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

2006－DBS－140 (I) (25)
2006／7／12

スーパーインポーズドコーディングを用いたXML文書キーワード索引手法

三木健士[†] 横田治夫[††,†]

XML文書の中から与えられた複数のキーワードを全て含む部分文書を高速に抽出する手法として、スーパーインポーズドコードを利用した手法を提案する。XML木の葉ノード毎に割り当てたbit列を階層構造でスーパーインポーズすることで、複数キーワードが全て含まれる部分木の親を判定する。さらに、木のノードにDewey Orderのラベルを付け、キーワードに対応する葉ノードのDewey Orderのラベル間の共通部分検出で共通親を探す方法、その手法とスーパーインポーズドコードを組み合わせた手法、スーパーインポーズドコードにBit-Indexで索引付して効率を高めた方法を提案し、それぞれの手法の検索コストを見積もり比較する。

Keywords Indexing of XML Documents using SuperImposed coding

Takeshi Miki[†] and Haruo Yokota[††,†]

We propose a new method using superimposed codes to extract XML subdocuments containing multiple keywords from an XML document. The method assigns a bit sequence to a leaf node of the XML tree and superimposes the bit sequences in the hierarchical structure of the XML tree to distinguish the common ancestor nodes for leaf nodes containing all the keywords. We also propose three methods to find the common ancestor nodes: detecting the common parts in the index Dewey Order codes, combining the index Dewey Order and superimposed codes, and applying a bit index to the superimposed codes. We estimate execution costs for these proposed methods to compare them.

1. はじめに

XML文書の中から必要な部分文書をXMLの構造を考慮して抽出する要求が高まっている。そのひとつとして、与えられた複数のキーワードを全て含む部分文書を効率よく抽出する方法が研究されている[1]. 例えば、シェークスピアの作品をXML化したものの中かから、母、王、兄弟というキーワードを台詞に含む部分文書を探したい場合、一人の台詞に入っている部分、複数人の台詞に入っている部分などが考えられる.

XML文書はラベル付木としてモデル化されることから、上記のような検索は、キーワードを含む葉の最低共通親 (Lowest Common Ancestors: LCAs) [2,3]を探す問題に置き換えることができる. ただし、実際には、全てのLCAを抽出したい訳ではなく、最も小さなLCA[3]や、最小接続木 (Minimum Connecting Trees: MCTs) を探す必要がある. [1]では、索引の付いたXML木に対して入れ子ループを用いる方法やスタックを用いる方法等について提案を行っている.

本稿では、複数キーワードに対してXML木のMCTを効率よく検索する新たな手法として、スーパーインポーズドコードを利用した手法を提案する. 一般に、スーパーインポーズドコードは、文書ファイル毎にキーワードのシグニチャファイルを用意することで、文書ファイルに対するキーワード検索を高速に行うために使われる手法である[4,5]が、これをXMLのMCT検索に適用する. XML木の葉ノード毎に割り当てたbit列を階層構造でスーパーインポーズすることで、複数キーワードが全て含まれる部分木の親を判定することができる.

さらに、本稿では、木のノードにDewey Order[6]のラベルを付け、キーワードに対応する葉ノードのDewey Orderのラベル間の共通部分検出で共通親を探す方法、その手法とスーパーインポーズドコードを組み合わせた手法、スーパーインポーズドコードにBit-Indexで索引付して効率を高めた方法を提案し、それぞれの手法の検索コストを見積もって比較を行う.

† 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 学術国際情報センター
Global Scientific Information & Computing Center, Tokyo Institute of Technology

2. 関連研究

部分文書検索としては、検索要求においてそれに類似した文書の一部分だけを検索するというパッセージ検索がある [7]. これを XML に適用し、XML 文書検索で文書構造と文書の内容に基づく順位付けをし、あらかじめ検索対象を絞る手法が提案されている [8]. XIRQL [9] も同様に順位付けをし、検索対象を絞る. この二つの研究の違いは順位付けの計算モデルの違いにある. しかし、これらの研究は、単に XML 部分文章と問い合わせキーワード間の類似度を計算する枠組を提供しているのに過ぎず、検索対象をあらかじめシステム設計者が 決めている点で汎用性に欠ける. 汎用性を持たせるために、XML 部分文章の統計量を利用し検索対象である部分文章の絞り込みを行なう方法も提案されている [10]. これらの研究は、基本的に処理コスト削減のために検索対象の部分文書となりえないものを除去してから検索するフィルタリングが中心となっている.

