Vol.2010-DBS-150 No.8
Vol.2010-IFAT-99 No.8
2010/8/4

Valuable LCA 探索に用いる索引のデータサイズの削減

小林 径[†]　　横田 治夫[††,†]

XML文書に対するキーワード検索は利点が多いが，適切な出力部分木を抽出することが重要である．XMLタグに出現する語を考慮することで，精度，F値の面で優れる部分木を抽出するVLCAを根とする手法が提案されている．しかし，これまで提案されているVLCA探索手法ではキーワードが増えた場合の抽出手法の性能に問題があった．我々は効率的なVLCS探索のための索引構築手法としてBitMapKBSI法を提案し，性能改善の効果を示している．ただ，これまでの実現方法では索引の容量の面で課題が残っていた．本稿では，索引の容量を削減するために，XMLノードのシグネチャを用いない方法，シグニチャの重複を除く手法，同一ノードのシグニチャ情報を共有する手法の3種類の改善手法を提案し，実験によってその性能および作成する索引データサイズを検証する．

Reduction of the data size of the index for effective Valuable LCA search

Kei Kobayashi[†] and Haruo Yokota[††,†]

The keyword search for XML documents has a number of benefit. It is important to derive appropriate output subtrees for the XML keyword search. A method deriving the precise output subtrees in which its root is VLCA by considering words appeared in XML tags. However, there is a problem of the performance to search for the VLCA in the case of increasing keywords. We have proposed BitMapKBSI as an index method to derive the VLCA effectively. However, there still remains a problem of index size. In this paper, we propose three improvement methods to reduce the index size: without signatures, removing duplication of signatures, and sharing the signatures for the same node. We evaluate the performance and index size for these three methods through experiments.

1. はじめに

近年，XML(Extensible Markup Language)文書の増加に伴い，XML 文書から必要な情報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 文書からキーワード検索を行うことで，情報を抽出する方法に着目した．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，結果を得ることができる．従って，専用の問い合わせ言語(Xpath[1]，XQuery[2]等)の事前学習が不要であること，また，事前に必要な，検索する文書の構造情報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では，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がとられてきた[3]．LCA を根とする部分木は，全ての検索語を必ず含む点では有効である．しかし，LCA それ自体では，ユーザにとって重要か否かを区別しないため，どの LCA が重要であるかが分からなくなる問題がある．

この問題に対しての提案の 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) がある[4]．VLCA は，精度と 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[4]で示している．しかし，その VLCA を抽出する手法は，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．

そこで，この問題に対して，我々はこれまでに，VLCA の判定をより効率的に行うことで，効率的に VLCA を抽出するための提案を行ってきた．文献[5]において，文書に出現するタグ名にビット列を割り当て，ビット演算を用いて LCA が VLCA か判定する手法を提案した．さらに，文献[6]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手法を提案し，既存手法 VLCAStack との比較実験によってその効果を示してきた．しかし，[6]で提案した手法は，作成する索引データサイズの大きさに課題があった．

そこで，本稿では，[6]で提案した方法をベースとして，かつ課題である索引データサイズを削減する方法を 3 つ考案し，その性能およびデータサイズを検証する．

[†] 東京工業大学 大学院情報理工学研究科 計算工学専攻
Department of Computer Science ,Graduate School of Information Science and Engineering,
Tokyo Institute of Technology
[††] 東京工業大学 学術国際センター
Global Scientific Information and Computing Center, Tokyo Institute of Technology

Vol.2010-DBS-150 No.8
Vol.2010-IFAT-99 No.8
2010/8/4

方法の１つ目は，VLCA 候補を得るために用いたノードに対するシグネチャを使わない方法である．[6]での分析から，索引データ量に対し，多くを占めるのはノードのシグネチャであった．従って，それを使わない方法によってデータサイズが大きく減少する効果を得ることができる．

２つ目は，ノードシグネチャの重複配置を減らす方法である．ノードラベルが定まればノードシグネチャも定まる．よって，ノードラベルをキーとして，ノードシグネチャを結合することで，同一ノードに対応するノードシグネチャの重複出現が無くなることによって，索引データサイズを削減する．

３つ目は，葉ノードとルートノード間のノードに関する情報を共有することで，重複を無くす方法である．同一の葉ノードに存在する全ての索引語のエントリに，その葉ノードからルートノードに至る間に出現するノードに対応する情報を格納していたため，重複があった．しかし，出現するノード自体は同一のノードであるため，それに対応する情報を,同一の葉ノードに出現する索引語のエントリ間で共有することで,索引データサイズを減少することができる．

