N-gram索引における複合検索条件の効率的な処理方法

小川泰嗣[†] 松田 透[††] 橋本信次[†††]

N-gram索引のための単一検索語の効率的な処理方法として冗長n-gram法が提案されている。これは検索処理を、検索語から抽出される複数のn-gramを含む文書を検索する候補文書特定と、n-gramが文書中で連続位置にあるか調べる位置検査の2段階に分離し、位置検査をできる限り省略することで検索を高速化するものである。本論文では、位置検査の省略という考えをAND, OR, ANDNOT演算子に対して拡張し、複合検索条件処理を高速化する。さらに、AND, OR演算子が入れ子になっている場合には、子ノード数に応じてOR標準形に変換することで検索処理を高速化する。新聞記事5年分を用いた評価により、これら手法の有効性が確認できた。

Efficient evaluation method of complex queries in n-gram indexing

OGAWA YASUSHI ,[†] MATSUDA TORU [††] and HASHIMOTO SHINJI [†††]

In the redundant n-gram method proposed for query processing in n-gram indexing, retrieval is accelerated by dividing the processing into two steps — one is finding the potential documents that have all the n-grams in the query word, and the other is checking the proximity constraints among the n-grams — and by reducing the unnecessary proximity check. This paper extends this method to complex queries with AND, OR, ANDNOT operators. In addition, we selectively convert complex queries with both AND and OR operators to the OR normal form according to the number of child nodes in these operators. The results of experiments using five years of newspapers showed that the extensions worked quite well.

1. はじめに

文書電子化とコンピュータネットワークの進展に伴い、蓄積された大量の文書群から所望の文書を見つけ出す文書検索技術の重要性が高まっている。特に、最近ではインターネットのサーチエンジンや電子図書館に見られるように検索対象が大規模になっているので、検索速度の向上が強く求められている。

高速検索の実現には、索引の利用が不可欠である[4)17)]。英語など多くの言語においては、単語を索引付けの単位として索引を作成することが一般的である。しかし、日本語文書を対象とする場合、日本語ではスペースなどによって単語の切れ目が明示的に示されないことが問題となる[11)12)]。そこで、n-gram（n文字組）を索引単位とするn-gram索引[3)14)]が広く使用されている[11)]。

しかし、n-gram索引にも問題点がある。検索速度の観点からは、単語索引であれば単一の索引単位として処理される検索語が複数のn-gramに分割・処理されることが問題となる。すなわち、nより長い検索語の処理では、分割の結果生じたn-gramが文書中で検索語を構成していることを確認する必要がある。出現文書ごとのn-gramの出現位置（先頭からのオフセット）を索引に記録しておき、検索時にその出現位置を突き合わせることで検索語の存在を確認できるが[1)9)]、この位置検査処理が検索時間が増大させる☆。

N-gram索引を用いた検索の高速化手法としては、文字種等に応じてnを調整する[1)8)18)]、圧縮を施すなど索引構造を工夫する[1)10)]、n-gramの処理順序や選択方法を工夫する[9)13)]などが提案されている。しかし、これらの提案は基本的に単一の検索語の処理を高速化するものである。ところが、実際の検索場面では、複数の検索語をAND, OR, ANDNOT等の演算子で組み合わせた検索条件（以下、複合条件）が使用

† （株）リコー ソフトウェア研究所
Software Research Center, Ricoh Co., Ltd.
†† （株）リコー オフィスシステム開発センター
Office System Development Center, Ricoh Co., Ltd.
††† リコーシステム開発（株）
Ricoh System Kaihatsu Co., Ltd.

☆ 日本語には複合語が容易に作成されるので、構成語を索引単位とした場合、単語索引でも位置検査が必要になる。これを回避するためには、登録時に全ての可能な限りの複合語を索引に登録する必要がある[5)]。

されることが多いにもかかわらず、筆者の知る限り複合条件の高速化を正面から扱った研究はない。もちろん、単一の検索語の処理が高速化されれば複合条件の処理も高速化される[10]。しかし、複合条件にあわせた最適化を行なうことができれば、一層の高速化が期待できるはずである。

本論文では、冗長 n-gram 法[13]という単一検索語のための高速化手法を複合条件に拡張することで、複合条件の検索処理を高速化する。冗長 n-gram 法の基本的な考え方は、位置検査処理コストを削減することで検索を高速化するというものである。複合条件では、条件に含まれる検索語に対する検索結果から最終的な検索結果が生成されるので、検索語の処理順序に応じて位置検査を行わなくても良い場合がある。本論文では、そのような無駄な位置検査を行わないことで検索処理を効率化する方法を提案する。さらに、検索場面で使用されることの多い AND 演算子のなかに OR 演算子が入れ子になった検索条件（混合条件）の効率的な処理についても検討する。

本論文の構成は以下の通りである。つぎの章では n-gram 索引の登録・検索の基本的な処理方法、冗長 n-gram 法とその改良について述べる。3 章では、位置検査の削減という考えを様々な演算子に適用する方法を提案し、4 章では、混合条件処理の効率化について検討する。5 章では、新聞記事を用いた評価結果および考察を示す。最後の 6 章はまとめである。

2. N-gram 索引

2.1 登録処理

文書登録時には、対象文書から部分的に重複するものも含めて、全ての n-gram を抽出する。ただし、文書の末尾部分にある n 文字未満の単語を検索可能とするため、末尾部分からは n 文字未満の n-gram も抽出する[18]。例えば、bi-gram（$n=2$ の n-gram を bi-gram と呼ぶ）索引では、「携帯電話」という文書から「携帯」「帯電」「電話」「話」を抽出する。最後に「話」を 1 文字で抽出するのが末尾処理である。

