XML Trees. In *IEEE Transaction on knowledge and data engineering,No4*, pp. 525–539, 2006.

[8] Guoliang Li, Jianhua Feng, Jianyong Wang, and Lizhu Zhou. Effective Keyword Search for Valuable LCAs over XML Documents. In *CIKM*, pp. 31–40, 2007.

[9] Wenxin Liang, Takeshi Miki, and Haruo Yokota. Superimposed Code-Based Indexing Method for Extracting MCTs from XML Documents. In *DEXA*, pp. 508–522, 2008.

[10] Igor Tatarinov, Stratis D.Viglas, Kevin Beyer, Jayavel Shanmugasundaram, Eugene Shekita, , Chun Zhang. Storing and querying ordered XML using a relational database system. In *Proceeding of the 2002 ACM SIGMOD international conference*, pp. 204–215, 2002.

[11] Y.Xu and Y.Papakonstantinou. Efficient keyword search for smallest lca in xml database. In *SIGMOD*, pp. 527–538, 2004.

[12] 三木健士, 横田治夫. 検索キーワードを含む最小 XML 部分文書抽出のための索引手法. In *DEWS2007*, 2007.

[13] 小林径, 横田治夫. Superimposed Code を用いた XML 中の Valuable LCA 探索手法. In *DEIM Forum 2009*, 2009.

[14] 小林径, 横田治夫. タグ名をビットマップ化した索引による効率的な Valuable LCA 探索手法. In *DEIM Forum 2010*, 2010.

[15] 小林径, 横田治夫. Valuable LCA 探索に用いる索引の データサイズの削減. 情報処理学会データベースシステム研究会, 2010.8.