第5章　ランキング検索処理の高速化

この章では位置省略に基づくランキング検索処理の高速化法を提案する。まず単一文字列の高速化手法を提案し、つぎに論理演算子への対応を検討する。なお、ランキング検索では自然言語文による検索が一般的であるが、ひとたび自然言語文から適切な検索文字列が選択された後は OR として処理されるので、論理演算子の検討により自然言語文もカバーされる。

5.1　従来の高速化手法とその問題点

5.1.1　基本方式

2.3 節で n-gram 索引を用いた場合の基本的なランキング検索手法について説明した。この基本方式で問題となるのは、索引単位である n-gram と異なる長さの検索文字列に対しては n-gram 索引からスコア計算に必要な検索文字列の文書頻度と文書内頻度を直接的には得ることができず、複数の n-gram — n よりも短い場合には検索文字列と先頭部分が一致する n-gram、n よりも長い場合には検索文字列に含まれる n-gram — を用いて文書頻度・文書内頻度を求めるため、検索コストが大きくなることである。n より短い場合については、通常 $n = 2$ であることから一文字の検索文字列が該当するが、実際には一文字の検索文字列はそれほど多くないこと、複数の n-gram 索引の組み合わせ [Aka96a, Sug96] により比較的容易に回避できること等を考慮するとあまり重大な問題とはならない。一方、n を大きくすることには限界があるため、長い検索文字列の処理を回避することは不可能であり、ブーリアン検索と同様にその高速化が重要である [Oga00a, Oga00b]。

長さ n の検索文字列については、索引から得られる n-gram としての頻度情報をそのまま用いればよく、検索は高速である。2.3 節で述べたように、n-gram の頻度は単語としての頻度とは異なるが、検索精度面からは有意な差がないことが実験的に示されている [Kan98, Jon98, Oga97c, Oga99b] ので、ここでは n-gram の頻度でスコア計算を行うこととする。

長い検索文字列のコストについて、コスト増大の原因である位置検査についてさらに検討する。位置検査が必要であるのは次の 2 つのステップである。

- 文書頻度（ＤＦ）の算出

  検索文字列についてブーリアン検索を行い、検索結果の文書数を文書頻度とする。

- 文書内頻度（ＴＦ）の算出

  検索文字列を構成する n-gram の文書内での出現位置を照合することで検索文字列の出現位置を全て求め、その出現回数を文書内頻度とする。

ブーリアン検索では、検索文字列が出現する文書において高々 1 つの出現位置を求めれば良かったのに対し、ランキング検索では、文書内頻度を求めるために全ての出現位置を求める必要があり、位置検査の検索速度に与える影響は大きい。

```
void LongTermNode::rankingRetrieve(RankingResult& result)
{
  // 最初の子ノードについて検索結果を求める
  child[0].rankingRetrieve(result);
  // 2番目以降の子ノードを処理する
  for (int i = 1; i < child.size(); ++i) {
    int df = child[i].getDF();
    int docId = 1;
    while (1) {
      // 該当文書の文書 ID の決定
      docId = child[i].findNext(docId);
      if (docId == maxId) {
        // これ以上候補文書がないので、検索終了
        break;
      }
      // スコア計算
      float score = calculate(df, child[i].getTF(docId));
      RankingResult::Iterator entry = result.find(docId);
      if (entry == result.end()) {
        // これまでの検索結果に含まれていないので、検索結果に追加
        Entry newentry;
        newentry.docId = docId;
        newentry.score = score;
        result.push(newentry);
      } else {
        // これまでの検索結果に含まれているので、スコアを加算
        (*entry).score += score;
      }
      ++docId;
    }
  }
}
```

