関係データベースを用いたXML情報検索システムの開発

清水敏之[†] 寺田憲正[††,☆] 吉川正俊[†]

利用者が文書に対して検索を行う際に，検索要求が検索結果文書中のどの部分と関連しているのかが分かることは有用である．たとえば，大量の学術論文の中から，ある話題に関連する章や節などの部分文書のみを拾い読みしたいという要求は高いと考えられる．近年，様々な文書がXMLで構造化されている．XML文書の構造を利用することで部分文書も対象とする検索が可能となる．そのため，XML文書に対する情報検索に関する研究がさかんになってきた．そこで我々はXML文書に対してキーワード集合による検索を行うシステムであるKikori-KSを開発した．キーワード検索は，XML文書のスキーマを知っている必要がなく，XML専用の問合せ言語に対する知識も必要ないため，多くの一般的な利用者が利用可能である．検索結果として入れ子するXML部分文書を単位として用いるため，我々は検索結果提示インタフェースが重要であると考え，今回開発したKikori-KSでは，XML文書検索のための検索結果表示インタフェースを用意した．キーワード検索に必要な情報は関係データベースに格納し，利用者が入力したキーワード集合に対して関連するXML部分文書を取得し，インタフェースを通じて利用者に提示する．我々は大量のXML部分文書を効率的に扱うために，実体化結合ビューを生成して検索の高速化を実現した．広く利用されている関係データベースを用いることで汎用性の高いシステムが構築できる．INEXテストコレクションを利用した実験では，Kikori-KSが実用的な検索速度と比較的高い適合性を持つことを確認した．

Development of an XML Information Retrieval System Using Relational Databases

Toshiyuki Shimizu,[†] Norimasa Terada[††,☆] and Masatoshi Yoshikawa[†]

Identifying meaningful document fragments is a major advantage achieved by encoding documents in XML. In scholarly articles, such document fragments include sections, subsections and paragraphs. XML information retrieval systems need to search document fragments relevant to queries from a set of XML documents. We present Kikori-KS, an effective and efficient XML information retrieval system for XML documents. Kikori-KS accepts a set of keywords as a query. This form of query is simple yet useful because users are not required to understand XML query languages or XML schema. To meet practical demands for searching relevant fragments in XML documents, we have developed a user-friendly interface for displaying search results. Kikori-KS was implemented on top of a relational XML database system developed by our group. By carefully designing the database schema, Kikori-KS handles a huge number of document fragments efficiently. Our experiments using INEX test collection show that Kikori-KS achieved an acceptable search time and with relatively high precision.

1. はじめに

近年，多くの文書がXMLで記述されている．XML文書はタグを用いて構造化された文書であり，XMLを用いることで文書中の部分文書を指定することができる．文書中の部分文書の特定はXMLを用いて文書を記述する利点の1つである．たとえば，論文をXMLで構造化して記述することで，節や段落などの部分文書の取得が可能になる．

XML文書の増加にともない，XML情報検索システムの需要が高まっている．XML情報検索システムは問合せに対して関連するXML部分文書を取得するシステムである．利用者は文書そのものではなく，XML部分文書というより細かな部分のみを閲覧することができる．

XML情報検索システムのための問合せとしては様々なものが提案されている．たとえば，W3CのXML問合せ言語へ全文検索機能を拡張する仕様で

† 京都大学情報学研究科
Graduate School of Informatics, Kyoto University
†† 名古屋大学情報科学研究科
Graduate School of Information Science, Nagoya University
☆ 現在，TIS株式会社
Presently with TIS Inc.

ある “XQuery 1.0 and XPath 2.0 Full-Text”[1] などがある．しかし，このような問合せ言語の利用には専門的知識が必要とされるため一般利用者にとっては不向きであり，また XML 文書のスキーマがあらかじめ分からない場合には利用できない．WEB 上の文書に対する WEB 検索エンジンの検索のように，キーワード入力による利用者に対して直感的な検索を提供することは重要である．キーワード検索には，異なるスキーマを持つ XML 文書やスキーマを持たない XML 文書に対しても検索を行うことができるという利点もある．本研究では，XML 文書集合に対してキーワード集合による問合せを入力し，関連する XML 部分文書を順序付けして出力する Kikori-KS システムを開発した．

入れ子を持った大量の XML 部分文書を扱わなければならないため，XML 情報検索では従来にはなかった新たな問題が生じた．まずあげられるものとしては，利用者インタフェースの問題がある．XML 情報検索システムは，部分文書の入れ子を考慮して検索結果を利用者に提示すべきである．我々は Kikori-KS システムにおいて FetchHighlight インタフェースを開発した．FetchHighlight インタフェースでは，関連する文書を関連度順で表示し，さらにその文書中で関連する部分文書を文書順で表示する．

次にあげられる問題点として，検索速度の問題がある．一般的に XML 部分文書は XML 文書よりも非常に数が多い．我々が実験で用いた XML 文書集合では，16,819 の文書に対して 16,080,830 の部分文書が存在した．XML 情報検索システムは，このような膨大な数の XML 部分文書を効率的に扱う必要がある．Kikori-KS では，我々のグループで開発している関係データベースを用いた XML データベースである XRel[2] を基にした関係スキーマを利用し，検索に必要な情報を格納した．XRel において利用した XML データベースシステムのための表に加え，索引語の重みを保持する表を用いた．入力されたキーワード集合は自動的に SQL に変換され処理される．高速な検索を実現するために実体化結合ビューを生成し，さらに SQL を洗練して，我々の以前の手法[3] に比べて約 9 倍の高速化を実現した．

現在，INEX[4] では XML 情報検索に関して活発に研究が行われているが，その主な焦点は検索結果の適合性であり，検索速度に関する議論はほとんどない．XML 情報検索システムにおいて，文献 5) が我々の知る限り唯一検索速度に関して述べている論文である．文献 5) の主な焦点は構造とキーワードを組み合わせた問合せにおける top-k 検索の高速化である．適合性と速度を両立する実用的な XML 文書に対する検索エンジンは非常に有用であるといえる．

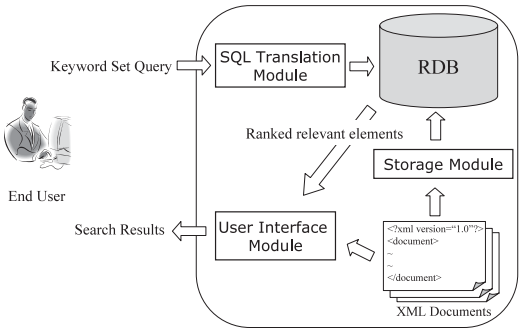


図 1 Kikori-KS システム概要
Fig. 1 An overview of Kikori-KS system.

