論文 / 著書情報
Article / Book Information

論題(和文)	Superimposed Code を用いた XML 中の Valuable LCA 探索手法
Title(English)	A Superimposed Code-based Valuable LCA Detection Method over XML Documents
著者(和文)	小林径, 梁文新, 横田治夫
Authors(English)	Kei KOBAYASHI, Wenxin LIANG, Haruo YOKOTA
掲載誌(和文)	DEIM2009論文集
Citation(English)	Proceedings of DEIM2009
Vol, no, pages	, , pp. B7-5
発行日 / Pub. date	2009, 3

Superimposed Codeを用いたXML中のValuable LCA探索手法

小林　径† 梁　文新††,††† 横田　治夫†††

† 東京工業大学 工学部 情報工学科
†† (独) 科学技術振興機構
††† 東京工業大学 学術国際情報センター
E-mail: {kei, wxliang}@de.cs.titech.ac.jp,yokota@cs.titech.ac.jp

あらまし　近年のXMLデータの増大により，XML文書に対する効果的かつ効率的なキーワード検索の必要性が増してきている．これまで，XMLキーワード検索では，全キーワードを含む最小部分木の根であるLCA(Lowest Common Ancestor)，あるいは部分木に他のLCAを持たないSLCA(Smallest LCA)を求めることが一般的であった．我々は，全文検索の手法であるスーパーインポーズドコード索引をXML文書に適用し，B+treeによるキーワード索引と組み合わせることで効率的にSLCAを求める手法を提案してきた．しかし，SLCAは階層構造中のキーワードの出現のみを考慮していることから，SLCAを根とする部分木には冗長な部分が含まれたり，逆に必要な部分が含まれなかったりする問題がある．この問題に対し，LCAから各キーワードを含むノードまでの間の同一タグ名の出現を考慮するVLCA(Valuable LCA)が提案されている．本稿では，タグ名にbit列を割り当て，bit演算によって効率的にVLCAの判定を行う手法を提案し，既存のBrute-forceアルゴリズムとの比較実験によりその効果を示す．

キーワード　XMLキーワード検索，スーパーインポーズドコード，LCA，SLCA，VLCA

A Superimposed Code-based Valuable LCA Detection Method over XML Documents

Kei KOBAYASHI†, Wenxin LIANG††,†††, and Haruo YOKOTA†††

† Department of Computer Science, Faculty of Engineering, Tokyo Institute of Technology
†† Japan Science and Technology Agency
††† Global Scientific Information and Computing Center, Tokyo Institute of Technology
E-mail: {kei, wxliang}@de.cs.titech.ac.jp,yokota@cs.titech.ac.jp

Abstract　With the increase of XML data in recent years, the necessity of effective and efficient keyword search for XML documents has been increasing. In most of the existing XML keyword search methods, the search results are commonly represented by subtrees rooted at LCAs(Lowest Common Ancestors) or SLCAs(Smallest LCAs) that contain all the search keywords. In our previous work, we have proposed an efficient method to determine SLCA by combining the superimposed coding technique and the Keyword B+tree index. However, SLCAs have false-positive and false-negative problems because they only consider the containment relationships of the nodes including keywords. To cope with these problems, VLCA(Valuable LCA) that considers the semantic features based on the name of tags appeared in the paths from LCAs to the leaf nodes containing keywords has been recently proposed. We propose a novel method for efficiently detecting VLCAs, in which each tag name is assigned a unique bit sequence, and the VLCA can be simply determined by applying bit operations over the paths from LCAs to the leaf nodes containing any keyword. We perform experiments to evaluate the proposed methods comparing with the Brute-force algorithm. The experimental results indicate that the proposed methods outperform the Brute-force algorithm, particularly when the frequency and the number of keywords increase.

Key words　XML keyword search, superimposed code, LCA, SLCA, VLCA

1. はじめに

近年，XML(Extensible Markup Language) の普及によって，文書サイズが増大してきている．特に，Web 上のオンライン百科事典である Wikipedia [9] など，大規模 XML データが Web 上に多く存在する．XML の大規模化に伴い，XML 文書から必要な情報を効率よく，かつ効果的に検索する手法の必要性が高まってきている．XML 文書から必要な情報を検索する方法の一つとして，キーワード検索がある．XML キーワード検索では，XML が木構造を構成することから，キーワードを全て含む部分の根のうち最低にある LCA(Lowest Common Ancestor) を探す方法が一般的である．

しかし，LCA は，キーワードを含む部分が多いと，膨大になり、その内から利用者の検索要望に応じるものを判り難くなるという問題点がある．この問題に対しては，LCA のうちで，LCA を根とする部分木の中に他の LCA を持たない LCA である SLCA(Smallest LCA) が提案されている [11]．我々は，SLCA を効率的に求めるための手法として，全文検索の手法であるスーパーインポーズドコードの索引 [5] [4] を XML 文書に対して適用し，キーワード B+tree の索引と組み合わせる手法である KBSI(Keyword B+tree with SuperImposed code) 手法を提案し，既存手法の Stack-based アルゴリズムを用いる手法 [6] と比較することでその効果を示してきた [8] [12].

しかし，SLCA は，LCA 間の含有関係のみを考慮しているため，SLCA を根とする部分木には意味的に冗長な部分を含むものが存在する問題や，意味的に必要な部分を含む部分木の根である LCA が，その下位階層に他の LCA を持つために，SLCA からは除かれてしまうという問題が存在する．このような SLCA の問題点に対して，LCA からキーワードを含むノードまでのパス上に存在するノードのタグ名を考慮した VLCA(Valuable LCA) が提案されている [7].

