テキストマイニングシステムTAKMIのRDBを利用した実装方法

猪口　明博†　　武田　浩一†

† 日本アイ・ビー・エム株式会社 東京基礎研究所　〒 242–8502 神奈川県大和市下鶴間 1623–14
E-mail: †{inokuchi,takedasu}@jp.ibm.com

あらまし　テキストマイニングはテキスト情報から半自動的に傾向やパターンを発見するための技術である．弊社はコールセンタのテキストログの解析のためTAKMIを，ライフサイエンス/医療文献から知識発見を促進するためにMedTAKMIを開発した．ユーザがTAKMIを対話的に操作する過程で，TAKMIは各集計/マイニング機能を高速に動作させることができる．TAKMIの各機能は高速に動作するが，独自のデータ形式，独自の集計エンジンを用いているので，他のシステムとの統合や新たな機能の開発が容易ではない．これら問題を解決するために，本稿では，アノテーションされたキーワードを関係データベースにストアし，TAKMIの集計/マイニング機能をSQLで高速に実行する手法を提案する．従来手法の数千行以上のC++のプログラムが，提案手法では数行程度のSQLによって実現され，比較実験で従来手法を同等以上であることを示す．

キーワード　情報検索, データマイニング, テキストDB

Aggregation by a Relational Database for a Text Mining System, TAKMI

Akihiro INOKUCHI† and Kohichi TAKEDA†

† Tokyo Research Laboratory, IBM Japan,　1623-14, Shimotsuruma, Yamato, Kanagawa, 242-8502
E-mail: †{inokuchi,takedasu}@jp.ibm.com

Abstract　Text mining is a technology that makes it possible to discover patterns and trends semi-automatically from huge collections of unstructured text. We developed TAKMI and MedTAKMI to facilitate knowledge discovery from the very large text databases characteristic of customer relationship management and life science/healthcare applications. TAKMI can interactively mine a huge document collection and provide fast computations for each function. However, since TAKMI uses a proprietary index as a modified DTM and an proprietary aggregate engine, and is implemented in C++, it is not easy to expand it to develop other functions and to integrate it with other systems. In this paper, we propose an SQL-based method for storing annotated words in a relational database and computing each function of TAKMI by SQL. Although the original TAKMI was implemented in a few thousands of lines of C++ code, the proposed method is implemented in a few lines of SQL and is comparable with the original TAKMI.

Key words　Information Retrieval, Data Mining, Text DB

1. はじめに

近年，グループウェアの普及，インターネットの普及などにより情報の取得，集積が容易になり，テキストで記述された文書データの多くが蓄積されるようになった．大量に蓄積された文書データから知識発見を行うために技術としてテキストマイニングが注目されている[6].

大量の文書から欲しい情報が簡単に見つかり，そこから知見を得ることができれば非常に有益であるが，目的の文書を探し出すことがそもそも容易ではない．通常，文書データはその有益性を判断する担当者が目を通して，キーワードなどを付与し，これらを分類することで検索可能なデータベースの構築を行っている．また，キーワードを付与せずに，全文検索を行って，目的の文書を発見するという手法もあるが，表記の揺れがあった場合に，発見できないなど問題も多い．

企業などでは，顧客から問い合わせなどをコールセンタで対応し，そのログをテキストとして蓄積している．コールセンタのコスト削減や将来の機能拡張，商品の不具合の早期発見などに役立てようと蓄積されたコールセンタログを分析して，新たな規則性や知見の発見を目的として，テキストマイニングが活用されている．テキストデータから単純にキーワードを取り出してデータマイニングの手法を用いるよりも，自然言語処理技

術を用いることで，より効率的にテキストデータを分析できる技術が報告されている [4]．コールセンタ向けテキストマイニングシステム IBM TAKMI (Text Analysis and Knowledge MIning) では，自然言語処理技術を用いて，文書中から「何がどうした」という概念を取り出すことで，コールセンタにおける大量の文書を解析し，顧客からの問い合わせ内容分析の迅速化に貢献した．

一方，生物医学の分野では，MEDLINE と呼ばれる生物医学文献データベースが広く使われている．MEDLINE とは，米国立医学図書館 [1](NLM(注1)) の米国立バイオテクノロジー情報センタ [2](NCBI(注2)) により管理されている巨大な生物医学文献アーカイブであり，1960 年代からの生物医学文献が登録されている．タイトル，アブストラクトなどの非構造化データと著者名，著者所属，公開日，掲載紙などの構造化情報が PubMed [3] により公開されており，近年，PubMed の文書データから知見を得ようとする研究が盛んに行われている．弊社では，生物医学文献に特化した MedTAKMI (IBM TAKMI for Biomedical Documents) を開発し，提供している [5]．

表 1 は，人手による文書分析とテキストマイニングによる文書分析の長所，短所を示している．人手による分析では，個々のテキストの分析精度は高く，多様で曖昧な分析要求に柔軟に対処できるが，人手での処理には限界があり，大量の文書を分析できず，サンプリングされた文書のみが対象となる場合が多い．また，分析結果は分析者の主観に依存する場合が多々ある．一方，テキストマイニングによる分析では，個々のデータの分析で失敗しノイズを含む可能性があるものの，全データを対象とすることで，大半のデータが正しく解析できていれば，多様な誤りを低頻度に分散させることができる．また個々の処理時間が短いことから，解析者が試行錯誤に分析することが可能である．