これらの方法について，VLCAStack および BitMapKBSI 手法との比較実験を行い，その性能およびデータサイズについて評価を行った．

2. 研究の前提

本章では，本研究の手法の抽出対象である VLCA についてその概要を説明し，その問題点についても述べる．

2.1 Valuable LCA (VLCA)

ここでは，最初に LCA(Lowest Common Ancestor)について説明し，次に本稿で述べる手法の抽出対象である VLCA(Valuable LCA)の概要を述べる．

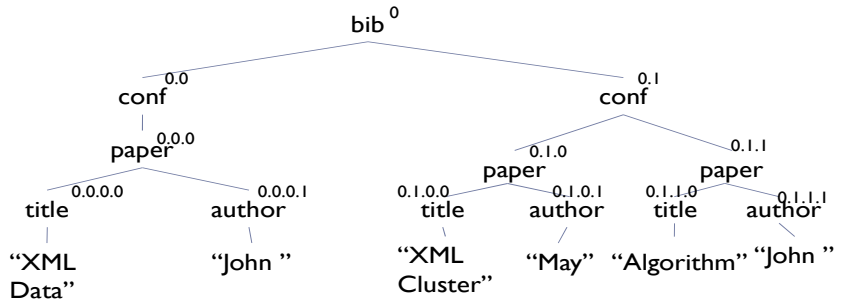

図 1 : XML 文書の例

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，木にはDeweyOrder[7]によるラベルを付す．

２つの葉ノードの LCA とは，それらの最低の共通祖先のノードである．LCA を根とする部分木は，必ず全検索語を含む．ただし，LCA の中で，どの LCA がユーザの情報要求を満たすか否かを区別しないため，精度面での課題がある．

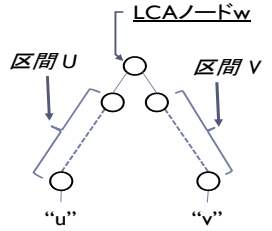

図 2 : 区間 U,V の木構造中での例

次に，VLCAの定義を述べる．ノードwを，2 つのノードu，vのLCAとする．ノードwがVLCAであるとは，ノードwとu間，wとv間に存在するノードに，同名タグをもつノードが存在しないLCAである．図 2で示した，区間U，Vおよびノードwの和集合に属すノードのタグ名に重複がない場合である．

例 1　LCAとVLCA

図 1 の文書に対して，2 つの検索語”XML”,”John”を与えた場合を考える．

最初に，LCA の例を与える．”XML”の出現するラベル 0.0.0.0 のノードと，”John”の出現するラベル 0.0.0.1 のノードの LCA は，ラベル 0.0.0 のノードである．同様に，XML の出現するラベル 0.1.0.0 のノードと，John の出現するラベル 0.1.1.1 のノードの LCA は，0.1 のラベルを持つノードである．

次に，VLCA の例を与える．前述の 2 つの LCA のうち，ラベル 0.0.0 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 0.1.0 と 0.1.1)するため, VLCA でない．

また，この時，ユーザは title に”XML”を含み，author が”John”の文献を探したい要求があったとすると, LCA を根とする部分木ではこれを満たさないもの(ラベル 0.1 および 0)も含むが，VLCA(0.0.0 のみ)ではこの要求を満たすもののみを漏れなく返しているのがわかる．従って，VLCA の方が良い結果を返す．

2.2 既存のVLCA抽出手法の問題点

VLCA を抽出する既存手法として，総当たり方式で VLCA を判定して抽出する Brute-Force アルゴリズムおよび，キーワードが出現する葉ノードラベルを，文書出現

Vol.2010-DBS-150 No.8
Vol.2010-IFAT-99 No.8
2010/8/4

順に 1 回だけスタックに格納することで, VLCA 判定を行う組数を Brute-Force 法より減らすことによって効率化した手法である VLCAStack 手法が，文献[4]で提案されている．検索効率面では，Brute-Force アルゴリズムよりも VLCAStack が勝る．しかし，両手法とも高頻出検索ワードでの性能が不十分である，という問題がある．

3. BitMapKBSI法

我々は，これまでに，VLCA 抽出手法の課題であった性能面を向上するための手法である BitMapKBSI 法[6]の提案を行い，VLCAStack との比較実験によりその有効性を示した．本章では，BitMapKBSI 手法の概要を述べる．