そこで，本研究では，この VLCA に着目し，VLCA を文献 [7] で提案されている既存手法よりも効率的に求める手法の実現を目指す．まず最初に，ノードのタグに出現するキーワードに対して bit 列を割り当て，下位のキーワードから上位の LCA までのパスに従って bit 演算を行うことで VLCA を求める手法である TS(TagSignature) 法の提案する．次に，KBSI 手法を用いて，先に VLCA 候補となるノードを与え，それが VLCA であるかを判定するのに bit 演算を用いる手法を用いる KBSITS(KBSI with TS) 手法を提案する．これらの提案手法に対し，既存手法 Brute-force アルゴリズム [7] との比較実験を行う．

以下では本論文の構成を述べる．まず，2 章では，SLCA の説明を行い，SLCA を効率よく求める KBSI 手法の説明を行う．続いて 3 章では，文献 [7] で述べられている SLCA の問題点と，それに対して提案された VLCA について説明する．4 章では，最初に VLCA を効率的に抽出するために，bit 演算を用いる手法である TS 法を提案し，次に KBSI 手法と TS 法を組み合わせた KBSITS 手法を提案する．5 章では，既存手法 Brute-force アルゴリズムと提案手法による比較実験を行い，提案手法の評価を行う．第 6 章では，まとめと今後の課題について述べる．

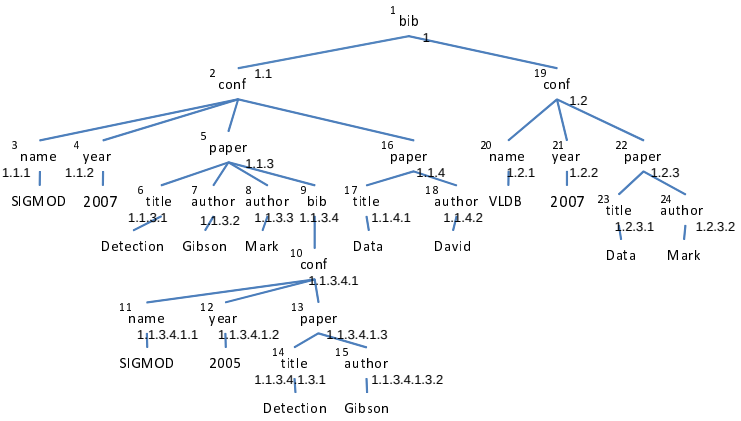

図 1　XML 文書の例

2. SLCA

2.1 LCA (Lowest Common Ancestor) と SLCA(Smallest LCA)

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，木構造を構成している．XML キーワード検索では，与えられたキーワードを全て含む部分木の根の中で最低にあるものである LCA(Lowest Common Ancestor) を求める方法が一般的によくとられている．しかし，キーワードを含む部分の数が多くなると，あるキーワードを含む 1 つのノードの先祖ノードに対して，複数の LCA が出現する．このとき，どの LCA が一番ユーザが欲しい情報であるかが判り難くなるという問題点がある．また、キーワードの数と文章サイズが増えた場合、抽出される LCA の総数が更に膨大になり、判定がより難しくなるという問題点もある．このような LCA の問題点に対して，LCA の中で，部分木中に他の LCA を含まない部分木の根である SLCA(Smallest LCA) が提案されている [11].

LCA および SLCA の例として，図 1 の XML 文書に対して，キーワード {″SIGMOD″,″ 2007″} を与えた場合を考える．なお，XML 文書には、DeweyOrder [10] によるラベル付けを行い，また，説明のため，ラベルの他にノードに対し番号を付した．LCA として，$\{bib(1), conf(2)\}$ が求まる。SLCA は，LCA の $conf(1)$ を根とする部分木に LCA である $conf(2)$ を含むため，これらのキーワードに対する SLCA は $conf(2)$ のみである．

2.2 SLCA を効率的に抽出する方法

前節で述べた SLCA を求める手法としてスタックを用いた手法が提案されている [6] が，キーワード数が増えた場合に効率が悪いという問題点があった．そこで，我々は、スーパーインポーズドコードとキーワードに対する B+tree 索引を組合わせた手法である KBSI(Keyword B+tree with SuperImposed code) 手法を提案し，スタックを用いた手法との比較実験によりその効果を示してきた [8] [12].

以下では，SLCA を効率的に抽出する手法である KBSI の概略を説明する．最初に，シグネチャを用いた索引を文書中の全てのノードに対して作成する手順を述べる．続いて，ノードのシグネチャを用いて LCA 候補となるノードを探す手順につい

キーワード	ワードシグネチャ
Detection	0100000001
Gibson	0100000100
SIGMOD	0100010000
2005	0001000001
2007	0001000010
Mark	0000101000
David	0010100000
Data	1000000001
VLDB	0000100010

図 2　図 1 の文書に出現するキーワードのワードシグネチャの構成例

て説明し，最後に KBSI 手法による SLCA 探索の概略を述べる．

2.2.1　シグネチャを用いた索引方法

（1）　ワードシグネチャの生成

最初に，文書中の葉ノードに出現するキーワードを全て抽出する．次に，それぞれのキーワードに対し固定長の異なる bit 列を割り当てる．これをワードシグネチャと呼ぶ．

