全文検索における複合語を考慮した転置リストの併合処理

数原　良彦† 　植松　幸生† 　安田　宜仁† 　井上　孝史† 　片岡　良治†

† 日本電信電話株式会社 NTT サイバーソリューション研究所　〒 239–0847 神奈川県横須賀市光の丘 1–1
E-mail: †{suhara.yoshihiko,uematsu.yukio,yasuda.n,inoue.takafumi,kataoka.ryoji}@lab.ntt.co.jp

あらまし　転置インデクスを用いた全文検索エンジンにおいて複合語の検索を行う際，クエリを構成する単語の転置リストの連接処理が必要となる．これらの処理に掛かる時間は，検索エンジンが行う処理時間の大半を占めており，効率的な併合演算は検索速度の改善につながる．連接処理には文書 ID の併合処理に加えて，単語の出現位置の照合処理が必要となる．この二段階の併合演算において，それぞれどのような手法を用いればよいかは明らかではない．本研究では転置リストが与えられた際に，適切な併合処理手法を選択する手法を提案し，連接処理の処理速度を改善を試みる．提案手法を含め，これら二段階の併合処理における手法の組み合わせを網羅的に比較し，連接処理における適切な組み合わせを調べる．実験では，実際の日本語コーパスを用いて実使用を想定した条件における処理時間の比較を行った．実験結果によって，文書 ID の併合処理において提案手法，単語の出現位置の照合処理に線形併合手法を用いた組み合わせによって最も高速に連接処理を行うことが示された．

キーワード　全文検索，転置インデクス，連接処理，併合演算

Intersection Algorithm in Inverted List for Phrase Query Processing

Yoshihiko SUHARA†, Yukio UEMATSU†, Norihito YASUDA†, Takafumi INOUE†, and Ryoji KATAOKA†

† NTT Cyber Solutions Laboratories, NTT Corporation　1–1 Hikarinooka, Yokosuka-shi, Kanaagwa 239–0847, Japan
E-mail: †{suhara.yoshihiko,uematsu.yukio,yasuda.n,inoue.takafumi,kataoka.ryoji}@lab.ntt.co.jp

Abstract　Searching phrase query using full-text search engine with inverted index, proximity processing for inverted lists of terms that compose the input phrase query is needed. The time cost for proximity processing takes over greater part of processing time of search engine. Therefore, efficient proximity processing will improve total searching speed. Proximity processing consists of DocID intersection and position list intersection. It is not known that what kind of intersection algorithm is effective for each of these two intersection processing. In this paper, we propose the method that select the appropriate intersection algorithm when the input lists are given. We conducted comparative experiments using Japanese corpus to analyze the effectiveness for each combination of fundamental algorithms and proposed method, The experimental results showed that the combination of proposed method as an intersection method for DocID and linear merge as an intersection method for position list can handle proximity processing faster.

Key words　full-text search, inverted index, proximity processing, intersection algorithm

1. はじめに

情報の爆発的な増加に伴い，ウェブをはじめ，あらゆる分野で検索システムが利用されている．中でも，転置インデクス (inverted index) と呼ばれる索引情報を用いることで高速に検索を行う全文検索システムが広く利用されている．

一般的な全文検索システムでは，ある単語が出現する文書の情報を転置インデクスに格納しておくことで，図書の索引と同様に入力された検索クエリが含まれる文書を高速に検索することができる．具体的には，各単語について当該単語が出現する文書識別子 (文書 ID; docID) リストを事前に構築し，検索時に検索クエリの単語に対応する文書 ID リストを取得することで全文検索を実現する．索引の単位となる単語のことを索引語 (index term)，文書 ID リストを転置リスト (inverted list) と

呼ぶ.

索引語の単位や転置リストの形式は様々なものがある [1]．日本語の場合，文書を形態素解析器を用いて，分割された形態素を索引語として用いる全文検索システムが多い.

全文検索システムにおいて「神奈川」と「横須賀」の 2 つのクエリを含む文書を検索する場合，2 つの索引語に対応する転置リストを走査し，双方に含まれる文書 ID を検索結果として取得する．このように，2 つの転置リストが与えられた際に，共通の文書 ID を取得する操作を併合処理と呼ぶ.

