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

2008－DBS－144 (7)
2008－GN－ 66 (7)
2008／1／24

対話的テキストマイニングのためのタグ付け用検索基盤

楠村幸貴† 神谷俊之†

近年，電子化された文書の増加によりテキストマイニングのニーズが高まっている．テキストマイニングではシステムの開発時に構造化処理をどのように行うかが課題となっている．そこで我々は対話的なテキストの構造化を可能にする基盤技術の開発を目指している．このためには，タグの追記が容易であり高速に行えること，また，追記されたタグ情報を用いたパターン検索が高速に行えることの両方が重要である．本稿では，この目的のために開発したLRインデックスとタグ付け用検索基盤システムについて述べ，その評価を行う．

A Data Management System for Interactive Text Mining

YUKITAKA KUSUMURA† and TOSHIYUKI KAMIYA †

The demand of text mining system is increasing. We aim at providing a data management system for interactive text mining. For this purpose, the data management system must be able to not only search ttext by the pattern of tags but also update tags. In this paper, we present the LR-index structure, which is an index structure for the fast searching and fast updating of tags.

1. はじめに

近年，電子化された文書の増加によりテキストマイニングのニーズが高まっている．テキストマイニングでは非構造情報であるテキストを分析しやすくするために，テキストに対して繰り返し構造化処理を行う．例えば，Webページを用いてレストランに関する分析を行う際には，まずテキストに対して形態素解析器や固有表現抽出器などのタガーを用いテキストに言語情報などを抽出する．次にこれらの情報を参照するタガーを利用しレストラン名や住所，メニューなどを抽出する．さらにこれらの情報を利用し，対象のレストランの評判情報などを抽出する，といった処理が挙げられる．

よってテキストマイニングを行う際には，開発者がテキストに対してどのような構造化処理を行うかを設計する必要がある．しかし，テキストに対してどのような処理を行うとテキストから有益な知識が抽出できるかは予め自明でない．そこで開発者は種々のタガーや分析プログラムを用いて対象のテキストにタグを付加して構造化を行い，結果を見ながらタグ付け対象の辞書の強化，抽出ルールや抽出アルゴリズムの変更，付加されるタグ自体の変更を行うなどの試行錯誤を行っている．しかし，このようなシステムの開発時においては，データの管理が煩雑であることが多い．従来，このようなデータを管理するシステムには，関係データベース(RDB)やXMLデータベース(XMLDB)が挙げられる．しかし，これらのシステムは固定的な構造（あるいはスキーマ）において高い更新性能や検索性能を提供するものであり，タグの追記や変更に伴う構造の変化に向いておらず時間がかかるという問題がある．

このことから従来，開発者はこの構造・スキーマの変更に人手と時間をかけて対応するか，あるいは予め小さな量の文書データを対象に，構造化処理のみの精度という観点で構造化の処理プロセスの調整を行うか，を行い試行錯誤にかかる手間を削減する作業を行ってきた．しかし，これらの作業は専門的な知識や経験が必要であり簡単には行えないといった問題がある．

このような背景から我々はこのようなシステム開発を支援するために，タグの動的な追加・削除を可能にするデータ管理システムとして，タグ付け用検索基盤の開発を目指す．図1にタグ付け用検索基盤を示す．タグ付け用検索基盤とは，文書データベースから文書データをインポートし，タガーに対してタグ更新，タグパターン検索，タグKWIC検索の3つの機能を提供する基盤ツールである．

† NEC サービスプラットフォーム研究所
Service Platforms Research Laboratories, NEC Corporation

タグ更新は，タガーに対して文書中の任意の位置を指定し，任意のタグを付加・削除・更新できる機能である．タグパターン検索は，タグや文字列の組み合わせによる文書検索機能である．例えば，タグと文字列を組み合わせたパターンとして「[属性:企業名 {NEC}] の [固有表現:人名]」が挙げられる．この検索クエリは，タグ名が「属性」でその値が「企業名」であるタグが付加された「NEC」という語,「の」という語，タグ名が「固有表現」でその値が「人名」であるタグが付加された任意の語が連続して登場する部分を返却せよ，という意味になる．この機能により，タグの付加やタグに関する統計値算出を高速化できる．タグ KWIC 検索は，あるタグや文字列，またその組み合わせによるパターンをクエリとし，その付近の文字列や別のタグを検索する機能である．この機能により，あるタグ付け対象の事例からその付近のタグと文字列を収集でき，タグを抽出するルールの作成を高速化できる．