図 2 は，図 1 の文書に出現する全てのキーワードに対してワードシグネチャの生成を行った例である．

（2）　ノードシグネチャの生成

次に，ワードシグネチャを用いて文書中の全てのノードに対応するノードシグネチャを生成する．ここでは，その概要を説明する．

（a）　葉ノード LN_i に対応するノードシグネチャ NS(LN_i) を計算する．方法は，LN_i の含むすべてのキーワードに対応するワードシグネチャの論理和をとればよい．

（b）　すべての節ノードについて，対応するノードシグネチャの値を求める．ある節ノード N_i に対応するノードシグネチャ NS(N_i) は，ノード N_i の持つ全ての子ノードのノードシグネチャの論理和をとることで求めることができる．この手順を，XML 木の下位階層から根に向かって，再帰的に求めていくことにより，文書中の全てのノードのノードシグネチャを生成することができる．

図 3 は，図 1 の文書に出現するノードに対して，図 2 で構成したワードシグネチャを用いて，ノードシグネチャを生成した例の一部である．

2.2.2　LCA 候補の決定

m 個の検索キーワード kw_1,$kw2$,...,kw_m が与えられた場合を考える．まず各検索キーワードのワードシグネチャを取得し，それらの値全ての論理和をとる．得た値を問い合わせシグネチャといい，Q と表す．式で表すと以下のようになる．

$$Q = WS(kw_1) \vee WS(kw_2) \ldots \vee (kw_m) \quad (1)$$

次に，この問い合わせシグネチャ Q と，あるノード u のノードシグネチャ $NS(u)$ に対して論理積演算を行い，

$$Q \wedge NS(u) = Q \quad (2)$$

ノードラベル	ノードシグネチャ
1.1.3.4.1.3.1	0100000001
1.1.3.4.1.3.2	0100000100
1.1.3.4.1.1	0100010000
1.1.3.4.1.2	0001000001
1.1.3.4.1.3	0100000101
1.1.3.4.1	0101010101
⋮	⋮
1.1	1111111111
1.2	1111111111
1	1111111111

図 3　図 1 の文書のノードに対するノードシグネチャの構成例

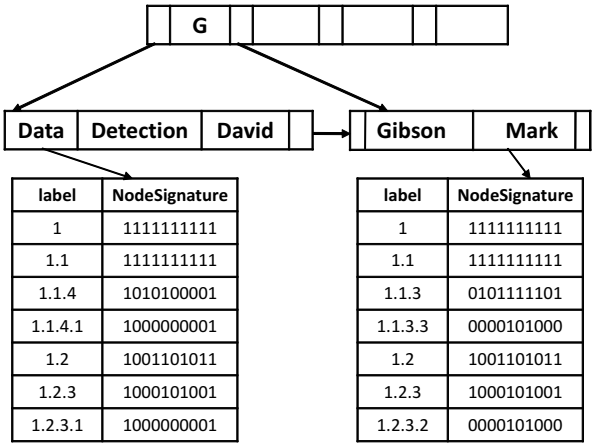

図 4　図 1 の XML 文書に対して KBSI 索引を作成した例

を満すなら，ノード u を根とする部分木に全てのキーワードを含む可能性がある．即ち，式 2 を満たすノードは LCA ノードの可能性があるということである．しかし，式 2 を満すノードでも，実際にはキーワードが全て含まれない場合も存在する．このようなものを false drop と呼ぶ．

2.2.3　KBSI (Keyword B+tree with SuperImposed code) 手法

SLCA を抽出する手法である KBSI 手法の説明を行う．まず，検索を行う前段階として，出現するキーワードごとに，そのキーワードの出現する葉ノードの全ての先祖ノードのノードラベルと，それに対応するノードシグネチャの組を，B+tree の索引に格納する．

図 1 の XML 文書に対して，KBSI で用いるキーワード B+tree 索引を構成した例が，図 4 である．以後，説明のために，KBSI で用いるキーワード B+tree 索引を，KBSI 索引とよぶ．

次に，与えられた検索キーワードに対する問い合わせシグネチャ Q と，KBSI 索引のノードシグネチャを比較することで，予め LCA に成りえないノードラベルを除くことによって検索を効率化する．

実際に検索を行う際は，以下の手順に従う．

（1）　入力キーワードに対する問い合わせシグネチャ Q を作成する．

（2）　すべてのキーワードについて，各入力キーワードに一致す

るエントリのうちで，そのノードシグネチャ要素と Q が式 2 を満たすエントリのラベルを全て抽出する．(LCA 候補を絞り込む)

（3） 抽出したラベル要素のうち，全キーワードで出現したラベルをラベル集合 CW に追加する．

（4） 得られたラベル集合 CW の各要素について，Dewey Order のラベルを解析することで，ある要素の先祖ノードラベルとなっている全てのノードラベルを集合 CW から削除することで SLCA を得る．

全入力キーワードのエントリを参照していることから，得られた SLCA はその部分木には確かに全キーワードを含むので，2.2.2 で述べた false drop も発生しない．

3. VLCA

本章では，SLCA の問題点と、その問題点に対して文献 [7] で提案されている VLCA(Valuable LCA) を紹介する．

3.1 SLCA の問題点

SLCA は，LCA 間の階層関係のみを考慮しているため，その部分木に冗長な部分が含まれる問題 (false positive) と，必要な部分を含む LCA であるのに，SLCA からは除外される問題 (false negative) がある [7]．以下ではその例を述べる．