日本語で書かれた文書のインデクスを構築する場合，ひとつのクエリが複数の索引語に分割されることがある.「神奈川県」という複合語クエリが形態素解析によって「神奈川」と「県」という索引語に分割された場合，併合処理を行うだけでは，索引語が正しく隣接して出現していることを保証できないため，同一文書内に 2 つの索引語が離れて出現しているような文書も取得してしまい，検索精度が低下するおそれがある．複合語を正しく検索するために，索引語が文書のどの位置で出現したかということを転置インデクスに保持しておき，検索の際に出現位置を調べる必要がある．このように，索引語が同一文書において正しい位置関係で出現しているかを確認する処理を照合処理と呼び，複合語の検索における併合処理，照合処理を併せて連接処理 (proximity processing) と呼ぶ.

特に日本語を扱う全文検索システムにおいては，この連接処理が検索処理全体におけるコストの大半を占めており，連接処理の高速化は，検索速度を向上する上で最も重要な要因のひとつとなっている．本研究では，この連接処理を改善することで，全文検索システムの処理速度の高速化を目指す.

転置インデクスにおける併合演算手法については今まで数多く研究があるが [2] [3] [4] [5] [6] [7]，我々の知る限り，複合語を検索するため，位置情報を保持した転置インデクスにおいて文書 ID の併合処理と，位置情報の照合処理を含めた連接処理全体における各手法の組み合わせについて検証を行ったものはない．本研究では，併合処理と照合処理それぞれにおいて，適切な併合演算手法を選択する方法を提案し，日本語コーパスを対象に構築したインデクスを用いて，提案手法を含めた各手法の組み合わせについて検証を行う.

2. 全文検索における連接処理

本節では，位置情報を保持した転置インデクスと連接処理の具体的な方法について述べる．図 1 に，位置情報を保持した転置インデクスを用いる全文検索システムにおける連接処理の流れを示す.

位置情報を保持するインデクスは，各索引語について文書 ID と，文書内のどの位置に出現したかという情報を保持している．例えば図 1 において，東京という索引語に対応する転置リストは，東京が文書 1 の 3 番目，65 番目，69 番目に出現する情報を保持している.

次に複合語が検索クエリとして入力された際の連接処理の流れを図 1 に従って述べる．まず，ユーザによって入力された検索クエリを形態素解析によって索引語に分割する．ここでは

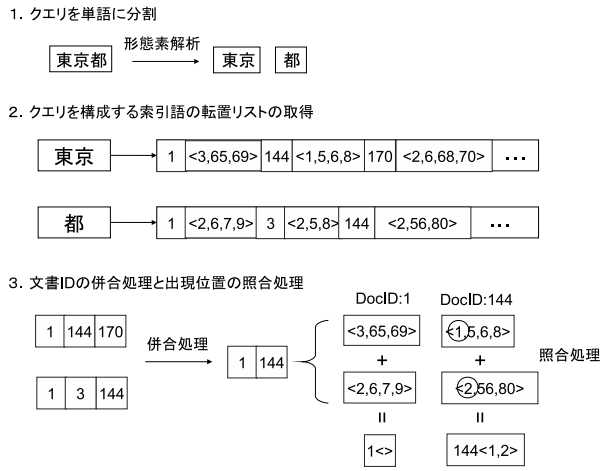

図 1　位置情報を保持した転置インデクスにおける連接処理

「東京都」という検索クエリが形態素解析によって「東京」と「都」の 2 つの索引語に分割されている．次にそれぞれの索引語に対応する転置リストを取得し，まず文書 ID について併合処理を行う．例では，2 つの転置リストに文書 1 と文書 144 が共通に存在する．共通の文書 ID が見つかると次に索引語が正しい位置関係で出現しているかを確認する．この操作を照合処理と呼ぶ．例では「東京」の次に出現する「都」を検索したいため，照合処理では同一文書における「都」の出現位置が「東京」の出現位置に 1 を加えたものを探索する．例における文書 1 ではこのような位置情報が見つからないが，文書 144 において位置 1 と位置 2 が見つかり，文書 144 の先頭で「東京都」が出現していることがわかる.

