論文 / 著書情報
Article / Book Information

論題(和文)	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
Title(English)	An Effective Valuable LCA Detection Method using Index with Bitmap Codes for Tag Names
著者(和文)	小林径, 横田治夫
Authors(English)	Kei Kobayashi, Haruo Yokota
出典(和文)	, , , C8-3
Citation(English)	, , , C8-3
発行日 / Pub. date	2010, 3

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 (O)

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
†† 東京工業大学 学術国際情報センター
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における VLCA(Valuable LCA) 判定コスト減少のために，文書に出現する全てのタグ名をビットコード化し，葉ノードと当該ノード間に出現するタグ名に対応するビットコードの論理和として転置索引に格納する．VLCA 判定時のタグ名重複出現の検出にビット演算を用いることで効率化を実現する．評価のために，既存手法 VLCAStack との比較実験を行った結果，特により高頻出な語での検索において，提案手法は既存手法より特に良い性能を示した．

キーワード XML キーワード検索，LCA，VLCA

An Effective Valuable LCA Detection Method using Index with Bitmap Codes for Tag Names (O)

Kei KOBAYASHI† and Haruo YOKOTA††,†

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

Abstract To decrease costs of judging Valuable Lowest Common Ancestors (VLCAs) for XML keyword search, we propose a method of assigning bit map codes to all tag names appeared between leaf nodes and the node containing the keywords. The bit map codes are stored into each entry of the keyword in a B-tree index to reduce the recalculation costs for bit map. We have implemented and compared the method with the existing VLCAStack algorithm. The experimental results indicate that the proposed method outperforms the VLCAStack algorithm, particularly in the case of keyword appearance frequency is high.

Key words XML keyword search, LCA, VLCA

1. はじめに

近年，XML(Extensible Markup Language) 文書の普及に伴い，ユーザの XML 文書から必要な情報のみを効率的に抽出したいという要求も増大している．この要求に対し，XML 文書からキーワード検索によって情報抽出を行う手法に着目する．キーワード検索の利点は，ユーザがキーワードを入力するだけで結果を得ることができる点にある．よって，問い合わせ言語，文書構造の事前知識なしで検索を行うことが可能である．

XML 文書はラベル付き木構造を構成することから，キーワード検索による情報抽出に対して，全ての検索語を含む葉ノードの最低の共通祖先である LCA(Lowest Common Ancestor) を抽出する手法がとられてきた．LCA を根とする部分木は，全検索語を含むという点で有効であるが，文書規模が大きくなると，LCA 数も多くなるため，どの LCA が重要であるかが分かりにくくなる．この問題に対する提案の中の 1 つが，中間ノードのもつタグ名情報を考慮した LCA である VLCA(Valuable LCA) である [5]．VLCA は，精度と F 値の面で LCA よりも優れることが文献 [5] で示されている．しかし，その抽出手法は，高頻出なキーワードでの検索の場合に性能が十分でないという問題があった．

そこで，我々は，既存手法より効率的な VLCA 抽出手法を実現するために，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