表 1　人手，テキストマイニングによる長所，短所

	長所	短所
人手による分析	個々のテキストの分析精度が高い ・多様で曖昧な分析要求に柔軟に対応出来る	・大量のデータを対象に出来ない ・作業量が膨大 ・分析者の主観に依存
テキストマイニング	・全データを対象に出来る → 大局的な分析が可能 ・比較 ・深堀 ・作業時間が短い → 試行錯誤がやり易い	・個々のデータの分析で失敗しノイズを含む可能性

表 2 は自然言語技術により処理された後の膨大な文書データを活用する技術を処理の深さの順に並べたものである．人手での分析で対象を絞り込むためには検索技術が有効であるが，検索結果で得られた文書集合の全体の傾向を把握するには個々の文書に目を通す必要がある．検索結果の文書集合の傾向を把握するために，キーワードを集計して，様々な統計値に基づいて評価することで文書集合の傾向が得られる．後述するカテゴリビューも集計機能の一つであるが，一般に集計処理は検索処理よりも負荷が高い．表 2 のレベル 3 では，単語や文書の組み合わせの処理が必要となり，負荷の大きな処理となる．テキストマイニングシステム TAKMI の活用では，対話的に試行錯誤する操作性を重視しており，そこで如何に集計処理を高速に動作させるかが重要である．また，お客様への提供の観点から，分析者の要求に合致するようにカスタマイズできることが重要であると考える．すなわち，拡張性や他システムとの統合性が重要となる．

表 2　膨大な文書データを活用する技術

処理の深さ	処理の概要	機能 (用途)	技術的要素	処理対象（データ内容 表現形式）	自然言語処理内容
レベル 1	検索	目を通す対象を絞り込む	情報検索	文字列 単語列	単語の抽出（語の原型・基本型への置換など）
レベル 2	集計	全体的にどのような内容が含まれているかをキーワードレベルで把握	集計	単語の集まり	単語分布状況の分析
レベル 3	分類・整理	全体的にどのような内容が含まれているかをドキュメント間の関係から把握	クラスタリング クラシフィケーション	Vector Space Model	状況の分析 関係の分析

TAKMI の各機能は高速に動作するが，独自のデータ形式，独自の集計エンジンを用いているので，他のシステムとの統合や新たな機能の開発が容易ではない．これら問題を解決するために，本稿では，アノテーションされたキーワードを関係データベースにストアし，TAKMI の集計/マイニング機能を SQL で高速に実行する手法を提案する．本稿の構成は以下の通りである．2 節でテキストマイニングシステム TAKMI の構成を簡単に紹介する．3 節では，ドキュメント-用語行列を用いた従来の実装方法をカテゴリビューと呼ばれるマイニング機能を例に説明する．4 節では，アノテーションされたデータを関係データベースにストアし，カテゴリビューの機能を SQL で高速に実行する手法を提案する．5 節では，MEDLINE からの約 50 万アブストラクトを用いて，従来手法と提案手法の比較実験を行う．6 節で拡張性について考察し，7 節で本稿のまとめを述べる．

2. TAKMI

TAKMI の構成は構文解析，情報抽出，関係抽出等の処理を行う前処理機能と検索，集計操作を行うランタイム機能からなる．本節では，これらの構成要素を簡潔に述べる．詳細は文献 [4]〜[6] を参照されたい．

2.1 前　処　理

TAKMI システムの前処理では，入力テキストは JSA (Japanese Syntactic Analyzer) [7] や CCAT [8] によって構文解析される．一般的な統計的構文解析器は，新聞記事など特定のコーパスから統計的手法を用いて学習しており，コールセンタのテキストログや生物医学文献から学習されているわけではないので，分野の異なるテキストデータに対して高い精度を得ることが困難な場合がある．図 1 に示されるように前処理の第

(注1)：United States National Library of Medicine
(注2)：National Cancer for Biotechnology Information

1 ステップでは，入力として与えられたテキストデータは，はじめに固有表現の判定，同義語の検出が行なわれる．生物医学文献には，例えば，“1,2-dihydroxy-1,2-dihydroxynaphthalene dehydrogenase” といった物質名や “repetitive sequence-based polymerase chain reaction” という現象名が出現する．このような語を含むテキストから正しい固有表現を抽出することは困難である．そこで，辞書を用いて単語の境界と品詞をこれらの語に与えておく．続いて同義語の検出について説明する．生物医学文献には，例えば，“DNA” と “Deoxyribo Nucleic Acid” のように，同じ概念を表す表現が異なる表記で表される場合がある．これらの語が異なる概念として扱われると，次節で説明する集計機能で本来同一の概念が別々に集計される問題がある．従って，“DNA” と “Deoxyribo Nucleic Acid” の正書形 (canonical form) として，それぞれに “Deoxyribo Nucleic Acid” と登録しておくことで，これらを同一視する．

第 2 ステップでは，専門用語辞書によってアノテーションされたテキストを構文解析する．第 3 のステップでは，キーワードの正書形とカテゴリツリーの節点のペアからなるカテゴリ辞書を用意し，キーワードに対しカテゴリを付与する．例えば，“brain”，“lung”，“liver” などのキーワードに対し，臓器カテゴリを付与し，キーワードとカテゴリパスのペアを抽出する．また構文解析の結果から名詞，動詞などを抽出し，さらに動詞-名詞などの 2 項関係，名詞-動詞-名詞などの 3 項関係を抽出する．これはキーワード単位の集計からドキュメント集合の頻出単語の傾向を把握することができるが，実際のテキストにどのようなことが書かれているのかを把握することができないためである．そこで，コールセンタを対象とするようなテキストマイニングシステムでは，テキストから「何がどうした」という情報を取り出すだけで，大まかな問題を把握するのに有用となる．一方，生物医学文献を対象とする場合には，「何が何にどう作用した」を取り出すことがより重要である．例えば，“Smoking increases risk of lung cancer.” というテキストから “smoking...increase...risk” という NVN の 3 項関係が抽出される．

