情 報 学 基 礎 57－2
自 然 言 語 処 理 136－2
（2000. 3. 21）

大規模な構造化文書データベースにおける インデクシングと検索の手法

井形 伸之　難波 功
{igata, namba}@flab.fujitsu.co.jp
富士通研究所
〒211-8588 川崎市中原区上小田中 4-1-1

本稿では、大量の構造化文書に対する検索要求を全文検索エンジンを用いて高速に処理する手法について提案する。本手法では、文書の構造と内容に対して2種類のインデックスを用いる。構造用のインデックスには文書集合の木構造を格納する。内容用のインデックスには term と term が出現する field ID および文書 ID を格納する。検索時には、構造用のインデックスを参照し、階層的な構造に関する検索要求を平らな構造に関する検索要求 (field または zone 検索) に変換する。実際に XML 文書を用いた実験を行ない、大規模な構造化文書データベースにおける本手法の有効性を確認した。

A Method of Indexing and Searching for Large Scale Structured Document DataBases

Nobuyuki Igata　Isao Namba

Fujitsu Laboratories Ltd.
4-1-1, Kamikodanaka, Nakahara-ku, Kawasaki, Kanagawa, 211-8588 Japan

In this paper, we propose a high-speed query processing method for a large scale structured document set using a full text search engine. Our method uses two types of indexes for document structures and contents: *Structure Index* stores tree structure of document set, and *Text Index* stores term, field IDs, and document IDs. In searching method, hierarchical structured queries are converted into flat structured queries (so called field or zone query) by accessing *Structure Index*. Through some experiments on XML documents, we proved the effectiveness of our method in large scale structured document databases.

1 はじめに

XML(eXtensible Markup Language) の登場以来、XML に対する人々の関心は高く、XML を処理するアプリケーションに大きな期待が寄せられている。XML はインターネット時代の文書記述言語と位置付けられ [1]、その周辺技術も含めて、今後、大きく発展するものと予想される。

一方、インターネットの流行を支えてきたアプリケーションの 1 つに検索エンジンがある。大量の文書からユーザの求める文書を高速に発見する検索エンジンは、インターネットを利用する上で必要不可欠なアプリケーションと言えるが、現在の検索エンジンに対するユーザの満足度は必ずしも高くない。

今後、インターネット上に流れる情報の多くが XML で記述されることを考えると、検索エンジン自体が XML 文書の構造を認識し、処理する利点は大きい。例えば、文書内容の検索だけでなく文書構造の検索をも実装した検索エンジンには、(1) 文書の構造を用いた高度な検索要求による検索結果の高精度化、(2)HTML 文書では不可能だった新しい検索サービスの提供、(3) 構造化文書を処理するデータベースの文字列検索性能の強化、などの多くメリットが考えられる。

文書の構造を検索する方式については、これまでにもいくつか提案されてきたが、論文 [2] によれば、「文書の内容に着目した検索方式の研究と比較して、文書の構造と内容に着目した検索方式の研究は少なく、十分な検証が行なわれていない」と言われている。

そこで、本稿では、文書の構造と内容に基づいた検索要求（以下、構造・内容検索と呼ぶ）を処理する全文検索エンジンのインデクシングおよび検索方式について検討する。特に、(1) 大量の構造化文書が対象 (大規模対応)、(2) 文書の構造に対する高度な検索要求 (高機能性)、(3) 実装または既存資産の改良のコスト (実装の容易さ)、の 3 点に注目する。

2 関連研究

SGML(Standard Generalized Markup Language) の制定と共に、データベースの分野では、構造化文書の格納・検索に関する研究が盛んに行なわれている。例えば、構造化文書をオブジェクト関係データベースに格納し、構造と内容に基づく問い合わせを SQL に変換して検索する手法 [3] がある。しかしながら、既存のデータベースで提供されている検索機能だけでは、部分文字列検索に対する十分な性能が得られないため、全文検索エンジンを併用した商用システムも多い。

一方、情報検索の分野でも 80 年代の後半から構造・内容検索に関する方式がいくつか提案されている [2]。また、日本においても、構造化文書を対象とした商用の検索システム [4, 5] がある。