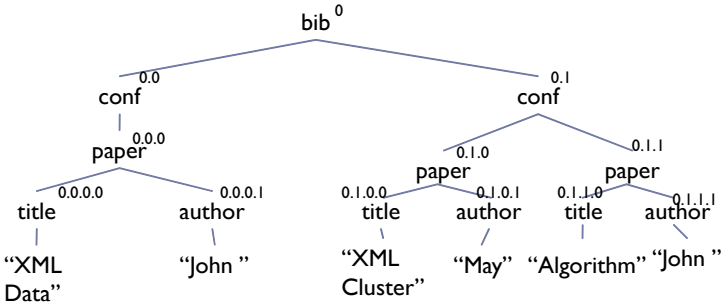

図 1　XML 文書の例

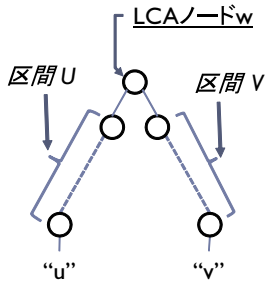

図 2　区間 U,V の木構造中の位置

表 1　タグ名のビット列割り当て例

タグ名	ビットコード
Bib	10000
conf	01000
paper	00100
title	00010
author	00001

した [9]．既存手法では文字列リストを用いて情報の保持および判定を行うが，同等の操作を全てビット列の論理演算のみで行うことができるため有効である．さらに，我々が以前提案した，LCA 候補を効率的に求める KBSI 手法 [6] [8] とビット列を用いる手法を組み合わせて VLCA を抽出する KBSITS 手法も提案した [9]．しかし，この手法は VLCA 判定の際に，タグ名をその都度求めていたため，計算コストが高いという問題があった．

本研究では，LCA が VLCA かどうかを判定する際，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する．このようにすることで，検索時の出現タグ名情報を得るコストが減少することから，VLCA 判定にかかる計算コストが減少するため，有効である．また，この手法の効率面の有効性を示すために，既存手法 VLCAStack [5] を実装し，比較実験を行った．

2. 研究の前提

本章では，提案手法を述べる上で必要な事項について説明する．

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 [5] 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．

例 1 (LCA の例):
図 1 の語 Cluster を含むノード 0.1.0.0 と，語 May を含むノード 0.1.0.1 の LCA は 0.1 である．

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 2 (VLCA の例) :
図 1 では，XML を含むノード 0.0.0.0 と，John を含むノード 0.0.0.1 の LCA ノード 0.0.0 は VLCA だが，逆に 0.1.0.0 と 0.1.1.1 の LCA 0.1 は，間に paper タグが重複出現するため VLCA でない．

2.1.1 VLCA を抽出する既存手法とその問題点

VLCA を抽出する既存手法としては，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 [5] 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．

2.2 文書に出現するタグ名のビットコード化

我々は，VLCA の判定に必要な，文書に出現する全ノードタグ名をビットコード化し，出現するタグ名をビット列で保持し，VLCA 判定をビット列の論理演算のみで行う手法を提案した [9]．以下でその説明をする．

2.2.1 タグ名のビットコード化

最初に，文書に出現する全タグ名を抽出する．次に，全タグ名について，1 の位置が重ならないようにビット列を割り当てる (例:表 1)．ビット長は，出現するタグの種類数に等しいが，その種類数は，ほとんどの文書では，テキストに出現する語の種類数ほど多くはならず限定的である ([9] 参照)．

2.2.2 ビット演算を用いた判定手法

PU，PV を，図 2 の区間 U と区間 V に出現する全ノードタグ名のビットコードの論理和とする．U と V のタグ名重複出現を確認するためには，PU と PV の論理積を取り，結果が 0 であれば，1 の位置が重ならないようにビット列を割り振ったため，重複出現がないのが判る．

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．

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
Argorithm	1000100
Cluster	0100010
Data	0000110
John	1001000
May	0110000
XML	0010001

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，タグ paper の重複出現が検出できる．

2.3　VLCA 候補を効率的に抽出する手法

VLCA 候補となるノードの絞り込みに，我々が提案した XML 文書にスーパーインポーズドコード [3] [4] 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 [6] [8]．以下ではその概要を述べる．

2.3.1　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 2)．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．

例 4 (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．

2.3.2　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，

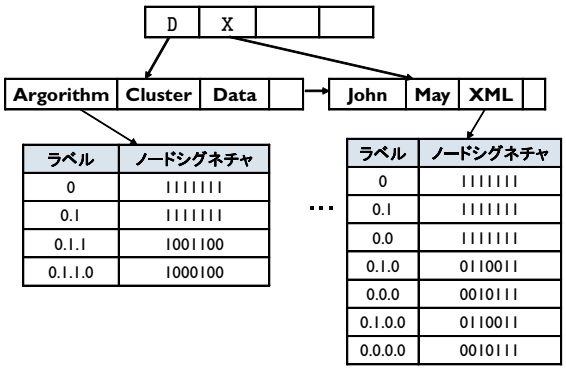

図 3　B+tree 索引の構成例

フォールスドロップは発生しない (詳細は, [6] [8] を参照).