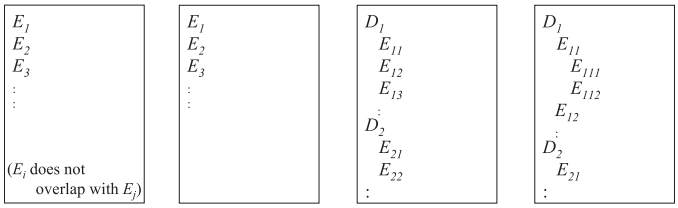
以下，まず，2 章で，Kikori-KS システムの概要を説明する．そして，3 章では，我々が実装したインタフェースに関して説明する．4 章で，XML 文書の関係データベースへの格納について述べ，5 章で，SQL への問合せ変換について述べる．6 章では，検索モデルについて述べる．7 章では，適合性と速度に関する実験結果を考察し，最後に，8 章でまとめと今後の課題に関して述べる．

2. システム概要

Kikori-KS システムの概要図を**図 1** に示す．XML 文書はあらかじめ解析され，XRel[2] を基にしたスキーマを利用してキーワード検索に必要な情報が関係データベースに格納される．我々は実体化結合ビューを生成して，検索の高速化を狙った．利用者がキーワード集合による問合せを入力すると，システムは入力されたキーワード集合を自動的に SQL に変換し，関連する部分文書を順序付きで取得する．そして，関係データベースから取得された情報を基に検索結果閲覧のための利用者インタフェースを構築する．実際に取得された部分文書の内容を閲覧する際には対応する XML 文書も参照する．

3. 出力インタフェース

XML 情報検索システムでは検索対象は XML 部分文書であり，検索結果が入れ子になっている場合がある．そのため，検索結果を提示するためのインタフェースが重要である．INEX 2005[4] では利用者の検索行動を考え，XML 情報検索のための 3 つの検索戦略を定義している．INEX では検索対象となる XML 部分文書は XML 文書中の任意の要素ノードを根とする部



(a) Focussed (b) Thorough (c) FetchBrowse (d) FetchHighlight

図 2 それぞれの検索戦略に対応するインタフェース
Fig. 2 Interfaces for each strategy.

分木であり，Kikori-KS でも同様に要素を検索単位と考えている．以下に INEX の 3 つの検索戦略を示す．

- **Focussed**
  検索結果の要素に入れ子がある場合は，先祖子孫関係にある部分文書のうち，関連度の高い 1 つの要素のみを答とし，関連度の高い順に要素を取得する．
- **Thorough**
  検索結果の入れ子を考慮せず，単純に関連度の高い順に要素を取得する，
- **FetchBrowse**
  文書という単位を重要であると考え，初めに文書単位で関連度の高い順に取得し（Fetch），次にその文書内で関連度の高い要素を関連度順に取得する（Browse）．

図 2 (a)–(c) に INEX のそれぞれの検索戦略に対応する直感的なインタフェースのイメージを示す．Focussed と Thorough では，要素が関連度の大きな順に表示される．つまり，要素 E_i の関連度が要素 E_j の関連度より大きいとき，かつそのときに限り要素 E_i は要素 E_j より上位に表示される．さらに Focussed では，すべての E_i，E_j ペアにおいて E_i と E_j は入れ子しない．FetchBrowse では，文書 D_i の関連度が文書 D_j の関連度より大きいとき，かつそのときに限り文書 D_i は文書 D_j より上位に表示され，ある文書 D_d 中の要素に関しては，要素 E_{di} の関連度が要素 E_{dj} の関連度より大きいとき，かつそのときに限り要素 E_{di} は要素 E_{dj} より上位に表示される．

INEX の 3 つの検索戦略は XML 情報検索システムの評価のために考えられており，我々はこれらの検索戦略が必ずしもインタフェースに適合するものではないと考えた．実際に検索結果を利用者に提示するためのインタフェースを意識して，我々は FetchBrowse と同様に，まず文書単位で関連の高い順に取得して提示することが重要であると考えた．しかし，FetchBrowse とは異なり，ある特定の文書の中の関連する要素を提示する際には，関連度順で提示するよりも，

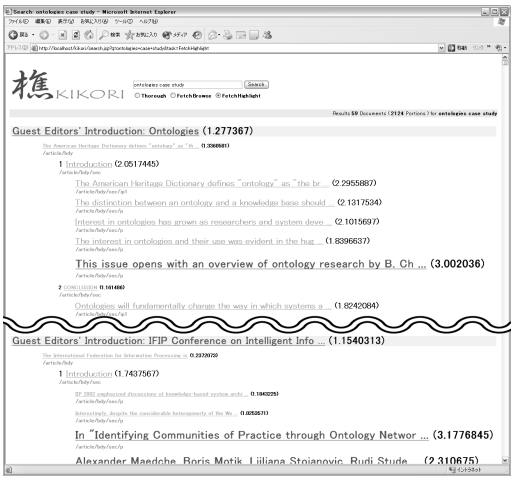


図 3 FetchHighlight インタフェース
Fig. 3 FetchHighlight retrieval interface.