一般に、大量の文書を高速に検索する全文検索エンジンのインデクシング方式には、inverted file [6] が使用される。これまで提案されてきた構造化文書の検索手法の中で、inverted file の枠組で実装可能な手法を以下にまとめる。

Hybrid Model [7]: 文書をいくつかの field に分割し、field 情報を文書 ID や出現位置情報 (offset) と共に inverted file に登録する方式。最も単純な手法であり実装も容易。検索速度も高速。ただし、field はフラットな構造となるため、階層的な構造の照合を行なうためには他の枠組が別途必要となる。ある field における offset の範囲を全てのテキストにおいて共通となるようにすることで、特に field 情報を持たずに field を特定する方式 [4] もある。

Overlapped Lists [8]: 各要素の領域情報 (開始位置、終了位置) を通常の単語リスト情報と同様に inverted file に登録する方式。従来のブーリアン代数だけでなく、領域判定用の命令 (リージョン代数:region algebra) が追加される。論文 [8] の GC-list(Generalized Concordance list) では、ネスト構造の照合ができない。

Structure Index + Text Index : 構造に関するインデックス (以下、構造インデックス) と内容に関するインデックス (以下、テキストインデックス) の 2 種類のインデックスを持つ方式。構造インデックスを木構造で持つ方式 [5] や、代表的なノードに関する階層情報のみ持ち、構造の照合を高速化する手法 (Proximal Node[9]) などがある。テキストインデックスには Hybrid Model が使用される。他手法と比較して構造の照合に対する記述能力が高い。

上記以外の方式としては、検索結果を文書 ID で返すのではなく、検索結果となる要素の情報を格納し

たデータベース上のポインタを返し、全文検索エンジンの外部において、文書構造の照合や文書IDを得る方式(Lists of Refences[10])がある。また、インデックスを持たない形式では、オートマトンを用いた照合手法[11]も提案されている。

3 構造化文書検索と基本方針

3.1 対象とする構造化文書

本稿で対象とする構造化文書の例を図1に示す。図1は、繰り返しや欠損のある半構造データを示したものである。要素のネストや兄弟要素の出現順序が任意のものも対象とする。ただし、リンクや参照といった木構造で表現されない構造は対象外とする。

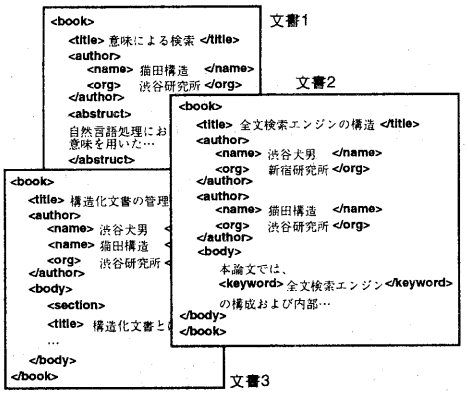

図 1: 構造化文書の例

3.2 構造化文書検索に求められる機能

従来から構造化文書検索に求められていた機能を以下にまとめる。これらの機能は、実装上、いくつかのレベルに分けることができる。レベルが上位になるに従って、高度な構造・内容検索が可能となるが、検索エンジン内部での処理が複雑になり、速度低下の原因となる。

機能レベル 1: 文書構造の要素が指定できること。『題名』に含まれる文字列と『本文』に含まれる文字列を区別して検索するなど。

機能レベル 2: 文書構造の親子関係が指定できること。『文書』の下の『題名』と『参考文献』の下の『題名』を区別して検索するなど。

機能レベル 3: 文書構造の兄弟関係が指定できること。ある『章』の『章題』に文字列Aを含み、同じ『章』の『本文』に文字列Bを含む文書を検索するなど。

機能レベル 4: 要素の出現位置や順序が指定できること。ある要素の3番目の子供に含まれる文字列を検索するなど。

機能レベル 5: W3C (World Wide Web Consortium) で勧告される XML 問い合わせ言語の記述能力と同等の検索条件が指定できること。

現在、W3C では XML 問い合わせ言語の標準化作業が行なわれているが、これはデータ指向 (data-oriented) を含んだ言語仕様[12]となることが予想され、文字列のみを扱う inverted file だけでの対応は難しい。また、「『ある要素の3番目の子供を得る』というような検索要求は個々の文書に極端に依存した検索要求であり、汎用のデータベースには必要ない」といった主張もある[13]。