3 つ以上の索引語の場合，先に短い転置リスト同士の連接処理を行い，得られた結果に対して残りの転置リストとの連接処理を行うことで最終的な結果を取得するという戦略が一般的である [2].

このように，照合処理では出現位置の差を考慮する必要があるが，与えられた索引語対の出現位置の差は一定であるため，照合処理は基本的に併合処理と同じ演算を行う．本稿では，併合処理と照合処理を併せて併合演算，併合演算を行うアルゴリズムを併合演算手法と呼ぶ.

2.1 併合演算手法

次に，転置リストの併合演算手法について詳しく述べる．本研究で扱う転置リストは，文書 ID，位置情報が昇順にソートされている一般的な実装を想定している．そのような転置リストにおける基本的な併合演算手法は大きく 2 種類に分けることができる [2].

（1） 線形併合に基づく手法

（2） 二分探索に基づく手法

線形併合 (linear merge) に基づく手法は，マージソートと同様に 2 つの転置リストを先頭から比較していき，同じものが見つかった場合に併合を行うというもので，2 つのリストを先頭から最後まで走査する．線形併合に基づく手法には様々な発展手法があるが，本研究では基本的な線形併合手法を用いる.

二分探索に基づく手法では，線形併合に基づく手法と異なり，

短い方のリストの各要素について，長いリストに含まれているかを二分探索によって探索するという方法で併合演算を実現する．2 つの転置リストの長さの差が大きい場合，二分探索に基づく手法の方が，線形併合に基づく手法に比べ高速に処理が可能である．二分探索に基づく手法についても，いくつか発展手法があるが，本研究では基本的な二分探索手法を用いる．

3. 提案手法

本研究では併合処理と照合処理において，与えられたリストの長さを元に，適切な併合演算手法を選択することで連接処理全体を高速に行う手法を提案する．

2.1 で述べたように，最適な併合演算手法は，処理対象の 2 つのリスト長に依存している．しかしながら，連接処理を行うために処理するリスト組の長さの比率は分散が大きいことが知られており [6]，ひとつの固定された手法では，その手法が苦手とするリスト組を処理することで，処理時間が長くなることが予想される．

提案手法では，併合処理，照合処理毎に与えられた 2 つのリスト長をもとに，計算量に基づいた選択基準によって最適な手法を選択し，選択された手法を用いて併合処理，照合処理を行うことで，高速な連接処理を実現する．本研究においては，選択対象を線形併合手法と二分探索手法に限定し，選択された手法を用いて併合演算を行う手法を提案手法とする．

提案手法の選択基準について詳しく述べる．併合を行う転置リストの長さを N, M $(M < N)$ とすると，計算量は，線形併合手法が $O(M+N)$，二分探索手法が $O(M \log_2 N)$ となり，以下の式において等号が成り立つ際に 2 つの手法の計算量が等しくなる．

$$M + N = M \log_2 N \tag{1}$$

式 (1) より，N を入力として，線形併合に基づく手法が二分探索に基づく手法に対して有効に働く分岐点である M の値を返す関数 $f_M(N)$ を求めることができる．

$$f_M(N) = N(\log_2 N - 1) \tag{2}$$

$f_M(N) > M$ の場合，二分探索手法が有効であり，$f_M(N) < M$ の場合，線形併合手法が有効となる．したがって，式 (2) を元に，2 つの転置リストの長さ M, N が与えられた際に，適切な手法を選択することができる．

式 (2) では，対数の計算を行う必要がある．この計算コストを減らすために，$f_M(N)$ の直線近似を試みる．$N = \{10, 20, \ldots, 1000\}$ について M の値を計算し，プロットした点について最小二乗法に基づいて直線近似を行ったところ，相関係数 0.997 で $M = 0.105N + 8.600$ が得られた．これを $f'_M(N)$ とする．$f_M(N)$ と $f'_M(N)$ を図 2 に示す．提案手法では，併合演算が行われる毎に $f'_M(N)$ を計算し，実際の M と比較を行うことでどちらの併合演算手法を用いるかを決定する．例えば，文書 ID の併合処理において $N = 100, M = 20$ の長さのリスト組の併合演算を行う場合，$f'_M(100) = 10.5 + 8.6 = 19.1$ より，$f'_M(100) < M$ となり，線形併合手法を選択する．