例 3.1-1 false positive の例として，図 1 の XML 文書に対して，検索キーワード {″*Data*″,“*Mark*″} を与えた場合を考える．これらの SLCA は，{*conf*(2), *paper*(22)} である．しかし，SLCA のうちの 1 つであるノード conf(2) はそれぞれ異なる paper(論文) 要素にまたがるノードである author(18) と title(6) の LCA であるから，検索の結果としてふさわしくない．このように，SLCA には，結果にふさわしくない LCA を SLCA に含む問題がある．

例 3.1-2 false negative の例として，図 1 の XML 文書に対して，検索キーワード {″*Detection*″,″ *Gibson*″} を与えた場合を考える．これらの SLCA は，ノード {*paper*(13)} である．一方で，LCA である paper(5) は，下位階層に LCA である paper(13) を持つため，SLCA にはならない．しかし、LCA ノード paper(5) は，同一の論文要素に全てのキーワードを含む部分を持つため，それを根とする部分木は，結果として必要な部分を含んでいるのに，SLCA とならない．このように，SLCA には，結果に必要な部分をその部分木含む LCA を SLCA から除外するという問題がある．

3.2 VLCA (Valuable LCA)

3.1 で述べた SLCA の問題に対して，キーワードを含む葉ノードから LCA までのパス上に存在するノードに同じタグ名が存在しない LCA である VLCA(Valuable LCA) が提案されている [7]．ここではまず，VLCA を導入する上で必要となる同質，異質の定義を述べる．

同質 (Homogenous)/異質 (Heterogenous)： 2 つのノード u と v が与えられたとする．それらの LCA ノードを w とする．ノード w から u およびノード w から v のパス上に出現するノードの集合を順に $uSet$,$vSet$ と呼ぶ．$wSet$=$uSet \bigcup vSet$ とする．

（1） ノード u と v が同質であるとは，$wSet$ にノード u と v の組み合わせ以外に同じタグ名を持つノードのペアが存在しない場合である．

（2） ノード u と v が異質であるとは，$wSet$ にノード u と v の組み合わせ以外に同じタグ名を持つノードのペアが存在する場合である．

以下では，実際に例をあげて説明する．

例 3.2-1 図 1 の XML 文書中の 2 つのノード {*author*(8), *title*(17)} について考える．それらの LCA はノード conf(2) である．この LCA からの 2 つのパス conf(2) → author(7) および conf(2) → title(17) 上に出現するノードの和集合には，同じタグ名を持つノード {*paper*(5), *paper*(16)} が存在するので，2 つのノード {*author*(8), *title*(17)} は異質である．同様に，ノード {*title*(23), *author*(24)} の組について考えると，これらの LCA ノード paper(22) からそれらのノードへのパスに出現するノードの和集合には，同名タグをもつノードの組がないので，2 つのノード {*title*(23), *author*(24)} は同質である．

上で述べた同質/異質の概念を用いることにより，VLCA の定義を紹介する．

VLCA: m 個のノード v_1,v_2,...,v_m が与えられたとする．これらの LCA ノードを w とする．w が VLCA であるとは，m 個のノードから 2 つを取る組み合わせについて，全てのノードの組が同質であるならば，ノード w は VLCA である．逆に，1 つでも異質であるノードの組が存在するならば，ノード w は VLCA でない．以下で例を挙げる．

例 3.2-2 検索キーワード {″*Data*″,″ *Mark*″} を図 1 の XML 文書に対して与えた場合を考える．キーワード ″*Data*″ を含むノード author(8) と ″*Mark*″ を含むノード title(17) については，これらは先の例より異質であると判っているので，これらの LCA であるノード conf(2) は VLCA でない．同様にして，キーワードを含むノードの組 {*author*(8), *title*(23)}, {*author*(24), *title*(17)} は，パス上に同名タグを持つので，これらの LCA の bib(1) は VLCA でない．一方，キーワード”Data” を含むノード title(23) と”Mark” を含むノード author(24) については，パス上に同名タグを持たないので，それらの LCA であるノード paper(22) は VLCA である．以上より，検索キーワード {″*Data*″,″ *Mark*″} に対する VLCA はノード paper(22) である．3.1 で述べた最初の例と比較すると，SLCA の false positive 問題が VLCA では回避されていることが判る．同様の手順で，検索キーワード {″*Detection*″,″ *Gibson*″} について調べてみると，ノード title(7) と author(6) の VLCA として paper(5) が抽出されることにより，3.1 で述べた例と比較することで，SLCA の false negative の問題が VLCA では回避されることが判る．

この VLCA を求める手法として，Brute-force アルゴリズムおよび，スタックを用いた手法 [6] をベースに，VLCA の部分集合である CVLCA(Compact VLCA) を求める手法である VLCAStack が提案されている [7]．しかし，これらの手法は，高頻度キーワードに対する検索性能に問題がある．

ファイル名	データサイズ	タグに出現するキーワードの種類
psd7003.xml	683 MB	66
dblp.xml	127 MB	36
reed.xml	277 KB	16
SigmodRecord.xml	467 KB	12
modual-3.0.xml	1MB	23

表 1　Washington XML Data Repository [2] の XML データセットのデータサイズとタグに出現するキーワードの種類

タグ名	タグシグネチャ
bib	1000000
conf	0100000
name	0010000
year	0001000
paper	0000100
title	0000010
author	0000001