3.1 シグネチャを用いたVLCA候補ノードの抽出

BitMapKBSI 法は，最初に VLCA になり得る候補ノードを絞り込む．その際に，我々が提案した XML 文書にスーパーインポーズドコード[8],[9]とキーワード B+tree 索引を組み合わせた手法である KBSI 手法を用いる[10][11]．以下ではその概要を述べる．

3.1.1 シグネチャを用いた索引方法

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(例:表 1)．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．

表 1: ワードシグネチャ割り当ての例

キーワード	ワードシグネチャ
Argorithm	10000100
Cluster	00100010
Data	00000110
John	10010000
May	01100000
XML	00100001

例 2 ノードシグネチャの計算例

図 1の，葉ノード 0.0.0.0 のノードシグネチャは，テキストに語XML，Dataを含むことから, これらのワードシグネチャの論理和の 0010111 である. また, 中間ノード 0.0.0 のノードシグネチャは，このノードの全ての子であるラベル 0.0.0.0 と 0.0.0.1 のノードシグネチャの論理和で求めることができる．

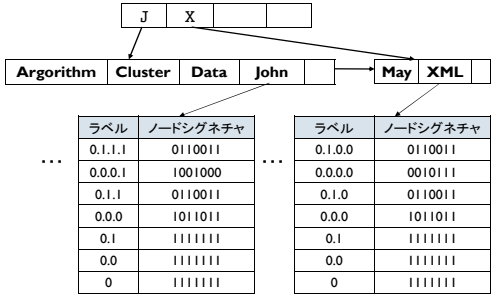

図 3: B+tree 索引の構成例

3.1.2 効率的にVLCA候補を得る手法

図 3のように，キーワードをエントリとし，それを含む葉ノードとそれらの先祖ノードラベルの値と，ノードに対応するノードシグネチャの値をB+tree索引に格納する．検索語が入力されたら，全検索語に対応するビット列の論理和であるクエリシグネチャQを作成する．次に，索引の各検索語のエントリにあり，かつノードシグネチャNSとQの論理積の結果がQとなるノードラベルを抽出する．

全検索語で抽出されたノードラベルは，そのノードを根とする部分木のテキストに全検索語を含むので，LCA である．また，全検索語のエントリを参照していることから，得られた候補ノードを根とする部分木は，確実に全キーワードを含むので，フォールスドロップは発生しない．(詳細は，[10][11]を参照)

3.2 VLCA判定を行うための索引の構成

3.1の方法で得たVLCA候補が，VLCAか否かを判定するために，VLCA候補と，検索語を含む各葉ノード間(図 2の区間U,Vに相当)に出現するノードタグ名が必要である．以下では最初に，効率化のために，ノードタグ名をビットコード化する方法を述べ，次に，ビットコード化したノードタグ名の情報を索引に格納する手法の概要を述べる．

3.2.1 タグ名のビットコード化とビット演算によるVLCA判定法

我々は，VLCA かどうかの判定の効率化のために，XML 文書に出現する全てのタグ名に対して，表 2のように 1 の立つ位置が重ならないようにビット列を割り当て，VLCAかどうかの判定に必要なタグ名情報をビット列として扱い，ビット演算で判定を行う方法を提案した[5]．ビット演算によるVLCAの判定は，次のように行う．

Vol.2010-DBS-150 No.8
Vol.2010-IFAT-99 No.8
2010/8/4

表 2：タグ名ビット列割り当て例

タグ名	bit列
bib	10000
conf	01000
paper	00100
title	00010
author	00001

(1) 最初に，図 2の区間Uに相当する部分に出現する全てのノードのタグ名のビットコードの論理和を取る．区間Vについても同様の操作を行う（それぞれの結果を順にPU,PVと呼ぶ）．
(2) PUとPVの論理積を取る．結果が 0 なら図 2のLCAノードwはVLCAである．0 でないなら，wはVLCAでない．

以下で例を用いて説明する．

例 3　ビット演算によるVLCA判定の例

図 1で，XMLを含むラベル 0.1.0.0 および，Johnを含むラベル 0.1.1.1 のLCAである 0.1 について考える．区間U，Vに存在するノードはそれぞれ，U={0.1.0，　0.1.0.0}，V={0.1.1，　0.1.1.1}であるから，PU，PVの値はそのノードの持つタグ名のビットコード論理和である．したがって，PU=00100∨00010=00110．PVも同様に計算し，00101となる．そして，これらの論理積が 00100 であることから，VLCAでないこと，およびタグpaperの重複出現が検出できることがわかる．