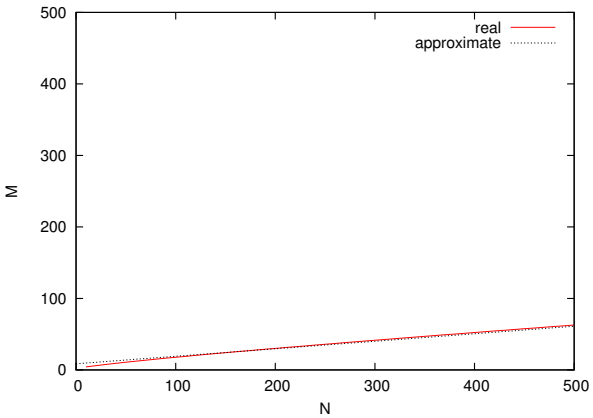

図 2　各転置リスト長における適切な手法の判別

表 1　構築した転置インデクスの文書数とインデクスサイズ

Corpus size	document number	index size (MB)
10%	946,892	673
20%	1,817,047	1,428
30%	2,842,649	1,910
40%	4,187,561	2,756
50%	5,329,753	3,447
60%	6,519,843	4,185
70%	7,661,855	4,854
80%	8,789,761	5,547
90%	9,900,361	6,200
ALL	10,865,904	6,773

4. 実　　験

連接処理における提案手法の有効性を検証するため実験を行った．

4.1 実験条件

本実験には，オープンソースの日本語全文検索エンジン FT-Search (注1)を元に，転置リストのデータ構造と連接処理部分の併合演算手法を変更した検索エンジンを用いた．形態素解析器には ChaSen [9] を用いた．

検索対象として NTCIR-4 Web などのタスクで用いられた NW100G-01 ウェブコーパス (注2) を用いた．これは，主に JP ドメインの Web 上で提供される文書を収集したもので，約 1,100 万の Web ページで構成されている．NW100G-01 のうち，HTML タグを除去したサイズが 20KByte 未満のページを対象にインデクスの構築を行った．コーパスサイズによる比較を行うため，NW100G-01 データセットを 10 分割し，全体の約 10%から 100%のコーパスサイズについて合計 10 個の転置インデクスを構築した．分割は NW100G-01 ウェブコーパスの各文書に付与された ID に基づいて行った．構築した転置インデクスサイズとインデクスした文書数について表 1 に示す．

連接処理の時間を計測するために使用した検索クエリは，NTCIR Workshop NTCIR-4 WEB [8] の Navi 1 と Info 1 タ

(注1)：http://www.ingrid.org/~harada/interface/index.html
(注2)：http://research.nii.ac.jp/ntcir/permission/ntcir-4/perm-ja-WEB.html

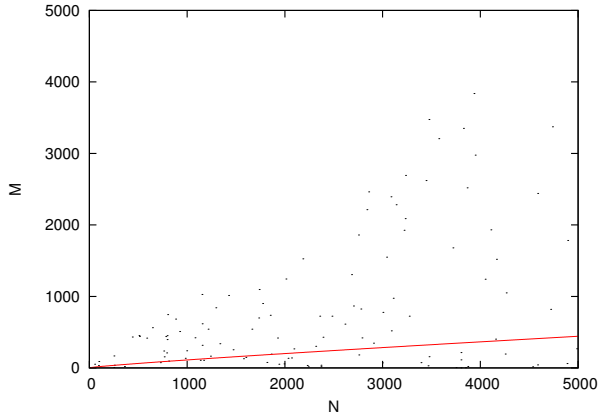

図 3　文書 ID の併合処理における 2 つのリスト長の分布 (10%)