2.2 検索処理

文書検索時には、検索語の長さ（以下 m 文字とする）に応じて処理方法が異なる。

- $m > n$ の場合
  検索語が索引単位の n-gram より長い場合に相当する。検索語から長さ n の n-gram（$m-n+1$ 個）を抽出し、それら全てを含む文書を特定する。しかし、n-gram がバラバラに出現していたのでは検索語を含むことにならない。例えば、「携帯式電話機の帯電」という文書は、「携帯電話」から抽出される「携帯」「帯電」「電話」を全て含んでいるが、「携帯電話」自体は含んでいない。そこで、全ての n-gram を含む文書において n-gram が連続した位置にあることを検査する必要がある。
- $m = n$ の場合
  検索語と索引単位の n-gram が等しい長さの場合に相当する。検索語と等しい n-gram が索引に登録されていれば、その n-gram が出現していた文書が検索結果となる。
- $m < n$ の場合
  検索語が索引単位より短い場合に相当する。索引に登録されている n-gram で、先頭 m 文字が検索語に一致するもののいずれかを含む文書が検索結果となる。そこで、そのような n-gram の全てを OR 演算子で結合したものを検索条件として、検索を行なう。例えば、検索語が「話」であれば OR(話, 話あ, $\cdots$, 話遥) が検索条件となる。

N-gram 索引では、n の値が検索性能に影響する。n が大きくなると、索引サイズは大きくなり、登録時間も増加する。一方、検索時間への影響は、検索語の長さ m に応じて異なる。検索語が n より長い場合、n が大きいほど検索語の分割数が少なくなるので検索時間は短縮されるが、検索語が n より短い場合、n が小さいほど検索語を展開する n-gram 数が少なくなるので検索時間は短縮される。これら影響のバランスを考え、$n=1,2$ が使用されることが多い[1)6)10)]。

2.3 冗長 n-gram 法

N より長い検索語処理では、前節で述べたように、n-gram が文書中で検索語を構成しているか検査する必要がある。そのためには、登録時に n-gram の文書中での出現位置を索引に記録し、検索時に n-gram の出現位置の突き合わせを行えばよい[1)9)15)]。なお、この場合、索引ファイル形式として出現位置を記録可能な転置ファイルを用いる必要がある[11)]。

位置検査処理のコストを削減することで検索処理を高速化する手法として冗長 n-gram 法が提案されている[13]。冗長 n-gram 法では、検索処理を、検索語から抽出された n-gram を全て含む文書を特定する候補文書特定と、候補文書において n-gram が連続した位置に出現し検索語を構成するか調べる位置検査の 2 段階に分けて処理する。そして、2 つの処理において使用する n-gram 群を以下のように使い分けることで、位置検索のコストを減らし、検索を高速化する。

- 候補文書を少なくするため、候補文書特定では、検索語から抽出される全 n-gram を使用する。

表 1 単一検索語に対する検索時間 (cold start)
Table 1 Response time of single word queries(cold start)

文字数	従来	冗長 n-gram	改良冗長 n-gram
1	1.274	1.272 (−0.2%)	1.274 (±0.0%)
2	0.087	0.086 (−1.1%)	0.086 (−1.1%)
3	0.507	0.507 (±0.0%)	0.508 (+0.2%)
4	0.480	0.492 (+2.5%)	0.486 (+1.3%)
5	0.636	0.573 (−9.9%)	0.539 (−15.3%)
6	0.632	0.684 (+8.2%)	0.657 (+4.0%)
7	0.745	0.636 (−14.6%)	0.593 (−20.4%)
8	0.699	0.656 (−6.2%)	0.639 (−8.6%)
9	0.661	0.643 (−2.7%)	0.567 (−14.2%)
10	0.655	0.678 (+3.5%)	0.634 (−3.3%)
(平均)	0.638	0.623 (−2.4%)	0.598 (−6.3%)
(平均′)	0.644	0.623 (−3.3%)	0.588 (−8.7%)

表 2 単一検索語に対する検索時間 (warm start)
Table 2 Response time of single word queries(warm start)

文字数	従来	冗長 n-gram	改良冗長 n-gram
1	0.595	0.595 (±0.0%)	0.596 (+0.2%)
2	0.013	0.013 (±0.0%)	0.013 (±0.0%)
3	0.120	0.119 (−0.8%)	0.119 (−0.8%)
4	0.087	0.062 (−28.7%)	0.067 (−23.0%)
5	0.078	0.051 (−34.6%)	0.048 (−38.5%)
6	0.079	0.050 (−36.7%)	0.048 (−39.2%)
7	0.062	0.028 (−54.8%)	0.026 (−58.1%)
8	0.061	0.021 (−65.6%)	0.022 (−63.9%)
9	0.034	0.021 (−38.2%)	0.018 (−47.1%)
10	0.044	0.020 (−54.5%)	0.018 (−59.1%)
(平均)	0.117	0.098 (−16.2%)	0.098 (−16.2%)
(平均′)	0.063	0.036 (−42.9%)	0.035 (−44.4%)

- 個々の位置検査のコストを削減するため、位置検査では、検索語を被覆し、かつ各 n-gram の文書頻度（n-gram を含む文書数）の和が最小となる n-gram 群を使用する。

なお、検索処理手順には、文書順（document-at-a-time）と索引単位順（term-at-a-time）の二つがある[2)7)16)]。文書順では、一つずつ文書を取り上げ、その文書が検索語を含んでいるか検査して、検索結果を決定していく。これに対し、索引単位順では、まず最初の索引単位に該当する文書集合を中間結果とし、それ以降は残りの索引単位を一つずつ取り上げ、その索引単位に該当する文書集合と前段の中間結果との積集合を新たな中間結果とするという処理を繰り返す。N-gram 索引における長い検索語処理では、中間結果として n-gram の位置情報を保持しておく必要のない文書順の方が効率的であり[2)]、冗長 n-gram 法でも文書順を採用している[13)]。

2.4 冗長 n-gram 法の改良

冗長 n-gram 法では、候補文書決定に検索語から抽出される n-gram の全てを使用している。しかし、そうした n-gram の中には候補文書を絞り込むのに有効でないものもあり、候補決定に多くの n-gram を使用することでかえって検索時間が増大することもある[13)]。例えば、検索語が「携帯電話」であるとき、冗長 n-gram 法では、「携帯」「帯電」「電話」の 3 つを候補文書決定に、「携帯」「電話」の 2 つを位置検査に使用する。ところが、「帯電」を含む文書がデータベースに多数登録されていれば、「帯電」を使用しても候補文書を絞り込めない。この問題の解決には、絞り込みに有効な n-gram のみを候補文書決定に追加的に使用することにすればよい。論文 13) の考察では、前後に位置する位置検査に用いる n-gram よりも文書頻度の小さいものを絞り込みに有効と判断し、候補特定に用いる改良方式が検討されている。