全文検索エンジンに実装されるべき機能に関しては速度面・効率面からも議論が残るが、本稿では機能レベル1～4を対象とし、用途により、上記機能レベルを容易に変更できる枠組を提供する。

3.3 設計の基本方針

構造・内容検索を実装した全文検索エンジンを設計するに当たり、本稿では以下の点を考慮した。

大規模対応: 全文検索エンジン内部で繰り返し実行される処理をできるだけ避け、大規模な文書データベースを対象とした場合に、高速性を維持できる設計とする。例えば、構造に関する情報を offset に持たせる設計では、文書量の増加と共に offset 演算の回数が増加するため、大規模対応は難しいと考えられる。

高機能性: 3.2節の機能における機能レベル4までを対象とする。ただし、機能の充実度と速度性能はトレードオフの関係にあるため、用途に合わせて機能レベルを容易に変更できる柔軟な設計とする。

実装の容易さ: 既存の検索エンジンの枠組を壊すことなく、構造・内容検索を実現する設計とする。これにより、機能追加による実装コストを

抑え、既存資源を有効利用できる。例えば、現在商用化されている検索エンジンの中で、リージョン代数 [8] を実装しているものは少なく、リージョン代数による構造・内容検索の実現は実装コストが高いと考えられる。

その他、問い合わせ言語は特に規定せず、内部的なデータ形式を用いて表現することにより、問い合わせ言語に依存しない設計とした。

4 インデクシングおよび検索手法

3.3節の基本方針を踏まえ、本手法では「Structure Index + Text Index」モデルを用いる。構造インデックスのデータ構造を木構造とし、それぞれの機能レベルに合わせた構造インデックスの作成方法を示す。また、階層的な構造に対する検索要求をフラットな構造を検索する Hybrid Model 用の Query に変換する手法を提案する。

4.1 インデクシング手法

インデクシング手法の基本的なアイデアは、文書を要素単位で分割し、分割された要素が持つテキストデータの1つ1つに field ID を割り当て構造インデックスおよびテキストインデックスに登録することである。構造インデックスは各 field の階層的な構造を管理し、テキストインデックスには検索キーが出現する文書 ID と field ID を登録する。

構造インデックス 構造インデックスは、複数の文書の中で共通する構造を重ね合わせて、検索対象となる文書の構造を1つの大きな木構造で表したものである。木構造中の1つのノードが1つの field に対応する。共通する構造の判定には、経路情報 (パス)、要素名、兄弟要素における出現位置など、いくつかの判定条件が考えられるが、検索エンジンがサポートする機能レベルに合わせて以下の3つのタイプを選択可能とする。

タイプ 1: 親ノードが同一であり要素名が同一の場合に同一の構造であると判定。機能レベル 2 までサポート。

タイプ 2: 親ノードが同一、要素名が同一、兄弟要素中の同じ要素名のノードにおける出現位置が同一の場合に同一の構造であると判定 (論文 [5] 方式)。機能レベル 3 までサポート。

タイプ 3: 親ノードが同一、要素名が同一、兄弟要素中の出現位置が同一の場合に同一の構造であると判定。機能レベル 4 までサポート。

図 1の文書例における各文書の構造を木構造で表現したものと、上記タイプ別に作成した構造インデックスの概念図を図 2に示す。各ノードは、そのノードに割り振られた field ID や兄弟要素中の出現位置 (offset) の情報を持つ。また、親ノードへのリンク (親リンク)、長男ノードへのリンク (子供リンク)、兄弟ノードへのリンク (兄弟リンク) により木構造を表現する。さらに、検索時に同一の要素名を持つノードを高速に参照するためのリンク (要素リンク) も持たせ、同じ要素名を持つ次のノードにリンクを張る。