図 2 は TAKMI の前処理の例である．“Repetitive sequence-based polymerase chain reaction affects deoxyribonucleic acids” が入力として与えられたとき，“deoxyribonucleic acids” の正書形として “Deoxyribo Nucleic Acid” が，品詞として “proper noun” が与えられる．構文解析語，それぞれのキーワード，2 項関係，3 項関係に対し，カテゴリが割り当てられる．カテゴリは木構造であり，ドットによって節点名が区切られているものとする．

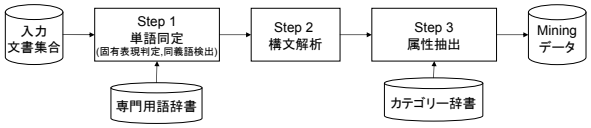

図 1　TAKMI の前処理

2.2　検索，マイニング機能

TAKMI はドキュメント集合全体，あるいはキーワード検索や全文検索によって絞り込まれたドキュメント集合に対して，様々なマイニング機能を適用し，ドキュメント集合の特徴を得たり，所望のドキュメントを検索するためのシステムである．キーワード検索では，ユーザが与えたキーワードか，その同義語をもつドキュメント集合が返される．マイニング機能には，カテゴリビュー，2 次元 Map ビュー，トピック分析ビュー，増減分析ビュー，時系列ビュー，パターン検索ビューと呼ばれる機能などを有する．カテゴリビューは指定されたカテゴリに属するキーワードに対するドキュメント数の分布や，指定されたカテゴリのサブカテゴリ毎のドキュメント分布を調べる機能であり，絞り込んだドキュメント集合に頻出するキーワード一覧や特徴語一覧を得ることができる．例えば，ある病名で絞り込んだドキュメント集合に対して，遺伝子のカテゴリを指定することで，その病気と頻繁に共起するキーワードや病名特有の遺伝子の一覧をドキュメント数と共に得ることができる．2 次元 Map ビューは 2 つのカテゴリの属するキーワードの相関/関連の強さを調べるための機能である．例えば，遺伝子間の関連や，遺伝子-病名間の関連などを調べることができる．その他，増減分析ビューは，頻度の時間的変化から，ある項目が急に増えたり減ったりする様子を表示するための機能，トピック抽出ビューは，頻度の時間的変化に着目し，時期別に目立って現れた項目の変遷を表示するための機能，時系列ビューは時系列的な頻度の変化を表示するための機能である．

さらにユーザは検索機能やマイニング機能で得られた結果を用いてさらにドキュメントの絞り込みを行うことができる．対話的に操作できる機能は TAKMI の重要な特徴であり，ユーザは検索とマイニングを柔軟に切り替えながら，ドキュメント集合の特徴や，所望のドキュメントを得ることができる．

シナリオ: 自己免疫学に関する 3 つの病名 (狼瘡 (lupus)[注3]，乾癬 (psoriasis)[注4]，間接リウマチチ (rheumatoid arthritis))