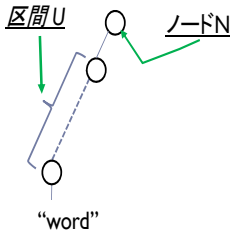

図 4：葉ノードとノード N の位置関係

3.2.2 ビットマップ論理和の格納

VLCA候補がVLCAか否かを判定するためには，得たVLCA候補ノードと検索語が存在する葉ノード間に存在するノードのタグ名に対応するビット論理和(3.2.1のPU,PV)があればよい．その値も図 3のB+tree索引に一緒に格納する．

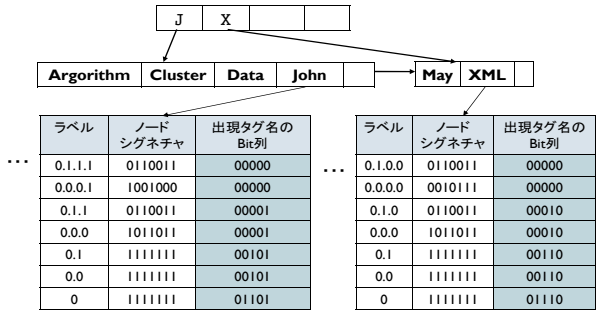

図 5：B+tree 索引にタグ名情報を追加した例

キーワード”word”のエントリに存在するノードラベルLに，タグ名のビット論理和を格納する．その値は，基本的には図 4の区間U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，[6]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 5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求める必要があったタグ名に関する情報を得るための計算コストを削減することになり，探索の効率化を実現できる．

3.3 BitMapKBSI手法

ここでは，3.2で構成した索引を用いて，検索語に対するVLCAを抽出する手法である BitMapKBSI 手法の手順は，以下のようになる．

(1) 3.1.2で述べた手法を用いてVLCA候補となるノードラベルを求める．
(2) (1)で得た候補ラベルについて，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を求める．

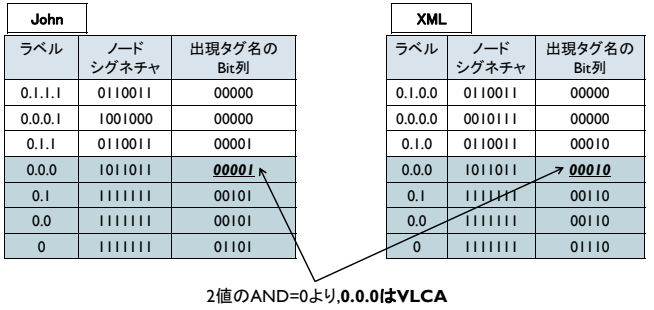

図 6：BitMapKBSI 手法による VLCA 探索の例

Vol.2010-DBS-150 No.8
Vol.2010-IFAT-99 No.8
2010/8/4

例 4　BitMapKBSI手法によるVLCA探索の例

図 5の索引を用いて，2 つの検索語”XML”,”John”のVLCAを求める

(1) 表 1を用いて問い合わせシグネチャQを求めると，Q=1011001 となる．次に，Q とノードシグネチャ項の値のANDがQになるラベルを全て抽出し，両方の語に出現するラベルを探索した結果が，図 6の色付けした行である．

(2) (1)で得た候補ラベル 0,0.1,0.0,0.0.0 のそれぞれについて，それらのタグ名のビット列項の値同士で AND 演算を行い，結果が 0 だったラベル 0.0.0 のみが VLCA であることがわかる．

3.4　BitMapKBSI手法の課題

[6]において，VLCAStack との比較実験を行うことで，BitMapKBSI 手法の効果を示した．しかし， 図 5からもわかるように，ラベルの他に 2 種類のビット列を保持する点，および各索引語が，葉ノードの全先祖ノード情報を保持すること，また，同一のデータの重複が多く存在すること，およびノードシグネチャのビット長が大きく，これが占める容量が大きいことから，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る．

4.　本稿で試行する方法

本章では，3.4で述べた索引データサイズの課題を解決するために，データサイズを BitMapKBSI 法より削減するための 3 つの方法について説明する．

4.1　ルートノードに関する情報の削除(全方法共通)