2.3.3　これまでに提案した VLCA 抽出法

KBSI 手法を用いて得た各 VLCA 候補に対する VLCA 判定を，2.2 で述べたビット演算を用いる方法で行う KBSITS 手法を提案した [9]．さらに，この手法と，2.1.1 で言及した既存手法 Brute-Force アルゴリズム [5] との比較を行い，特に高頻出検索語での効果を示した．しかし，性能面で Brute-Force アルゴリズムに勝る VLCAStack [5] との比較は行っていなかったが，本稿では，提案手法と VLCAStack を比較対象とした実験を行う．

3.　本稿の提案

この章では，これまでに我々が提案した KBSITS 手法 [9] の改善案を述べる．次に，それに基づいた索引の構成法，およびそれを用いた VLCA 探索手法である BitMapKBSI 手法を提案する．

3.1　ビットマップ論理和の格納

KBSITS 手法では，得た VLCA 候補と各葉ノード間のタグ名のビット列の論理和 (2.2.2 の PU，PV に相当) を求める必要があるが，この値は事前に計算が可能である．従って，論理和の結果を索引に格納することで，検索時の計算コストを削減できるため，検索効率面で有効である．

3.2　キーワード B+tree 索引の拡張手法

B+tree 索引に，タグ名情報ビット列の論理和も一緒に格納する手法を提案する．その手順を以下で述べる．

キーワード kw のエントリに存在するノードラベル L に格納するビット列の計算法は，以下の 3 パターンに場合分けされる．なお，説明のため，ノードラベル L の指すノードを N と表記する．図 4 は，以下で場合分けする 3 通りの N の木の中での位置である．

(1) N が葉ノードの場合:
ビット長がタグ名ビットコードに等しく，値が全 0 のビット列を格納する．
(2) N が中間ノードの場合:
(2-a) 語 kw を含む葉から N に至るパスが 1 本の場合:
N に至るまでに出現したタグ名のビット列のすべての論理和の結果を格納する．
(2-b) 葉から N に至るパスが複数存在する場合:

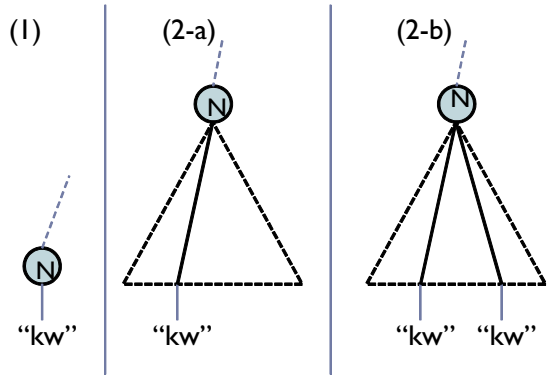

図 4　ノード N と葉ノードの木の中での位置関係

全てのパスについて 2-a と同様に論理和を求めた後，得た全種類の論理和演算結果を，連結して格納する．この操作を全語のラベルについて同様に行う．

例 5 (索引に格納する論理和演算の計算例) :
図 1 の文書を用いてその計算例を示す．索引付けを行うノードの各場合について述べる．
(1) の場合 :
0.0.0.1 は葉ノードであることから，語 John，ラベル 0.0.0.1 は 00000 を格納する．
(2-a) の場合 :
語 John，ラベル 0.0 のノードは中間ノード，かつ葉ノード (ラベル 0.0.0.1) から 0.0 へ至るパスは 1 本である．これらの間に出現するタグは，paper, author であるから，これらのビット列の論理和である 00101 を格納する．
(2-b) の場合 :
語 John，ラベル 0 のノードは中間ノードで，かつ葉ノードから 0 に至るパスが 2 本存在する (0.0.0.1 および 0.1.1.1 から 0 に至るパス)．よって，各パスの論理和値を 2-a と同様に計算すると，どちらも conf, paper, author が出現し，これらの論理和は 01101 となる．従って，求まった 1 種類のビットパターンである 01101 を格納する．