要素の文書順（開始タグの出現順）で提示することが有効であると考えた．これらのことを考えて，我々は FetchHighlight インタフェースという新しいインタフェースを設計した．FetchHighlight インタフェースでは，まず XML 文書が関連度順に並べられ，その中で関連する部分文書が文書順で並べられて表示される．図 2 (d) に FetchHighlight インタフェースの直感的なイメージを示す．要素は XML 木中の深さに応じてインデントされて表示される．

さらに，関連度の高い部分文書が文書中のどの位置にあるのかを分かりやすくするために，アウトライン要素を検索結果の要素と同時に出力することにした．アウトライン要素はシステム管理者が設定した特定の構造を持つ要素であり，たとえ関連度が 0 であっても出力するものとする．たとえば，論文 XML 文書におけるアウトライン要素としては節，小節，小々節に対応する要素を用いることが妥当であると思われる．

実際の FetchHighlight インタフェースを**図 3** に示す．これは INEX 2005 の問合せの 1 つである “ontologies case study” というキーワード集合で検索した例である．関連度の高い部分文書に対応するアン

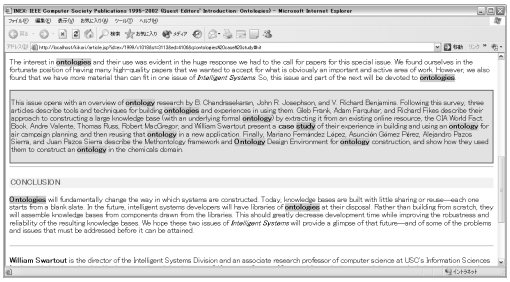


図 4 部分文書表示
Fig. 4 Highlighted document fragment.

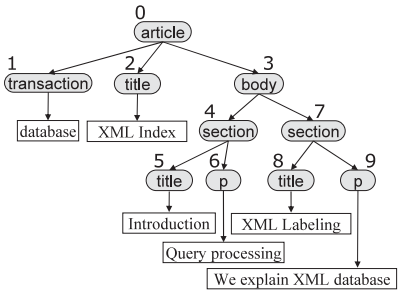


図 5 木構造で表した XML 文書の例
Fig. 5 An example of XML tree.

カーテキストほど大きな文字で表示することで強調表示している．FetchHighlight インタフェースで検索結果を閲覧することで文書全体としては関連度が低くても，部分文書では関連度が高いものも容易に識別できる（図 3 の下部）．また，部分文書が文書順で並んでいるため，利用者は関連度の高い部分文書が多く存在する部分を識別することができる．アンカーテキストをクリックすることで，その部分文書を含む文書を取得し，対応する部分文書を強調して表示する（**図 4**）．

Kikori-KS では結果を提示するためのインタフェースとしてこの FetchHighlight インタフェースのほかに Thorough での要素の取得に対応した Thorough インタフェースと FetchBrowse での要素の取得に対応した FetchBrowse インタフェースも用意した．Focussed は検索結果の要素中から入れ子が取り除かれているため，冗長な内容の閲覧を防止できるが，取得された要素以外の要素の関連度は分からず，XML 情報検索システムの利点を損なう可能性がある[6] ため実装しなかった．たとえば，ある関連度の高い要素 E_1 が，それよりも関連度の低い要素 E_2 の子要素になっている場合，Focussed では E_1 のみを出力し，E_2 は出力しないため，E_2 が E_1 以外の部分にも問合せに関連する内容を含むような場合でも E_2 に関する情報は失われてしまうことになる．Focussed に対応する単純なインタフェースは Thorough インタフェースと同等であると思われるが，FetchHighlight インタフェースを用いることで，利用者は Focussed で取得すべき要素を閲覧することができる．

4. 関係データベースへの格納

本章では，XML 文書と文書に含まれる索引語の重みを関係データベースにどのように格納したかに関して説明する．索引語の重みに関しては 6 章で述べる．実際に XML 文書を格納する際には，データ的な要素を排除し，文書的な要素のみを検索対象として格納する我々のグループが文献 3)，7) で提案した手法を用いた．

我々はまず XRel[2] を基にした関係スキーマを考えた．そして，そのスキーマを基に実体化結合ビューを生成して問合せ処理の高速化を狙った．以下にこれらのスキーマを**図 5** の XML 文書を利用した例を交えて説明する．なお，これらのスキーマを用いた問合せ処理時間に関する実験結果は 7.2 節で報告する．

4.1 関係スキーマ

XRel は XML 文書のスキーマに依存しない関係スキーマを利用する構造写像アプローチをとっている．XML データベースのための表に加えて索引語の重みを保持する表を用意した．また，FetchHighlight インタフェースではアウトライン要素が必要であるため，すべての要素の中からアウトライン要素を抽出した表を用意した．さらに，インタフェースの構築のために必要な付加的な情報を追加して以下のスキーマを用いることにした．

- `Document (docID, file)`
- `Element (docID, elemID, pathID, st, ed, label)`
- `Path (pathID, pathexp)`
- `Term (term, docID, elemID, tfipf)`
- `Outline (docID, elemID, pathID, st, ed, label)`

`docID`，`elemID`，`pathID` はそれぞれ文書，要素，経路を識別する識別子である．`file` には実際の XML 文書のファイル名を格納するが，`Document` 表中に XML 文書の内容全体を文字列として格納することも考えられる．`docID` と `elemID` の組合せを用いることで，XML 文書集合中でその要素を一意に識別できる．`st`，`ed` は各要素の開始，終了バイト位置であり，部分文書の切り出しと FetchHighlight インタフェースでの結果要素のインデント表示のために使われる．`label` の値は結果を表示する際にその要素を示すための文字

Document

docID	file
0	doc1.xml

Path

pathID	pathexp
0	#/article
1	#/article#/transaction
2	#/article#/title
:	:

Element

docID	elemID	pathID	st	ed	label
0	0	0	1	236	XML Index
0	1	1	10	44	database
0	2	2	45	68	XML Index
:	:	:	:	:	:

Term

term	docID	elemID	tfipf
database	0	0	0.3
database	0	1	0.1
:	:	:	:
xml	0	0	0.3
xml	0	2	0.4
:	:	:	:

Outline

docID	elemID	pathID	st	ed	label
0	4	4	75	143	Introduction
0	7	4	144	219	XML Labeling

図 6 関係データベースへの格納例
Fig. 6 A storage example.