一方、XML 文書の索引に関しては、Xpath [11] のような XML の問合わせ言語を前提とした索引の研究が多数なされている [12–14]. しかし、MCT の検索の場合にはそのまま適用することは難しい.

本稿では、部分文書は葉ノードにのみ格納されているとし、各ノードに Dewey Order のラベルを付けて、キーワードに対する Btree 索引からラベルを参照する方法とスーパーインポーズドコードを用いた方法およびスパーインポーズドコードを索引付けした方法を比較する.

3. シグネチャファイル

ここでは、まず提案の前準備として、文書検索に利用されるスーパーインポーズドコードを用いたシグネチャファイルによる索引手法の概要を述べる.

3.1 シグネチャファイルの構成方法

まず、シグネチャファイルを用いた索引では、全文書ファイルに出現するキーワードを抽出し、それぞれのキーワードに異る bit 列を割り当てる. これをワードシグネチャという. 例として、二つのファイル（図 1、2）に出現するキーワードのワードシグネチャを示す（図 3）. この後の処理のために、ワードシグネチャ中の 1 の生起確率は一般に低く抑えられる.

次に、対象となる文書ファイル中に含まれるキーワードのワードシグニチャの論理和を取ることでその文書ファイルに対するスーパーインポーズドコードを作成する (図 4,5). このスーパーインポーズドコード

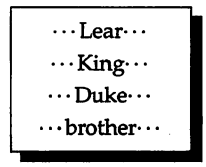

図 1 file-A

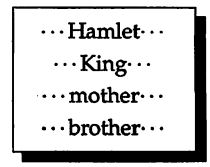

図 2 file-B

Lear	1000001
Hamlet	0100001
King	0100010
Duke	0101000
mother	0100100
brother	1100000

図 3 ワードシグネチャ

キーワード	シグネチャ
Lear	1000001
King	0100010
Duke	0101000
brother	1100000
論理和	1101011

図 4 file-A のシグニチャ

キーワード	シグネチャ
Hamlet	0100001
King	0100010
mother	0100100
brother	1100000
論理和	1100111

図 5 file-B のシグニチャ

を対応づけた表 (図 6) がシグネチャファイルである.

3.2 問い合わせ方法

問い合わせが与えられると、その問い合わせに含まれるキーワードの集合から問い合わせシグネチャと呼ばれる bit 列 (Q) を作成する. この生成方法は、問い合わせ中の各キーワードのワードシグネチャの論理和を取ることで生成する. 次に、シグニチャファイル中の各ファイル (F_i) に対応するシグニチャ(S_i) との論理積を取り、その結果が問い合わせシグネチャ (Q) と等しければ、つまり次式を満たせば、そのファイル (F_i)

ファイル名	シグネチャ
file-A	1101011
file-B	1100111

図 6 シグニチャファイル

問い合わせキーワード	問い合わせシグネチャ	file-A	file-B
King, brother	1100010	actual drop	actual drop
King, mother	0100110	no match	actual drop
Lear, King	1100011	actual drop	false drop

図 7 問合わせとドロップの例

は検索結果候補となる.

$$Q \wedge S_i = Q \quad (1)$$

このような候補はドロップと呼ばれる. ワードシグニチャの組合わせにより、ドロップの中には、実際には全てのキーワードが含まれないものもある. そのようなドロップをフォールスドロップと呼ぶ. これに対して、実際に全てのキーワードを含むものをアクチュアルドロップと呼ぶ. ドロップ中からフォールスドロップを見分ける処理は、フォールスドロップリゾリューションと呼ばれる. フォールスドロップの起こる確率は、ワードシグネチャの与え方による. この確率を最小化する研究 [15] は既に行われており、本稿の範囲ではないので、詳しくは他の研究を参考にされたい.