我々はこのような機能を持った基盤を提供することで，テキストマイニングを行うシステムの開発時のコストを削減でき，また，テキストを対話に構造化するテキストマイニングツールを構築することができると考えている．

タグ付け用検索基盤を開発することを目指した場合，その技術課題はタグの更新の高速性と検索の高速性を両立させることが難しいという点にある．通常のインデックス構造を用いると，検索を高速化するためにタグの更新時の処理が大きくなり，更新が大幅に遅くなる．またタグの更新を高速化するためには，検索時のデータ処理が大きくなり，検索が大幅に遅くなるという特徴がある．そこで本論文では，高い更新性能を保ったまま検索を高速化するために，タグに関する構造情報を最小限にしたデータモデルを採用した上で，テキスト情報が固定的であるという事実に着目しタグと左右の語に関するインデックス (LR インデックスと呼ぶ) を作成するというアプローチを取る．

本論文の構成は次のようになっている．本論文ではまず 2 章において関連研究を述べ，3 章においてタギング用検索基盤と LR インデックスについて述べる．さらに 4 章においてその評価実験について述べ，最後に 5 章においてまとめを行う．

2. 関連研究

本研究の他にも，タグと文字列を用いた文書検索システムがいくつか存在している．これらのシステムは大きく，RDB をベースに用いたもの，XMLDB をベースに用いたもの，全文検索エンジンをベースに用いたものに分けられる．

RDB をベースに用いたシステムには茶器[1] がある．RDB は表構造を管理するシステムであり，本来はタグのみをその特性に応じたスキーマで管理すべきである．しかし，茶器のようにタグと単語の組み合わせに関する検索を実現するために，文書データをすべて表で表現するアプローチがある．茶器では単語と言語情報のタグを用いた検索を実現するために表 1 に示すテーブルを利用している．この表の各行は形態素ごとに作成され，その形態素の ID，その形態素の右に存在する形態素の ID，その形態素に付加される品詞などが保持される．検索時はこのテーブルに対し SQL を発行することにより，品詞と文字列に関するパターンを元に文を検索できる．しかし，RDB ではデータを行単位で管理するため，検索時に多くのランダムディスクアクセスが発生するため検索速度が遅くなってしまうという問題がある．さらに，この方法では形態素ごとにレコードを作成するため，文書量が大きいとレコード数が RDB で扱うには大きくなり過ぎ，RDB 上の実装では最適なデータ構造を取れない．例えば，100 万文書中にそれぞれ 1000 単語含まれているとするとレコード数は 10 億行になってしまう．

また，XMLDB を用いたシステムには，ひまわり[2] と清水らのシステム[3] がある．しかし，このインデックスは階層構造に関するクエリに特化しており，タグとその付近の部分文字列を用いたクエリを出すためにはテキストに関するインデックスを用意する必要がある．そこで，ひまわりでは Suffix Array を用いたインデックスを，清水らのシステムでは B+Tree を改良した COB-tree と呼ばれるインデックスを用いている．しかし，通常 XMLDB は構造に関するインデックスを作成しているため，タグの更新を行うと関連する範囲すべての構造を更新せねばならず，更新に時間がかかるという問題がある．

また，全文検索エンジンを用いたシステムには，ANNIC[4] と Cafarella らのシステム[5] などのシステムがある．これらはオープンソースの全文検索エンジン Lucene[6] を元に実装されているが，両者にはインデックス構造に差がある．ANNIC では単語の登場位置を記憶する転置インデックスとタグの登場位置を記憶す

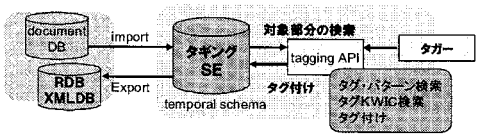

図 1 タグ付け用検索基盤

表 1 茶器の表構造

形態素 ID	右の形態素 ID	文節 ID	文 ID	形態素	品詞	原形
1	2	1	1	彼	名詞	彼
2	3	1	1	は	助詞	は
3	4	2	1	奈良	名詞	奈良

る転置インデックスの両方を持つ．検索時は，例えば「[企業名] の [人名]」というクエリに対して，[企業名] の登場位置のリスト，「の」の登場位置のリスト，[人名] の登場位置のリストをすべて読み込んだ上で位置関係の比較を行う．しかし，これらの条件は一般的過ぎるため大量の登場位置のリストを読み込む必要があり，検索に時間がかかる．