図 5.1: スコア合成法によるランキング検索処理

5.1.2 スコア合成法

基本方式を高速化する手法として、検索文字列のスコアを構成 n-gram のスコアから合成して求める方法（以下、スコア合成法）が従来から提案されている。

スコア合成法では、ある文書に対する検索文字列のスコアを、その検索文字列を構成する n-gram のスコアから合成して求める。n-gram のスコアは、スコア計算式に n-gram の頻度情報を適用することで求められる [Aka97, Oga97b]。一方、n-gram のスコアから検索文字列のスコアを合成する方法としては、算術和・最小値・最大値等さまざまな計算式が考えられる [Oga97c]。スコア合成方式を算術和とした場合、文書 d における検索文字列 t のスコアは検索文字列中の各 n-gram のスコアを合計することで得られる。

$$score(d,t) = \sum_{g \in t} score(d,g) \tag{5.1}$$

スコア合成方式を算術和とした場合の検索アルゴリズムを図 5.1 に示す。基本的には図

```
void LongTermNode::rankingRetrieve(RankingResult& result)
{
  int docId = 1;
  while (1) {
    // 候補文書の文書 ID の決定
    docId = findNextCandidate(docId);
    if (docId == maxId) {
      // これ以上候補文書がないので、検索終了
      break;
    }
    // 候補文書の検査
    float score = 0.0;
    for (int i = 0; i < child.size(); ++i) {
      // 検査結果が真ならば結果に追加
      score += calculate(child[i].getDF(), child[i].getTF(docId));
    }
    Entry entry;
    entry.docId = docId;
    entry.score = score;
    result.push(entry);
    ++docId;
  }
}
```

図 5.2: AND 型のスコア合成法によるランキング検索処理

3.18 に示した OR 演算子のブーリアン検索と同じで、子ノードを順に処理する。2 番目以降の子ノードに対しては、それまでの検索結果に含まれている文書が検索された場合にはその時点でのスコアに現在の子ノードのスコアを加算し、これまでにない文書であれば検索結果にその文書とスコアの組を追加するという処理手順を取る。

この方式によれば検索文字列を含む文書を特定する必要がないので、n-gram の位置検査は不要であり、検索時間の短縮が見込まれる。その一方で、検索文字列を実際には含まない文書が検索されることもあるため、基本方式と比較して検索文書数は増大する。また、ある文書におけるスコア計算について比較すると、頻度合成法では 1 回スコア計算するのに対し、スコア合成法では、その文書に出現する n-gram 数だけのスコア計算を行う。したがって、スコア計算回数は増大し、検索時間の増大が懸念される。結局、検索時間は検索対象・検索条件・実装等に依存すると考えられる。

上述のスコア合成方法の基本方式（各 n-gram を含む文書を OR したものが検索結果となるので、**OR** 型と呼ぶこととする）を高速化するには、以下の方法が考えられる。

- スコア計算対象（検索結果）を少なくする
- 対象文書におけるスコア計算回数を少なくする

スコア計算対象の削減方法としては、以下の 2 つがある。

- AND 型

```
void LongTermNode::rankingRetrieve(RankingResult& result)
{
  int docId = 1;
  while (1) {
    // 候補文書の文書 ID の決定
    docId = findNextCandidate(docId);
    if (docId == maxId) {
      // これ以上候補文書がないので、検索終了
      break;
    }
    // 候補文書の検査
    if (checkCandidate(docId) == true) {
      float score = 0.0;
      for (int i = 0; i < child.size(); ++i) {
        // 検査結果が真ならば結果に追加
        score += calculate(child[i].getDF(), child[i].getTF(docId));
      }
      Entry entry;
      entry.docId = docId;
      entry.score = score;
      result.push(entry);
    }
    ++docId;
  }
}
```

図 5.3: PROX 型のスコア合成法によるランキング検索処理

検索文字列を構成する n-gram の全てを含む文書を検索文書とする。

AND 型の検索処理を図 5.2 に示す。スコア計算対象文書は全ての n-gram を含む文書であり、ブーリアン検索の候補決定に用いた`findNextCandidate`関数（図 2.6）で見つけられる。スコア計算は、対象文書において各 n-gram のスコアを合計することで求めている。

- PROX 型

AND 型では、OR 型と比較すれば検索文書数は少なくなるものの、検索文字列を含むか否かの検査は行っていないので、検索文字列を含まない文書が検索されることがある。そこで、検索文字列の有無を検査し、検索文字列を含む文書をスコア計算対象文書とする方式が考えられる [Aka97]。この方式を**PROX** 型と呼ぶ。この方式では、検索文書数は基本方式と同じになるが、スコア計算が n-gram 単位に行われるので、ランキング結果は異なる。