実際に，全索引語についてこの操作を行った結果が図 5 である．図中の色付けした部分が追加した情報である．このように索引を構成することで，従来手法では逐一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が削減できる．

ただし，ここで作成した索引は，ノードシグネチャの値および出現タグ名のビット論理和を，索引語が出現するノード自身と，それの先祖ノード全てに対して格納するから，容量が通常の転置索引に対して大きくなる．

3.3　BitMapKBSI 手法

ここでは，3.2 で構成した索引を用いて，与えられた検索キーワードに対する VLCA を抽出する手法である提案手法 BitMapKBSI 手法の手順について述べる．

1　2.2.2 で述べた手法を用いて，VLCA 候補となるノードラベルを絞り込む．この段階での処理については，従来手法からの変化はない．

2　前段階で得た候補ラベルについて，同ラベルのタグ名のビット列同士の AND 演算を行い，結果が 0 となるものが VLCA である．

例 6 (提案手法による VLCA 探索手順の例) :
図 5 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．

1　表 2 のワードシグネチャを用いて Q を計算するために，XML と John の値の論理和をとると，Q=1011001 となる．次に，Q と，図 5 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，図 6 の色を塗った部分である．

2　前で求めた候補ラベル 0, 0.0, 0.1, 0.0.0 で，タグ名のビット列同士で AND 演算を行うと，0.0.0 のみが演算結果が 0 となることから，VLCA は 0.0.0 のみとなる．

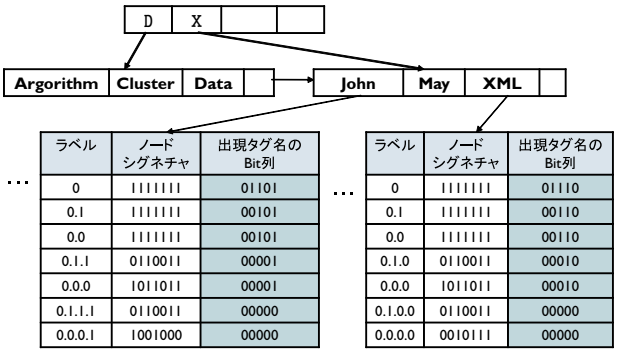

図 5　B+tree 索引にタグの情報を追加した例

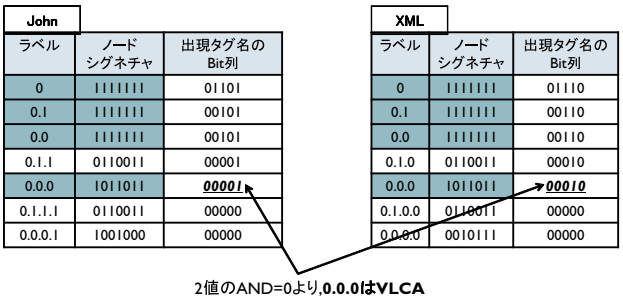

図 6　提案手法による VLCA 探索例

4.　評 価 実 験

提案手法の有効性を調べるために，提案手法および既存手法 VLCAStack アルゴリズム [5] を実装し，キーワードの頻度を変えて，文書に対しキーワード検索を行い，提案手法の評価を行う．

4.1　比較対象 VLCAStack

VLCAStack は，検索キーワードを含む葉ノードを，文書に

表 3　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

表 4　キーワードの出現頻度

頻度	キーワードの出現するノード数
低頻度	1-20
中頻度	20-200
高頻度	201-

出現する順に 1 回だけスキャンし順次スタックに積む手法である．このため，VLCA 判定を行う候補数が総当りよりも格段に減少し，計算効率が向上している．しかし，VLCA 判定時に文字列リストを用いてタグ名重複出現を検出する方法を用いており，この過程における性能改善は行われていない．

4.2　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，両手法ともシングルスレッドで実装し，ノードシグネチャ長=4096bit とした．実験で使用する XML 文書は，Xmark の XML 文書生成器 xmlgen [2] を用い，実験 1 では規模変更子を 0.01 として作成した文書 xmlgen001(size=1.13MB) および，実存するデータセットである SIGMOD Record [1](size=473KB) の 2 種類の文書を，実験 2 では xmlgen の規模変更子を 0.1 として作成した文書 xmlgen01(size=11.60MB) を用いて実験を行う．