図 6 のシグネチャファイルに対して、いくつかのキーワードの組みに対する検索結果を図 7 に示す.

4. 提案手法

本稿では、上述のスーパーインポーズドコーディングを XML の部分木検索である MCT 検索に用いる方法を提案する. 以下では、簡単化のため、キーワードは XML 木の葉ノードのみに出現するものとする. 葉ノード以外のノードに出現するキーワードも基本的には同様に扱うことができるが、ここでは扱わない.

文書検索のシグニチャファイルでは、文書単位でスーパーインポーズドコードを生成して、問合わせシグニチャと比較していた. 本提案では XML 木の葉ノードで同様のスーパーインポーズドコードを生成するとともに、XML の木構造の階層構造に対してもコード間で論理和演算を行うことで、部分木単位のスーパーインポーズドコードを生成する. こうすることで、問い合わせ中のキーワードをすべて含む部分木を抽出することができる. 以下、この手法を SuperImposed for XML Tree (SIXT) と呼ぶ.

しかし、単に XML 木の全てのノードにスーパーインポーズドコードを割り当てただけでは、全ノードを操作する必要がある. このため、XML 木のノードに Dewey Order [6] のラベルを付けて索引に格納し、キーワードに対応する葉ノードの Dewey Order のラベル間の共通部分検出で共通親を探す方法、その手法とスーパーインポーズドコードを組み合わせた手法、スーパーインポーズドコードに Btree で索引付して効率を高めた方法を提案する. それぞれを、Keyword B+tree with Dewey Label (KBDL)、Keyword B+tree with SuperImposed Code (KBSI)、Bit Indexed SuperImposed Code (BISI) と呼ぶ. 以下、それぞれの手法に関して説明を行う.

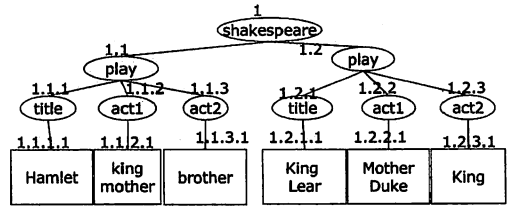

図 8 Dewey Order によるラベリング

4.1 KBDL: Keyword B+tree with Dewey Label

KBDL では、XML 木のノードに Dewey Order のラベルを付けて B+tree の索引に格納し、キーワードに対応する葉ノードの Dewey Order のラベル間の共通部分検出で共通親を探す.

Dewey Order [6] は、XML 順序木の各ノードのラベルは親のラベルに兄弟間の順序を連結したものである. Dewey Order によるラベリングの例を図 8 に示す. Dewey Order でラベリングされたノードは、ラベルを比較することで子孫・祖先関係を知ることができる. つまり、二つの葉ノードのラベルを比較することで、共通の祖先を知ることができる.

例えば、*Hamlet* と *King* を含むノードを持つ共通部分木を知りたいとする. 図 8 を見るとわかるように、その共通部分木の根ノードは、*play* である. *Hamlet* と *King* のそれぞれを含むノードのラベル 1.1.1.1 と 1.1.2.1 を比較すると共通部分は 1.1 であり、そのラベルは *play* を指している.

KBDL では、出現キーワードに対して B+tree を構成し、その各葉ノード（以下キーワードノード KN_i と呼ぶ）にキーワードと一緒にそのキーワードの XML 木中の出現する葉ノードの Dewey Order を格納する.

検索時は、n 個の問い合わせキーワード $q_1, q_2, .., q_n$ に対し、B+tree によりそれぞれのキーワードノード $KN_1, KN_2, ..., KN_n$ を得る. 問い合わせキーワード q_i と q_j を含む XML 木のノードの共通の親は、KN_i と KN_j の Dewey Order のラベルを比較することで得ることができる. よって、$KN_1, KN_2, ..., KN_n$ の中のキーワードの組みのラベルを順番に比較していくことで、全てのキーワードを含む共通の親を得ることができる.