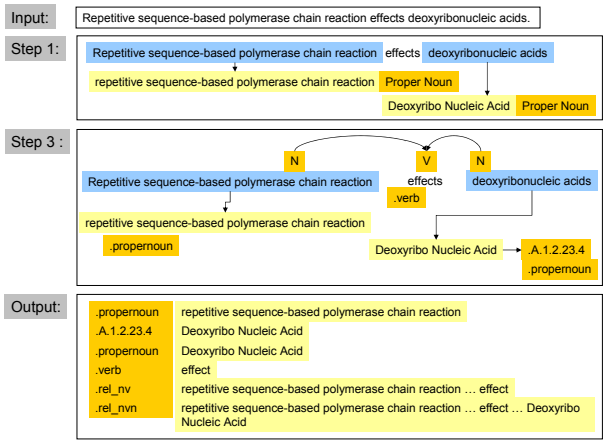

図 2　TAKMI の前処理の例

(注3)：結核などの原因により皮膚の状態が破壊された病変．

(注4)：皮膚の紅斑の上に、表皮角層の上層が、銀白色の雲母状の大小の角質片状となる慢性皮膚炎．

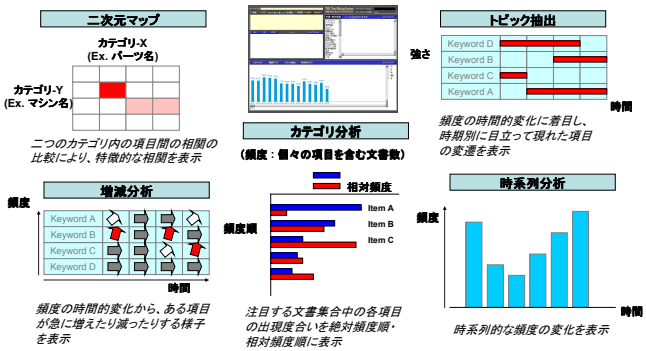

図 3　TAKMI のマイニング機能

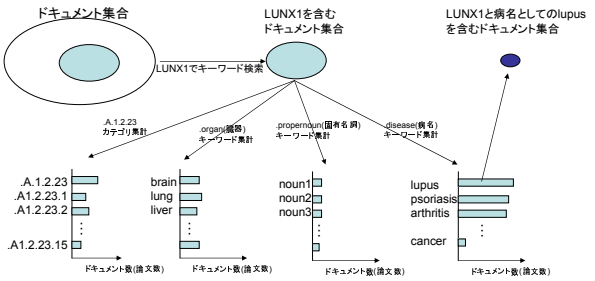

図 4　解析シナリオ

はそれぞれ異なる遺伝子に関連していることが知られているが，近年これらの病名が RUNX1 として知られるたんぱく質が結合する遺伝子の欠陥によって引き起こされることが明らかになった [9]〜[11]．RUNX1 がそれぞれの病名と関連があることを事前に知っていれば，それに関する論文を通常の検索エンジンで検索することは容易である．しかし，RUNX1 がどのような病名と関連があるかを調べたい場合には，図 4 に示すように RUNX1 で絞り込んだドキュメント集合に対して，病名カテゴリを指定してカテゴリビューを実行することで，RUNX1 と狼瘡，乾癬，間接リウマの関連が強いことに気づくかもしれない．図 4 は論文数の分布の図となっているが，TAKMI では文献 [5] に示されるような統計的指標に基づく特徴量も表示される．TAKMI は前処理で lupus, psoriasis, rheumatoid arthritis に病名というカテゴリを割り当てているので，キーワード全体でなく，病気のキーワードのみに限定して関連を調べることができる．さらに，TAKMI の GUI 上で例えば，“lupus” を選択することで，RUNX1 と病名としての lupus を含むドキュメントに絞り込んで，対話的に解析を繰り返すことができる．

3. 従 来 手 法

本稿では説明の簡単化のため，カテゴリビューのみについて説明するが，他の機能も本稿で述べる手法により容易に拡張可能である．従来手法で，TAKMI は各機能を高速に実行するためにドキュメント-用語行列 (Document-Term Matrix ：DTM) を用いている．ドキュメントの集合を $D = \{d_1, d_2, \ldots, d_m\}$，用語の集合を $T = \{t_1, t_2, \ldots, t_n\}$ とする．TDM は $m \times n$ の行列 $M = (m_{ij})$ であり，その要素 m_{ij} は t_j が d_i で何回出現したかを表している．この行列をそのまま記憶するには多くの

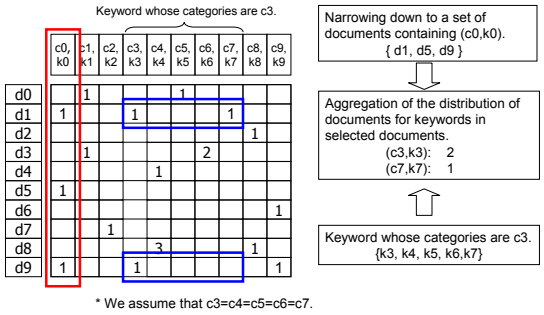

図 5　カテゴリビューの実行例

メモリ量を要するが，DTM は非常に疎な行列であるので，非ゼロの要素とインデックスを記憶することで，実メモリ上にストアすることができる．

2.1 節で述べたように，各キーワードにはカテゴリが割り当てられているので，TAKMI は以下のような DTM を用いている．DTM の各行は従来と同様に各ドキュメントに相当する．各列はカテゴリとキーワードの対 $P = \{(c_1, k_1), (c_2, k_2), \ldots, (c_n, k_n)\}$ の 1 つに相当する．図 2 の例では，(c_j, k_j)=(.A.1.2.23.4, DNA) などである．同じカテゴリでも異なるキーワード，あるいは異なるカテゴリでも同じキーワードの場合もあるので，$i \neq j$ に対し，$c_i = c_j$，あるいは $k_i = k_j$ の場合もありうる．さらに $p_j = (c_j, NULL)$ の場合もあり，p_j に対する m_{ij} はカテゴリ c_j に直接属するキーワード，あるいは c_j の下位カテゴリに属する任意のキーワードが d_i に何回出現したかに相当する．これは，後述するカテゴリ集計を高速に実行するためのものである．

図 5 は，ユーザがカテゴリが c_0 であるキーワード k_0 を含むドキュメントに絞り込んだ後，カテゴリ c_3 を指定したとき，TAKMI がどのようにカテゴリビューを実行するかを表している．はじめに (c_0, k_0) を含むドキュメント集合 $\{d_1, d_5, d_9\}$ に絞り込む (処理 1)．それと同時に，カテゴリ c_3 に属するキーワード $\{k_3, k_4, k_5, k_6, k_7\}$ を得る (処理 2)．その後，$\{d_1, d_5, d_9\}$ に現れるキーワードに対するドキュメントの分布 $\{(c_3, k_3) : 2, (c_7, k_7) : 1\}$ が返される (処理 3)． これをキーワード集計と呼び，図 4 の下図の右 3 つに相当する．また，カテゴリビューにはカテゴリ集計機能があり，図 4 の下図の左に相当する．. これはユーザが指定したカテゴリ c_3 と子カテゴリ $c_3.1, \cdots, c_3.s$ に対するドキュメントの分布を返す．この場合，処理 2 で $(c_3.*, NULL)$ の集合が返される．処理 3 は処理 1 や 2 に比べ計算負荷が高いので，実際の実行時には処理 3 の直前に 1000 件程度にサンプリングして，処理 3 は実行される(注5)．

TAKMI の各機能は高速に動作するが，独自のデータ形式，独自の集計エンジンを用いているので，他のシステムとの統合や新たな機能の開発が容易ではない．これら問題を解決するために，次節では，アノテーションされたキーワードを関係データベースにストアし，SQL で高速に集計する手法を提案する．