ただし、論文 13) では、改良方式の有効性を実験によって確認していない。そこで、5 章で用いる新聞記事・環境のもとで、論文 13) の検索語を処理する時間を測定した。表 1 は索引ファイルが全くメモリ上に読み込まれていない cold start での、表 2 はファイルがキャッシュされている warm start での検索時間（秒単位）である。表における「従来」は論文 13) でベースラインとして採用した検索語の先頭から重ならないように n-gram を切り出す方式を指し、冗長 n-gram 法・改良冗長 n-gram 法欄の括弧内の数字は、ベースラインに対する増減比である。なお、(平均′) は高速化手法の効果が期待できない 1 ～ 3 文字を除いた 4 ～ 10 文字に対する平均値である。

これら表からわかるように、改良方式の方が単純な冗長 n-gram 法よりも検索時間が短い☆。特に、改良方式は cold に対して効果が大きいことがわかる。

3. 複合条件への拡張

AND, OR, ANDNOT 演算子を含む複合条件に対しても、各検索語の検索処理に改良冗長 n-gram 法を適用すれば、検索を高速化できる。しかし、改良冗長 n-gram 法の基本アイデアである不要な位置検査の省略を AND, OR, ANDNOT 演算子に直接適用することができれば、さらなる高速化が期待できる。本章では、AND, OR, ANDNOT 演算子のための、無駄な位置検査を行わないアルゴリズムを提案する。

3.1 AND 演算子への拡張

子ノードの検索結果の集合積を自ノードの検索結果とする AND 演算子の処理を考える。なお、AND 演

☆ 検索プログラムを改良したため、検索対象が 1 年分増加しているにもかかわらず、検索時間は論文 13) よりも短縮された。

(1) 検索語 i を最初の検索語とする。
(2) targetDocId = 1*とする。
(3) 検索語 i について targetDocId 以上で最小の文書を検索する（検索文書を matchDocId とする)。検索できない場合、検索処理を終了する。
(4) matchDocId > targetDocId ならば、targetDocId = matchDocId とする。i が最初の検索語であれば i を次の検索語に、そうでなければ i を最初の検索語にし、ステップ (3) に戻る。
(5) 現在の targetDocId では検索していない次の検索語があれば、i をその検索語にしてステップ (3) に戻る。
(6) targetDocId が該当文書であるので、targetDocId を検索結果に加える。
(7) ++targetDocId し、i を最初の検索語にしてステップ (3) に戻る。

図 1 AND 演算子の基本アルゴリズム
Fig. 1 Naive algorithm for AND

(3) 検索語 i について targetDocId 以上で最小の候補文書を検索する（検索文書を matchDocId とする)。検索できない場合、検索処理を終了する。
(6) **targetDocId が AND 条件としての候補文書であるので、全ての検索語について位置検査を行う。すべての検索語が存在していれば、** targetDocId を検索結果に加える。

図 2 AND 演算子の拡張アルゴリズムでの変更部分
Fig. 2 Modified steps for extended algorithm for AND

算子を 2 項演算子とする場合と、多項演算子とする場合があるが、本論文は後者の立場をとる。

AND 演算子の処理方法としては、中間結果を保持しておく必要のない文書順が効率的であることが知られている [2)16)]。文書順による処理では、全ての索引語が出現する文書を文書 ID の小さい順に見つければ良く、アルゴリズムは図 1 のようになる。

N-gram 索引では、検索語の長さに応じて処理方法が異なることを考慮しなければならない。検索語の長さが n に等しい場合は、単語索引と状況は同じであり、上記アルゴリズムをそのまま用いればよい。短い場合は、AND と OR の入れ子条件になるので、4 章であらためて検討する。長い場合には、位置検査が発生するため、その回数によって検索時間が影響される。

* 文書 ID は 1 以上の値を取る。

(1) 最初の検索語について該当する文書をすべて検索し、検索結果とする。
(2) i を次の検索語とする。次の検索語がない場合、検索処理を終了する。
(3) targetDocId = 1 とする。
(4) 検索語 i について targetDocId 以上で最小の文書を検索する（検索文書を matchDocId とする)。検索できない場合、ステップ (2) に戻る。
(5) matchDocId が検索結果に含まれていなければ、matchDocId を検索結果に加える。
(6) targetDocId = matchDocId + 1 とし、ステップ (4) に戻る。

図 3 OR 演算子の基本アルゴリズム
Fig. 3 Naive algorithm for OR

そこで、これ以降、長い検索語を含む AND 条件について考察する。

基本アルゴリズムでは、ステップ (3) において検索語を確実に含む文書を検索しているので、その途中で見つかる候補文書全てについて位置検査を行う。しかし、長い検索語に対する位置検査は、全ての検索語について候補文書であると判断された文書についてのみ実施すれば十分である。そこで、拡張アルゴリズムでは、ステップ (3) を位置検査を行わない候補文書の検索とし、全ての検索語について候補文書となる文書が特定されたステップ (6) ですべての検索語について改めて位置検査を行なう。図 2 にアルゴリズムの変更部分を示す。この修正により、位置検査回数は大幅に減り、検索時間の短縮が期待できる。

検索中に必要な記憶領域についても検討しておく。検索結果については、ステップ (6) で位置検査を行って AND 条件を完全に満たす文書 ID をセットするので、基本アルゴリズムと変わらない。検索中に使用する一時変数は、targetDocId と各子ノードごとに保持する直前の検索において渡された文書 ID で、基本アルゴリズムと同じである。したがって、拡張アルゴリズムによって必要な記憶領域が増大することはない。

3.2 OR 演算子への拡張

子ノードの検索結果の集合和を自ノードの検索結果とする OR 演算子の処理を考える。本論文では、AND 同様、OR も多項演算子として扱う。

まず、単語索引に対する OR 演算子の処理方法を説明する。OR 演算子では、AND 演算子とは異なり、検索単位順が文書順よりも効率的である [7)]。これは、OR 演算子を文書順で実現するためには、ある時点での全ての検索語に該当する文書のなかで最小の文書 ID