これに対し，Cafarella らのシステムは単語の登場位置を記憶する転置インデックスのみを持つ．ただし単語の転置インデックス内において，各登場位置の情報に加え，付近に登場するタグの情報を格納するという方法を取っている．よって「[企業名] の [人名]」というクエリに対しては，「の」の登場位置のリストを読み込み，それぞれの登場位置情報を参照した上で左に企業名があり，右に人名がある部分だけを取り出す．この方法では，単語に関する登場位置のみをシーケンシャルに読み込むため高速に検索を行うことができる．しかし，この方法はタグが複数並んだクエリを出すためには単語インデックス内に大量のタグ情報を入れなければならないという問題がある．また，例えばある位置に [企業名] というタグを新たに追加するためにはその付近の位置にある複数の単語をキーとする登場位置をすべて更新しなければならず遅い，といった問題がある．

これらのシステムは，タグと文字列に関するパターンで検索を実現することに焦点をあてたものであり，本来タグの更新を想定していない．このため，タグの追加や削除といった作業に大きな時間がかかってしまうという問題がある．本研究ではタグの更新性までを想定したインデックス構造を提案し，そのインデックスを用いたタグ付け用検索基盤を開発する．この点が本研究の新規性である．

3. タグ付け用検索基盤

本章ではタグ付け用検索基盤を説明する．このためにまず，3.1 節でタグのデータモデルについて述べる．次に 3.2 節で本基盤が提供する API について述べ，3.3 節において検索クエリの構文を説明する．さらに，3.4 節において，LR インデックスについて述べる．次に，3.5 節においてタグ付け用検索基盤の全体アーキテクチャについて述べる．

3.1 タグのデータモデル

本基盤では，タグの階層構造を扱わず，タグを登場位置だけで管理するスタンドオフアノテーション[8] のデータモデルを用いたデータ管理を行う．図 2 にこのモデルを用いたタグの定義を示す．本基盤ではタグをタグ名，タグ値，文書上の登場位置から成る情報として定義する．タグ名とはそのタグの属性を表現するための情報であり，「品詞」や「固有表現」といった文字列がこれにあたる．タグ値とはそのタグ名の属性における値を意味し，「名詞」や「組織名」といった文字列がこれにあたる．登場位置とはタグが付加される位置であり，図では実線の矢印がこれにあたる．登場位置は開始位置と終了位置によって表現され，それぞれ文書の先頭からの文字数で表現される．

このモデルでは，タグの階層情報を記述できないため複数の属性情報を持つアノテーションは複数のタグとして管理される．よってタグの階層構造に関するクエリ（例えば，「名詞/組織名」など）が書けない．しかし，このようなクエリを実現するためには XMLDB のようにタグの階層構造に関するインデックスを管理しなければならず，タグの更新速度が大きく低下する．我々は，テキストマイニングにおいては階層構造に関するクエリを実現することよりも更新の高速化を実現することが重要であると判断し，このデータモデルを採用した．

3.2 提供する API

本基盤ではタグ更新，タグパターン検索，タグ KWIC 検索の三つの機能を提供するために，update 関数，search 関数，read 関数の 3 つの関数を用意する．タグ更新は update 関数によって実現される．タグパターン検索，タグ KWIC 検索は search 関数と read 関数を組合すことで実現される．

update 関数はタグの更新を行う関数であり，命令種，タグ名，タグ値，文書番号，開始位置，終了位置を引数とする．このうち，命令種とは更新が追加/削除/値の更新のどれであるかを示す識別子である．タグ名とタグ値は，追加/削除/更新されるタグのタグ名とタグ値を意味する．文書番号，開始位置，終了位置はこの三つの値でタグが追加/削除/更新される位置を

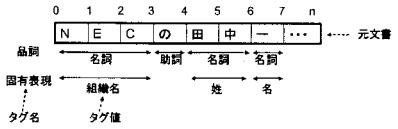

図 2 タグのデータモデル

```
Q → PH
PH → PH KEY
PH → KEY
KEY →文字列
KEY → [TAG]
KEY → [TAG {文字列} ]
TAG →タグ名:タグ値
TAG →タグ値
```

図 3 検索クエリ Q の構文

意味しており，文書番号はある文書を示す ID 値，開始位置と終了位置は文書の先頭からの文字数で表現される．

search 関数はタグに関するパターンの検索を行う関数であり，検索クエリを引数とし，検索クエリで指定されたパターンが登場する位置のリストとして，文書番号，開始位置，終了位置のリストを出力する．検索クエリの構文に関しては 3.3 節で述べる．

read 関数はある位置にあるタグと文字列を読み込む関数であり，文書番号，開始位置，終了位置を引数とし，引数で指定される部分に存在する文字列と，その範囲に付加されているタグ名，タグ値，開始位置，終了位置のリストを返却する．