図 5　タグシグネチャの構成例

4. VLCA を抽出する手法の提案

この章では，最初に bit 演算による同質判定アルゴリズム提案する．次に，それを用いて，与えられたキーワードに対して葉ノードから LCA へパスに従って VLCA を求める手法である TS 手法を提案する．次に，KBSI 手法によって最初に VLCA 候補を与え，それが VLCA かを判定することで VLCA を求める KBSITS 手法を提案する．

4.1 同タグ名の出現を検出する手法

ここでは，LCA からキーワードを含むノードへのパス上に存在するノードのタグ名に出現するキーワード対して，bit 列を割り当て，bit 演算を適用することで同タグ名出現を検出することによって，節 3.2 で述べた同質/異質の判定を行う手法を提案する．

4.1.1 タグシグネチャの割り当て方法

まず，文書中のノードのタグに出現する全てのキーワードを抽出する．次に、それぞれのキーワード毎に 1 の位置が異なる bit 列を割り当てる．この bit 列をタグシグネチャと呼ぶ．この bit 列は，bit 長がタグに出現するキーワードの種類に等しく，1 が出現する回数は 1 回であり，1 の出現する位置は他のどの bit 列とも異なるように割り当てる．大規模な XML データベースでも，節ノードのタグを表すキーワードの種類がそれほど多くない（表 1 参照）ため，多くの場合ノードタグに割り当てた bit 列の長さもそれほど長くならないと考えられる．図 5 は，図 1 の XML 文書に出現するキーワードに対して，タグシグネチャを構成した例である．

次に，文書中の各ノードのノードラベルと，そのノードの持つタグ名に対応するタグシグネチャの値を対応させる．これをノードタグシグネチャと呼ぶ．図 6 は，図 1 の XML 文書に出現するノードに対して，図 5 で生成したタグシグネチャを用いて，ノードタグシグネチャを生成した例の一部である．

ノードラベル	ノードタグシグネチャ
1	1000000
1.1	0100000
1.1.1	0010000
1.1.2	0001000
1.1.3	0000100
1.1.3.1	0000010
1.1.3.2	0000001
1.1.3.3	0000001
1.1.3.4	1000000
⋮	⋮

図 6　ノードタグシグネチャを構成した例の一部

2 つのラベルが同質かどうかを判定するアルゴリズム
入力: ノードラベル $\lambda(u)$，$\lambda(v)$
出力: *true*　(ノード u，v が同質である場合)
　　false　(ノード u，v が異質である場合)

```
0:  begin
1:   if λ(u) = λ(v)
2:    return true
3:   else
4:    λ(w) = getLCA((u), λ(v))
5:    ENS ← σ(λ(u)) ∨ σ(lambda(v))
6:    for n = λ(u), λ(v)
7:     while (parent(n) ≠ λ(w))
8:       n ← parent(λ(n))
9:        if (ENS = ENS ∨ σ(n))
10:           return false
11:       else
12:           ENS ← ENS ∨ σ(n)
13:    end;
14:   end;
15:   if (ENS = ENS ∨ σ(λ(w)))
16:      return false
17:   else
18:      return true
19:  end;
```

getLCA(label1,label2):2 つのラベルの LCA を返す
parent(label):親ノードラベルを返す

図 7　ノードタグシグネチャを用いた同質判定アルゴリズム

4.1.2 ノードタグシグネチャを用いた同質判定アルゴリズム

ここでは，3.2 で述べた 2 つのノードの同質判定を，4.1.1 で作成したノードタグシグネチャを用いて行うアルゴリズムの提案を行う．以下では，あるノード n のノードラベルを $\lambda(n)$ と表記し，あるノードラベル lb に対応するノードタグシグネチャの値を $\sigma(lb)$ と表記する．

ノードタグシグネチャを用いた同質判定アルゴリズムを図 7 に示す．ここで，事前にノードラベルとノードタグシグネチャの対応表 (図 6) が得られているものとする．

以下図 7 に沿って説明を行う．2 つのノード u,v のノードラベル $\lambda(u), \lambda(v)$ が与えられたとする．最初に，これらのラベルが等しい場合は同質と判断できる (図 7:line1-2)．そうでない場

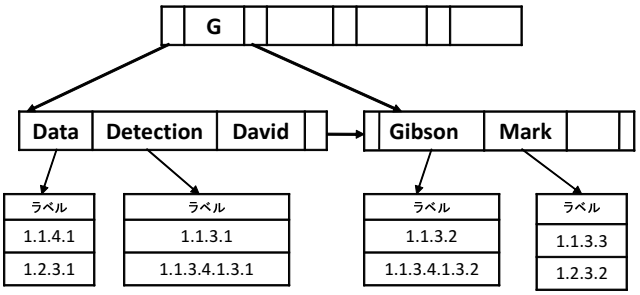

図 8　キーワードに対する B+tree インデックスの例

最初にラベル 1.1.3.1 に対して親ノードラベルを求めると (1.1.3.1) = 1.1.3 = $\lambda(w)$ より ENS には何もしない
次にラベル 1.1.3.4.1.3.2 の親ノードラベルが LCA に一致 するまで以下の操作を行った $ENS \leftarrow ENS \vee \sigma(1.1.3.4.1.3) = 000011 \vee 000100 = 0000111$ $ENS \leftarrow ENS \vee \sigma(1.1.3.4.1) = 0000111 \vee 0100000 = 0100111$ $ENS \leftarrow ENS \vee \sigma(1.1.3.4) = 0100111 \vee 1000000 = 1100111$ $parent(1.1.3.4) = 1.1.3 = \lambda(w)$ より ENS には何もしない
最後に LCA ノードラベル 1.1.3 に対して $ENS \vee \sigma(1.1.3) = 1100111 = ENS$ となるからこれらは異質と判る