4.2 SIXT: SuperImposed for XML Tree

SIXT では、図 8 の XML 木のそれぞれ葉ノードに

Lear	1000001
Hamlet	0100001
King	0100010
Duke	0101000
mother	0100100
brother	1100000

1.1.2.1 のシグネチャ	
king	0100010
mother	0100100
論理和	0100110

図 9　(左) ワードシグネチャの例, (右) 葉ノードのシグニチャの例

1.1.1	0100001
1.1.2	0100110
1.1.3	1100000
1.1	1100111

図 10　葉ノード以外のシグニチャ

出現するキーワードに対してワードシグニチャを割り当てる.

ワードシグニチャの生成する時、W 個 1 を持つ固定長ビット列 (Fbit) を作成する関数として、以下では次のようなハッシュ関数を前提とする.

$$H(x) = \{(0|1)^* | \text{長さ } F, 1 \text{ の個数 } W\} \quad (2)$$

- 入力 : x にはキーワード
- 出力 : ビット列

このハッシュ関数を用いて図 8 に現れるキーワードに対してワードシグネチャを計算したものが図 9 (左) である. また、葉ノードのシグニチャの例として、1.1.2.1 のシグネチャは図 9 (右) のようになる.

次に葉ノード以外のシグネチャは、そのノードのすべての子ノードのシグネチャ(ノードシグネチャ) を求め、それらの論理和をとる. これを再帰的にルートまで繰り返し、すべてのノードシグネチャを作成する.

例えば図 8 の 1.1.1 の *play* のノードシグネチャを求めたい時は、その子ノード 1.1.1.1〜1.1.1.3 のノードシグネチャを求め、それらの論理和を計算する (図 10).

同様に、XML 木のすべてのノードシグニチャを求め、それぞれのノードのラベルとノードシグネチャの組をまとめたノードシグネチャファイル (図 11) に作る.

検索の際、文書ファイルの時と同様にまず問い合わせシグネチャ(Q) を求め、そのシグネチャとノードシグネチャファイルのノードシグニチャ (NS_i) が以下の式を満たすか調べる.

$$Q \wedge NS_i = Q \quad (3)$$

その式が成り立つノードはその問い合わせキーワードを含むノードを子ノードとして含む. つまり、すべての問い合わせキーワードを含む部分木を得ることができる. ただし、フォールスドロップを含む可能性がある.

signature	label(element node name: node name)	Keywords
1000001	1.1.1.1(text node)	Hamlet
1000001	1.1.1(title:element node)	
1100111	1.1(play:element node)	
0100110	1.1.2.1(text node)	king mother
...	...	...

図 11　SIXT のノードシグニチャファイル

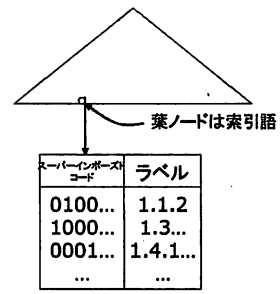

図 12　KBSI の構成

4.3 KBSI: Keyword B+tree with Superimposed Code

Keyword B+tree with Superimposed Code (KBSI) 手法は、KBDL と SIXL の手法を組み合わせてものである.

この手法では、KBDL のキーワードノードに SIXL を使用する. つまり、KBDL ではキーワードと Dewey Order のラベルの組みのみが入っているキーワードノードに SIXT で求めた XML 木そのノードからルールまでのシグネチャも一緒に格納する (図 12). 検索においては、問い合わせに含まれるキーワードの中の任意のひとつのキーワードでまず B+tree を走査し、対応するキーワードノードを得た後、そのキーワードノード中のシグニチャの中で、他の問い合わせキーワードのワードシグニチャを含む上位ノードのラベルを見つける.