TermExt

term	docID	elemID	file	st	ed	label	pathexp	tfipf
database	0	0	doc1.xml	1	236	XML Index	#/article	0.3
database	0	1	doc1.xml	10	44	database	#/article#/transaction	0.1
:	:	:	:	:	:	:	:	:
xml	0	0	doc1.xml	1	236	XML Index	#/article	0.3
xml	0	2	doc1.xml	45	68	XML Index	#/article#/title	0.4
:	:	:	:	:	:	:	:	:

OutlineExt

docID	elemID	file	st	ed	label	pathexp
0	4	doc1.xml	75	143	Introduction	#/article#/body#/section
0	7	doc1.xml	144	219	XML Labeling	#/article#/body#/section

図 7 実体化結合ビューの例
Fig. 7 A storage example of materialized join view.

列であり，インタフェース中でアンカーテキストとして用いられる．たとえば，XML で構造化された論文において節に対応する要素の `label` の値には節のタイトルを用いることができる．`pathexp` は根ノードからの経路表現であり，経路情報も考慮する際に使用される．要素中の索引語の重みは `Term` 表の `tfipf` に格納される．格納する索引語は小文字で統一されて扱われ，ステミング処理を施し，ストップワードは取り除いた．

管理者はあらかじめ経路を指定するなどの方法でアウトライン要素を定義しておく．FetchHighlight インタフェースではアウトライン要素の取得が必要となる．アウトライン要素は検索時に動的に取得することも可能であるが，あらかじめ `Element` 表中からアウトライン要素を抽出し，`Outline` 表を生成しておくことで高速な検索が可能になる．`Outline` 表は FetchHighlight インタフェースのみに利用される．

図 5 の XML 文書に関し，この XML 文書のファイル名を doc1.xml とし，`elemID` は図 5 中で要素に付与された番号であるとし，アウトライン要素は/article/body/section の経路上の要素であるとすると，この関係スキーマを利用した関係データベースへの格納の一部は**図 6** のようになる☆．

Kikori-KS ではキーワード検索のみを考えているが，この関係スキーマを用いることでキーワード検索だけでなくキーワードと構造を組み合わせた検索なども可能となると思われる．

4.2 実体化結合ビュー

XML 情報検索システムでは膨大な数の XML 部分文書を扱わなければならず，検索速度の向上も重要な課題の 1 つである．そのため，我々は 4.1 節の関係スキーマを基に実体化結合ビューを生成することを考えた．格納容量よりも検索速度を重視し，冗長に情報を格納することで高速な検索を目指す．

Kikori-KS では，入力されたキーワードを条件とし，

☆ なお，図 6，図 7 中の `tfipf` の値は簡単化のため 6 章の式で求まる値とは異なるものを用いている．

Term 表の term にマッチする行を取得し，Element 表と docID，elemID ペアを用いて結合して結果を取得することになる．また，経路情報も補助情報として出力し，得られた要素がアウトライン要素かどうかを判別するために Path 表との pathID を用いた結合も必要になってくる．これらの結合をあらかじめ行った実体化結合ビューを生成することで問合せ処理の高速化を狙う．生成した実体化結合ビューのスキーマは以下のようになる．

- TermExt (term, docID, elemID, file, st, ed, label, pathexp, tfipf)
- OutlineExt (docID, elemID, file, st, ed, label, pathexp)

TermExt 表は，Term 表，Element 表，Path 表，Document 表を対応する識別子で結合した実体化結合ビューである．TermExt 表の行数は Term 表の行数に等しい．同様に OutlineExt 表は，Outline 表，Path 表，Document 表を対応する識別子で結合した実体化結合ビューである．

図 5 の XML 文書を格納した図 6 のデータからこの実体化結合ビューを生成すると**図 7** のようになる．

5. 問合せ変換

システムはキーワード集合が入力されると適切な SQL 文に変換し，要素の関連度を計算する．Kikori-KS システムでは自動的にキーワード集合をそれぞれの検索戦略に対応する SQL 文に変換する．

キーワード集合中のそれぞれのキーワードに対応する索引語の情報を取得し，6 章のランキングモデルに基づいて関連度の計算を行う．FetchHighlight と FetchBrowse では XML 文書単位での関連度が必要であるが，XML 文書の関連度はその XML 文書の根ノードの関連度と等しいと考え，他の要素と同様に関連度を計算し，順序付けに用いる．さらに FetchHighlight インタフェースではアウトライン要素を同時に出力するため，Outline 表（実体化結合ビューが利用可能な場合は OutlineExt 表）から必要なアウトライン要素を取り出して検索結果要素との和集合を得る．

ここで，“ontologies case study” というキーワード集合が入力された場合において，FetchHighlight を用いたときの実体化結合ビューが利用できない場合の SQL と利用可能な場合の SQL の具体例をそれぞれ**図 8** と**図 9** に示す．入力されたキーワードはステミングされて処理される．