図 9　同質を調べる過程の例

合はそれらの LCA ノード w のノードラベル $\lambda(w)$ を得る (図 7:line3). 次に，判定用 bit 列 ENS を用意する．その初期値として，ノード u,v に対応するノードタグシグネチャの論理和演算の結果を代入する (図 7:line5). ノードラベル $\lambda(u), \lambda(v)$ それぞれについて，その親ノードラベルを得て，自身のラベルに代入し (図 7:line8)，bit 列 ENS と論理和演算を行う (図 7:line9). ただしこのとき，論理和演算結果がもとの ENS と等しい場合は，すでに同じタグ名が出現しているので (bit の 1 の位置と，タグ名が一意に対応するよう定めたからそう言える)，異質と判断できる (図 7:line10). そうでない場合は，得た演算結果を ENS に代入する (図 7:line11-12). この手順を親ノードラベルが LCA のノードラベルに一致するまで再帰的に繰り返す (図 7:line7-14). 最後に，LCA のノードラベル $\lambda(w)$ と ENS との論理和演算を行い，その結果が ENS と等しいなら異質，そうでないなら同質と判断できる (図 7:line15-19).

4.2　TS (Tagsignature) 手法

4.1 で述べた判定法を用いて VLCA を求める手法である TS(Tag signature) 法の提案を行う．以下では，TS 法の VLCA の抽出方法を説明する．なお，検索を行う前に，文書に出現するキーワードに対して，その語の出現するノードの親ノードラベルのみを格納するキーワード B+tree インデックスおよびノードタグシグネチャの表が構成されているものとする．(図 8 は，図 1 の XML 文書のキーワードに対してキーワード B+tree インデックスを構成した例の一部である．)

m 個のキーワード k_1,k_2,...,k_m が与えられたら，最初にそれぞれのキーワードを含むラベルの集合をインデックスより得る．τ_i とは，キーワード $k_i(1 \leqq i \leqq m)$ を含む，インデックスより得たノードラベルの集合を表すこととする．まず，得た各キーワードに対するラベル集合から，要素をひとつずつとることで調べる対象となる m 個のノードラベルの組 $\{label_1, label_2, \ldots, label_m\}$ を選ぶ (ただし，$label_i \in \tau_i$ である)．このラベルの組から，2 つのラベルを取る全ての組み合わせ $\forall$i,j $\{label_i, label_j\}$(ただし，i≠j) に対して，各組み合わせ毎に 4.1.2 で提案した同質/異質の判定アルゴリズムを用いて同質であるかを調べる．全て同質であれば，それらの LCA は VLCA であると判断できる．一方，全ての組を調べる過程で，異質な組み合わせが出現したなら，それらの LCA は VLCA でないと判定できる．この手順を，全ての集合の各要素の，全ての組み合わせ $\{label_1, label_2, \ldots, label_m\}$[ただし $\forall\ label_i \in \tau_i$] について行うことで，m 個のキーワードに対する VLCA を求めることができる．以下でキーワードに対する VLCA を求める例を示す.

例 4.2　図 1 の文書にキーワード {“*Detection*”,” *Gibson*”} を与えた場合を考える．最初に，キーワードの B+tree インデックス (図 8) より，ノードラベル集合 $\tau_{Detection}$={1.1.3.1; 1.1.3.4.1.3.1} および τ_{Gibson}= {1.1.3.2; 1.1.3.4.1.3.2} を得る．次に，2 つの集合の要素の全ての組み合わせについて同質かどうかを調べる．最初に，{1.1.3.1, 1.1.3.2} の組について，LCA ノードラベルを求めると，{1.1.3} となる．次に，それらのノードラベルに対応するノードタグシグネチャを図 6 から得て，bit 列 ENS の初期値を求めると，

$ENS = \sigma(1.1.3.1) \vee \sigma(1.1.3.2) = 0000011$

となる．次に，それらの親ノードラベルを求めると，どちらも {1.1.3} であり，LCA に一致するので，最後に，LCA ノードラベル {1.1.3} のノードタグシグネチャσ(1.1.3) と ENS との論理和をとると，$ENS \vee \sigma(1.1.3) \neq ENS$ となるので同質であるとわかる．よって，{$VLCA$ = 1.1.3 : *contentnode* = 1.1.3.1; 1.1.3.2} が求まる．次に，{1.1.3.1, 1.1.3.4.1.3.2} の組について考える．LCAw のラベル $\lambda(w)$={1.1.3}　で，ENS の初期値は，ENS=0000011 となる．以下の手順については，過程が冗長になるため，過程を図 9 に記したのでそちらを参照されたい．図より，LCA ノードラベルについて調べたときに異質と判断されるので，{1.1.3.1, 1.1.3.4.1.3.2} の組は VLCA とならない．同様にして組 {1.1.3.4.1.3.1, 1.1.3.2} および {1.1.3.4.1.3.1, 1.1.3.4.1.3.2} も調べることができ，{1.1.3.4.1.3.1, 1.1.3.4.1.3.2} を調べた結果として，{$VLCA$ = 1.1.3.4.1.3 : *contentnode* = 1.1.3.4.1.3.1; 1.1.3.4.1.3.2} が求まる．以上より，キーワード {“*Detection*”,” *Gibson*”} に対する VLCA は，VLCA={1.1.3; 1.1.3.4.1.3} となる．