KBDL では、問い合わせキーワードに対してそれぞれのキーワードノード求めた後すべてのラベル比較して共通の親を求めたが、KBSI では、1 つのキーワードのキーワードノードを走査するだけで共通の親を求めることができる.

4.4 BISI: Bit Indexed Superimposed Code

Bit Indexed SuperImposed Code(BISI) は、SIXT と Bit-Index を利用した索引である.

SIXT に使用するシグネチャの長さ F とすると、Bit-Index は長さ F の配列を使い、配列の k 番目の位置に、ノードシグネチャのビット列の位置 k が 1 である全てのノードシグニチャとそのノードラベルを格納し

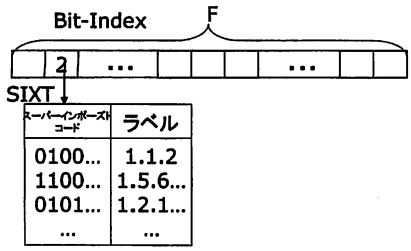

図 13 BISI の構成（$k = 2$ の場合）

たファイルへのポインタを持つ.

問い合わせに対して、SIXT では XML 木の全てのノードシグニチャを走査する必要があったが、BISI ではあるひとつのキーワードのワードシグニチャで 1 が立っている bit で絞り込まれたノードシグニチャだけ走査すればよいことになる.

例えば複数の問い合わせキーワードのあるキーワードのシグニチャの 2bit 目が 1 だった場合に、図 13 のように Bit-index の $k = 2$ からポイントされているファイルのシグニチャに対して SIXT と同様の処理をすることで部分木を見つけることができ、検索速度が速くなる.

5. コスト見積り

この章では、提案手法の索引と比較する手法として転置索引を最悪の場合の処理コストの見積り、提案手法間の比較を行う.

見積り使う XML 文書の XML 木を図 14 に表す. このときの見積りにおけるパラメタとして、以下を考える.

- S:問い合わせキーワード数
- K:全キーワード種類数
- k:B+木のファンアウト（ノードのエントリ）数
- N:XML 木の全葉ノード数
- w:1 ノード当たりの平均キーワード種類数
- F:1 キーワード当たりの平均ノード数
- b:1 bit 比較コスト
- B:1 String 比較コスト
- r:ラベルの平均 String 数
- l:シグネチャのビット長
- n:検索キーワードの平均 String 長
- d:Dewey Oeder の平均ラベル長

5.1 KBDL の見積り

KBDL では、B+tree の葉ノードに Dewey Order のラベルを入れる. パラメタから、エントリ数が k、B+tree のファンアウトが k であるので, 木の高さは

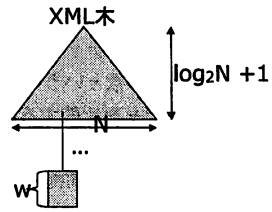

図 14 見積りにおける XML 文書木

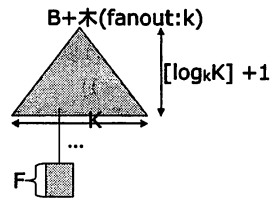

図 15 KBDL に使用する構造

$[log_k K] + 1$ となる (図 15).

最悪の場合、問い合わせキーワード 1 つに対して、それぞれの B+tree の各階層エントリをすべて比較することになるので、葉ノードまでの比較回数は $S \times k \times ([log_k K] + 1)$ となる. 各 String の比較コストは B で、キーワード長は n なので比較コストは、以下の式である.

$$S \times k \times ([log_k K] + 1) \times B \times n \quad (4)$$

ここまでで、問い合わせキーワードを含む XML 木のノードのラベルが得られる. 次にそれらのラベルを比較して、すべてのキーワードを含むノード、つまり部分木の根ノードのラベルを得るためのコストを求める. 1 キーワード当たりのラベル数は F なので S 個のキーワードを入力とすると比較回数は F^{S-1} である. そして、ラベルの長さ d であるので比較コストは以下の式になる.