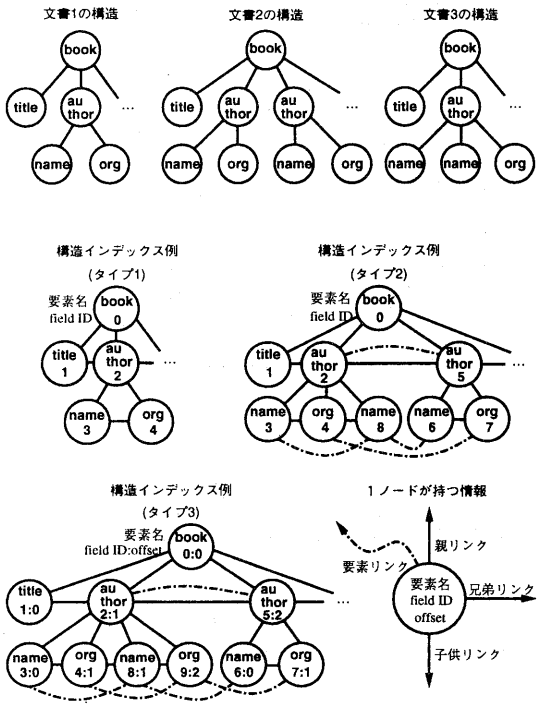

図 2: 構造インデックスの概念図

テキストインデックス テキストインデックスは、通常の inverted file に検索キーが出現する field ID を追加した形式 (Hybrid Model) となる。図 3にテキストインデックスの概念図を示す。

通常の inverted file を Hybrid Model に拡張する最も簡単方法は、検索キーを登録する辞書部に field

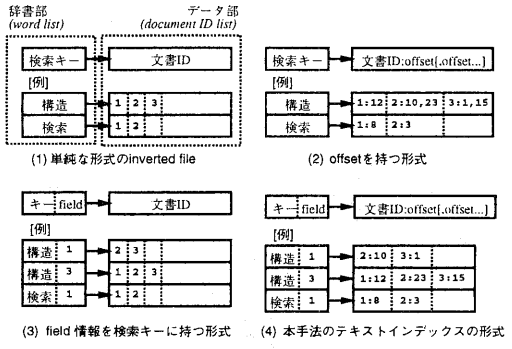

図 3: テキストインデックスの概念図

ID の情報を付加する方法である (図 3(3))。この方法は、データ部に出現位置情報 (offset) を持たない形式の inverted file でも簡単に Hybrid Model に拡張することができる。

offset を持つ inverted file(図 3(2)) の場合、field 毎に offset の値を調整し、field ID を持たない方法 [4] がある。日本語検索エンジンのほとんどが、offset 情報を持ち、N-gram の隣接演算を行なっていることを考えると、この方法も従来システムからの拡張が容易であると言える。ただし、offset に対する演算は全文検索エンジン内部において最も時間の掛かる処理であり、文書量の増加に比例して offset に対する演算回数が増加する。

以上の点を考慮し、本手法では、inverted file の辞書部に field ID を持つ形式を採用した。この形式は、文書構造の複雑さと比例して辞書に登録される検索キーが増加する欠点があるが、検索速度は文書量の影響を受けにくい特徴を持つ。また、field ID とは別に、N-gram の隣接演算や近接演算のための offset を inverted file のデータ部に持つ形式とした。

4.2 検索手法

以下に構造インデックスとテキストインデックスを用いた構造・内容検索の検索手順を示す。

1. 構造と内容による検索要求から、内部的なデータ形式である問い合わせ木を作成、
2. 構造インデックスを参照し、問い合わせ木を Hybrid Model 用の Query (field ID に対する照合条件も含んだブーリアン式。以下、Hybrid Query と呼ぶ) に変換、
3. テキストインデックスを参照し、Hybrid Query を真とする文書 ID の集合を得る

問い合わせ木 問い合わせ木の作成手順は、システムが使用する問い合わせ言語に依存するため、ここでは問い合わせ木のデータ構造を示すのみとする。

図 4に問い合わせ木の概念図を示す。問い合わせ木の各ノードは、検索要求中の 1 つの要素に関する検索条件を表し、階層的な構造に関する検索条件を木構造によって表現する。図 4に示した検索要求は Unordered Model[14] の例であるが、Ordered Model の場合ならば出現位置に関する検索条件を各ノードに持たせればよい。

Hybrid Query への変換 問い合わせ木を Hybrid Query へ変換するアルゴリズムを図 5に示す。基本動作は、構造インデックスの木構造と問い合わせ木の木構造の照合である。