3 つの方法全てついて，ルートノードに関する情報を削除する．検索結果としてルートノードを与えることは，実際，ユーザにとって有用性の高いものではないためである．なお，以後の説明図中の横赤線は，ルートノードに対応する部分を削除したことを表している．

4.2　ノードシグネチャ値を用いない方法(NS法)

データ容量が大きくなる要因の一つに，ノードシグネチャの値がある．なぜなら，一般的にシグネチャのビット長の方が大きいからである．そこで，図 7のように，ノードシグネチャの値を用いない索引を構成することで，索引のデータサイズを抑える方法を考案する．

4.2.1 VLCA抽出手順

3.3(1)に代替する操作を，シグネチャなしで行う．最初に，検索語のエントリに含まれるカラムを全てスキャンする．全検索語のエントリに出現したノードラベルを，VLCA候補のノードラベルとして抽出する．以降の処理は，3.3(2)のVLCA判定方法と同様である．

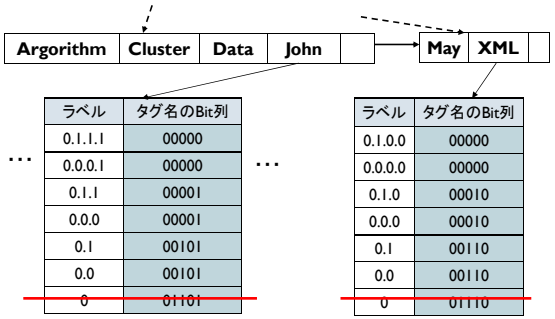

図 7：ノードシグネチャを用いない索引の構成例

4.3　シグネチャテーブル法(ST法)

同一ノードに対するノードシグネチャの値が，複数箇所に重複して存在する．例えば，図 5の中のノードラベル 0，0.0，0.1 等に，重複が見受けられる．この重複を無くすことによって，索引データサイズを削減する方法を提案する．ノードラベルが定まればノードシグネチャの値も定まることから，図 8のように，ラベルをキーとして，

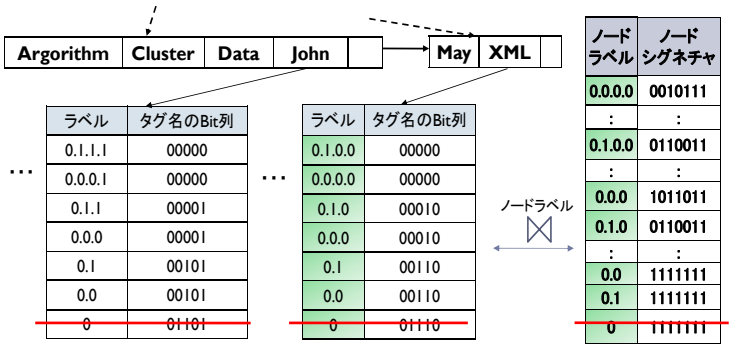

図 8：ノードラベルによる結合の例

パス bit 列項とノードシグネチャを結合する構造をとれば，ノードシグネチャの重複配置が無くなることで，索引データ総量を BitMapKBSI 法のものよりも抑えることが可能である．

4.3.1 VLCA抽出手順

最初に結合処理を行うことで，図 5と同等のデータを得ることとなる．従って，VLCA抽出の手順は，シグネチャ結合後は3.3の方法をそのまま適用する．

Vol.2010-DBS-150 No.8
Vol.2010-IFAT-99 No.8
2010/8/4

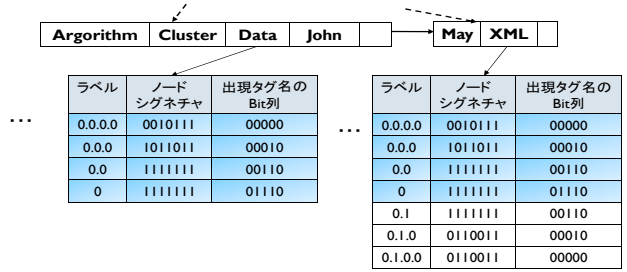

図 9：索引中の同一データ重複出現の例

4.4 ノード情報共有法(CN法)

葉ノードとルートノード間のノードに関する情報の重複を無くすことによって，索引データサイズを索引する方法を考案する．3.2で述べた構成法で作成される索引は，同一のリーフノードに存在する全ての索引語のエントリ中に，葉ノードからルートノードに至る間に出現するノードについての同一のシグネチャおよびタグのビット列の内容を重複して保持している．たとえば，図 9の中では，葉ノード 0.0.0.0 に出現する索引語”Data”,”XML”のそれぞれのエントリに，重複した内容の情報を保持しているのがわかる(図中の青色を付けた項).