$$F^{S-1} \times B \times n \quad (5)$$

よって、KBDL のコストは以下のようになる.

$$S \times k \times ([log_k K] + 1) \times n \times B + F^S \times d \times B \quad (6)$$

F を N と K を用いて表すと、$F = \frac{wN}{K}$ となる. これを上式に代入した上で N に関するオーダーで考えると、第 2 項のみとなり、

$$O(F^{S-1}) = O((\frac{wN}{K})^{S-1}) = O(N^{S-1}) \quad (7)$$

となる. つまり、XML 木の全葉ノード数の問い合わせキーワード数のオーダーとなる.

5.2 SIXT

SIXT では、まず、問い合わせキーワードのそれぞれ

のワードシグニチャから問い合わせシグニチャを生成し、各中間ノードがそのシグニチャを包含するかどうかの比較を行う．この比較回数という面ではXML木の全中間ノードとシグニチャを比較する必要がある．

XML文書の一般的な構造は多分木であるので、各中間ノードが必ず分岐するという前提の下で葉ノードがN個の場合のもっとも中間ノード数が多くなるものは完全2分木である．葉ノードがNの完全2分木の葉以外のノード数は$N-1$であることから，シグニチャのビット長をl、ビット比較演算のコストをbとすると、提案手法の検索コストは以下の式となる．

$$l \times b \times (N-1) \quad (8)$$

よって、Nに関するオーダーで考えると$O(N)$となる．

5.3 KBSI

KBSIの問い合わせ構造のB+tree中の検索方法はKBDLと同じなので、最初のキーワードノードに至るまでの比較コストは以下の式になる．

$$S \times r \times ([log_k K]+1) \times n \times B \quad (9)$$

KBSIの場合、最初のキーワードノードを求めた後は、そのキーワードノード内の格納されているラベルと比較すれば部分木を得ることができる．キーワードノード内のラベル数はFで、比較対象はシグネチャなので1bit比較コストbであり、シグネチャ長lであるため、比較コストは以下の式になる．

$$S \times k \times ([log_k K]+1) \times n \times B + b \times l \times F \quad (10)$$

KBDL同様にNに関するオーダーとしては、第2項のみとなり、$F = \frac{\alpha N}{K}$を代入すると、$O(\frac{\alpha N}{K}) = O(N)$となる．

5.4 BISI

BISIでは、ある1つの問い合わせキーワードによって指定したBit-Indexの位置に格納されているSIXT手法のシグネチャファイルと問い合わせシグネチャを比較する．

Bit-Index長（シグネチャ長）はl、ワードシグニチャ中の1の数がWであるから、全てのbitの1の生起確率が等しいとすると、あるbitからポイントされるシグニチャファイル中のシグニチャの数は$N \times \frac{W}{l}$となる．一般には、Wはシグニチャの特性上、小さく抑えられる．

上記より、1bit当りの比較コストはbであることから、BISIの比較コストは、

$$W \times b \times N \quad (11)$$

となり、オーダーとしては、同じく$O(N)$である．

5.5 見積り比較

オーダーのレベルでの提案手法の検索コストの比較をまとめると、KBDLは$O(N^{S-1})$、SIXT、KBSI、BISIはすべて$O(N)$である．係数から見ると、一般的には$l > W > \frac{\alpha l}{K}$と予想されることから、KBDL > SIXT > BISI > KBSIという順番で検索コストが下がると予想される．

6. おわりに

XML文書に対して複数の検索キーワードをすべて含むXML部分木を高速に得ることができる索引手法として、スーパーインポーズドコードを用いるSuper-Imposed for XML Tree (SIXT)を提案した．XML木の葉ノード毎に割り当てたシグニチャを階層構造でスーパーインポーズすることで、複数キーワードが全て含まれる部分木の親を判定する．