PROX 型の検索処理を図 5.3 に示す。スコア計算対象文書は検索文字列を含む文書なので、図 3.6 と同じ方法を用いて決定するのが効率的である。スコア計算は、位置検査によって検索文字列を含むと判断された場合について行えばよく、その場合の計算方法自体は AND 型の場合と同じである。

検索結果数（$\#Result(\mathtt{X})$ で手法X の検索結果数を表す）は以下のような関係になる。

$$\#Result(\text{OR 型}) \geq \#Result(\text{AND 型}) \geq \#Result(\text{PROX 型}) = \#Result(\text{基本方式}) \quad (5.2)$$

高速化の２つ目の方法である対象文書におけるスコア計算回数の削減には、検索に使用する n-gram 数を少なくすれば良い。これはブーリアン検索の高速化でも使用された方法であり、スコア合成法に対しても同じものが適用できる。3 章に示したように、n-gram の選択法には単純パス法、最小頻度法、選択 n-gram 法があり、n-gram 数（$\#Ngram(\mathtt{X})$ で手法X の n-gram 数を表す）は以下のような関係となる。

$$\#Ngram(\text{全 n-gram}) \geq \#Ngram(\text{選択 n-gram}) \geq \#Ngram(\text{単純パス}) = \#Ngram(\text{最小頻度パス}) \quad (5.3)$$

このように使用する n-gram を変えると、OR/AND 型においては検索結果数にも影響する。OR 型では右から左に向かって検索件数が少なくなり、AND 型では逆に大きくなる。単純パスと最小頻度パスを比較すると、n-gram 数は同一であるが、各 n-gram の文書頻度は異なっており、最小頻度パスの方が文書頻度の小さいものが選択されているため、検索結果は同数になるとは限らない。PROX 型については検索件数には影響しないが、スコア計算に使用する n-gram の個数が少ない方が高速になると考えられ、右側の手法の方が高速である。

5.1.3 従来手法の問題点

基本方式とスコア合成法という従来手法同士を比較し、その問題点を明らかにする。

処理コストについては以下のように考えられる。スコア合成法については、OR 型・AND 型は位置検査が不要なのに対し、PROX 型は検索文字列が出現する文書を特定するために位置検査が必要となる。しかし、PROX 型を基本方式と比較した場合、基本方式では、文書頻度・文書内頻度のそれぞれを求める場面で位置検査を行ない、しかも文書内頻度を求める際にはスコア計算対象文書における全ての出現位置を求めなければならないのに対し、PROX 型では、スコア計算対象文書を求める際に 1 文書につき高々 1 回位置検査を行えばよいので、PROX 型の処理コストは基本方式の半分以下と考えられる。結局、検索時間は検索対象・検索条件・実装などに依存するとは言え、AND 型・PROX 型や、n-gram 数の削減等の高速化を適用すれば、スコア合成法の方が基本方式より高速であると考えられる。

一方、検索精度の点からは、基本方式が優れていると考えられる。これは、スコア合成法は n-gram 単位にスコア計算を行うので、検索モデルの背景にある検索文字列の頻度情報とスコアに関する仮説が無視されてしまうからである。具体的には以下のように説明できる。2.3.1 節で述べたように、検索文字列のスコアは文書頻度から算出される弁別性と文書内頻度から算出される代表性から求められる。ところが、検索文字列を構成する n-gram は検索文字列が出現していない文書にも出現することがあるため、検索文字列の文書頻度よりも構成 n-gram の文書頻度は大きくなり、n-gram レベルでは弁別性は低く見積もられることになる。一方、検索文字列が出現している文書について考えると、n-gram が検索文字列の構成要素としてではなく他の単語の構成要素として出現することもあるため、検索文字列の文書内頻度よりも構成 n-gram の文書内頻度は大きくなり、n-gram レベルでは代表性は高く見積もられることになる。このように、検索文字列レベルの重要性が誤って見積られるこ

とから、スコア合成法は基本方式よりも検索精度は低いと考えられるのである。なお、構成n-gram 数が多いカタカナ語や英単語において、検索文字列と構成 n-gram の頻度情報の差が大きく、この問題が顕著になると考えられる。

以上をまとめると、基本方式は検索精度は高いが検索が遅く、スコア合成法は検索は速いが検索精度が低いという問題点があり、従来手法では検索速度・検索精度の両立をはかることができない。