4.3　経過時間と探索時間の手法間での比較（実験 1）

文書中のキーワードの出現するノード数をキーワードごとにカウントし，その値によって表 4 に示すような，低頻度，中頻度，高頻度の 3 つの部類に，全てのキーワードを分類しておく．2 つの文書それぞれについて，各頻度について，そこからランダムに選んだキーワードで検索を各手法に対して 100 回行う．キーワード数 2 の場合について，平均経過時間および平均探索時間をとる．なお，経過時間とは，キーワードを入力してから結果が全て出力されるまでの時間を意味する．また，平均探索時間とは，経過時間から，データベースクエリのターンアラウンドタイムを除いた時間を意味する．注意点としては，提案手法と既存手法ではクエリで指定する条件が異なるため，ターンラウンドタイムも異なる．提案手法では，語のほかに，ノードシグネチャと問い合わせシグネチャQ を満たすタプルのみを取得する条件を指定するので (2.3.2 参照)，単なるキーワード転置索引のみを使用する VLCAStack よりも，探索以外の時間がかかる．

図 7～図 9 は，低中高それぞれの場合における経過時間および探索時間の結果である．

両手法とも，頻度を高くすると経過時間，探索時間ともに増加する傾向が見て取れるが，探索時間の増加量では，提案手法の方が著しく優れるのがわかる．また，探索時間においては提案手法が全ての場合で勝った．一方で，経過時間では低頻度

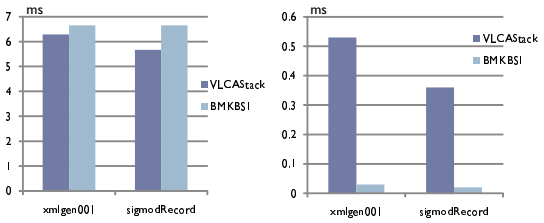

図 7　低頻度での経過時間 (左) と探索時間 (右)

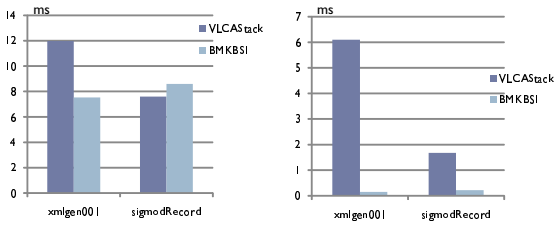

図 8　中頻度での経過時間 (左) と探索時間 (右)

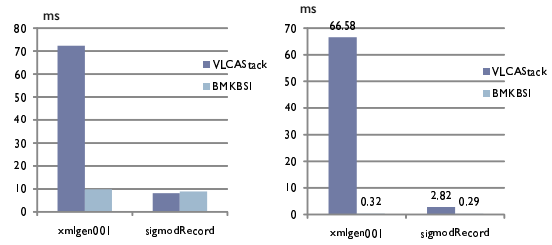

図 9　高頻度での経過時間 (左) と探索時間 (右)

時および SigmodRecord の中，高頻度時の場合において劣った．しかし，劣った全てのケースにおいて，経過時間の値自体が小さく (高々10ms)，両手法の差も小さい (高々3ms) であることがわかる．さらに，提案手法は，全ての場合において 10ms 以下の経過時間を示していることから，検索ワードの出現数の増加に強いと言える．

4.4　キーワード出現ノード数の変化に対する手法間の比較 (実験 2)

検索ワードの出現回数の増加の影響をさらに詳しく分析するために，次のような実験を行った．検索文書 xmlgen01 を用いて，キーワードの出現するノード数に対する変化を測定する．出現回数に応じて，0-200，200-400，400-600，600-1000，1000-1400 の 5 つのクラスに分類する。各クラスについて，そのクラスに属す語を与えた検索を 100 回行い，その平均経過時間を計測した．