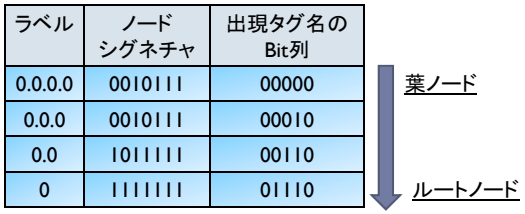

図 10 : 1 つの葉ノードとルートノード間情報の例

リーフノードラベルの値が定まれば，それらの各先祖ノードに至るまでに出現するタグ名がわかることから，まず，全葉ノードに関して，それぞれ図 10のように葉ノードからルートノード間に存在するノードについてのシグネチャ，および出現するタグ名のビット列(3.2.1のPU,PVの値に相当する)の値を作成する．

例えば，図 10は，葉ノードラベル 0.0.0.0 に関して作成した例である．

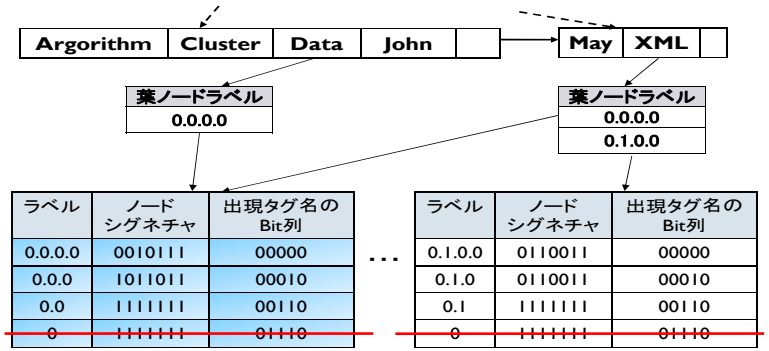

図 11：葉-ルートノード間情報を共有する索引構成例

同一の葉ノードに出現する索引語について，図 11 のように，同一の葉-ルートノード間についての情報を共有することで，図 9に見られるような，同一の内容の重複出現は発生しない．従って，使用する索引の容量を，BitMapKBSI手法のものよりも削減することができる．

4.4.1 VLCA抽出手順

4.4で述べた構成の索引を用いた場合のVLCA抽出の手順は，基本的には3.3での方法と同様である．

検索語のエントリ中に存在する，全ての葉ノード値に対応する，葉-ルートノード間の情報を入手する．その後の手順は，3.3と同様に，両検索語で出現するノードラベルの，タグ名のビット列同士をAND演算することで，VLCAを求めることが可能である．

4.5 性能と索引データサイズ

本章で述べた 3 つの方法は，いずれも索引データサイズの面で BitMapKBSI 手法より小さいが，性能では劣るものと予想される．

5. 評価実験

本稿で考案した手法の有効性を調べるために，4章で述べた 3 手法，BitMapKBSI手法，および VLCAStack アルゴリズムを実装し，その性能，および作成する索引データサイズによる比較をすることで，本稿の手法の評価を行う．

5.1 比較対象VLCAStack

VLCAStack は，検索語を含む葉ノードラベルを，文書出現の順に各 1 回のみスキャンし順次スタックに積む．従って，VLCA 判定をする組み合わせ数が総当たり法より少なく済むことで，効率性を高めている．しかし，VLCA 判定自体の効率化はされて

Vol.2010-DBS-150 No.8
Vol.2010-IFAT-99 No.8
2010/8/4

いないため，高頻度語での検索では，VLCA 判定回数が増加するので，検索性能が著しく悪化する問題がある．

5.2 今回の実験でのCN法の実装

今回の実験では，4.4節で述べたCN法の実装を，他手法と同様に関係データベースを用いて行うために，図 10のテーブルに，葉ノードラベルの列を付加し，値として葉ノードラベルを格納した．葉ノードラベル値をキーとして結合することで，葉ノード-ルートノード間の情報を得る方法を採用した．