5.2 本研究における高速化の考え方

本論文では、スコア合成方法とは異なるアプローチにより、検索精度を低下させることなく検索を高速化をはかる。そのアプローチとは、検索文字列として頻度に基づいてスコアを計算するという基本方式の原則を生かすことで検索精度を維持しつつ、以下のような2つのアイデアによって高速化を達成するものである。

- 計算順序を工夫することで、無駄な位置検査を無くす
- 頻度情報を近似的に求めることで、位置検査を省略する

前者を順序入れ替え法、後者を頻度推定法と名付ける。

ランキング検索の高速化は検索結果に影響するか否かによって危険な（unsafe）手法、安全な（safe）手法に分けられる [Bro95b]。この視点からは、順序入れ換え法は安全、頻度推定法は危険な手法に分類できる。なお、データベースの高速化等では結果に影響するような危険な手法は通常許されない。しかし、文書検索の分野ではあいまい性を含む自然言語を対象とし、スコア計算もさまざまな仮説に基づくものであるため、高速化前の結果が絶対的に正しいわけではない。そのため、結果に影響するような高速化も、検索精度への影響が小さい範囲であれば許容される。

以下、両手法を詳しく説明するとともに両者の組み合わせについても検討する。

5.3 順序入れ替え法

スコア計算には、文書頻度と文書内頻度の両方が必要である。基本方式は、文書内頻度を求めると同時にスコアを計算しているため、文書内頻度を求める前に文書頻度（すなわち、検索文字列を含む文書数）を求めておく。したがって、文書頻度・文書内頻度のそれぞれを求める際に、検索文字列を構成する n-gram の位置検査が必要であり、索引を2回に渡って走査し、位置検査を重複して実施しなければならない。

この問題に対し、順序入れ換え法は以下のように対応する。文書頻度を求める際に検索文字列が出現している文書については文書内頻度（全ての出現位置の個数）を求めることにすれば、検索文字列が出現している文書（すなわちスコア計算対象文書）の全てが求まり文書頻度が確定した時点には、各文書の文書内頻度も求まっている状態になる。この状態から、あらためて各文書のスコアを計算することとすれば、索引の走査は1回となり、重複して位置検査を行う必要がなくなる。なお、この方法を実現するには、スコア計算対象文書における文書内頻度[1]を一時的に記録する必要があるが、それにはランキング検索結果におけるス

```
void LongTermNode::rankingRetrieve(RankingResult& result)
{
  int docId = 1;
  while (1) {
    // 候補文書の文書IDの決定
    docId = findNextCandidate(docId);
    if (docId == maxId) {
      // これ以上候補文書がないので、検索終了
      break;
    }
    int tf = getTF(docId);
    if (tf > 0) {
      // 検索文字列を含む文書について文書内頻度を記録する
      Entry entry;
      entry.docId = docId;
      entry.score = tf;
      result.push(entry);
    }
    ++docId;
  }
  // スコアを計算する
  int df = result.size();
  for (int i = 0; i < df; ++i) {
    result[i].score = calculate(df, result[i].score);
  }
}
```

図 5.4: 順序入れ替え法によるランキング検索処理

コアの格納場所を利用すればよい（詳細はアルゴリズムの説明のなかで述べる）。

検索アルゴリズムを図5.4に示す。このアルゴリズムでは、スコア計算対象文書の文書内頻度の記録に、最終的にはランキング検索結果をセットする配列をそのまま利用している。まず、スコア計算対象文書ごとにスコアの代わりに文書内頻度を記録しておく。全ての検索文字列を含む文書が求め終わった時点で、文書頻度が確定するので、それと最初のステップで文書ごとに求めておいた文書内頻度を用いてスコアを計算する。

この手順によれば、基本方式で最初に文書頻度を求めるために行っていたブーリアン検索が不要となり、その中で行われていた位置検査分だけ検索コストを削減できる。もちろん、スコアを計算するために第2ステップでは検索結果を走査する必要があるが、これはメモリ上の単純な処理であり、検索時間への影響は極めて小さい。