構造インデックス中の要素リンクを用いることで、問い合わせ木のノードに対応する構造インデックスのノードを高速に探索し、親子関係の照合を行なう。この時、Hybrid Query の項となるノード (以下、項ノードと呼ぶ。AND, OR 等のブーリアン記号用のノードを記号ノードと呼ぶ) を作成する。項ノードには、テキストデータの照合条件、対応する構造インデックスのノードの field ID、および兄弟を持つ場合には親ノードの field ID(group ID と呼ぶことにする) の 3 つの情報を持たせる。

図 6に、タイプ 2 の構造インデックスを用いた Hybrid Query への変換例を示す。始めに、問い合わせ木に対応する構造インデックス中のノードに、項ノードを登録する。また、子供を複数持つ問い合わせ木のノード (例では<author>) に対応する構造インデックス中のノードをスタックに入れる (図 6(1))。次に、スタックに積まれたノードを基点に構造インデックス中に登録された項ノードを組み立てる (図 6(2))。スタックが空となったら部分的に組み立てた AND 式を OR 結合し、最終的な Hybrid Query を得る (図 6(3))。図 6(4) は、図 6(3) の各項を「検索キー **in** field ID」の形式で文字列表現したものである。

図 5の変換アルゴリズムは、すべてのタイプの構造インデックスに使用できる。ただし、検索要求が兄

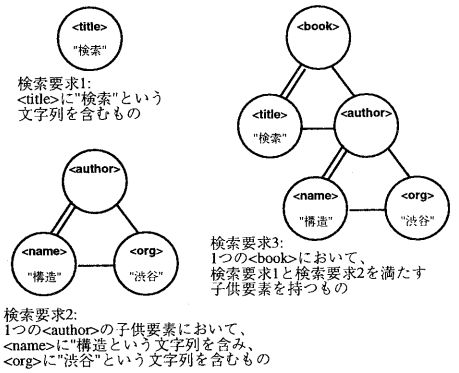

図 4: 問い合わせ木の概念図

Step 1: 親子関係の照合と項ノードの作成

問い合わせ木のノードを深さ優先で巡回し、各々のノードに対して以下の処理を行なう。

Step 1.1: 問い合わせ木のノードに対応する構造インデックスのノード集合を取得し、親子関係の照合を行なう。

Step 1.2: 上記でマッチした構造インデックス中の各ノードに対し、以下の処理を行なう。

Step 1.2.1: 問い合わせ木のノードが複数の子供ノードを持つ場合、構造インデックスのノードをスタックに保存する。

Step 1.2.2: 問い合わせ木のノードがテキストデータの照合条件を持つ場合、項ノードを作成し、構造インデックスのノードに登録する。

Step 1.2.3: 構造インデックスのノードから対応する問い合わせ木のノードにリンクを張る。

Step 2: 兄弟情報の照合と Hybrid Query の組み立て

Step 2.1: Step 1.2.1 のスタックに保存されたノード (以下、グループポイントと呼ぶ) を 1 つ 1 つ取りだし、グループポイントより下のノードに登録されている項ノードに対し、以下の処理を行なう。

Step 2.1.1: 問い合わせ木へのリンク先が同一である項ノードを OR 結合 (OR の記号ノードを用意し、項ノードをまとめて OR 式を作成)。

Step 2.1.2: グループポイントの field ID と同じ値の group ID を持つ項ノードおよび OR 式を AND 結合する。AND の記号ノードの field ID にはグループポイントの field ID を入れる。グループポイントに対応する問い合わせ木のノードが兄弟を持つ場合には、AND の記号ノードの group ID にグループポイントの親ノードの field ID を入れる。

Step 2.1.3: グループポイントに対応する問い合わせ木へのリンク先が持つ子供ノードの数と Step 2.1.2 の AND 式の子供ノードの数を比較し、数が一致しない AND 式を廃棄する (兄弟関係の照合)。