(注5)：サンプリング数は設定可

4. 提案手法

カテゴリツリーとアノテーションされたキーワードをストアするために以下のように，CATEGORY テーブルと KEYWORD テーブルを定義する．

```
CATEGORY (PATH CHARACTER,
          DESCRIPTION CHARACTER,
          PARENT CHARACTER),
KEYWORD (DOCID INT,
          KEYWORD CHARACTER,
          PATH CHARACTER)
```

CATEGORY テーブルの各レコードは，カテゴリツリー上の各節点であり，PATH はルートから節点までのパス，DESCRIPTION は節点名，PARENT は親節点のパスである．一方，KEYWORD テーブルの各レコードはアノテーションされた各キーワードであり，DOCID はドキュメント ID，KEYWORD はキーワード，PATH はカテゴリパスである．あるカテゴリ *path* に属する各キーワードに対するドキュメントの分布を集計するための SQL は

```
SELECT KEYWORD, COUNT(DISTINCT DOCID)
  FROM KEYWORD WHERE PATH='path'
  GROUP BY KEYWORD,                    (1)
```

であり，*path* に対するカテゴリ集計のための SQL は

```
SELECT DESCRIPTION, COUNT(DISTINCT ID)
  FROM CATEGORY AS A, KEYWORD AS B
  WHERE A.PATH='path' AND
   (B.PATH='path' OR B.PATH LIKE 'path.%')
  GROUP BY DESCRIPTION
UNION ALL
SELECT DESCRIPTION, COUNT(DISTINCT ID)
  FROM CATEGORY AS A, KEYWORD AS B
  WHERE PARENT='path' AND B.PATH LIKE
     'path.%' AND (A.PATH=B.PATH
     OR RIGHT(B.PATH,LENGTH(B.PATH)
     -LENGTH(A.PATH)) like '.%'
     AND LOCATE(A.PATH,B.PATH)=1 )
  GROUP BY DESCRIPTION,                (2)
```

で表される．ここで，*path* は図 5 の c_3 に相当する．

SQL(2) は文字列関数を含むので，多くの計算時間を要する．3 節で述べたような $(c_j, NULL)$ に相当するレコードをストアするのも 1 つの方法であるが，アノテーションされるキーワード数やカテゴリの節点数は非常に大きいので，提案手法では，木構造の祖先子孫判定を行うために前順-後順法を用いる [12]．この手法は図 6 に示すように，木の各節点に前順，後順を割り当てて，2 節点の前順，後順を比較することで包含判定を行う手法である．もし A が B の祖先であるならば，A の前順は B の前順より大きく，かつ A の後順は B の後順未満のときに限る．SQL(2) の文字列関数は前順，後順の大小関係に置き換えられるので，提案手法は高速に集計処理を行うことができる．

以上より，CATEGORY テーブルと KEYWORD テーブルを以下のように修正する．

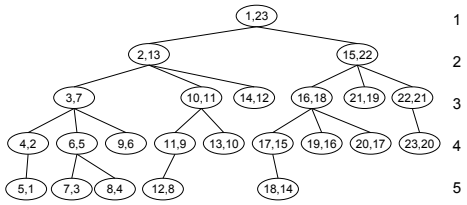

図 6　木の前順，後順，深さ

```
CATEGORY (PATH        CHARACTER,
          DESCRIPTION CHARACTER,
          PREORDER    INT,
          POSTORDER   INT,
          DEPTH       INT,
          PARENT      INT),
KEYWORD (DOCID     INT,
         KEYWORD   CHARACTER,
         PREORDER  INT,
         POSTORDER INT)
```

CATEGORY テーブルの PREORDER, POSTORDER, DEPTH は，節点の前順，後順，深さであり，PARENT は親節点の前順である．KEYWORD テーブルの PREORDER，POSTORDER はアノテーションされたカテゴリの節点の前順，後順である．

節点 *path* の前順，後順を *pre*，*post* とすると，新たなテーブルの定義に従って，SQL(1)，(2) は

```
SELECT KEYWORD, COUNT(DISTINCT DOCID)
  FROM KEYWORD WHERE PREORDER=pre
  GROUP BY KEYWORD                     (3)
```

及び，

```
SELECT DESCRIPTION, COUNT(DISTINCT DOCID)
  FROM CATEGORY AS A, KEYWORD AS B
  WHERE A.PREORDER>=pre AND A.POSTORDER<=post
    AND B.PREORDER>=pre AND B.POSTORDER<=post
    AND B.PREORDER>=A.PREORDER
    AND B.POSTORDER<=A.POSTORDER
  GROUPY BY DESCRIPTION                (4)
```

となる．SQL は全データに対して集計処理を行う場合の SQL であるが，事前に複数のキーワードでドキュメントが絞り込まれている場合は，SQL(3) は

```
SELECT KEYWORD, COUNT(DISTINCT DOCID)
  FROM KEYWORD
  WHERE PREORDER=pre AND DOCID IN (sql)
  GROUP BY KEYWORD,
```

となる．ここで *sql* は絞り込まれたドキュメント集合の ID を返す SQL であり，図 5 の例では，$\{d_1, d_5, d_9\}$ を返すことに相当する．実際の実装では，索引をより良く活用するために POSTORDER<=*post* の代わりに PREORDER<*post*+*dep* を用いたり，副問い合わせを用いている．