図 10 は，実験 2 の結果である．出現数 600 以下の場合では，両手法間に大きな差は見られなかった．一方，600-1000，1000-1400 の場合において，提案手法は既存手法よりも優れた性能を示した．また，出現数の増加に対する経過時間の増加率でも，提案手法の方が優れていることがわかる．結果より，高頻出語での検索では提案手法が優れた性能を示すことがわかる．

4.5　使用したデータ容量の比較

3.2 で述べたように，提案手法は作成する索引の容量が増え

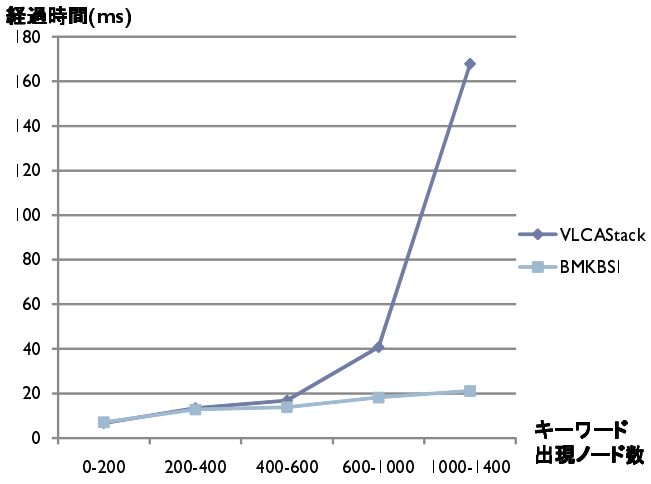

図 10　各出現数での経過時間

表 5　転置索引と提案手法で追加したデータのサイズ比較

	キーワード転置索引	ノードシグネチャ	タグ名論理和bit列
SigmodRecord	548KB	49MB	125KB
xmlgen001	3MB	307MB	52MB
xmlgen01	14MB	2800MB	455MB

るという問題がある．確認のために，今回実験で用いた 3 種類の文書について，提案手法で作成した索引に追加したデータのサイズを確認する．

表 5 は，VLCAStack の用いたキーワード転置索引 (表 5 の色付きの項)，および提案手法で用いたノードシグネチャ，タグ名の論理和のビット列の総データ容量である．実際に，提案手法では使用する索引のサイズが著しく大きいことがわかる．ただし，データサイズの算出方法はそれぞれ，キーワード転置索引については pg_relation_size の情報から，それ以外の 2 つの値については，タプル数と平均ビット長の積をとることでデータ容量の推定値を算出した．

5. まとめと今後の課題

本稿では，VLCA 抽出手法の効率化のために，キーワード B+tree 索引に，葉ノードからのタグ名のビット列論理和を格納することで，計算コストを削減した手法である BitMapKBSI 手法を提案した．この手法と，既存手法 VLCAStack との比較実験を行った結果，提案手法は，特に高頻出な検索ワード時，およびよりサイズの大きい文書においてよい性能を示した．しかし，提案手法で作成する索引のデータサイズが普通の転置索引よりも大きくなるという問題が残る．今後の課題としては，さらに大きなサイズの文書での比較実験を行い，傾向を観測する必要がある．また，検索語 3 以上の場合への抽出手法の拡張を実現する必要がある．

謝　　辞

本研究の一部は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] UW XMLData Repository. http://www.cs.washington.edu/research/xmldatasets/.

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

[3] 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.

[4] 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.

[5] Guoliang Li, Jianhua Feng, Jianyong Wang, and Lizhu Zhou. Effective keyword search for valuable lcas over xml documents. In *KICM*, pp. 31–40, 2007.

[6] Wenxin Liang, Takeshi Miki, and Haruo Yokota. Superimposed code-based indexing method for extracting mcts from xml documents. In *DEXA*, pp. 508–522, 2008.

[7] 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.

[8] 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.

[9] 小林径, 梁文新, 横田治夫. SuperimposedCode を用いた XML 中の ValuableLCA 探索手法. In *DEIM2009*, pp.B7-5,2009.