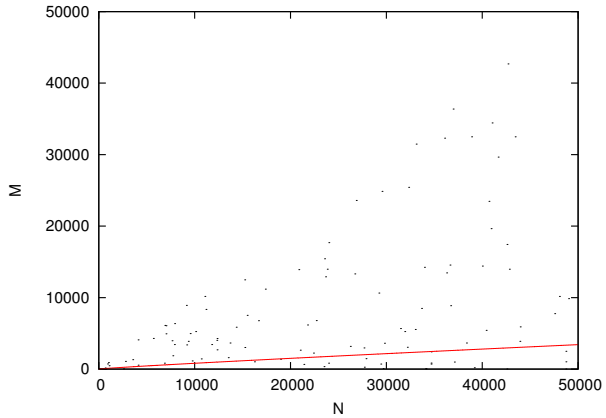

図 4　文書 ID の併合処理における 2 つのリスト長の分布 (ALL)

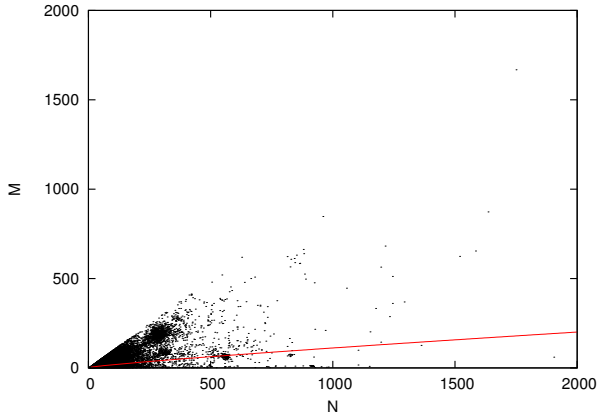

図 5　位置情報の照合処理における 2 つのリスト長の分布 (10%)

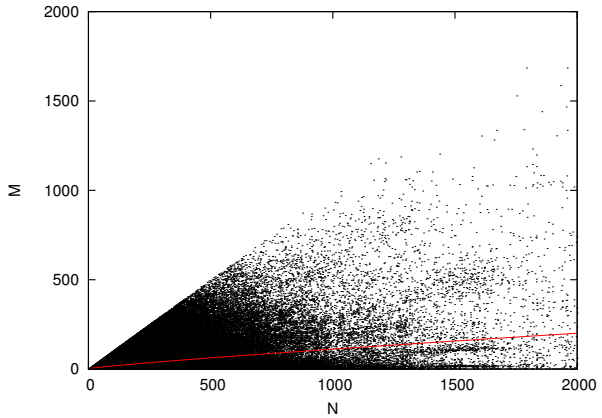

図 6　位置情報の照合処理における 2 つのリスト長の分布 (ALL)

スクで用いられたクエリ群を使用した．入力クエリが複数の形態素に分割される場合に連接処理が行われるため，クエリ群の中から ChaSen によって複数の形態素に分割される 387 個のクエリを選択した．

本実験は Intel Xeon X7350 2.93GHz CPU (128GB memory) Cent OS 5.2 の環境で行った．

4.2　予備実験

各手法の組み合わせにおける連接処理時間の比較を行う前に，構築したインデクスについて予備実験を行った．

4.2.1　併合演算を行うリスト長の分析

併合演算を行う 2 つのリスト長の分析を行う．図 3, 4 に 10%インデクス，ALL インデクスを対象に実験で用いたクエリを検索する際に行われる併合処理における文書 ID リスト長のプロット，図 5, 6 に同様に照合処理における位置情報リスト長をプロットしたものを示す．それぞれ長い方のリスト長を N，もう一方を M としている．両方の図に描かれている曲線は，$f_M(N)$ である．両図において，$f_M(N)$ の下側にあるプロットは，二分探索が有効になるリスト組であることを表しており，上側にあるプロットでは線形併合が有効になる．　図 3 と図 4 を比較すると，コーパスサイズに応じてスケールが変化しているが，分布に大きな違いが見られないことがわかる．コーパスサイズが大きくなるにつれて，転置リストに含まれる文書 ID も同じように増加するためだと考えられる．