(4) 検索語 i について targetDocId 以上で最小の候補文書を検索する(検索文書を matchDocId とする)。検索できない場合、ステップ (2) に戻る。

(5) matchDocId が検索結果に含まれていなければ、検索語 i について位置検査を行なう。検索語が出現していれば、matchDocId を検索結果に加える。

図 4 OR 演算子の拡張アルゴリズムでの変更部分
Fig. 4 Modified steps for extended algorithm for OR

を持つ文書を見つけなければならない。しかし、最小の文書を見つけるには、各検索語について最小の文書が何かを常に管理しておく必要があり、その処理の負荷が大きいからである。一方、索引単位順であれば、処理途中では常に 1 つの検索語のみが処理対象となっているので、このような問題は発生しない。索引単位順によるアルゴリズムを図 **3** に示す。

基本アルゴリズムでは、AND の基本アルゴリズムのステップ (3) と同様に、ステップ (4) の検索処理なかで常に位置検査が行われている。しかし、ステップ (5) では、そのようにして検索された文書がこれまでに処理された検索語によって検索されていたのであれば検索結果に追加されないので、ステップ (4) で行った位置検査が無駄になることがある。したがって、ステップ (4) では位置検査を伴わない候補文書の検索を行い、検索された候補文書が検索結果に含まれていない場合にのみ、ステップ (5) で位置検査を行なう方が効率的である。図 **4** に拡張アルゴリズムの変更部分を示す。なお、AND 同様、必要な記憶領域が増大することはない。

3.3 ANDNOT 演算子への拡張

第 1 子ノードの検索結果と第 2 子ノードの集合差を自ノードの検索結果とする ANDNOT 演算子の処理を考える。ANDNOT 演算子は、第 2 子ノードが補集合を求める NOT 演算子である AND 演算子と捉えることも可能であり、処理手順としては AND と同様に文書順が効率的である。ANDNOT 演算子の基本アルゴリズムは図 **5** のようになる。なお、ステップ (5) で使用される maxDocId は文書 ID の最大値である (maxDocId + 1 は、全ての登録文書の文書 ID よりも大きい値になる)。

位置検査を削減するには、ステップ (2), (5) では位置検査の伴わない候補文書の検索を行い、ステップ (3), (4) の正解文書判定において位置検査を行うこととすればよい。図 **6** に拡張アルゴリズムの変更部分を

(1) targetDocId = 1, matchDocId2 = 0 とする。

(2) 検索語 1 について targetDocId 以上で最小の文書を検索する(検索文書を matchDocId1 とする)。検索できない場合、検索処理を終了する。

(3) matchDocId1 < matchDocId2 ならば、matchDocId1 が該当文書であるので、matchDocId1 を検索結果に加える。targetDocId = matchDocId1 + 1 とし、ステップ (2) に戻る。

(4) matchDocId1 == matchDocId2 ならば、matchDocId1 は該当文書ではない。targetDocId = matchDocId1 + 1 とし、ステップ (2) に戻る。

(5) matchDocId1 > matchDocId2 ならば、検索語 2 について matchDocId1 以上で最小の文書を検索する。検索文書を matchDocId2 とするが、検索できなければ matchDocId2 = maxDocId + 1 とする。ステップ (3) に戻る。

図 5 ANDNOT 演算子の基本アルゴリズム
Fig. 5 Naive algorithm for ANDNOT

(2) 検索語 1 について targetDocId 以上で最小の候補文書を検索する(検索文書を matchDocId1 とする)。検索できない場合、検索処理を終了する。

(3) matchDocId1 < matchDocId2 ならば、**matchDocId1 に対し検索語 1 の位置検査を行う。検索語 1 が存在していれば、この文書が該当文書であるので、**matchDocId1 を検索結果に加える。targetDocId = matchDocId1 + 1 とし、ステップ (2) に戻る。

(4) matchDocId1 == matchDocId2 ならば、**検索語 1 ・検索語 2 について位置検査する。検索語 1 が存在し検索語 2 が存在していなければ matchDocId1 が該当文書であるので matchDocId1 を検索結果に加える。**targetDocId = matchDocId1 + 1 とし、ステップ (2) に戻る。

(5) matchDocId1 > matchDocId2 ならば、検索語 2 について matchDocId1 以上で最小の候補文書を検索する。検索文書を matchDocId2 とするが、検索できなければ matchDocId2 = maxDocId + 1 とする。ステップ (3) に戻る。

図 6 ANDNOT 演算子の拡張アルゴリズムでの変更部分
Fig. 6 Modified steps of extended algorithm for ANDNOT

示す。必要な記憶領域は、AND・OR 同様、基本アルゴリズムと同じである。

4. 混合条件の効率化

前章では、演算子がひとつの場合を扱った。ところが、現実の検索場面では、複数の演算子が組み合わせられることもある。特に、ひとつの概念を複数の検索語の OR 演算子で表現し、それらを AND 演算子で結合した条件が多く用いられる[17)19)]。さらに、n-gram 索引では、3.1 節で述べたように、AND 演算子のなかに短い検索語が含まれる場合、短い検索語は OR 演算子で展開されるので、ユーザが AND 演算子しか用いていなくとも、OR 演算子と AND 演算子が入れ子になった検索条件の処理が必要になる。そこで、この章では OR 演算子と AND 演算子が入れ子になった条件（混合条件）について検討する。

AND 演算子の子ノードとして OR 演算子が存在する場合、AND 演算子の処理は文書順で行われるため、その子ノードである OR 演算子の処理も文書順で行われる。ところが、前述のように OR 演算子の文書順処理は非効率であるので、混合条件の処理も非効率となる。一方、混合条件は、ド＝モルガンの法則を使って等価な OR 標準形— OR 演算子の子ノードとして AND 演算子が存在する形式 — に変換することができる[17)]。OR 標準形であれば、OR 演算子は索引単位順、その子ノードである AND 演算子は文書順と、それぞれに適した処理手順を用いることができるので、検索処理も効率化できる。