3.3 検索クエリ

図 3 に検索クエリ Q の構文を示す．この構文ではタグに関する条件を「[」と「]」という区切り文字で表現し，その内部にタグ名とタグ値を「:」という区切りで記述する．また，タグが付加されている文字列を「{」と「}」という区切り文字で表現することとしている．また，タグに関する条件 TAG に関して，本構文ではタグ名のみを条件とするクエリを想定しない☆．例えば「[固有表現：組織名 {NEC}] の [品詞：人名] 社長」といったクエリは，タグ名が「固有表現」でタグ値が「組織名」であるタグが付加された「NEC」という語，「の」という語，「人名」というタグが付加された任意の語，「社長」という語，が隣接して登場する部分を返却せよ，という検索要求を意味する．

3.4 インデックス構造

LR インデックスを説明する前に，転置インデックス[7] を用いた検索に関して説明する．転置インデックスはあるパターンをキーとし，そのパターンが登場した位置のリスト（登場位置リスト）を参照できるようにしたデータ構造である．このとき，タグや単語に関するパターン検索を実現する方法には，単一アプローチと連結アプローチがある．

単一アプローチとは ANNIC[4] が用いたアプローチであり，単一の語をキーとしたインデックスと単一のタグをキーとしたインデックスを作っておく方法である．この方法は検索時にクエリ内の語やタグに関してそれぞれ膨大な登場位置リストを読み込む必要があり，検索が重いという特徴がある．

連結アプローチとは 2 つの語やタグに関するパターンをキーとしたインデックスを作成する方法である．例えば「[固有表現:組織名] の [固有表現:人名]」というクエリに対しては，「[固有表現:組織名] の」というパターンの登場位置リストと，「の [固有表現:人名]」というパターンの登場位置リストを用意しておけば良い．この方法を用いると，より限定した条件で登場位置リストを用意できるため，検索時に読み込む登場位置リストのサイズが減り，検索を高速化できる．しかし，本基盤のように文字列に複数のタグを付加することを想定すると，隣接関係が増えるためインデックス量が膨大になる上，更新に時間がかかってしまうという問題がある．図 4 にこの例を示す．この例では「NEC の田中」という文字列とその文字列に付加されたタグを示しており，文字列の下に描かれた長方形がタグを意味する．このとき破線の矢印がそれぞれインデックスに作成されるキーの組み合わせを意味する．このように複数のタグが付加されると，2 つの語/タグの隣接関係の組み合わせが増加するためインデックスが膨大になる．また，新たにタグをする場合も同様に付近の語とタグに対する組み合わせに対してインデックスを更新しなければならずコストが高い．

そこで我々は，あるタグに対して固定的なテキストだけを連結したキーを持ち，左右の方向性を持たせたインデックス構造を用いる．図 5 に LR インデックスの構造を示す．LR インデックスでは，あるタグに対し，L リスト，R リスト，タグ文字という 3 つの情報を保持する．L リスト/R リストとは，そのタグのの左/右に登場した語のリストであり，それぞれ登場位置リストへの参照を保持する．この登場位置リストはあるタグとその左/右の語が隣接するパターンをキーとする登場位置のリストである．なお，以下の説明では，あるタグとその左/右の語に関する条件をそれぞれ A，B とするとき，この隣接関係に関する参照を「A →B」と表現し，A を一次キー，B を二次キーと呼ぶ．さらに，L リストに対する参照を左参照，R リストに対する参照を右参照と呼ぶ．タグ文字とは，あるタグに対し，そのタグが付加された文字列に含まれる文字のリストであり，メモリ上で管理される．例えば，[固

☆ ただし，この構文では，タグ値のみを条件としたクエリが記述可能である．これはあるタグ値が本基盤内のデータにおいて一意である場合，タグ名を省略できるものとしているためである．

有表現:国名] タグに対するタグ文字は「日本中国米アメ … 」になる.