表 3：実験環境

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

5.3 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 3に示す．ただし，全ての手法はシングルスレッドで実装し，ノードシグネチャ長を 4096bit とした．実験で使用する文書は，Xmark の XML 文書生成器 xmlgen[12]の規模変更子を 0.1 として作成した XML 文書である xmlgen01(size=11.60MB)を使用して実験を行う．

5.4 検索語の出現数の増加に対する各手法間の性能比較

検索語の出現頻度の増加に対する性能の変化を詳しく分析するために，次のような実験を行った．文書 xmlgen01 を用いて，検索キーワードが出現するノード数に応じて，与える検索語を，0-200，201-400，401-600，601-1000，1001-1400 の 5 つのクラスに分類する．各クラスについて，そのクラスに属す検索語を 2 語与えた検索を 100 回行い，その平均経過時間を測定した．

図 12は，この実験の結果である．NS法，CN法においては，検索時間の増加が，既存手法VLCAStackに対して著しく小さいことが見て取れる．従って，高頻出語での検索においては本稿の提案手法も有効であることが分かる．また，結合処理を含む方法であるTS法およびCN法の結果が，BitMapKBSI手法およびNS法に対してやや劣る傾向が見られる．

我々の提案した手法間での性能をより詳しく分析するために，経過時間をデータベース検索時間と判定処理時間の 2 つに分けた結果も分析する．なお，データベース検索時間とは，データベースクエリのターンアラウンドタイムを意味し，判定処理時間とは，全体の経過時間からデータベース検索時間を除いた時間を意味する．ただし，与えるクエリは各手法で異なる点，および経過時間が，データベース検索時間と判定処理時間の合計に相当する点に注意する．