ただし、n-gram 索引では、AND 演算子の子ノードとして短い検索語が含まれる場合、非常に多くの子ノードを持つ OR 演算子が AND 演算子の子ノードになることを考慮しなければならない。こうした条件では、OR 標準形への変換コストがかかるので、変換することで検索時間が増大する可能性がある。したがって、混合条件を常に OR 標準形に変形することが望ましいとは限らない。本論文では、この問題に対し、OR 標準形に変換した場合の子ノードの数を見積もり、それがある値（変換閾値）以下であるときに限って OR 標準形に変換することとする。

5. 評　　価

5.1 評価方法

5.1.1 対象文書

本論文で提案した n-gram 索引における複合条件の効率的な処理方法を評価した。評価には新聞記事 5 年分（毎日新聞 CD-ROM 91 ～ 95 年版）を使用した。この CD-ROM では記事ごとにキーワードなども付与されているが、見出しと本文を合わせて 1 文書として登録を行なった。5 年分の合計で、登録件数は 496997 件、テキストサイズは 441.5 MB（1 件当たり 932 B）である。索引には bi-gram 索引を用い、索引サイズは 869.0 MB となった。

5.1.2 検索条件

検索条件としては、以下の 5 セットを用意した。これら検索条件は、社内で稼働している検索システムのログから、検索結果が少なくとも 1 件あり、かつ下に示す制約条件を満たすように選択した。

AND 条件　3 文字以上の検索語を少なくとも 1 つ含む 2 文字以上の検索語を AND 結合したもの。検索語数が 2, 3, 4, 5 のもの各 10 個、計 40 個
例：AND(複写機, 消費電力, 発熱, 低減)

OR 条件　3 文字以上の検索語を少なくとも 1 つ含む 2 文字以上の検索語を OR 結合したもの。検索語数が 2, 3, 4, 5 のもの各 10 個、計 40 個
例：OR(内線, 構内回線, ＰＢＸ)

ANDNOT 条件　3 文字以上の検索語を少なくとも 1 つ含む 2 文字以上の検索語を ANDNOT 結合したもの 30 個
例：ANDNOT(液晶, ディスプレイ)

混合条件 1　1 文字と 3 文字以上の検索語を少なくとも 1 つ含む検索語を AND 結合したもの。検索語数が 2, 3 のもの各 10 個、4 のもの 8 個、5 のもの 2 個、計 30 個☆
例：AND(インクジェット, 紙)

混合条件 2　3 文字以上の検索語を少なくとも 1 つ含む検索語を AND, OR で結合したもの 30 個☆☆
例：AND(OR(感光体, ドラム),OR(冷却, 風))

5.1.3 評価指標

評価指標として、位置検査回数と検索時間を測定した。位置検査回数は各検索条件中の検索語ごとの位置検査回数の合計であり、検索条件の各セットごとに平均値を求めた。検索時間は、SUN Ultra30 (CPU:UltraSPARC II 296MHz) を使用し、索引ファイルはローカルディスクに置いて測定した。索引ファイルが全くメモリ上に読み込まれていない cold start 条件と、データがキャッシュされている warm start 条件について測定した[13)]。各検索条件について 5 回ずつ測定し、最大値と最小値を除いた 3 回の平均値を求め、さらに各セットごとに平均を取った。

☆ 検索ログから、検索語数 4, 5 のものを 10 個用意することができなかった。
☆☆ 平均検索語数は 6 個となった。

表 3 AND 条件の評価結果
Table 3 Evaluation results of AND queries

		基本	拡張	
従	位置検査回数	234.8	140.4	(−40.2%)
	検索時間 (cold)	0.695	0.661	(−4.9%)
来	検索時間 (warm)	0.047	0.028	(−40.4%)
改	位置検査回数	193.3	90.0	(−48.8%)
	検索時間 (cold)	0.667	0.639	(−4.2%)
良	検索時間 (warm)	0.037	0.021	(−43.2%)

表 4 AND 条件における検索語数と位置検査回数の関係
Table 4 Number of query terms vs. number of the proximity checks in AND queries

検索語数	基本	拡張	
2	37.3	19.9	(−46.8%)
3	418.8	214.5	(−48.8%)
4	209.3	106.9	(−48.9%)
5	107.9	54.6	(−49.4%)

表 5 OR 条件の評価結果
Table 5 Evaluation results of OR queries

		基本	拡張	
従	位置検査回数	2842.2	2757.0	(−3.0%)
	検索時間 (cold)	0.904	0.896	(−0.9%)
来	検索時間 (warm)	0.169	0.168	(−0.6%)
改	位置検査回数	1596.7	1382.4	(−13.4%)
	検索時間 (cold)	0.761	0.740	(−2.8%)
良	検索時間 (warm)	0.108	0.097	(−10.2%)

表 6 OR 条件における検索語数と位置検査回数の関係
Table 6 Number of query terms vs. number of the proximity checks in OR queries

検索語数	基本	拡張	
2	125.8	124.6	(−1.0%)
3	1495.6	1389.7	(−7.1%)
4	3441.9	3062.8	(−11.0%)
5	1323.6	952.6	(−28.0%)

5.2 測定結果

5.2.1 AND 条件

ここでは、単一検索語の処理方式として（2.2 節でも説明した）検索語の先頭から重ならないように n-gram を切り出す従来法と改良冗長 n-gram 法の二つの場合について、AND 演算子の処理方法として基本と拡張の二つのアルゴリズムを使用した計 4 つの組合わせを評価した。評価結果を表 3 に示す（これ以降、検索時間は全て秒単位）。基本は基本アルゴリズム、拡張は拡張アルゴリズムの結果であり、拡張の括弧内の数字は基本に対する増減比を表す。

表 3 から、位置検査回数が減少し、拡張アルゴリズムが有効であることがわかる。検索時間も cold・warm ともに減少しているが、増減比は warm では位置検査回数の減少と同程度であるのに、cold ではかなり小さくなっている。これは単一検索語でも見られた現象であり、位置検査回数の減少がディスクアクセスの削減に直接的には結び付かないことに起因する [13]。