図 8 の SQL では，3 行目から 9 行目にかけて入力されたキーワード集合に関連する要素を関連度を含めて取得し，11 行目から 20 行目にかけてアウトライン要素を取得している．関連度は 6 章のモデルに基づき，入力キーワード集合に対応する索引語集合のうち，その要素が含んでいる索引語の重みの和となる．また，アウトライン要素に関しては入力されたキーワード集合を含む文書におけるアウトライン要素のみを取得する．入力キーワードに関連する要素やアウトライン要素を取得する際には同時に Path 表との結合を行う．入力キーワードに関連する要素とアウトライン要素の和集合を 10 行目でとっている．さらに，21 行目から 25 行目にかけて入力されたキーワード集合を含む文書の関連度を取得している．XML 文書の関連度はその XML 文書の根ノードの関連度と等しいと考えるが，根ノードの elemID はつねに 0 であることを利用している．2，26，27 行目で Document 表と取得された要素の情報と取得された文書の情報を結合する．最後に

```
 1:SELECT d.file, r1.st, r1.ed, r1.score,
   r1.pathexp, r1.label
 2:FROM Document d,
 3:  ((SELECT t.docID, t.elemID, e.st, e.ed,
     e.label, p.pathexp, SUM(t.tfipf) AS score
 4:  FROM Term t, Element e, Path p
 5:  WHERE t.docID = e.docID
 6:  AND t.elemID = e.elemID
 7:  AND e.pathID = p.pathID
 8:  AND t.token IN
     ('ontologi', 'case', 'studi')
 9:  GROUP BY t.docID, t.elemID, e.st, e.ed,
     e.label, p.pathexp)
10:  UNION
11:  (SELECT o.docID, o.elemID, o.st, o.ed,
     o.label, p.pathexp, 0 AS score
12:  FROM Outline o, Path p
13:  WHERE o.pathID = p.pathID
14:  AND o.docID IN
15:    (SELECT t.docID
16:    FROM Term t
17:    WHERE t.elemID = 0
18:    AND t.token IN
       ('ontologi', 'case', 'studi')
19:    GROUP BY t.docID)
20:  GROUP BY o.docID, o.elemID, o.st, o.ed,
     o.label, p.pathexp)) r1,
21:  (SELECT t.docID, SUM(t.tfipf) AS score
22:  FROM Term t
23:  WHERE t.elemID = 0
24:  AND t.token in
     ('ontologi', 'case', 'studi')
25:  GROUP BY t.docID) r2
26:WHERE d.docID = r1.docID
27:AND r1.docID = r2.docID
28:ORDER BY r2.score DESC, r2.docID ASC,
   r1.st ASC
```

図 8 SQL の例
Fig. 8 An example of SQL.

```
 1:SELECT r1.file, r1.st, r1.ed, r1.score,
   r1.pathexp, r1.label
 2:FROM
 3:  ((SELECT t.docID, t.elemID, t.file,
     t.st, t.ed, t.label, t.pathexp,
     SUM(t.tfipf) AS score
 4:  FROM TermExt t
 5:  WHERE t.token IN
     ('ontologi', 'case', 'studi')
 6:  GROUP BY t.docID, t.elemID, t.file,
     t.st, t.ed, t.label, t.pathexp)
 7:  UNION
 8:  (SELECT o.docID, o.elemID, o.file,
     o.st, o.ed, o.label, o.pathexp, 0 AS score
 9:  FROM OutlineExt o
10:  WHERE o.docID IN
11:    (SELECT t.docID
12:    FROM TermExt t
13:    WHERE t.elemID = 0
14:    AND t.token IN
       ('ontologi', 'case', 'studi')
15:    GROUP BY t.docID))) r1,
16:  (SELECT t.docID, SUM(t.tfipf) AS score
17:  FROM TermExt t
18:  WHERE t.elemID = 0
19:  AND t.token IN
     ('ontologi', 'case', 'studi')
20:  GROUP BY t.docID) r2
21:WHERE r1.docID = r2.docID
22:ORDER BY r2.score DESC, r2.docID ASC,
   r1.st ASC
```

図 9 実体化結合ビューを利用した SQL の例
Fig. 9 An example of SQL using materialized join view.

28 行目で文書の関連度と要素の出現位置に基づいて順序付けし，1 行目で結果を取得する．

図 9 の実体化結合ビューが利用できる場合の SQL も，図 8 の SQL と同じ流れであるが，識別子による結合があらかじめ行われているため，その結合に対応する操作が必要なくなる．

また，Kikori-KS では単純なキーワード集合による検索に加え，検索結果の部分文書に必ず含まれていてほしいキーワードや，含まれていてほしくないキーワードによる検索結果の絞り込みができる．これらはそれぞれ問合せを入力する際にキーワードの前に “+” と “−” を付けることで指定する．

Kikori-KS は利用者が選択した検索戦略に対応した SQL の処理結果を適切に整えてインタフェースを構築する．

6. ランキングモデル

本章では，我々が Kikori-KS で用いたランキングモデルと索引語重み付け手法に関して説明する．索引語の重み情報はあらかじめ計算しておくことで，問合せ時に高速な検索を実現する．

XML 情報検索システムの場合でも，通常の情報検索システムと同様に検索結果は問合せに対する関連度によって順序付けされていることが望まれる．関連度を算出するためのモデルとして，我々は広く利用されているベクトル空間モデルを用いることにした．ベクトル空間モデルでは，索引語の重みを要素として，問合せと文書（XML 文書の場合は要素）をベクトル化し，その類似度を求めることで関連度を算出する．類似度としては問合せベクトル Q と要素ベクトル E の内積を用いた．

$$Score(Q,E) = \sum_{t \in Q,E} weight(t,Q) * weight(t,E) \quad (1)$$

問合せベクトル中の索引語の重み（$weight(t,Q)$）としては単純に索引語 t が問合せ中に出現するかどうかの 2 値（出現したら 1，出現しなかったら 0）を用いるが，要素中のある索引語 t の重みである $weight(t,E)$ は t の網羅性と特定性を反映した値を用いるべきである．非構造化文書においては，*tf* (term frequency) と *idf* (inverse document frequency) を組み合わせて利用する *tf-idf* を用いることが有用であるが，構造化されている XML 文書に対しては，その構造を利用したより有効な指標が提案されている[8)〜10)]．文献 9) では *Category* という概念を導入して XML 文書中の要素をカテゴリ分けし，それぞれのカテゴリ中での特定性を用いることを提案している．実際には根ノードからの経路ごとにカテゴリ化する実装を行っている．我々はこれを *ipf* (inverse path frequency) と呼び，Kikori-KS で用いることにした．さらに，我々は文献 11) で提案されている重み付けの式を基にして，同じ経路を持つ要素中の平均要素長を用いて正規化要素長（*nel*）を計算し，*ipf* と組み合わせ，さらに小さな要素に対する対処を施して，$weight(t,E)$ として以下の式を用いた．

$$weight(t,E) = \frac{ntf * ipf}{nel * penalty} \quad (2)$$

$$ntf = 1 + ln(1 + ln(tf)) \quad (3)$$

$$nel = \left((1-s) + s * \frac{el}{avgel_p}\right) * \left(1 + ln(avgel_p)\right) \quad (4)$$

$$ipf = ln\frac{N_p + 1}{ef_p} \quad (5)$$

$$penalty = 1 + ln\left(max\left(\frac{el_t}{el}, 1\right)\right) \quad (6)$$

ntf は正規化された索引語出現回数（*tf*）であり，*nel* は要素長を反映した正規化指標である．また，*ipf* は経路中の索引語の特定性である．ここで，*el* は要素中