LR インデックスの更新時には左右の語だけに注目してＬリストとＲリスト，タグ文字の３箇所を更新すれば良い．例えば，「ＮＥＣの田中氏は,,,」というテキスト内の「田中」に [固有表現:姓] というタグを付加する場合，「[固有表現:姓] →の」という左参照で得られる登場位置リストに新たにこの文書の登場位置を追加し，「[固有表現:姓] →氏」という右参照で得られる登場位置リストにこの登場位置を追加し，[固有表現:姓] というタグのタグ文字に「田中」という文字を追加すれば良い．

LR インデックスを用いた検索はこのインデックスの双方向性を利用して行われる．例えば，「[組織名] の [人名]」というクエリがある場合，「[組織名] →の」という右参照と「[人名] →の」という左参照を行うことで，検索を行うことができる．

また，「[形容詞][企業名]」といったタグが隣接しているクエリ（タグ隣接クエリと呼ぶこととする.）に関しては，次のように検索を行う．

タグ隣接クエリを「[A][B]」とするとき，その参照方法として，タグ右参照とタグ左参照を定義する．これらの参照はそれぞれ下記のアルゴリズムで行われる．

- タグ右参照
  - (1) 「A」の R リストを読み込み
  - (2) 読み込んだ「A」の R リストと「B」のタグ文字を比較し，「A」の右の語のうちタグ [B] が付加される可能性が無いものを削除.
  - (3) 残った「A」の R リストの先の登場位置リストを読み込み
  - (4) (3) で読み込んだ登場位置リストをマージ
- タグ左参照
  - (1) 「B」の L リストを読み込み
  - (2) 読み込んだ「B」の L リストと「A」のタグ文字を比較し，「B」の左の語のうちタグ [A] が付加される可能性が無いものを削除.
  - (3) 残った「B」の L リストの先の登場位置リストを読み込み
  - (4) (3) で読み込んだ登場位置リストをマージ

ただし，これらの参照でそれぞれ得られる登場位置リストは，厳密には「[A][B]」というパターンを含む登場位置リストではない．例えば，右側参照から得た登場位置リストは，タグ「A」と，その右にタグ「B」が付加される可能性のある文字列，が含まれる登場位置のリストである．このため「[A][B]」というクエリに対してはこのタグ右参照とタグ左参照を両方行うものとする．ただし，「[A][B] は」というクエリに対しては，「[B] →は」というタグと語による右参照があるため，「[A] → [B]」というタグ右参照のみがあれば良い．このように，LR インデックスではタグ隣接クエリに関しても左右のテキストに関する二次キーを用いて登場位置リストを絞り込みつつ参照することで相補的に検索対象を絞り込むことができる．

3.5 全体アーキテクチャ

図 6 にタグ付け用検索基盤のアーキテクチャを示す．タグ付け用検索基盤は主に順引きタグデータ，順引き文書データ，LR インデックス，単語インデックスの４つのデータを管理する．

順引きタグデータは文書番号，開始位置，終了位置を元にその範囲に付加されたタグデータを読み込むために利用される．我々はこの目的のために文書ごとにタグのデータだけを保持するハッシュテーブルを構築した．

順引き文書データは文書番号，開始位置，終了位置を元にその範囲の部分文字列を読み込むために利用される．本基盤ではテキストデータを文書単位で読み込むのではなく，文書中の任意の部分文字列だけを読み込むという用途が多い．このため我々は複数のテキストデータを一つのファイルにまとめ，あるテキスト中の任意の位置のみを読み込める API を持つ文書 DB を構築した．

LR インデックスはタグの登場位置を参照するために利用される．LR インデックスの構造は前節で説明したため割愛する．なお，本実装ではタグの右/左の語としてユニグラム (1 文字) を採用した．登場位置リストは「文書番号，開始位置，終了位置」というデータのリストを昇順で並べたものである．

単語インデックスは単語の登場位置を参照するために利用される．単語インデックスは通常の検索エンジンで作成される転置インデックスと同様のものである．なお，本実装では単語の区切り方法として形態素ごとではなく２グラムの区切りを採用した．これは形態素区切りを採用した場合，検索クエリ内の単語区切りと形態素の区切りが一致しないことがあるためである．また，単語転置インデックス内で参照される登場位置データは「文書番号，開始位置」というデータのリス

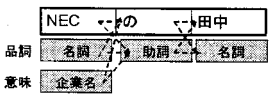

図 4 連結アプローチにおける２項の組み合わせ

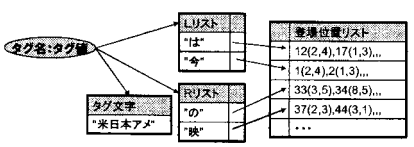

図 5 LR インデックス

トを昇順で並べたものである．

次に，検索アルゴリズムとタグの更新アルゴリズムに関してそれぞれ説明する．

3.5.1 更新アルゴリズム

タグの更新はタガープログラムあるいはユーザによって update 関数が呼び出されることで開始する．タグの更新手順を以下に示す．

(1) 文書番号，開始位置，終了位置を元に順引き用文書データを参照し，タグ付け対象の文字列，左の語，右の語を読み込む．
(2) タグ名，タグ値，右の語，左の語を元に LR インデックスを参照し，該当する登場位置リストを読み込む．
(3) 読み込んだ登場位置リストを前方から調べ，適切な位置に文書番号，開始位置，終了位置を挿入し，登場位置リストを保存する．
(4) タグ付け対象の文字列を元にタグ文字を更新する．