N-gram 選択法による相違を見ると、従来より改良の方が位置検査回数の削減率が高い。これは、改良方式では候補文書特定に使用する n-gram 数が多いので各検索語に対する候補文書中の誤検索が少ないためと考えられる。すなわち、検索語レベルの誤検索が少ないと、AND レベルの誤検索も減少するため、位置検査を省略できる可能性が高まるからである。

最後に、改良冗長 n-gram 法に関して、検索語数と位置検査回数の関係を調べた（表 4）。直感的には、検索語が多いほど AND 条件に対する候補文書が誤検索である可能性が低くなるので、位置検査回数の削減率は高くなると予想される。しかし、表 4 からわかるように、削減率は検索語数にはほとんど関係なく一定であった。これは、検索語が多くなるほど検索条件が厳しくなるので、ユーザは該当する文書数の多い検索語を選択し、各検索語の文字数は少なくなる傾向にある。その結果、検索語が多くなるにしたがって位置検査の不要な 2 文字検索語が増え、位置検査が削減されにくくなったと考えられる。実際、2 文字検索語の割合は、検索語数 2 の場合の 20% が検索語数 5 の場合には 28% に増加していた。

5.2.2 OR 条件

OR 条件に対する測定結果を表 5 に示す。ここでも、拡張アルゴリズムにより、位置検査回数が削減され、検索時間が短縮されている。ただし、AND 条件と比較すると短縮の割合は小さい。この差異が生じたのは、位置検査を省略できる状況が AND と OR で異なるからである。AND 演算子では、全検索語について候補文書と判断された文書以外では、各検索語の候補文書において位置検査を省略可能なので、実際に省略されることが多い。一方 OR 演算子では、各検索語の候補文書がすでに処理した検索語のいずれかで該当文書と判断された場合においてのみ位置検査を省略できるが、これに該当するのは複数の検索語がもともと同じ文書に出現している場合に限られるため、省略されることが少ないのである。なお、従来と改良を比較すると、AND の場合と同様、改良の方が効果が大きかった。

改良冗長 n-gram 法に関する、検索語数と位置検査回数の関係を表 6 に示す。ここでは検索語数に応じて位置検査回数の削減率が高くなっており、直感に一致している。これは、OR 条件では、検索語数が多い場

表 7 ANDNOT 条件の評価結果
Table 7 Evaluation results of ANDNOT queries

		基本	拡張	
従	位置検査回数	5894.0	3809.4	(−35.4%)
	検索時間 (cold)	0.745	0.693	(−7.0%)
来	検索時間 (warm)	0.204	0.163	(−20.1%)
改	位置検査回数	5870.6	3632.3	(−38.1%)
	検索時間 (cold)	0.752	0.682	(−9.2%)
良	検索時間 (warm)	0.198	0.149	(−24.7%)

合にも各検索語に該当する文書数に気を使うことなく比較的自由にユーザは検索語を選択できるため、AND 条件とは異なり、2 文字検索語の割合は変わらない。したがって、検索語数が多いことがそのまま位置検査回数の削減に結び付くからと考えられる。実際、2 文字検索語の割合は、検索語数 2 の場合が 20%、検索語数 5 の場合が 18% で、ほぼ一定であった。

5.2.3 ANDNOT 条件

ANDNOT 条件に対する評価結果を表 **7** に示す。ここでも、位置検査回数は減少し、検索時間も短縮されており、拡張アルゴリズムの有効性が確認できた。短縮効果の傾向は、OR 条件よりも AND 条件に近い。これは ANDNOT 演算子は AND 演算子の一種と捉えることができ、位置検査が省略できる場合が多いためと考えられる。

5.2.4 混合条件 1

これまでの評価結果から、複合条件においても単一検索語同様に改良冗長 n-gram 法の方が検索時間が短く、有効であることがわかった。そこで、混合条件の評価では改良冗長 n-gram 法のみを用いる。

まず、混合条件 1 に関する評価結果を示す。混合条件 1 に含まれる 1 文字検索語で、展開される n-gram 数が最小であったのは「雷」であり、展開数は 195 であった。そこで、OR 標準形への変換閾値を 1 (変換を行わない), 200, 500, 1000, 2000 と変化させることとした。改良冗長 n-gram 法に対する測定結果を表 **8** に示す。

変換閾値の影響を調べると、拡張アルゴリズムでは、閾値を大きくするにつれて位置検査回数は一時的に増加するものの 1000 以上では減少している。これは、検索条件が OR 標準形に変換されると、長い検索語が OR・AND 演算子の入れ子の下に位置するようになる (例えば AND(インクジェット, 紙) という検索条件は OR(AND(インクジェット, 紙に),AND(インクジェット, 紙の), …) になる)。そのため、OR・AND 演算子の位置検査省略の効果が相乗され、検査回数が減少するものと考えられる。一方、検索時間は閾値とともに増大している。これは、位置検査回数の減少よりも、OR 標準形への変換処理に要する時間の影響が強いからと考えられる。ユーザの立場からは検索時間の方が重要なので、(少なくとも今回の実装に関しては) 1 文字検索語が OR 標準形に変換されないように変換閾値は 200 以下にするのがよい。

基本アルゴリズムとの比較では、全ての変換閾値に対して位置検査回数・検索時間ともに小さく、1 文字検索語が AND 演算子の子ノードとなる場合にも拡張アルゴリズムが有効であることが確認できた。なお、基本アルゴリズムの場合、拡張とは異なり、位置検査回数も変換閾値とともに大きくなっている。これは、前段に例示したように、OR 標準形に変換すると長い検索語が検索条件中に複数回出現するが、基本アルゴリズムでは、長い検索語に対する検索のたびに位置検査が実施されるので位置検査回数も増大するからである。

5.2.5 混合条件 2

混合条件 2 では、評価用の検索条件そのものが OR を含んでおり、その子ノード数は 2 以上の様々な値をとる。したがって、OR 標準形変換閾値は 1, 2, 5, 10, 20, 50, 100, 200, 500, 1000, 2000 と変化させた。評価結果を表 **9** に示す。