の索引語の数，p は E の根ノードからの経路，$avgel_p$ はある経路 p に存在する要素の el の平均値，N_p は p に存在する要素の数，ef_p は経路 p に存在する要素の内 t が出現するものの数である．また，s は定数のパラメータであり，一般的に 0.2 を用いる[11]．

penalty はある程度小さい要素に対して重みを下げる指標である．XML 文書に対するキーワード検索で検索対象となる XML 部分文書は非常に小さなものから大きなものまでその大きさは様々である．小さな要素が検索結果として返されても，そこから得られる情報は少ない．従来の重み付け手法では，それぞれの文書が結果として妥当なある程度の大きさを持つことが前提となっているが，XML 部分文書のような非常に小さい文書が混在する場合には小さな部分文書のスコアを適切に下げることが有効であると予備実験で分かったため，*penalty* を導入した．現在の実装では，閾値 el_t には 60 を設定している．

7. 実　　験

実験には INEX プロジェクト[4] によって提供される XML 文書集合を使用した．INEX プロジェクトは XML 文書を対象としたテストコレクションを作成する国際プロジェクトであり，IEEE によって出版される論文を XML 文書化している．INEX プロジェクトでは XML 文書集合とともに，問合せ集合と問合せ結果の正解集合を提供している．我々が用いた INEX-1.9 テストコレクションは約 700 MB である．INEX では CO（Content Only）問合せと CAS（Content And Structure）問合せを考えており，この実験では，INEX 2005 の 40 個の CO 問合せを用いた．CO 問合せは基本的にキーワード集合を用いた問合せであるが，フレーズを用いた問合せも含んでいる．我々はフレーズは個々のキーワードに分解して扱った．実験環境は CPU：Intel Xeon 3.80 GHz（2 CPU），メモリ：8.0 GB，OS：Miracle Linux 3.0，RDBMS：Oracle 10g Release1 である．なお，RDBMS には 2.0 GB のメモリを割り当てた．

7.1 適 合 性

我々は Kikori-KS における Thorough と FetchBrowse の適合性を調査した．Kikori-KS のインタフェースの焦点は FetchHighlight インタフェースであるが，FetchHighlight の適合性は直接は計測できない．しかし，FetchHighlight と FetchBrowse はインタフェースでの表示が異なるだけであるため，FetchHighlight の適合性は FetchBrowse の適合性と同等であると考えられる．

XML 情報検索システムにおける検索結果の評価は現在研究が進行中の分野であるが，INEX 2005 では eXtended Cumulated Gain（XCG）を基にした評価尺度を用いている．理想的なシステムが出力する結果に対して，構築したシステムがどの程度良い結果を出力したかを考え，システム指向の評価尺度である ep/gr（effort-precision/gain-recall）とユーザ指向の評価尺度である nxCG（normalized extended Cumulated Gain）を用いている[12]．

我々は従来の評価尺度である精度再現率グラフに加えて，ep/gr グラフと nxCG グラフを得た．さらに，精度再現率に関して平均精度を，ep/gr に関して MAep（Mean Average effort-precision）の値を，nxCG に関して nxCG@10，nxCG@25，nxCG@50 のそれぞれの値を調査した．MAep は effort-precision の平均値であり，nxCG@k は k 位における理想的なシステムが取得した累積利得（Cumulated Gain）の値に対する構築したシステムが取得した累積利得の値の相対値である．これらの値は，INEX 2005 における公式の評価指標である[12]．評価には INEX で開発されている評価用のソフトウェアである EvalJ [13] を利用した．

INEX 2005 における XCG を基にした評価では，FetchBrowse の評価は文書の順序の評価である FetchBrowse-Article と文書中の要素の順序の評価である FetchBrowse-Element に分かれている．また，FetchBrowse ではそれぞれの文書中で関連する要素数が異なり，何位の nxCG 値を用いるのが適切か不明であるため，ep/gr に関してのみ評価を行う[12]．

INEX では多数の参加者が各々のシステムの適合性を競い合っている．図 10〜図 15 では Kikori-KS の結果とともに，他の INEX 参加者の結果も表示しており，それぞれ，**図 10** では Thorough の精度再現率グラフを，**図 11** では Thorough の ep/gr グラフを，**図 12** では Thorough の nxCG グラフを，**図 13** では FetchBrowse の精度再現率グラフを，**図 14** では FetchBrowse-Article の ep/gr グラフを，**図 15** では FetchBrowse-Element の ep/gr グラフを示している．また**表 1** に INEX 公式の評価指標である MAep，nxCG@10，nxCG@25，nxCG@50 における Kikori-KS の値とそれらの値が INEX 参加チーム中で何位に相当するかを示す☆．また，Thorough の平均精度は 0.0706（22 チーム中 6 位）であり，FetchBrowse の平均精度は 0.0794（11 チーム中 1 位）であった．

☆ 括弧内に (Kikori-KS の順位/参加チーム数) の形式で示している．

表 1 INEX 公式の評価指標の値と INEX 参加チーム中の順位
Table 1 The values of INEX evaluation indicators and the ranks in INEX participants.

	MAep	nxCG@10	nxCG@25	nxCG@50
Thorough	0.0617 (9/22)	0.2334 (10/22)	0.2493 (5/22)	0.2356 (4/22)
FetchBrowse-Article	0.1864 (8/16)	NA	NA	NA
FetchBrowse-Element	0.1304 (1/15)	NA	NA	NA

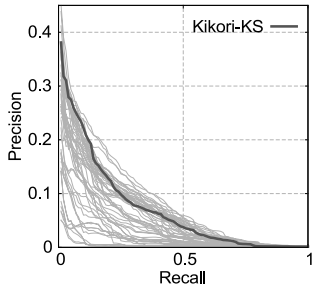


図 10 Thorough の精度再現率
Fig. 10 Precision/Recall of Thorough.