5. 実　　験

提案手法を Java で実装し，C++で実装されている従来手法と比較実験を行った．TAKMI のシステム全体は図 7 に示されるような Web サービスとして実装されており，検索エンジンを操作するようにブラウザ上で容易に操作できる．実験では client 側でカテゴリビューの問い合わせを発行し，TAKMI サーバーで集計操作を行い，結果を返す．従来手法は独自のインデックスファイルを用いて，集計操作を行う．提案手法は Java のプログラムが SQL を生成して，JDBC (Java Database Connectivity) 経由で，関係データベースにストアされたデータにアクセスし，SQL で集計操作を行う．TAKMI サーバーの CPU は Opteron 248 2.2GHz で，メモリが 3GB である IntelliStation 6224 を用いた．またこの計算機には Windows XP と DB2 Universal Database Enterprise Server Edition v8.2 がインストールされている．実験に用いたデータは 503,989 件の Medline アブストラクトであり，テキスト (非定型) データともに，定型項目 (著者や Mesh Term など) からなる．前処理後，アノテーションされたキーワードは 193,185,919 語，カテゴリ数は 340,154 節点，(c_j, k_j) の種類は 14,331,595 であった．

実験方法は，図 4 に示すようにドキュメント集合全体に対して，1 つのカテゴリを指定してキーワードビューを実行する．TAKMI サーバーが問い合わせを受けてからからの応答時間を計測する．この操作を全 340,154 カテゴリに対して，キーワード集計とカテゴリ集計操作を繰り返す．続いて，“cancer” で絞り込んだドキュメント集合に対して，同様に行う．計算時間は “cancer” によるキーワード検索の時間も含むが，各問い合わせは，自動的に生成されるので，ユーザの操作の時間は含まない．

図 8 は全データに対する計算時間である．図 8 の DTM と DB は従来手法と提案手法であり，KW と SUB は指定した

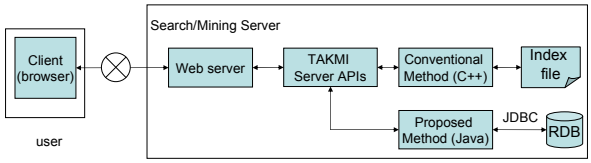

図 7　実 験 環 境

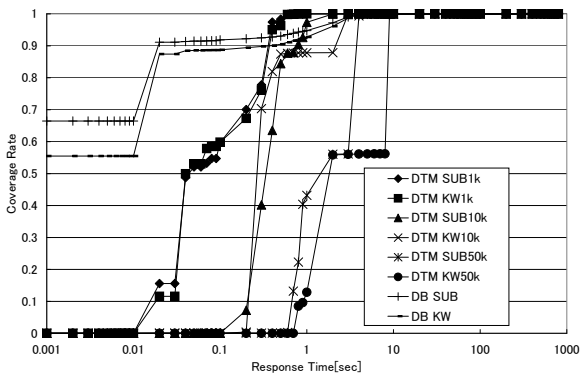

図 8　全データに対する計算時間

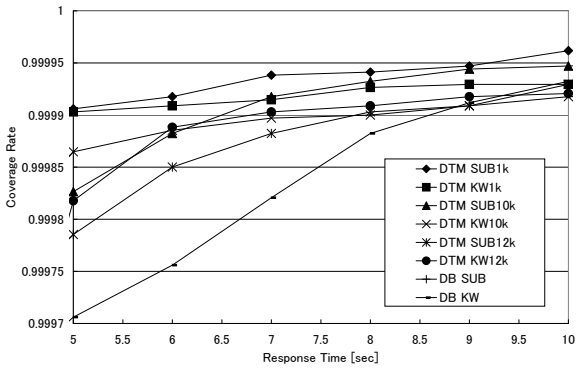

図 9　“cancer” で絞り込んだデータに対する実験結果

カテゴリに属するキーワード集計とカテゴリ集計に相当する．“DTM KW 1k” はサンプリングされた 1000 件に対する従来手法の計算時間であり，提案手法はサンプリングを行わず全データを用いた．図中の各点 (x, y) は，全カテゴリのうち y%のカテゴリが x 秒以内で計算できたことを示している．提案手法はキーワード集計において，0.1 秒で約 89%のカテゴリの結果を返すことができたが，従来手法はサンプリングされた 1000 件に対して約 60%，10,000 件に対して約 0.01%であった．続いて，“cancer” で絞り込んだ 11,914 件のドキュメント集合に対しても実験を行ったが，図 8 と同様であった．図 8 に示されるように，たいていのカテゴリに対して，提案手法は従来手法より高速に計算できた．

しかしながら,「インターネット上の Web ページはユーザがその URL を訪れてから 8 秒以内に表示されなければならない」という経験則の観点から実験結果を見たとき，10 秒付近でのカバー率が高いほうがより良い手法であると言える．図 9 は，“cancer” で絞り込んだデータに対する結果であり，横軸が 5 秒から 10 秒，縦軸を 99.97%から 100%の領域を拡大したものである．8 秒ルールの観点から見ると図 9 は従来手法の方が提案手法より良いと言える．表 3，4 は全データと “cancer” を含むドキュメントに絞り込んだデータに対する実験結果をまとめたものである．平均計算時間は提案手法の方が短いが，10 秒以内に結果を返せないカテゴリ数は提案手法の方が多くなる場合がある．

表 3　全データに対する実験結果

手法	平均計算時間［秒］	10 秒 †	100 秒 ‡
DTM KW 1k	0.143	1	0
DTM KW 5k	0.284	26	0
DTM KW 10k	0.526	16	0
DTM KW 50k	4.293	75	0
DB KW	0.185	10	0
DTM SUB 1k	0.138	2	1
DTM SUB 5k	0.176	2	0
DTM SUB 10k	0.405	5	1
DTM SUB 50k	2.091	33	0
DB SUB	0.137	20	2