LR インデックスを更新するためはタグ付け対象の文字列，左の語，右の語を読み込む必要があるため，(1) の処理が必要になる．しかし通常，タグ付けを行うプログラムやユーザはタグ付け位置の文字列情報を保持していることが多い．そこで，本基盤では update 関数の引数に左右の語とタグが付加される文字列を加えた fupdate 関数を実装している．fupdate 関数は (1) の処理を省略し，(2) 以降の処理のみを行う．

3.5.2 検索アルゴリズム

検索は，タガープログラムあるいはユーザによって search 関数が呼び出されることで開始する．検索手順を以下に示す．

(1) 検索クエリの解釈を行い，インデックスへの問い合わせ方法を決定する．
(2) 各問い合わせを実行し，LR インデックスと単語インデックスから登場位置を読み込む．
(3) 読み込んだ登場位置リストを比較し，隣接関係を調べる．隣接関係がクエリと等しければその文書番号，開始位置，終了位置を出力に加える．

なお，検索クエリの解釈では次の３つの手順により問い合わせの集合を作成する．

(1) クエリ内で左右に単語が存在するタグに関して，LR インデックスへの右参照あるいは左参照の問い合わせを作成する．
(2) 残ったタグに関して，LR インデックスへのタグ右参照あるいはタグ左参照の問い合わせを作成する．
(3) クエリ内の単語に関して，(1) において二次キーとしての参照が作成されなかった 2 文字以上の単語に関して，単語インデックスへの問い合わせを追加する．

この処理により，最小限の問い合わせにより検索を行うことができる．

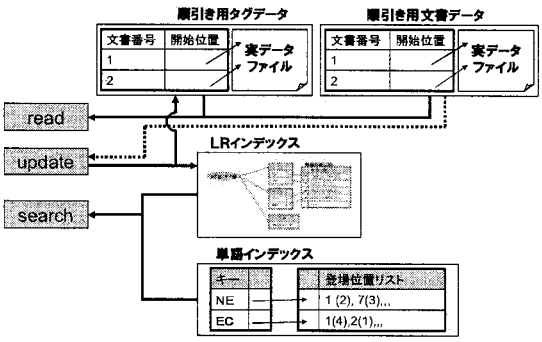

図 6 アーキテクチャ

4. 評価実験

本論文では，タグの更新性を保ったまま高い検索性能を実現するインデックス構造として LR インデックス構造を提案し，そのシステム実装を行った．そこで評価実験としては，インデックスのサイズ，検索速度，タグの更新速度，という３つの観点から従来の手法との比較評価を行う．

本実験では，比較対象の手法として単一アプローチの転置インデックスを用いる．しかし，この転置インデックスを実装するためにはタグの更新の効率も考えておく必要がある．あるタグをキーとした登場位置リストは昇順に並べておく必要があるため，新しく登場位置を追加するためには，そのタグに関する登場位置リストを前方から読み込み，適切な位置に挿入する必要がある．このため，単純に登場位置リストを作成すると，更新のたびにすべての登場位置リストを読み込まなければならず，更新の手間が大きい．

そこで我々は，Skipping[7] と呼ばれる手法を用いた．Skipping は登場位置リストを S 個ごとに分け，それぞれの部分に対しての参照を管理する方法である．このインデックス構造を図 7 に示す．この図では S = 1000 とし，千文書ごとに登場位置リストを分けた例を示している．Skipping を用いるとある登場位置を追加・削除するために，S 個の文書に関する登場位置リストだけを更新するだけで良い．しかし，この方法はパラメータ S によって検索性能と更新性能が大きく変化する．S を大きくすると，検索時の読み込みをシーケンシャルに行えるため速いが，更新時に多くのデータを更新する必要があり遅い．逆に S を小さくすると，検索時に分散した登場位置リストをそれぞれ読み込まなければならず遅いが，更新時にはデータの更新範囲が少なくて済むため速い．