[1]多くのスコア計算式では、式 (2.2) のように、スコアは文書頻度の項と文書内頻度の項の積で与えられる。この場合、第1ステップで文書ごとの文書内頻度を記録しておくのではなく、第1ステップでスコアの文書内頻度の項を計算しておくということも可能である。

5.4 頻度推定法

頻度推定法はスコア計算に必要な文書に関する頻度情報を近似的に求めることで位置検査を省略し、検索を高速化するものである。文書に関する頻度情報である文書頻度と文書内頻度のそれぞれの推定方法を以下で説明する。

5.4.1 文書頻度の推定

検索文字列が n-gram より長い場合、文書頻度を正確に知るためにブーリアン検索が必要となる。したがって、検索文字列中の n-gram を含む候補文書において本当に検索文字列が出現しているかを確認するために位置検査が必要となり、検索コストが増大する。この問題を解決するため、位置検査を行うことなく文書頻度を推定する。具体的な推定方法としては、以下の 2 つの方法が考えられる。

- AND 方式

  検索文字列を構成する n-gram を全て含む文書を特定し、その文書数を検索文字列の文書頻度とする。この AND 方式は、位置検査を行うべき候補文書を求め、その件数を検索文字列を含む件数の近似値とするものである。

  AND 方式で文書頻度を推定する場合の検索手順を図 5.5 に示す。基本的な検索の流れは基本方式（図 2.10）と同じであるが、最初にブーリアン検索をして検索文書を決定する部分が位置検査を伴わない候補文書の特定（`findNextCandidate` 関数）となっている点で基本方式と異なっている。

  文書頻度推定において求めた候補文書はブーリアン検索結果に保存しておき、スコア計算の際に試用する。スコア計算時には、位置検査を行い、文書内頻度が 0 の文書は最終結果に加えることはしていない。すなわち、検索文字列を含む文書だけが検索され、検索件数は正しい値となる。

- MIN 方式

  検索文字列を構成する n-gram の文書頻度の最小値を検索文字列の文書頻度とする。この MIN 方式では、各 n-gram の文書頻度を参照するだけであり、基本方式・AND 方式と異なり文書頻度を求める段階では転置リストを一切走査する必要がなく、転置リストの走査は文書内頻度を求めて文書スコアを計算する際の 1 回だけになる。

  MIN 方式の検索手順は図 5.6 の通りである。 4 行目等で`candidateNode->child`のようになっているのは、文書頻度の推定に 3.3.3 節の選択 n-gram 法の候補選択用の n-gram 群を用いることとしているためである。この部分は位置検査用の n-gram 群とすることも可能であり、その場合には単に`child`とすれば良い（性能への影響は後述）。

  スコア計算時には`findNextCandidate`関数で候補文書を求め、`getTf` 関数で求めた文書内頻度が 1 以上のものだけに対しスコアを計算している。すなわち、AND 方式同様、検索文字列を含む文書のみが検索される。

両者を比較すると、転置リストの走査が 1 回で済む MIN 方式が高速であるが、AND 方式の方が真の文書頻度との差異は小さいので検索精度の点では優れていると考えられる。

```
void LongTermNode::rankingRetrieve(RankingResult& result)
{
  BooleanResult bresult;
  // 位置検査を行わない近似検索を行う
  int docId = 1;
  while (1) {
    // 候補文書の文書 ID の決定
    docId = findNextCandidate(docId);
    if (docId == maxId) {
      // これ以上候補文書がないので、検索終了
      break;
    }
    bresult.push(docId);
    ++docId;
  }
  // 文書頻度の推定値を求める
  int df = bresult.size();
  // 文書ごとの処理
  for (int i = 0; i < df; ++i) {
    Entry entry;
    entry.docId = bresult[i];
    // 文書内頻度を求める
    int tf = getTF(entry.docId);
    if (tf > 0) {
      // スコア計算し、結果に追加
      entry.score = calculate(df, tf);
      result.push(entry);
    }
  }
}
```