†，‡ は 10 秒，100 秒で結果を返せないカテゴリ数

表 4 “cancer” で絞り込んだデータに対する実験結果

手法	平均計算時間［秒］	10 秒 †	100 秒 ‡
DTM KW 1k	0.177	24	5
DTM KW 5k	0.355	116	5
DTM KW 10k	0.511	28	7
DTM KW 12k	0.594	27	7
DB KW	0.267	23	0
DTM SUB 1k	0.195	13	1
DTM SUB 5k	0.352	65	0
DTM SUB 10k	0.484	18	1
DTM SUB 12k	0.534	24	2
DB SUB	0.413	706	5

表 4，及び図 9 の実験で，“cancer” に絞り込んだデータに対して，前順が pre であるカテゴリに対するキーワードの集計に対する SQL は

```
SELECT A.KEYWORD, COUNT(DISTINCT A.DOCID)
  FROM KEYWORD AS A, KEYWORD AS B
  WHERE A.PREORDER=pre  AND A.DOCID=B.DOCID
        AND B.KEYWORD='cancer'        (5)
```

で表される．提案手法において，あるカテゴリに対して多くの計算時間を要する要因の 1 つは 193,185,919 レコードからなる KEYWORD テーブルのセルフジョインである．提案手法を改善するために，以下の手法を用いる．テーブル T のドキュメント ID を返す関数を $id(T)$ とする．テーブル T を $id(T) = \cup_i id(T_i)$ かつ $id(T_i) \cap id(T_j) = \emptyset (i \neq j)$ を満たすように，複数のテーブル $T_i (i > 2)$ に分割する．SQL(5) は WHERE 節に A.DOCID=B.DOCID をもつので，分割された各テーブル毎に SQL(5) を実行後に総和をとることで，以下に示す SQL で共通のドキュメント ID を含まないテーブルのジョインを避けることができる．

```
SELECT KEYWORD, SUM(COUNT)
FROM (
  SELECT A.KEYWORD, COUNT(DISTINCT A.DOCID)
    FROM KEYWORD1 AS A, KEYWORD1 AS B
    WHERE A.PREORDER=pre  AND A.DOCID=B.DOCID
          AND B.KEYWORD='cancer'
  UNION ALL
  ...
  UNION ALL
  SELECT A.KEYWORD, COUNT(DISTINCT A.DOCID)
    FROM KEYWORDi AS A, KEYWORDi AS B
    WHERE A.PREORDER=pre  AND A.DOCID=B.DOCID
          AND B.KEYWORD='cancer'
  ) A
GROUP BY KEYWORD                    (6)
```

図 10 は “cancer” で絞り込まれたデータに対する，従来手法，1 つの KEYWORD テーブルに対する提案手法，KEYWORD を 10 分割した場合の提案手法を比較した実験結果である．KEYWORD テーブルを分割することで，10 秒におけるカバー率はキーワード集計で 99.993%から 99.999%に，カテゴリ集計で 99.79%から 99.93%に上昇している．

表 5 は，“cancer” で絞り込んだデータに対して，多くの異なり語が属するカテゴリに対する計算時間である．表 5 より提案手法が，異なり後が多いカテゴリに対しても，高速に動作することが分かる．

表 5 異なり語を多く含むカテゴリに対する計算時間

category	異なり語数	DTM KW 5k	DTM KW 12k	DB KW
.commonnoun	340,154	231.2	238.7	53.0
.propernoun	7,903	11.8	10.8	2.8
.verb	32,826	73.9	74.4	9.7
.adj	23,629	61.6	62.4	7.2
.rel_vn	1,337,891	171.3	177.7	7.6
.rel_nv	601,256	85.7	85.3	3.6
.rel_vnn	5,439,810	145.6	162.9	9.8
.rel_nvn	2,294,012	99.3	101.2	4.4
.rel_nnv	1,812,501	90.7	89.5	3.8
.affiliation	282,729	53.7	55.3	3.2
.pnsubstance	48,512	66.2	68.0	4.3
.majormesh	89,275	83.6	85.1	3.1
.minormesh	111,761	126.2	128.5	7.8
.chemical	10,146	17.9	18.2	3.1
.genesymbol	15,310	48.1	48.8	7.3
.protein	15,562	57.1	58.2	6.7
.biomedicalterm	91,670	133.7	137.8	21.1

6. 考　　察

提案手法は，集計操作を SQL で行うために既存の機能，新たな機能を容易に実現することが可能である．本節では，2 次元 Map ビューの実現方法，並列化，新たな機能構築の可能性について述べる．

2 次元 Map ビュー

本稿では説明の簡単化のためにカテゴリビューについて説明したが，2 次元 Map ビューも，以下のような SQL で実行できる．

```
SELECT A.KEYWORD, B.KEYWORD, COUNT(DISTINCT DOCID)
  FROM
    (SELECT DOCID, KEYWORD
      FROM KEYWORD WHERE PREORDER=pre1) A,
```

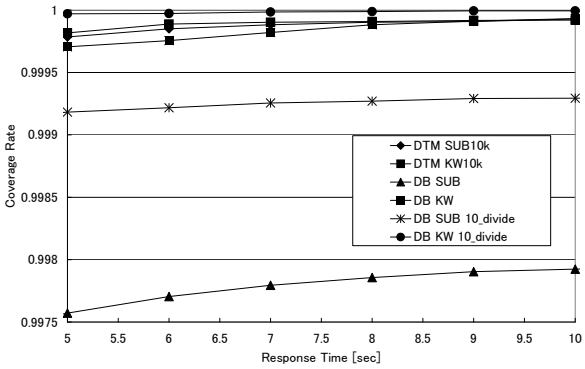

図 10 KEYWORD テーブルを 10 分割した場合の比較実験結果

```
    (SELECT DOCID, KEYWORD
      FROM KEYWORD WHERE PREORDER=pre2) B
  WHERE A.DOCID=B.DOCID
  GROUP BY A.KEYWORD, B.KEYWORD
```