そこで実験では，検索性を重視し S=10000 としたインデックス (Search インデックスとする) と更新性を重視し S=100 としたインデックス (Update インデックスとする) との 2 通りのインデックス構造を用いることとした．

実験データは次のように作成した．2007 年の 5 月

表 2 インデックス量

	LR	単一 S	単一 U
データ量	964MB	218MB	330MB

にブログを 25 万記事 (約 1GB) 収集し，形態素解析器 SEN を利用して品詞タグ（形容詞，固有名詞など）と固有表現タグ (姓，名，組織名，地名，国名) からなる 14 種類，200 万箇所のタグを付加した．以下，各評価結果について述べ，最後に実験のまとめを行う．

4.1 インデックス量

LR インデックス (LR)，Search インデックス (単一 S)，Update インデックス (単一 U) を用いて作成したインデックスのデータ量を表 2 に示す．タグに関するインデックスに注目すると，LR インデックスのデータサイズは従来のインデックスに比べ，約 3 倍〜4 倍になる．これには 3 つの理由がある．LR インデックスが左右方向に 2 重に登場位置リストを記憶するためと，文書リストが分散化されるためにディスクの使用効率が悪いためと，左右の語を記憶しなければならないためである．しかしながら，近年 HDD の大規模化が進んでおり，この程度のインデックス量の増加が大きな問題になることは少ないと言える．

4.2 検索速度

本研究で提案する LR インデックスは検索クエリ内のタグと語の位置関係に応じて検索速度が大きく変化する．このため我々は次のタイプ A からタイプ C の検索クエリを用いた．タイプ A はタグ間の隣接関係が存在しないクエリ (例：「[組織名] は [形容詞]」) である．タイプ B はタグ隣接クエリであるが，片側からの参照で良いクエリ (例：「[姓][名] 社長」) である．タイプ C はタグ隣接クエリであり，タグ右参照とタグ左参照の両方が必要な検索クエリ (例：「[姓][名]」など．) である．

表 3 に実験で用いた 13 個のクエリとそれぞれの検索における検索時間を示す．タイプ A のクエリに対しては LR インデックスは従来のインデックスに比べ，約 40 倍〜60 倍高速である．これは LR インデックスによる登場位置リストの読み込み量が大幅に削減した結果である．タイプ B のクエリに対しては，従来のインデックスに比べ，約 5 倍〜約 7 倍高速である．タイプ C のクエリに関しては従来の Update インデックスよりは速いものの，Search インデックスに対してほぼ同等の速度となっており，約 2〜3 秒の時間がかかっている．タイプ C のタグ隣接クエリに対しては，タグ文字によって読み込み量が削減できるものの，右/左の語を元に登場位置リストを読み込むため登場位置リスト全体をシーケンシャルに読み込めない点と，読み込んだ後のマージに時間がかかる点が速度低下の原因となっている．

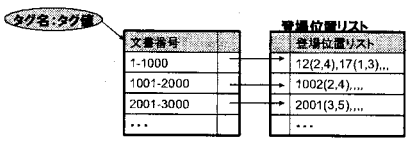

図 7 比較用のインデックス構造

4.3 更新速度

タグの更新に関しては実アプリケーションを想定したタグの追加例として，映画作品に関するタギングと企業名に関するタギングを行う．タギングは辞書ベースの方法を用いることとし，映画作品名の辞書として 2007 年度の公開映画作品名のリスト 151 個と，企業名の辞書として日経平均に用いられている 225 銘柄の名前のリスト 225 個を用意した．実験ではこの辞書を元に単語転置インデックスで登場位置を検索し，各文字列パターンが登場した位置をタグの登場位置とし，タグの追加を行った．ただし，このうち，LR インデックスへのタグの追記方法として，update 関数を用いた場合と fupdate 関数を用いた場合の 2 種類で計測した．これは左右の語とタグが付けられた文字列を読み込むため順引き用文書データにアクセスするコストがどの程度かを計測するためである．この結果を表 4 に示す．

この結果から，更新が最も高速であるのは Update インデックスである．Update インデックスは単一アプローチを取っており更新時に 1 箇所の更新で済む．これに対し LR インデックスは左右の語に関して登場位置リストを持つため 2 個所の更新が必要となる．よって fupdate 関数を用いた場合は，Update インデックスに対して約 2 倍の更新速度となっていることが確認できる．また，Search インデックスは単一アプローチを取っているものの，多くの登場位置リストを読み書きしなければならず，fupdate の約 5 倍の更新時間がかかっている．