同様に照合処理を行う位置情報リスト長の分布について分析を行う．図 5 と図 6 を眺めると，コーパスサイズによって分布が大きく変化していることがわかる．これらの図における位置情報リスト長は，当該索引語 (今回は形態素) が，併合された文書内にいくつ出現するかを表しているため，図 6 はコーパスサイズが増えるにつれて，クエリを構成する形態素をより多く含む文書が追加されたこと意味している．新しく追加された文書が含むクエリの単語頻度は既にインデクスされた文書の単語頻度と同じ分布をしていると予想していたため，この結果は直感に反する．ひとつの可能性としては，今回コーパスサイズの比較のためにデータセットを 10 分割したが，NW100G-01 データセットにあらかじめ付与されていた ID に基づいて分割を行ったため，クエリを構成する形態素を多く含むような文書がある部分セットに多く偏っていたことが考えられる．

10%インデクスの分析で 966,432 件行われた照合処理のうち，238,229 件 (24.7%) は，照合処理を行う 2 つの位置情報リストの長さが同じであった．位置情報リストが同じであることから，入力された複合語を構成する形態素の転置リストにおいて，複合語としてのみ出現していると推測できる．この値は，ALL インデクスを用いた場合，21,007,631 件中 2,321,455 件 (11.1%) となり，コーパスサイズが大きくなるにつれて，複合語以外に形態素が出現する回数が多くなるという直感に一致する結果が得られた．なお，文書 ID の併合処理において同一の文書 ID

リスト長を持つ組は 10%インデクスにおいて 445 件中 1 件で，ALL インデクスにおいて 482 件中 1 件であった．

4.2.2 連接処理における有効手法の分析

表 2 に各コーパスサイズの併合処理と照合処理において与えられたリスト長をもとに，式 (2) で示した f_M によって線形併合手法 (linear) または二分探索手法 (binary) が適切であると選択された件数を示す．これは図 3, 4, 5, 6 において $f'_M(N)$ の上側，下側にあるプロットの数と等しい．

表 2 より，20%インデクスの照合処理の結果を除き，有効手法の分布は全てのコーパスサイズで同じ傾向を示していることがわかる．有効手法は，コーパスサイズによらずほぼ同じ比率となっていることから，手法を固定することで約半数の併合演算においてもう一方の手法を選択した場合よりもコストの高い処理を行うことになる．これより提案手法が有効に働くことが推測できる．

4.3 連接処理時間の計測

併合演算手法の組み合わせにおける連接処理の処理時間の比較を行った．併合演算手法としては，提案手法 (proposed)，線形併合手法 (linear)，二分探索手法 (binary) について全ての組み合わせについて実験を行った．構築された 10 個のインデクスを対象に，4.1 で述べた 387 個のクエリの検索を行い，連接処理にかかる時間の測定を行った．時間の測定には gettimeofday(2) を用いた．各組み合わせによる検索を 10 回試行し，連接処理時間の平均を求めた．

実験結果を表 3 に示す．表 3 において linear-binary は文書 ID の併合処理に線形併合手法を，位置情報の照合処理には二分探索手法を用いる組み合わせであることを示している．結果より，全てのコーパスサイズにおいて proposed-linear が最も高速に連接処理を行うことがわかる．以下，結果の詳細と考察を行う．

位置情報の照合処理に用いる手法を固定した際の結果を比較する．表 3 より，照合処理にいずれの手法を用いた場合においても併合処理に proposed を用いる組み合わせが最も処理時間が短い．これより，文書 ID の併合処理には proposed が有効であることが示された．また，併合処理においては linear と binary の差がほとんどないことがわかる．