ここで，*pre1* と *pre2* はユーザに指定された 2 つのカテゴリの前順であり，各軸ともにキーワード集計の場合である．

並列化

5 節で KEYWORD テーブルの分割法について述べた．SQL(6) から分かるように分割された各テーブル毎の集計は独立に計算される．比較実験は単一 CPU の一台の計算機で行われたが，商用の関係データベースは並列計算をサポートしているので，ソースコードを修正することなく容易に並列計算でき，さらに分割の効果が望める．一方，C++で実装された従来方法は MPI (Message Passing Interface) などを用いた開発が必要となるが，関係データベースを用いることで短期間での開発が望める(注6)．

拡張性

前処理で好評，あるいは不評を意図する形容詞に対し，好評/不評のカテゴリが割り当てられているとする．「A だが B」という表現と「B だが A」という表現はニュアンスが異なるが，このような表現数の違いを調べるために，前処理されたデータに対し「同一文内で，好評の形容詞，逆接の接続詞，不評の形容詞の順に述べられているドキュメント数を形容詞のペア毎に集計せよ」という要求があったとする．これまで述べてきたように，TAKMI はドキュメント単位の計算を想定しているので，文単位の計算や出現位置の計算をするには大幅な拡張が必要となる．一方，提案手法は，このような現在の仕様を越える拡張に柔軟に対応できる．例えば，KEYWORD テーブルを

```
KEYWORD (DOCID     INT,
         SENTENCEID INT,
         POSITION  INT,
         KEYWORD   CHARACTER,
         PREORDER  INT,
         POSTORDER INT)
```

とする．ここで，SENTENCEID は文 ID，POSITION はキーワードの出現位置である．このようなテーブルに対して

```
  SELECT A.KEYWORD, C.KEYWORD, COUNT(DISTINCT DOCID)
  FROM
    (SELECT DOCID, KEYWORD
      FROM KEYWORD WHERE PREORDER=pre1) A,
    (SELECT DOCID, KEYWORD
      FROM KEYWORD WHERE PREORDER=pre2) B,
    (SELECT DOCID, KEYWORD
      FROM KEYWORD WHERE PREORDER=pre3) C
  WHERE A.DOCID=B.DOCID AND A.DOCID=C.DOCID
    AND A.SENTENCEID=B.SENTENCEID
    AND A.SENTENCEID=C.SENTENCEID
    AND A.POSITION<B.POSITION
    AND B.POSITION<C.POSITION
  GROUP BY A.KEYWORD, C.KEYWORD
```

という集計操作で，所望の集計結果が得られる．ここで，*pre1*, *pre2*, *pre3* はそれぞれ好評の形容詞，逆接の接続詞，不評の形容詞に相当するカテゴリの前順である．これまでの我々の経験から，適切な索引を作成することで，ある程度十分なパフォーマンスを得られる．

7. おわりに

本稿では，アノテーションが付与されたキーワードを関係データベースにストアし，TAKMI のカテゴリビューの機能を SQL で集計する機能を提案した．本稿では説明の簡単化のためにカテゴリビューについて説明したが，他の機能も SQL で実行可能である．C++で実装されている従来手法は数千行のソースコードであるが，提案手法は数行程度の SQL であるので，新たな機能を容易に開発することができる．また 5 節で示したように，提案手法は，従来手法と同等以上に動作することが確認された．

(注6)：従来手法の TAKMI の並列版は既に開発済みである．

文　献

[1] NLM
http://www.nlm.nih.gov/

[2] NCBI
http://www.ncbi.nlm.nih.gov/

[3] PubMed
http://www.ncbi.nlm.nih.gov/entrez/query.fcgi

[4] T. Nasukawa, & T. Nagano: Text analysis and knowledge mining system. *IBM Systems Journal* 40(4): pp. 967–984, 2001.

[5] N. Uramoto, H. Matsuzawa, T. Nagano, A. Murakami, H. Takeuchi, & K. Takeda: A text-mining system for knowledge discovery from biomedical documents. *IBM Systems Journal*, 43(3): pp. 516–533, 2004.

[6] 松澤, 長野, 村上, 浦本, 武田: ライフサイエンス向けテキストマイニングツール MedTAKMI, データベースシステム研究会研究報告, No.130–005

[7] 金山，鳥澤，光石，辻井: 3 つ以下の候補から係り先を選択する係り受け解析モデル, 自然言語処理, Vol. 7, No. 5, 2000.

[8] E. Charniak: *Statistical Language Learning*, MIT Press, 1994.

[9] S. Tokuhiro, et al. An intronic SNP in a RUNX1 binding site of SLC22A4, encoding an organic cation transporter, is associated with rheumatoid arthritis. *Nature Genetics*, Novermber 9, 2003.

[10] C. Helms, et al. A putative RUNX1 binding site variant between SLC9A3R1 and NAT9 is associated with susceptibility to psoriasis. *Nature Genetics*, Novermber 9, 2003.

[11] L. Prokunina, et al. A regulatory polymorphism in PDCD1 is associated with susceptibility to systemic lupus erythematosus in humans. *Nature Genetics*, 32, pp. 666–669, December, 2003.

[12] P. F. Dietz: Maintaining order in a linked list, *Proc. of the fourteenth annual ACM symposium on Theory of computing*, pp. 122–127, 1982.