Step 2.1.4: Step 2.1.3 の処理で、数が一致した AND 式をグループポイントに登録する。

Step 2.2: スタックが空となったら、構造インデックスに登録されている全ての項ノードを OR 結合する。

図 5: Hybrid Query への変換アルゴリズム

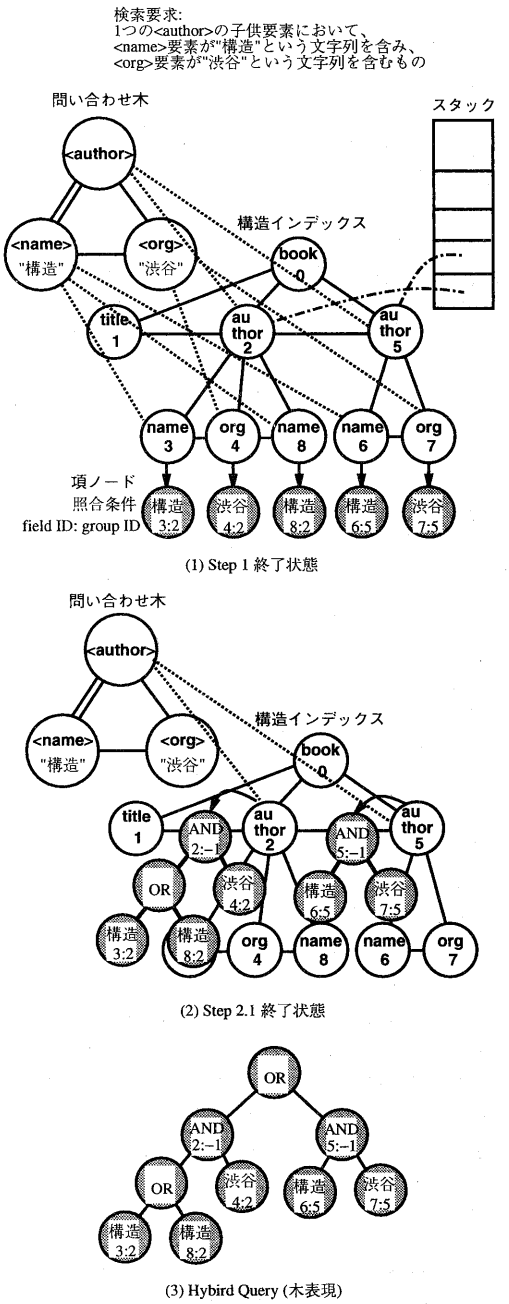

図 6: Hybrid Query への変換例

弟関係の指定を含むのであれば、タイプ2, 3の構造インデックスを用いなければならず、またOrdered Modelの場合なら、タイプ3の構造インデックスを用いて、Step 2.1.3 の兄弟関係の照合時に、各ノードの出現位置を比較することになる。

Hybrid Queryの評価 テキストインデックスを参照し、Hybrid Queryに該当する文書IDを得る。テキストインデックスを参照する手順は、fieldに関する条件が追加されている他は、従来の全文検索エンジンで実装されているブーリアン式の評価手順[6]と同様の手順となる。

5 評価

4節で示した手法を実装し、評価実験を行なった。構造インデックス作成プログラムとHybrid Queryへの変換ルーチン[1]を新規開発し、テキストインデックスの作成・検索プログラムには筆者らが開発した全文検索エンジン[15]を流用した。また、同エンジンにリージョン代数を処理するルーチンを追加してOverlapped Lists[8]を実装し、本手法との性能比較を行なった。

5.1 実験環境

Ultra Sparc II 296MHz x 2CPU, 主記憶1GBの計算機を使用し、検索対象にはJon Bosak氏がXML化したシェークスピアの戯曲[2]を用いた。

表1に検索対象文書の基本データを示す。文書数は37件と小さいが、平均文書サイズは 206.7 KBあり、通常の文書データ (新聞データ平均1KB,Webデータ平均2KB) と比較して、非常に大きい。また、1文書当たりの要素数は5,000個弱あり、十分に複雑な構造を持った文書集合であると言える。

さらに、大規模対応の検証として、表1のデータを10, 100, 200倍に膨らませた実験を行なった。

5.2 実験結果

表2, 3に表1のデータに対する各手法のインデックスサイズを示す。空白および記号により term を切り出し、インデックスに登録した (記号類を含む)。case foldingのみ使用し、stop words, stemmingといった技法[6]は使用していない。