また，update 関数と fupdate 関数を比較すると，update 関数の更新時間は fupdate 関数の約 2 倍であり，タギング時に左右の語とタグが付加される文字列を読み込む必要がある場合，約 2 倍の更新時間が必要であることがわかる．しかし，本基盤を用いてタギングを行うことを考えると，多くの場合タガー/ユーザは事前にこれらの文字列を検索済みであることが多い．よって通常はこれらの文字列をキャッシュに保持することで fupdate 関数の利用が可能であり，この点はそれほど大きな問題にならないと考えられる．

表 3 検索時間

タイプ	クエリ	ヒット数	LR[ms]	単一 S[ms]	単一 U[ms]
A	[組織名] が	4818	83	2497	3320
A	[組織名] は [形容詞]	375	96	2848	4821
A	[固有名詞] は [形容詞]	173	118	2963	5069
A	[組織名] と [組織名]	250	81	3057	5129
A	[組織名] の [姓] 社長	191	124	6089	8129
A	[組織名] の [姓][名] 教授	91	146	7460	10129
A	平均	-	108	4152	6100
B	[姓][名] 社長	379	1078	3307	4821
B	[国名] の [姓][名]	210	1229	6937	9015
B	[組織名] の [姓][名]	19	1142	7255	10267
B	[姓][名] が [姓][名] と結婚	17	1227	5562	7378
B	出演：[姓][名]	15	1143	4306	6001
B	平均	-	1164	5474	7496
C	[姓][名]	123078	2993	2524	3580
C	[形容詞][固有名詞]	1349	2346	2445	3985
C	平均	-	2670	2485	3783

表 4 更新時間

ドメイン	追加タグ数	update [s]	fupdate [s]	単一 S [s]	単一 U [s]
映画	6854	1458	799	3588	469
企業名	14580	3126	1506	7961	869

4.4 実験のまとめ

我々は本実験により下記の知見を得た.

- LR インデックスは従来のインデックスに比べ，3 倍〜4 倍のデータ量である.
- LR インデックスはタグと単語の組み合わせによる検索において従来のインデックスより数倍から数十倍高速であった.
- LR インデックスはタグの更新において従来の更新重視のインデックスの約 2 倍程度の更新時間でタグ情報を更新できる.

本実験で比較に使用した単一アプローチのインデックスは連結アプローチのインデックスに比べ更新が高速であり，単一アプローチによる更新重視のインデックスは最も更新が高速なデータ構造であるといえる. 本インデックスはその 2 倍程度の更新時間で，検索を数倍から数十倍高速化可能である. このことから我々は，LR インデックスはタギングのためのデータ管理システムにおいて有効であると考えている.

5. ま と め

本論文ではテキストマイニングシステムにおいて対話的なタギングを実現する環境を構築するため，タグ付け用検索基盤を提案した. さらに，このためにタグの更新速度を維持したまま検索速度を高速化する方法として，LR インデックス構造を提案し，1GB の文書データを対象として評価実験を行った. これにより，LR インデックスはデータ量が大きいものの，検索と更新の性能において従来の単純なインデックス構造を上回っていることを実証した. 今後は，より数百万記事単位の大規模なデータに対する検証実験を行った上で，本基盤上の API を利用した対話的テキストマイニング環境の開発を行う予定である.

参 考 文 献

1) 工藤拓, 松本裕治. Rdb を利用したタグ付きコーパス検索支援環境の構築. 情報処理学会 2001-NL-144, pp. 135–142, 2001.
2) 山口昌也, 田中牧郎. 多様な構造化テキストに対応した全文検索システム「ひまわり」. Proceedings of 日本語学会 2004 年度秋季大会研究発表会, pp. 144–145, 2005.
3) T.Shimizu and M.Yoshikawa. Full-text and structural indexing of xml documents on b+ tree. IEICE Trans on Info & Sys, Vol. E89-D, 2006.
4) N. Aswani, V. Tablan, K. Bontcheva, and H. Cunningham. Indexing and querying linguistic metadata and document content. Proceedings of RANLP2005, 2005.
5) MJ.Cafarella and O.Etzioni. A search engine for natural language applications. Proceedings of WWW2005, pp. 442–452, 2005.
6) Apache Lucene. http://lucene.apache.org/.
7) J. Zobel and A.Moffat. Inverted files for text search engines. ACM Computing Surveys, Vol. 38, No. 2, 2006.
8) H.Thompson and D.McKelvie. Hyperlink semantics for standoff markup of read-only documents. Proceedings of SGML97, pp. 227–229, 1997.