次に文書 ID の併合処理に用いる手法を固定した結果を比較する．表 3 より，併合処理にいずれの手法を用いた場合においても linear が最も短い処理時間を示している．文書 ID の併合処理と異なり，照合処理手法については proposed ではなく linear によって高速に処理されることがわかる．また，照合処理においては linear に比べ binary が最も処理時間が長い．これは binary が不得意とするようなリスト長比の照合処理を行うことでコストが増加したためだと考えられる．表 2 を見ると，照合処理において binary が有効の頻度は linear とほぼ同じであるため，処理時間も同じ程度だと予想できる．しかし，図 5 を眺めると，$N = 100$ 程度まで $N = M$ 付近にも分布し，また $N = 300, M = 200$ あたりにも部分的な分布が確認できる．このように binary ではコストが大きくなるリスト組が多いことがうかがえる．したがって，binary が有効となる頻度は同程度であるが，binary が有効ではない場合において，コストが相対的に大きくなるため，照合処理においては linear の方が平均的に高速に処理できることがわかる．

照合処理において proposed が最良の結果を得られなかった理由について考察を行う．文献 [5] で述べられているように，提案手法では併合演算の度に手法の選択を行うため，リスト長に基づく判断値を計算するコストと条件分岐のコストがかかってしまう．文書 ID の併合処理に比べ，位置情報の照合処理の回数は膨大であり，手法を選択するコストに比べ，照合処理手法を選択することで低減できるコストが小さいためだと考えられる．これにより，毎回手法を選択するのではなく，不適切な手法を選択するとコストが大きくなる場合にのみ手法の選択を行うような戦略によって更に高速な連接処理が可能になると推測される．

5. 関 連 研 究

転置リストの高速な併合演算手法については古くから盛んに研究されている．線形併合に基づく手法では，スキップリストと呼ばれるデータ構造を用いることで高速に線形併合が可能となることが知られている [2] [7]．Baeza-Yates は [4] は，分割統治法に基づいて二分探索手法を拡張した Baeza-Yates アルゴリズムを提案した．Baeza-Yates ら [5] は，各手法が有効になる分岐点を実験的に求め，併合処理毎にリスト長に基づいて適切な併合演算手法を選択するハイブリッド手法を提案している．我々の提案手法では計算量に基づいた選択基準を設けている点で異なる．Sanders ら [6] は，ランダムに文書 ID が付与されたインデクスを構築することで高速に処理可能な lookup アルゴリズムを提案している．また，従来手法と比較実験を行い，あらゆるリスト長比において lookup アルゴリズムが高速に処理可能であることを示している．

6. お わ り に

本稿では，併合演算手法の計算量に基づく選択基準を設けることで，併合演算対象のリスト長をもとに併合演算手法を選択し，高速に併合演算を実現する手法を提案した．実際の日本語コーパスを対象に異なるサイズのインデクスを構築し，連接処理を必要とする全文検索システムにおける併合処理と照合処理において，線形併合，二分探索，提案手法の組み合わせの連接処理時間の比較実験を行った．

実験結果より，文書 ID の併合処理において提案手法が最も高速に処理することが確認され，文書 ID の併合処理に提案手法，位置情報の照合処理に線形併合手法を組み合わせたものが最も高速に連接処理を行うことが確認された．

複合語クエリが入力された際に併合処理を行う文書 ID リスト組の長さの分布，照合処理を行う位置情報リスト組の長さの分布の分析を行った．併合演算を行うリスト長の分析を通じて，コーパスサイズを増加することで位置情報リスト長の分布が変化したことがわかった．原因としてクエリを多く含むような文書の偏りが考えられるため，データセットをランダムに分割した場合の分析を行う必要がある．

表 2　併合処理と照合処理における有効手法の分布

	有効手法 (併合処理)		有効手法 (照合処理)	
corpus size	linear	binary	linear	binary
10%	231 (49.7%)	234 (50.3%)	1,027,875 (53.1%)	908,979 (46.9%)
20%	240 (51.3%)	228 (48.7%)	1,716,511 (62.1%)	1,046,328 (37.9%)
30%	240 (50.6%)	234 (49.4%)	3,002,875 (52.3%)	2,737,987 (47.7%)
40%	243 (50.8%)	235 (49.2%)	4,375,055 (52.6%)	3,943,213 (47.4%)
50%	247 (51.7%)	231 (48.3%)	5,509,644 (52.6%)	4,972,789 (47.4%)
60%	246 (51.1%)	235 (48.9%)	6,730,307 (52.5%)	6,095,202 (47.5%)
70%	247 (51.4%)	234 (48.6%)	7,863,989 (52.5%)	7,111,051 (47.5%)
80%	248 (51.5%)	234 (48.5%)	8,999,972 (52.5%)	8,145,940 (47.5%)
90%	245 (50.8%)	237 (49.2%)	10,001,823 (52.2%)	9,142,836 (47.8%)
ALL	246 (51.0%)	236 (49.0%)	10,985,133 (52.3%)	10,022,498 (47.7%)