表4に検索要求「**<SPEAKER>**のテキストデータに"HENRY"を含むもの」(該当文書数:10件, 該当要素数:813件) における各手法の検索時間 (該当文書ID列を取得するまでの時間) を示す。また、文書件数を10, 100, 200倍してデータ量を増やした時の各手法の検索時間を図7に示す。各測定値は、10回の試行の平均値を取ったものである。

表 1: 検索対象の基本データ

文書数	37件
文書サイズ (1文書平均)	7.65MB(206.7KB)
総要素数 (1文書平均)	179,618個 (4,854.5個)
テキスト部のサイズ (1文書平均)	4.54MB(125.6KB)
総term数 (1文書平均)	860,359個 (23,253個)
単純パスの種類	57個
termの種類	22,948個

表 2: 構造インデックスサイズ

	type1	type2	type3	Overlap
filed数	57	50,028	56,749	—
index size(KB)	1.8	1563.4	1773.4	—

表 3: テキストインデックスサイズ

		type1	type2	type3	Overlap
登録キー数		31,933	877,504	896,766	22,989
index size (MB)	辞書部	0.48	13.4	13.7	0.35
	データ部	2.4	7.6	7.6	2.2
	total	2.88	21.0	21.3	2.55

表 4: 37件の検索時間 (単位: 秒)

	type1	type2	type3	Overlap
Query変換	0.004	0.030	0.035	—
辞書部参照	0.000	0.044	0.050	0.000
データ部参照&評価	0.001	0.040	0.047	0.002
total	0.005	0.114	0.132	0.002

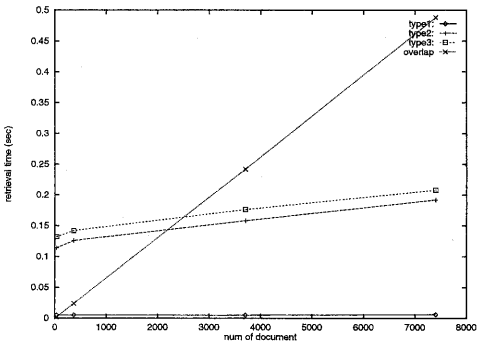

図 7: 文書量の増加と検索時間の推移 (単位: 秒)

[1] XML-QL[16] のサブセットを解釈し、問い合わせ木に変換するパーザをyacc/lexで実装し、検索エンジンの入力とした。これは、XQL[17]よりもXML-QLの方が複数文書に対する問い合わせを意識した言語仕様であるなどの理由による。

[2] http://metalab.unc.edu/pub/sun-info/standards/xml/eg/shakespeare.1.10.xml.zip

5.3 考察

タイプ2, 3では、構造の複雑さからfield数が爆発し(表2)、結果的にインデックスサイズを肥大させてしまう(表3)。また、<SPEAKER>に該当するfield数は、タイプ2で4790個、タイプ3で5487個存在するため、辞書部参照のオーバーヘッドが大きい(表4)。ただし、この処理時間は構造の複雑さだけに依存し、文書数には依存しないため、文書数が増えてもそれほど速度は落ちない(図7)。

タイプ1では、<SPEAKER>に該当するfield数は6個だけであり、Hybrid Queryへの変換処理時間以外は、ほぼHybrid Modelの高速性能が出ているものと思われる(表4)。

Overlapped Listsは、インデックスサイズこそ小さいが、文書数の増加と共にoffsetによる演算回数が増加するため、大規模なデータベースへの適用は速度的に不利であることがわかる(図7)。

実験データからは、インデックスサイズも小さく高速なタイプ1が優れた手法であると示されているが、逆にタイプ1は機能面において非力である。また、機能面で最も優れたタイプ3は、タイプ2と比較して性能差はほとんどない。

結論としては、(1)大量の文書データを処理するのならば、Overlapped Listsよりも本手法の方が速度的に有利である、(2)構造に関する複雑な照合を外部で高速に行なう枠組があるのならばタイプ1が最善である、(3)機能性を重視するならばタイプ3がよく、タイプ2にはメリットがない、等の知見が実験結果より得られた。