変換閾値の影響は、4 節で予想した通りの傾向が観測された。すなわち、拡張アルゴリズムでは、閾値が 1 の場合よりも 2 以上にした方が位置照合回数・検索時間が小さく、OR 標準形への変換が有効であることがわかる。しかし、閾値が 1000 以上では明らかに検索時間は増大しており、混合条件 2 に限って言えば閾値は 5 〜 500 程度であることが望ましい。なお、1000 以上で検索時間を増大させているのは検索条件中に 1 文字検索語が混ざっている検索条件 (例えば AND(OR(感光体, ドラム),OR(冷却, 風))) であった。ただし、別の検索条件 AND(OR(ネズミ, ねずみ, 鼠),OR(ケーブル, 配電盤, ショート)) では「鼠」の展開数が 28 と少ないため、OR 標準形に変換した方が検索は高速であった。したがって、1 文字検索語を OR 標準形への変換対象から単純に除外するよりも、本論文で提案した閾値による方法が適切と考えられる。

基本と拡張の両アルゴリズムを比較すると、変換閾値が 1, 2 の場合を除いては、拡張の方が高速である。前段で述べたように、変換閾値が 1, 2 では処理自体が非効率なので、拡張アルゴリズムが有効であると結論付けることができる。変換閾値が 1, 2 の場合に拡張が悪化しているのは、現在の OR 演算子の実装では、与えられた targetDocId 以上で最小の候補文書の検索が、OR の子ノードに対して位置検査を伴う検索

表 8 混合条件1の評価結果
Table 8 Evaluation results of MIX1 queries

閾値	位置検査回数			検索時間 (cold)			検索時間 (warm)		
	基本	拡張		基本	拡張		基本	拡張	
1	163.8	128.0	(−21.9%)	1.166	1.050	(−9.9%)	0.271	0.138	(−49.1%)
200	164.5	128.1	(−22.1%)	1.172	1.059	(−9.6%)	0.278	0.145	(−47.8%)
500	220.7	133.8	(−39.4%)	1.385	1.277	(−7.8%)	0.492	0.368	(−25.2%)
1000	257.2	111.6	(−56.6%)	1.446	1.383	(−4.4%)	0.551	0.470	(−14.7%)
2000	259.2	111.7	(−56.9%)	1.804	1.737	(−3.7%)	0.922	0.836	(−9.3%)

表 9 混合条件2の評価結果
Table 9 Evaluation results of MIX2 queries

閾値	位置検査回数			検索時間 (cold)			検索時間 (warm)		
	基本	拡張		基本	拡張		基本	拡張	
1	3454.6	4069.0	(+17.8%)	1.284	1.310	(+2.0%)	0.286	0.302	(+5.6%)
2	3534.4	4044.5	(+14.4%)	1.286	1.310	(+1.9%)	0.289	0.303	(+4.8%)
5	3149.1	2837.2	(−9.9%)	1.252	1.235	(−1.4%)	0.296	0.284	(−4.1%)
10	1850.9	1443.0	(−22.0%)	1.226	1.191	(−2.9%)	0.293	0.270	(−7.8%)
20	1811.7	1365.8	(−24.6%)	1.246	1.203	(−3.5%)	0.318	0.291	(−8.5%)
50	1783.5	1336.5	(−25.1%)	1.246	1.201	(−3.6%)	0.319	0.289	(−9.4%)
100	1500.4	1051.7	(−29.9%)	1.234	1.187	(−3.8%)	0.313	0.283	(−9.6%)
200	1500.4	1051.7	(−29.9%)	1.236	1.187	(−3.8%)	0.314	0.283	(−9.9%)
500	1500.4	1051.7	(−29.9%)	1.235	1.188	(−3.8%)	0.314	0.283	(−9.9%)
1000	1575.9	1165.8	(−26.0%)	1.245	1.200	(−3.8%)	0.325	0.295	(−9.2%)
2000	1252.3	715.5	(−42.9%)	1.314	1.266	(−3.7%)	0.395	0.361	(−8.6%)

を呼び出すようにコーディングされていることに原因がある。すなわち、OR 標準形への変換を行わず OR が AND の子ノードにある場合、AND が拡張アルゴリズムで動作すると、OR の子ノードに位置する検索語については図 2 のステップ (3) とステップ (6) の 2 ヵ所で位置検査が行われてしまうのである。OR 演算子の最小候補文書の検索の実装を改良すれば、この問題は回避できると思われる。

混合条件 1 、 2 の結果を総合すると、変換閾値としては 5 ～ 200 程度の比較的小さな値を使用するのが適切であると言えよう。その一方、この範囲であれば、検索時間はほぼ一定であることが確認できた。これはパラメータ調整にあまり気を遣う必要がないことを意味しており、検索システムにとって望ましい特性である。

5.3 考　　察

これまでの評価実験により、本論文で提案した複合条件の処理手法— 不要な位置検査の省略と子ノード数に応じた OR 標準形への変換 — が検索時間の短縮に有効であることが確認できた。この節では、はじめにでも触れた既存の n-gram 索引向けの高速化手法との関係について考察する。

既存の高速化手法の 1 つ目は文字種等に応じて n を調整するというものである [1)8)18)]。論文 13) に示されているように、n を調整する方法と提案手法のような検索手順の改良は異なる方向性のものなので、基本的に組み合わせ可能である。ただし、提案手法は長い検索語処理において発生する位置検査を省略することで高速化を実現するものなので、文字種に応じて n を大きくした場合には位置検査が不要なケースが増大し、高速化の効果は小さくなると考えられる。

2 つ目の高速化手法は圧縮等の索引構造の工夫である [1)10)]。圧縮技術の導入は索引の小型化と検索の高速化には必須の技術であり [4)17)]、本実験で使用した索引でも圧縮を適用している☆。したがって、前節までの評価結果は提案手法を圧縮と組み合わせた場合の結果であり、両者の組合わせが有効であることはすでに確認されたと言うこともできよう。少なくとも、提案手法はメモリ上の処理量を小さくするものなので、索引のファイル構造の差異が（特に cold start 条件での）高速化効果の大小に影響するにしても、検索時間を悪化させることはないと考えられる。