表 3　コーパスサイズ毎の各組み合わせの処理時間の結果

	Processing time [micro sec.]									
method	10%	20%	30%	40%	50%	60%	70%	80%	90%	ALL
linear-linear	1,377	2,078	4,318	7,007	8,036	9,970	12,514	13,903	16,242	17,803
linear-binary	1,803	2,226	6,055	8,143	11,185	14,569	15,357	16,205	19,925	21,173
linear-proposed	1,554	2,468	4,698	7,388	9,161	11,302	13,033	15,794	17,297	19,549
binary-linear	1,306	1,704	5,333	7,701	9,029	11,750	14,521	15,853	18,018	20,229
binary-binary	2,036	1,974	6,809	8,132	11,154	15,406	17,020	19,976	22,490	24,141
binary-proposed	1,390	2,522	5,678	8,194	9,962	11,932	16,035	17,644	19,834	21,591
proposed-linear	**1,033**	**1,231**	**3,070**	**5,239**	**6,233**	**7,558**	**8,903**	**9,661**	**12,240**	**12,243**
proposed-binary	1,560	1,677	4,529	6,654	8,084	11,825	12,773	13,798	15,911	16,625
proposed-proposed	1,052	1,375	3,770	5,501	7,428	9,057	10,141	11,272	12,919	13,889

本研究では組み合わせに基本的な併合演算手法のみを用いたが，関連研究で紹介した発展手法についても同様に実験を行う必要がある．

選択基準の計算コストを下げることができれば，提案手法を更に高速化することができると考えられるため，今後の課題としては，4.3 で述べたように不適切な手法を選択した際に大きなコストがかかるような場合のみ併合演算手法を選択する戦略について検討したい．

文　　献

[1] C. D. Manning, P. Raghavan and H. Schütze. Introduction to information retrieval. Cambridge University Press, 2008.

[2] A. Moffat and J. Zobel. Self-indexing inverted files for fast text retrieval. ACM Transactions on Information Systems, Vol.14(4), pp.349–379, 1996.

[3] E. D. Demaine, A. López-Ortiz and J. I. Munro. Adaptive set intersections, unions, and differences. In Proc. the 11th annual ACM-SIAM symposium on Discrete algorithms (SODA), pp.743–752, 2000.

[4] R. Baeza-Yates. A fast set intersection algorithm for sorted sequences. In Proc. the 15th Annual Symposium on Combinatorial Pattern Matching (CPM), pp.400–408, 2004.

[5] R. Baeza-Yates and A. Salinger. Experimental analysis of a fast intersection algorithm for sorted sequences. In Proc. 12th String Processing and Information Retrieval Symposium (SPIRE), pp.13–24, 2005.

[6] P. Sanders and F. Transier. Intersection in integer inverted indices. In Proc. 7th Workshop on Algorithm Engineering and Experiments (ALENEX), pp.71–83, 2007.

[7] T. Strohman and W. B. Croft. Efficient document retrieval in main memory. In Proc. 30th annual international ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR), pp.175–182, 2007.

[8] K. Eguchi, K. Oyama, A. Aizawa, and Haruko Ishikawa. Overview of the informational retrieval task at NTCIR-4 WEB, In Proc. 4th NTCIR Workshop on Research in Information Access Technologies Information Retrieval, 2004.

[9] 松本裕治, 北内啓, 山下達雄, 平野善隆, 松田寛, 高岡一馬, 浅原正幸 日本語形態素解析システム『茶筌』 version 2.2.1 使用説明書, 2000.