6 まとめ

本稿では、大量の構造化文書に対して高速に構造・内容検索を行なう手法を示した。実際のXML文書を用いた実験を行い、大規模な構造化文書データベースに対しても十分な速度性能が得られることを確認できた。また、同時に登録検索キーの爆発による性能低下といった欠点も確認された。今後の課題としては、(1)Hybrid Queryへの変換ルーチンの高速化、(2) inverted fileへの無駄なアクセスの削減による速度向上、(3)テキストインデックスの圧縮、などによる性能向上が挙げられる。また、レポート機能(該当する要素の情報など)や、異種データ(数値データなど)に対するインデクシングおよび検索方式についても検討する必要がある。

構造化文書検索に対する今後の要望として、構造と内容によるランキング検索があるが、本手法のHybrid Queryの評価の際に、拡張ブール検索手法[18]を用いることにより対応可能である。

参考文献

[1] 村田 真: XMLの背景と基礎, 人工知能学会誌, Vol.13, No.4, pp.507-514, (1998)

[2] R. Baeze-Yates, G. Navarro: Integrating Contents and Structure in Text Retrieval, *ACM SIGMOD Record*, Vol.25, No.1, pp.67-79, (1996)

[3] 吉川 正俊, 志村 壮是, 植村 俊亮: オブジェクト関係データベースを用いたXML文書の格納と検索, 情報処理学会論文誌, Vol.40, No.SIG6(TOD3), pp.115-131, (1999)

[4] 赤峯 享, 福島 俊一, 米山 千美, 清沢 治彦, 田中 俊行: 大規模テキスト並列検索エンジン RetrievalExpress(2) 構造化テキスト検索方式, 情報処理学会第55回全国大会, 第3分冊, pp.117-118, (1997)

[5] 多田 勝己, 岡本卓哉, 菅谷 奈津子, 加藤 寛次, 川下 靖司: 構造化文書対応全文検索システム Bibliotheca2 TextSearch の開発(3) — 構造指定全文検索方式 —, 情報処理学会第55回全国大会, 第3分冊, pp.111-112, (1997)

[6] W.Frakes, R. Baeza-Yates, editors: Information Retrieval: Data Structures and Algorithms, *Pretice-Hall*, Endlewood Cliffs, New Jersery 07632, (1992)

[7] R. Baeze-Yates: An Hybird Query Model for Full Text Retrieval System, *Technical Report DCC-1994-2*, Dept. of Computer Science, Univ. of Chile, (1994)

[8] C. Clarke, G. Cormack, F. Burkowski: An Algebra for Structured Text Search and a Framework for its Implementation, *The Computer Journal*, Vol.38, No.1, pp.43-56, (1995)

[9] G. Navarro, R. Baeza-Yates: Proximal Nodes: A Model to Query Document DataBases by Content and Structure, *ACM Tras. on Information System*, Vol.15, No.4, pp.400-435, (1997)

[10] I. MacLeod, A Query Language for Retrieving Information from Hierarchic Text Structures, *The Computer Journal*, Vol.34, No.3, pp.197-208, (1991)

[11] 村田 真: 構造化文書データベースのための文脈条件とパターン, 情処研報, 97-DBS-112, pp.25-32, (1997)

[12] P. Fankhauser, M. Marchiori, J. Robie: XML Query Requirements, W3C Working Draft 31 January 2000, http://www.w3.org/TR/2000/WD-xmlquery-req-20000131, (2000)

[13] 中津山 恒, 沼田 賢一: 文書データベース管理システム Xebec の検索について, 情処研報, 95-DBS-104, pp.25-32, (1995)

[14] P. Kilpeläinen and H. Mannila: Retrieval from hierarchical texts by partial patterns, *proc. ACM SIGIR'93*, pp.214-222, (1993)

[15] 松井くにお, 難波功, 井形伸之: 高速テキスト検索エンジン, 情処研報, 97-DD-7, pp.15-21, (1997)

[16] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, D. Suciu: XML-QL: A Query Language for XML, Submission to the W3C 19-August-1998, http://www.w3.org/TR/NOTE-xml-ql/, (1998)

[17] J. Robie, J. Lapp, D. Schach: XML Query Language (XQL), http://www.w3.org/TandS/QL/QL98/pp/xql.html, (1998)

[18] G. Salton, E. Fox, H. Wu: Extended Boolean information retrieval, *Comm. of ACM*, Vol.26, No.11, pp.1022-1036, (1983)