3 つ目の高速化手法は複合条件を構成する個々の検索語に対する処理手順の改良である [9)13)]。本論文で提案した手法は、複数の検索語の演算子によって規定される関係を利用して位置検査を省略するものであるので、個々の検索語の処理方法には依存しない。実際、5.2.1 ～ 5.2.3 節の評価実験により、2 種類の検索語処

☆ 使用した転置ファイルの概要は論文 13) に記されている。

理法に対して拡張アルゴリズムが有効であることが確認できた。

以上見てきたように、提案手法は既存の高速化手法とも矛盾することなく、組み合わせ可能と考えられる。

6. おわりに

本論文では、n-gram 索引のための複合条件の効率的な処理手法を提案した。基本的な考えは冗長 n-gram 法と同じであり、長い検索語処理に伴う位置検査をできるかぎり行わないことで検索時間を短縮する。本論文では、AND, OR, ANDNOT 演算子の処理アルゴリズムを見直し、位置検査の少ないアルゴリズムを提案した。さらに、AND, OR が入れ子結合した混合条件に対しては、OR 標準形に変換した場合の子ノードの数に応じて選択的に OR 標準形に変換するという方法を提案した。新聞記事 5 年分を用いて評価した結果、提案手法の有効性が確認できた。

今後は、本手法のランキング検索 [4)17)] への拡張、および本手法に適した索引のファイル構造の検討などを行ないたい。

参考文献

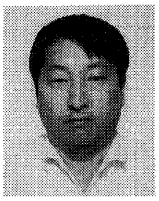
1) 赤峯亨, 福島俊一: 高速全文検索のためのフレキシブル文字列インバージョン法, アドバンストデータベースシステムシンポジウム'96 予稿集, 情報処理学会, pp. 35–42 (1996).
2) Brown, E.: Fast Evaluation of Structured Queries for Information Retrieval, *Proc. of 18th ACM SIGIR Conf.*, pp. 30–38 (1995).
3) Cavnar, W.: Using an n-gram-based document representation with a vector processing retrieval model, *Proc. of 3rd TREC*, pp. 269–277 (1995).
4) Frakes, W. and Basza-Yates, R.: *Information Retrieval: Data Structures and Algorithms*, Prentice-Hall, New Jersey (1992).
5) 堀池博巳, 久富丈志, 小澤義明: 日本語解析処理システム HAPPINESS について, 京都大学大型計算機センター広報, Vol. 24, No. 1, pp. 16–25 (1990).
6) 岩崎雅二郎, 小川泰嗣: 文字成分表による文字列検索の実現と評価, 研究会報告 DBS97, 情報処理学会, pp. 1–10 (1993).
7) Kaszkiel, M. and Zobel, J.: Term-ordered query evaluation versus document-ordered query evaluation for large document databases, *Proc. of 21st ACM SIGIR Conf.*, pp. 343–344 (1998).
8) 川口久光, 菅谷奈津子, 畠山敦, 多田勝己, 加藤寛次: n-gram 型大規模全文検索方式の開発～文字種適応型 n-gram インデックス方式, 第 53 回情報処理学会全国大会 (3), pp. 237–238 (1996).
9) 菊池忠一: 日本語文書用高速全文検索の一手法, 電子情報通信学会論文誌, Vol. J75-D-I, No. 9, pp. 836–846 (1992).
10) 松井くにお, 難波巧, 井形伸之: 大容量情報全文検索エンジン Terass, *FUJITSU*, Vol. 48, No. 3, pp. 240–243 (1997).
11) 道本健二, 真島馨: 高速全文検索の威力, 日経バイト, No. 156, pp. 142–168 (1996).
12) 小川隆一, 菊池芳秀, 高橋恒介: フルテキスト・データベースの技術動向, 情報処理, Vol. 33, No. 4, pp. 404–412 (1992).
13) 小川泰嗣, 松田透: N-gram 索引を用いた効率的な文書検索法, 電子情報通信学会論文誌, Vol. J82-D-I, No. 1, pp. 121–129 (1999).
14) Robertson, A. and Willett, P.: Applications of n-grams in textual information systems, *Journal of Documentation*, Vol. 54, No. 1, pp. 48–69 (1998).
15) 菅谷奈津子, 川口久光, 畠山敦, 多田勝己, 加藤寛次: n-gram 型大規模全文検索方式の開発～インクリメンタル型 n-gram インデックス方式, 第 53 回情報処理学会全国大会 (3), pp. 235–236 (1996).
16) Turtle, H. and Flood, J.: Query Evaluation: Strategies and Optimizations, *Information Processing and Management*, Vol. 31, No. 6, pp. 831–850 (1995).
17) Witten, I., Moffat, A. and Bell, T.: *Managing Gigabytes: Compressing and Indexing Documents and Images*, Van Nostrand Reinhold (1994).
18) 横山昌典: 高速全文検索エンジン, *FUJITSU*, Vol. 48, No. 2, pp. 155–158 (1997).
19) Zobel, J., Moffat, A. and Ramamhanrao, K.: Inverted files versus signature files for text indexing, Technical Report CITIR/TR-94-1, RMIT, The University of Melbourne (1994).

(平成 10 年 12 月 20 日受付)
(平成 11 年 3 月 27 日採録)

(担当編集委員 仲尾 由雄)

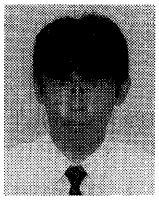
小川 泰嗣 (正会員)

1985 年東京大学工学部計数工学科卒業. 1987 年同大学大学院工学系研究科情報工学専攻修士課程修了. 同年 (株) リコー入社. 情報検索・データベース等の研究開発に従事.



松田 透

1983年東京大学教養学部基礎科学科卒業．1985年同大学大学院理学系研究科相関理化学専攻修士課程修了．同年(株)リコー入社．人工知能・文書処理・情報検索等の研究開発に従事．



橋本 信次

1985年日本電子専門学校 情報処理学部 電子情報処理科 卒業．同年(株)メイテック入社．1989年(株)サンシナップス入社．1994年リコーシステム開発(株)入社．