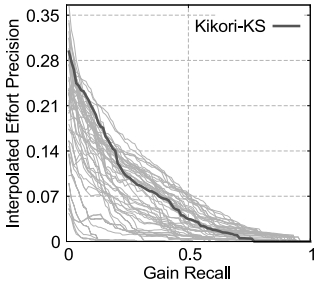


図 11 Thorough の ep/gr
Fig. 11 ep/gr of Thorough.

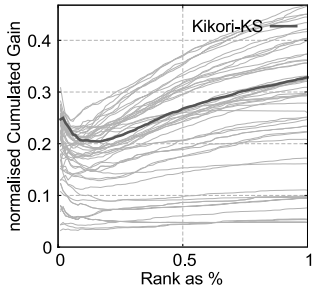


図 12 Thorough の nxCG
Fig. 12 nxCG of Thorough.

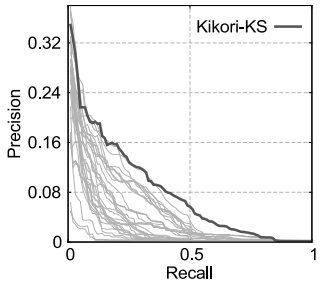


図 13 FetchBrowse の精度再現率
Fig. 13 Precision/Recall of FetchBrowse.

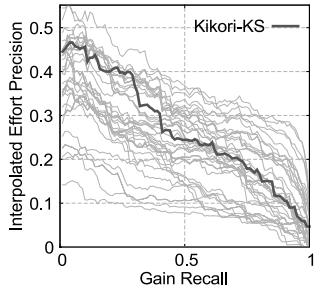


図 14 FetchBrowse-Article の ep/gr
Fig. 14 ep/gr of FetchBrowse-Article.

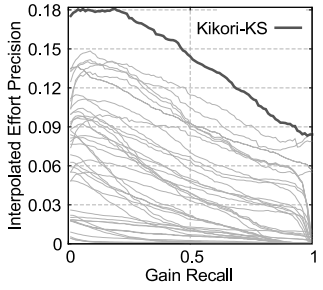


図 15 FetchBrowse-Element の ep/gr
Fig. 15 ep/gr of FetchBrowse-Element.

Kikori-KS は特に FetchBrowse において，他の参加者と比べて比較的高い適合性を実現しており，FetchBrowse-Element では INEX 参加チーム中トップの MAep を実現している．現在の Kikori-KS では FetchBrowse において上位に順序付けされた XML 文書中では少しでも関連する要素はすべて取得しており，取得した XML 文書ごとの要素順の評価を平均して求める FetchBrowse-Element では特に優位な結果が得られたと考えられる．FetchBrowse における文書と要素の取得方針やその評価手法に関しては今後の課題の 1 つである．また Thorough においても他のチームに比べて遜色ない適合性を実現している．Thorough は最も単純な検索戦略であるため，Thorough における適合性評価結果はシステム評価の基盤になると考えら

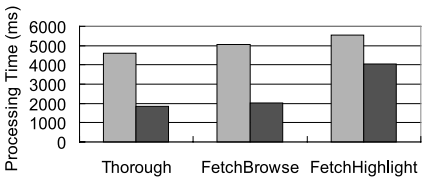


図 16 すべての CO 問合せにおける平均検索時間
Fig. 16 Processing time of all queries.

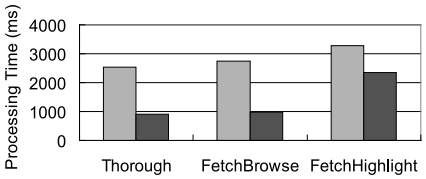


図 17 3 語以下の問合せにおける平均検索時間
Fig. 17 Processing time of queries under three words.

れる．Thorough の適合性向上のためには索引語の重み付け式の改善などが考えられる．

7.2 速　　度

検索速度に関する実験では，Thorough，FetchBrowse，FetchHighlight のそれぞれにおいて，実体化結合ビューがない場合とある場合で問合せ処理時間を調べた．40 個の CO 問合せを SQL に変換し，INEX で定められている上位 1,500 件の検索結果を取得するのにかかる時間を計測して，40 個の問合せの平均値を求めた．その結果を**図 16** に示す．それぞれ左側が実体化結合ビューがない場合で，右側が実体化結合ビューがある場合の処理時間である．FetchBrowse や FetchHighlight では上位 1,500 件の部分文書を取得する際に FetchBrowse で平均で約 73 件，FetchHighlight で平均で約 41 件の文書を取得した．FetchHighlight では結果にアウトライン要素も含んでいるため FetchBrowse と文書取得件数が異なっている．

実験に用いた 40 個の CO 問合せには多くのキーワードを用いているものもあるが，キーワードによる検索は一般的に 3 語以下によるものが多いと思われる．40 個の CO 問合せのうち，3 語以下のキーワードを用いている 17 個の問合せに関して，同様に検索時間の平均値を求めた．その結果を**図 17** に示す．

実体化結合ビューを用いることで検索の高速化を実現している．実体化結合ビューを用いると，Thorough で約 1.8 秒，FetchBrowse では約 2.0 秒，FetchHighlight では約 4.0 秒で検索が可能であった．また，3 語以下の問合せにおいては，それぞれ約 0.9 秒，約 1.0 秒，約 2.4 秒程度であった．我々はこの時間を実用的な検索時間であると考えている．