4.3　KBSITS 手法

ここでは，2.2 で述べた KBSI 手法で VLCA 候補となるノードのラベルを予め得て，その候補が VLCA かどうかを 4.1.2 で提案した VLCA の判定方法を用いて調べることで，VLCA を抽出する手法を提案する．

以下ではキーワードが与えられた際の検索手順を述べる．ただし，検索を行う前に，キーワードに対するキーワード B+tree インデックスおよび，ノードタグシグネチャ表 (図 6)，KBSI で用いる KBSI 索引 (図 4) が事前に構成されているものとする．

m 個のキーワード kw_1,kw_2,...,kw_m が与えられたとする．最初

表 2　実 験 環 境

CPU:	AMD Opteron2.2G *2
OS:	Linux 2.6.9
Memory:	6GB
HDD:	ATA/133 200 GB
Java VM:	J2SE 1.5.0
Database:	PostgreSQL 8.1.3

に，2.2 で説明された KBSI 手法を用い，その過程で VLCA 候補ラベル集合 CW を得る．この CW の要素全てに対して，それが VLCA であるかを調べる．

以下では，ある VLCA 候補ノード cw のノードラベル $\lambda(cw) \in CW$ に対して，それが VLCA であるかを調べる手順を説明する．まず最初に，cw を根とする部分木に含まれる各キーワードを含むノードラベルを，$\lambda(cw)$ と前一致するラベルを各キーワードごとにキーワード B+tree インデックスより取得する (ここで，ξ_i を，キーワード kw_i に対して得た，cw を根とする部分木に含まれるラベル集合と表記する)．得られた m 個のキーワードに対するラベル集合 $\xi_1,\xi_2,\ldots,\xi_m$ を得たら，それらの要素の全ての組合せに対して 4.1 で提案した VLCA の判定法を用いることで VLCA を抽出すればよい．ただし，判定を行う前に，調べる m 個のノードラベルの組 $(label_1 \in \xi_1, label_2 \in \xi_2, \ldots, label_m \in \xi_m)$ の LCA が，VLCA かどうかを調べるラベル $\lambda(cw)$ と一致するかどうかを調べなければならない．なぜなら，m 個のノードの LCA が VLCA だった場合，それらの LCA は VLCA になるが，そのノードは調べている VLCA 候補とは異なるためである．

以下で KBSITS 手法で VLCA を得る例を示す．

例 4.3　図 1 の文書にキーワード {"*Data*","*Mark*"} を与えた場合について説明する．2.2 の手順に従い，VLCA 候補ラベル集合 CW={1; 1.1; 1.2; 1.2.3} が得られるため，この各要素について順次調べる．

最初に，ラベル $\lambda(cw)$={1.2.3} について調べるとする．このラベル $\lambda(cw)$ に前一致するラベルを転置インデックス (図 8) から得ると，それぞれ ξ_{Data}={1.2.3.1}, ξ_{Mark}={1.2.3.2} となる．次に，集合 ξ_{Data},ξ_{Mark} の要素からひとつずつラベルをとる全ての組合せについて調べる．{1.2.3.1; 1.2.3.2} のラベルの組について，これらの LCA が $\lambda(cw)$={1.2.3} と一致するかどうかを調べる．この場合は一致するのでこれらのラベルに対して 4.1.2 の方法で同質および VLCA を判定すると，これらは同質であり，これらの LCA{1.2.3} は VLCA である．これで集合から要素を 1 つずつとる全ての組について調べたので，集合 CW から次の要素のラベルを調べる．次に，ラベル $\lambda(cw)$={1.2} について，キーワードを含む部分のラベルは，それぞれ ξ_{Data}={1.2.3.1},ξ_{Mark}={1.2.3.2} となる．ラベルの組 {1.2.3.1; 1.2.3.2} の LCA ラベルはラベル $\lambda(cw)$ に一致しない．さらに他のラベルの組も無いので，$\lambda(cw)$={1.2} については調べ終った．再び，集合 CW から次のラベルをとる．$\lambda(cw)$={1.1},$\lambda(cw)$={1} の場合についても上と同様の方法で調べることができ，VLCA が無いことが判る．結果として VLCA={1.2.3} を得ることができる．

表 3　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-

文書名	SigmodRecord	xmlgen001
サイズ	727KB	1.135 MB
要素数	16202	17131
タグのキーワード数	12	76

表 4　実験で使用する文書の情報

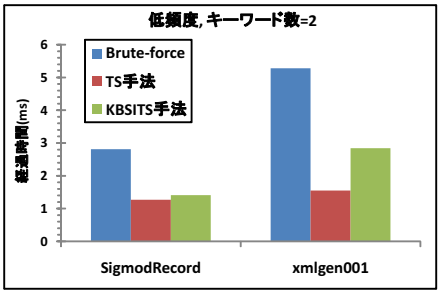

図 10　低頻度, キーワード数 2 の経過時間

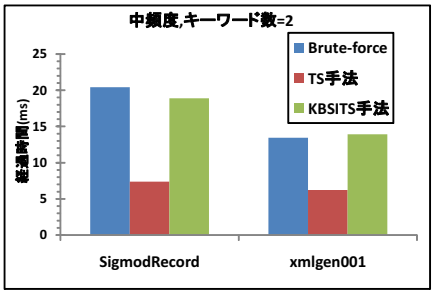

図 11　中頻度, キーワード数 2 の経過時間

5. 実　　験

提案した手法の有効性を調べるために，提案手法および Brute-force アルゴリズムを実装し，キーワード数および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．実験に用いた計算機，および環境を表 2 に示す．