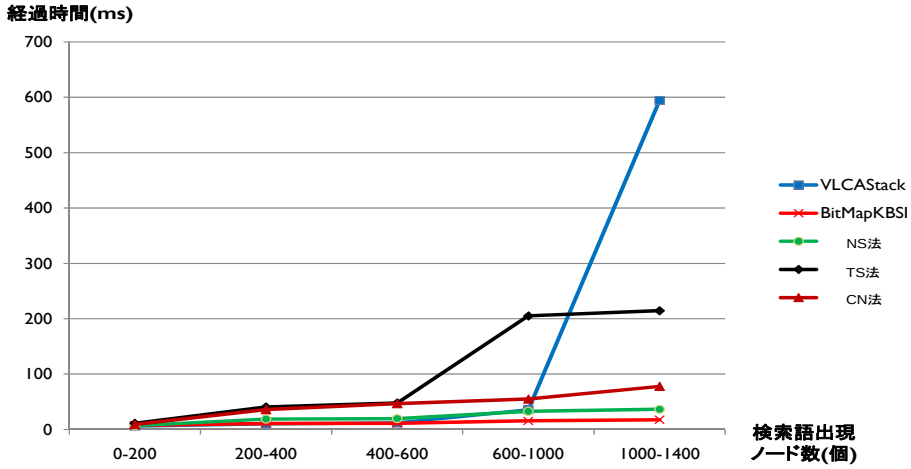

図 12：各クラスの検索語での平均経過時間

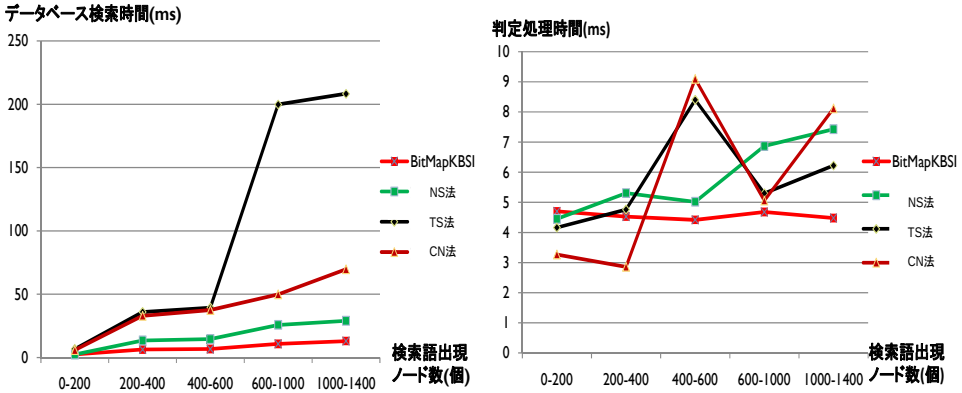

図 13：平均データベース検索時間と平均判定処理時間

図 13は，図 12の実験結果におけるクエリ経過時間と探索時間の値を示したものであ

Vol.2010-DBS-150 No.8
Vol.2010-IFAT-99 No.8
2010/8/4

る．クエリ経過時間では，結合演算を含むTS法，およびCN法の値が他の 2 つに比べて大きい．一方，探索時間で見ると，今回の実験では全てのケースにおいて 10ms以下と小さな値を示した．このことより，今回提案した 3 手法とも，検索語の出現頻度の増加に対して，VLCAStackより強いと言える．また，CN法については，クエリ経過時間を減らすような実装で実行することによって，より良い結果を示す可能性があると言える．

表 4：各手法の索引データサイズ

方法名	索引データサイズ
VLCAStack	**15.9MB**
BitMapKBSI	**3653.7MB**
NS法	**619.2MB**
TS法	**729.9MB**
CN法	**509.8MB**

5.5 索引データサイズの比較

表 4は，今回実験で使用した各手法で作成した索引データサイズである．なお，この数値は，postgreSQL の pg_total_relation_size の情報より取得した．

本稿で考案した方法は，いずれも BitMapKBSI 手法に対して NS 法では 83%，TS 法では 80%，CN 法では 86%と，大きな削減効果があった．ただし，元の文書サイズに対しては大きい．今回実験で用いた文書の場合，タグ名のビット列のビット長が 74 と比較的長いため，これが占める容量も大きいものとなった．より短いビット長の場合ならば，元の文書データに対してより小さな割合で済むのではないかと考える．

6. 本稿のまとめと今後の課題

本稿では，BitMapKBSI 手法をベースとして，かつ作成する索引データサイズを削減した手法を 3 つ考案し，その効果を試した．比較実験において，BitMapKBSI 手法より性能は落ちるが，VLCAStack よりも，高頻出語での検索時の有効性を確認した．また，データサイズの面からも，高い削減効果を確認した．

今後の課題として，より大きなサイズの XML データでの，性能と索引データサイズの変化を確認する必要がある．そのためには，パラメータによる分析，および実験的な観測を行う必要がある．また，本稿では試行できなかったが，方法 3 においては未だノードシグネチャの項に重複があるため，CN 法に TS 法を組み合わせた手法を構築したいと考える．さらに，使用したシグネチャの長さは，[11]の実験と同じ値を使用したが，この値の変化に対する影響を調べ，適切な設定を調べる必要がある．

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．

参考文献

1) XML Path Language (XPath) 2.0
http://www.w3.org/TR/xpath20/

2) XQuery 1.0: An XML Query Language
http://www.w3.org/TR/xquery/

3) Yu Xu and Yannis Papakonstantinou. : Efficient keyword search for smallest lcas in xml databases. Proc. of the 2005 ACM SIGMOD, pp.527-538, 2005.

4) Guoliang Li, Jianhua Feng, Jianyong Wang, and Lizhu Zhou. : Effective keyword search for valuable lcas over xml documents In CIKM, pp.31-40, 2007.

5) 小林 径, 梁 文新, 横田治夫. : SuperimposedCode を用いた XML 中の Valuable LCA 探索手法. DEIM2009, pp.B7-5, 2009.

6) 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. DEIM Forum 2010, C8-3, 2010.3.

7) IgorTatarinov，StratisD. Viglas，Kevin Beyer，Jayavel Shanmugasundaram，Eugene Shekita，and Chun Zhang. Storing and querying ordered XML using a relational database system. Proceedings of the 2002 ACM SIGMOD，pp. 204-215，2002.

8) Signature Files: An Access Method for Documents and Its Analytical Performance Evaluation. ACM Transactions on Office Information Systems, No. 4, pp267-288 ,October 1984.

9) C. Faloutsos and S. Christodoulakis. : Description and performance analysis of signature file methods for office filing. ACM Transactions on Office Information Systems，No. 3，pp. 237-257，July 1987.

10) Wenxin Liang, Takeshi Miki, and Haruo Yokota. : Superimposed Code-Based Indexing Method for Extracting MCTs from XML Documents. In DEXA2008，pp508-522，2008.

11) 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. DEWS2007，pp. C1-4，2007.

12) The xml benchmark project.
http://www.xml-benchmark.org