文献 5) では XML 情報検索システムにおいて top-k を高速に取得するための手法を提案している．我々の知る限り，XML 情報検索システムにおいて検索時間に関して述べている研究は文献 5) のみである．文献 5) では，XML 文書中の構造とキーワードを組み合わせた検索である CAS 検索が焦点であり，単純に要素を順序付けする Thorough を対象としている．一般に CAS 検索では構造による絞り込みが可能なため，CO 検索よりも高速に処理できると思われる．実験環境の違いや，キーワード検索と CAS 検索という対象とする問合せの違い，また包括的な検索と top-k 検索という検索戦略の違いなどから単純に Kikori-KS と文献 5) における検索時間を比較することはできない．図 16 の処理時間は上位 1,500 件を求めるのにかかった時間であるが，我々の手法では取得件数を変化させても，処理時間はほとんど変化がなかった．また，Kikori-KS では FetchBrowse でも Thorough と同程度の時間で処理できている．

FetchHighlight ではアウトライン要素の取得と検索結果への統合が必要であり，Thorough や FetchBrowse と比較して処理時間が長くなっている．もしアウトライン要素を必要としないのであれば，FetchBrowse と同程度の処理時間となる．

8. おわりに

本論文では，我々が開発した Kikori-KS システムに関して説明した．Kikori-KS はキーワード集合を問合せとする XML 情報検索システムである．我々は，特にインタフェースに着目し，FetchHighlight インタフェースを開発した．XML 文書と索引語の重み情報は関係データベースに格納され，検索時の処理高速化のために実体化結合ビューを生成した．そして，INEX テストコレクションを用いて実験を行い，Kikori-KS が実用的な検索時間と，比較的高い適合性を実現していることを確認した．

今後の課題としては，フレーズ検索のための重み付け手法とデータの格納手法の開発や，構造とキーワードを組み合わせた問合せへの対応，top-k 検索の実現などがある．

参 考 文 献

1) W3C: XQuery 1.0 and XPath 2.0 Full-Text (2006). http://www.w3.org/TR/xquery-full-text/
2) Yoshikawa, M., Amagasa, T., Shimura, T. and Uemura, S.: XRel: a path-based approach to

storage and retrieval of XML documents using relational databases, *ACM Trans. Internet Technology*, Vol.1, No.1, pp.110–141 (2001).
3) Fujimoto, K., Shimizu, T., Terada, N., Hatano, K., Suzuki, Y., Amagasa, T., Kinutani, H. and Yoshikawa, M.: Implementation of a high-speed and high-precision XML information retrieval system on relational databases, *Advances in XML Information Retrieval and Evaluation*, Lecture Notes in Computer Science, Vol.3977, pp.254–267, Springer-Verlag (2006).
4) INEX: INitiative for the Evaluation of XML Retrieval (2005). http://inex.is.informatik.uni-duisburg.de/2005/
5) Theobald, M., Schenkel, R. and Weikum, G.: An efficient and versatile query engine for TopX search, *Proc. 31st International Conference on Very Large Data Bases*, Trondheim, Norway, pp.625–636 (2005).
6) Clarke, C.L.A.: Controlling overlap in content-oriented XML retrieval, *Proc. 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Salvador, Brazil, pp.314–321 (2005).
7) Hatano, K., Kinutani, H., Amagasa, T., Mori, Y., Yoshikawa, M. and Uemura, S.: Analyzing the properties of XML fragments decomposed from the INEX document collection, *Advances in XML Information Retrieval*, Lecture Notes in Computer Science, Vol.3493, pp.168–182, Springer-Verlag (2005).
8) Cohen, S., Mamou, J., Kanza, Y. and Sagiv, Y.: XSEarch: A semantic search engine for XML, *Proc. 29th International Conference on Very Large Data Bases*, Berlin, Germany, pp.45–56 (2003).
9) Grabs, T. and Schek, H.-J.: ETH Zürich at INEX: Flexible information retrieval from XML with PowerDB-XML, *Proc. 1st Workshop of the INitiative for the Evaluation of XML Retrieval*, Schloss Dagstuhl, Germany, pp.141–148 (2002).
10) Amer-Yahia, S., Curtmola, E. and Deutsch, A.: Flexible and efficient XML search with complex full-text predicates, *Proc. 2006 ACM SIGMOD International Conference on Management of Data*, Chicago, USA, pp.575–586 (2006).
11) Liu, F., Yu, C.T., Meng, W. and Chowdhury, A.: Effective keyword search in relational databases, *Proc. 2006 ACM SIGMOD International Conference on Management of Data*, Chicago, USA, pp.563–574 (2006).
12) Kazai, G. and Lalmas, M.: INEX 2005 evaluation measures, *Advances in XML Information Retrieval and Evaluation*, Lecture Notes in Computer Science, Vol.3977, pp.16–29, Springer-Verlag (2006).
13) EvalJ: INEX Evaluation Package (2006). http://evalj.sourceforge.net/

(平成 18 年 12 月 20 日受付)
(平成 19 年 1 月 31 日採録)

(担当編集委員　池田 哲夫)

清水　敏之（学生会員）

2003 年名古屋大学工学部電気電子・情報工学科卒業．2005 年同大学大学院情報科学研究科博士前期課程修了．修士（情報科学）．現在，京都大学情報学研究科博士後期課程に在籍．XML データベース，特に XML 索引付け，XML 全文検索，XML 情報検索に興味を持つ．ACM，日本データベース学会各学生会員．

寺田　憲正

2005 年名古屋大学工学部電気電子・情報工学科卒業．2007 年同大学大学院情報科学研究科博士前期課程修了．修士（情報科学）．現在，TIS 株式会社勤務．

吉川　正俊（正会員）

1980 年京都大学工学部情報工学科卒業．1985 年同大学大学院工学研究科博士後期課程修了．工学博士．同年京都産業大学計算機科学研究所講師．同大学工学部助教授，奈良先端科学技術大学院大学情報科学研究科助教授，名古屋大学情報連携基盤センター教授，同大学情報科学研究科教授を経て，2006 年より京都大学大学院情報学研究科教授，現在に至る．1989〜1990 年南カリフォルニア大学客員研究員，1996〜1997 年ウォータルー大学客員准教授．XML データベース，多次元空間索引等の研究に従事．電子情報通信学会，ACM，IEEE Computer Society 各会員．日本データベース学会理事．