実験で使用する XML 文書は，Xmark の XML 文書生成器 xmlgen [3] を用い，規模変更子を 0.01 として作成した文書 xmlgen001 および，実存するデータセットである SIGMOD Record [1] を用いて実験を行う．実験に用いる文書に関する情報を表 4 にまとめた．

実験を行う前の準備として，KBSI 手法で用いる KBSI 索引 (図 4) および，ノードラベルとノードタグシグネチャの対応表 (図 6)，さらにキーワードに対するキーワード B+tree インデックス (図 8) を文書に対して予め作成する．また，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 3 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

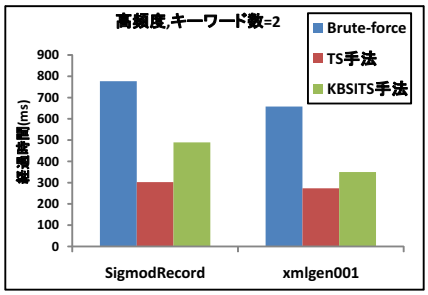

図 12　高頻度, キーワード数 2 の経過時間

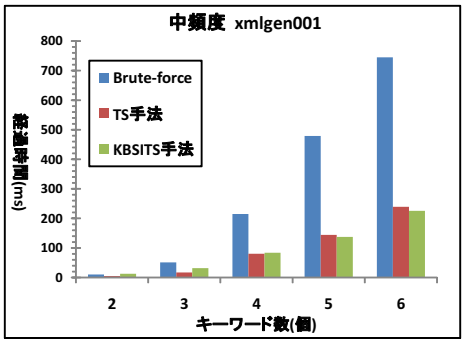

図 13　出現頻度を中に固定したときの経過時間

ワードで検索を各手法に対して 100 回行う．これをキーワード数を 2 個から 6 個まで変化させてそれぞれ行ったときの 100 回の平均経過時間をとる．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．

5.1　キーワードの頻度の与える影響

与えるキーワードの頻度による影響を調べる．キーワード数を 2 に固定し，それぞれの頻度に対する経過時間の結果が図 10-12 である．いずれの文書においても，キーワードの頻度を上げると経過時間が大きくなる傾向があるのがわかる．また，キーワードの出現頻度が高い方が既存手法と提案手法との経過時間差が大きい．このことより，キーワードの出現頻度の増加に対しては，提案手法の方が有効であると言える．

5.2　キーワード数の変化の影響

検索キーワード数の変化に対する影響を分析する．xmlgen001 に対して，キーワードの出現頻度を中に固定し，キーワード数を変えて実験を行った場合の経過時間を図 13 に示す．この結果より，全ての手法でキーワード数が増えることによって経過時間が増加する傾向があるのがわかる．また，キーワード数の増加による経過時間の増加量は，提案手法の方が少ないのがわかる．このことより，キーワード数の増加に対して，既存手法 Bruteforce アルゴリズムよりも提案手法の方が有効であると言える．また，キーワード数が増加すると KBSITS 手法が TS 手法よりも性能が良くなる．

5.3　結果に対する考察

既存手法 Brute-force アルゴリズムに比べて，提案手法は多いキーワード数の場合，および高い出現頻度の場合に有効であることがわかる．また，キーワード数が多い場合は、KBSITS 法の方が TS 法よりも有効であると確認した．

6.　まとめと今後の課題

本稿では，XML キーワード検索に関して，SLCA の持つ問題点に対して提案されている，VLCA に着目し，VLCA を既存手法よりも効率よく求める手法の実現のために，bit 演算によってを判定する手法を提案した．またキーワードに対し VLCA を抽出する手法である TS 法の提案を行った．次に，KBSI 手法と TS 法を組み合わせて VLCA を抽出する手法である KBSITS 手法の提案を行った．

Brute-force アルゴリズム [7] と提案手法に対して，実験を行ったところ，Brute-force アルゴリズムに対して提案手法はキーワードが多い場合，キーワード出現頻度が高い場合において特に検索時間に優れることを示した．また、提案した TS 法と KBSITS 法に関しては、検索キーワード数が多い場合に KBSITS の方がより有効であることを確認した．

今後の課題として，実験の対象の文書サイズを大きくした場合や，検索を行う対象の文書の種類も増やしてて実験を行う必要がある．さらに，既存手法 VLCAStack [7] との比較も行う必要がある．

謝　　辞

本研究の一部は，独立行政法人科学技術振興機構戦略的創造研究推進事業 CREST，および文部科学省科学研究費補助金特定領域研究 (#19024028) の助成により行なわれた．

文　　献

[1] ACM SIGMOD Record:XML Version. http://www.dia.uniroma3.it/Araneus/Sigmod/.

[2] UW XMLData Repository. http://www.cs.washington.edu/research/xmldatasets/.

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

[4] 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.

[5] 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.

[6] Vagelis Hristidis and Nick Koudas. Keyword proximity search in XML trees. In *IEEE Transaction on knowledge and data engineering,No4*, pp. 525–539, 2006.

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

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

[9] non-profit Wikipedia Foundation. Wikipedia. http://en.wikipedia.org/.

[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.