また、その効率化に関連して、木のノードにDewey Orderのラベルを付け、キーワードに対応する葉ノードのDewey Orderのラベル間の共通部分検出で共通親を探すKeyword B+tree with Dewey Label (KBDL)、KBDLとSIXTを組み合わせたKeyword B+tree with SuperImposed Code (KBSI)、SIXTにBit-Indexで索引付して効率を高めたBit Indexed SuperImposed Code (BISI)を提案した．

それらの提案手法に対して、いくつかのパラメタを設定して検索コストの見積もりを行った結果、XML木のサイズに対して、KBSIの検索コストが最小であると予想できる．ただし、今回はあくまでも見積もりであり、今後これらの手法を実装し、様々なXML文書対し比較検証する必要がある．また、今回の見積りにはフォールスドロップリゾリューションのコストが含まれていないことから、フォールスドロップリゾリューションを効率的に行う方法も検討する必要がある．さらに、[1]で提案されているスタックを用いた手法等との比較も必要である．一方、得られた部分木を区別して、もっとも適した部分木を提示する方法等も検討する必要があると考えている．

謝辞 また本研究の一部は，文部科学省科学研究費補助金特定領域研究(18049026)，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および東京工業大学21世紀COEプログラム「大規模知識資源の体系化と活用基盤構築」の助成により行なわれた．

参考文献

1) Hristidis, V. and Koudas, N.: Keyword Proximity Search in XML Trees, *IEEE Transaction on knowledge and data engineering*, No.4 (2006).
2) Li, Y., Yu, C. and Jagadish, H.V.: Schema-Free

XQuery., *VLDB*, pp.72–83 (2004).
3) Xu, Y. and Papakonstantinou, Y.: Efficient Keyword Search for Smallest LCAs in XML Databases., *SIGMOD Conference*, pp. 537–538 (2005).
4) Lee, D.L., Kim, Y.M. and Patel, G.: Efficient Signature File Methods for Text Retrieval, *IEEE Transaction on knowledge and data engineering*, No.3 (1995).
5) Faloutsos, C. and Christodoulakis, S.: Description and Performance Analysis of Signature File Methods for office filing, *ACM TOOIS*, No.3, pp. 237–257 (1987).
6) Tatarinov, I., Viglas, S.D., Beyer, K., Shanmigasundaram, J., Shekita, E. and Zhang, C.: Storing and Querying Ordered XML Using a Relational Database System, *Proc. ACM SIGMOD 2002*, pp. 204–215 (2002).
7) Salton, G., Allan, J. and Buckley, C.: Approaches to Passage Retrieval in Full Text Information System, *Proc. of the Sixteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp.49–58 (1993).
8) Hayashi, Y., J.Tomita and Kikui, G.: Searching text-rich XML documents with relevance ranking, *Proc. of ACM SIGIR 2000 Workshop on XML and Information Retrieval*, pp.204–215 (2000).
9) N.Fuhr and FroBjohann, K.: XIRQL: A query language for information retrieval in XML document, *Proc. of 24th Annual International ACM SIGIR conference on Research and Development in Information Retrieval*, pp.172–180 (2001).
10) 波多野賢治，絹谷弘子，吉川正俊，植村俊亮：XML 文書検索システムにおける文書内容の統計亮を利用した 検索対象部分文章の決定，電子情報通信学会論文誌， No.3, pp.422–431 (2006).
11) Clark, J. and S.DeRose: XML path language (Xpath) version1.0 (1999).
12) Goldman, R. and Widom, J.: Dataguides: Enabling query formulation and optimization in semistructured databases, *Proc. of VLDB*, pp. 436–445 (1997).
13) Raw, P.R. and Moon, B.: PRIX: Indexing and querying XML using prufer sequences, *Proc. of ICDE*, pp.288–300 (2004).
14) SHIMIZU, T. and YOSHIKAWA, M.: FULL-Text and Structual Indexing of XML Documents on B+-Tree, *IEICE TRANS. INF. & SYST.*, No.1 (2006).
15) Faloutsos, C. and Christodoulakis, S.: Description and performance analysis of signature file methods for office filing, *ACM Trans Office Information System*, No.3, pp.237–257 (1987).