図 5.5: AND 方式による推定文書頻度を用いたランキング検索処理

```
void LongTermNode::rankingRetrieve(RankingResult& result)
{
  // 子ノードの文書頻度の最小値を求める
  int df = maxId;
  for (int i = 0; i < candidateNode->child.size(); ++i) {
    if (candidateNode->child[i].getDf() < df) {
      df = candidateNode->child[i].getDf();
    }
  }
  int docId = 1;
  while (1) {
    // 候補文書の文書 ID の決定
    docId = findNextCandidate(docId);
    if (docId == maxId) {
      // これ以上候補文書がないので、検索終了
      break;
    }
    int tf = getTF(docId);
    if (tf > 0) {
      // 検索文字列を含む文書について文書内頻度を記録する
      Entry entry;
      entry.docId = docId;
      entry.score = calculate(df, tf);
      result.pushd(entry);
    }
    ++docId;
  }
}
```

図 5.6: MIN 方式による推定文書頻度を用いたランキング検索処理

```
void LongTermNode::getTF(DocumentID docId)
{
  // 子ノードの文書頻度の最小値を求める
  int tf = maxTf;
  for (int i = 0; i < candidateNode->child.size(); ++i) {
    if (candidateNode->child[i].getTf(docId) < tf) {
      tf = candidateNode->child[i].getTf(docId);
    }
  }
  return tf;
}
```

図 5.7: 文書内頻度の推定処理

文書頻度の推定に検索文字列中のどの n-gram を用いるかは検索性能に影響する。AND 方式・MIN 方式にかかわりなく、n-gram を多く使用するほど近似値と真の値との差は小さくなり、検索精度の点では有利と考えられる。一方、検索時間の点からは n-gram が少ないほどアクセスするデータ量が少なくなり、検索速度の点では優れている。6 章の評価実験では、3.3.3 節の選択 n-gram 法の候補文書決定用の n-gram を使用した。

5.4.2 文書内頻度の推定

検索文字列が n-gram より長い場合、文書内頻度を正確に知るには候補文書ごとに検索文字列の全ての出現位置を求める必要がある。すなわち、ここでも位置検査が検索コストを増大させることとなる。そこで、文書内頻度の場合と同様、位置検査を伴わない方法で文書内頻度を推定することで検索の高速化を試みる。具体的には、ある文書における検索文字列の文書内頻度をその検索文字列の構成 n-gram の文書内頻度の最小値によって推定する[2]。推定精度と速度に関しては、文書頻度の場合と同じく、使用する n-gram が多いほど精度は高くなるが速度は遅くなるというトレードオフがある。

検索手順は基本方式そのものでよく、図 2.10 の通りである。ただし、`getTf` 関数は基本方式の図 2.12 ではなく、n-gram の文書内頻度の最小値を求める図 5.7 のものを用いる。ここでは、推定値の精度を高めるため候補決定用の n-gram を対象に最小値を求めているが、位置検査用の n-gram を対象としてもよい。

5.5 順序入れ替え法と頻度推定法の組み合わせ

これまでは順序入れ替え法と頻度推定法を個別に説明してきたが、本節ではその組み合わせについて考察する。これら 2 つの手法は独立なものなので組み合わせが可能であるが、組み合わせても高速化の相乗効果が得られない場合もある。また、頻度推定法に関しては文書頻度、文書内頻度の推定について述べたが、それらの組み合わせについても検討する。

組み合わせ可能性については以下のことが言える。

[2]文書頻度推定の MIN 方式に相当する。文書内頻度については AND 方式に相当する推定方式は存在しない。

表 5.1: 順序入れ替え法と擬似頻度法の組み合わせ

	順序入れ替え	文書頻度推定	文書内頻度推定	検索件数
NNN	×	×	×	○
RNN	○	×	×	○
NAN	×	AND	×	○
~~RAN~~	○	AND	×	○
NMN	×	MIN	×	○
~~RMN~~	○	MIN	×	○
NNM	×	×	MIN	○
~~RNM~~	○	×	MIN	○
NAM	×	AND	MIN	AND
RAM	○	AND	MIN	AND
NMM	×	MIN	MIN	AND
~~RMM~~	○	MIN	MIN	AND

- 頻度推定法に関して、文書頻度の推定と文書内頻度の推定は組み合わせ可能である。ただし、両者を組み合わせるとランキング検索において位置検査を行わないことになるので、高速化が期待できる一方、検索文字列の有無を正確には判断できなくなる。検索文字列から選択された n-gram の全てが存在する文書を検索することとなり、検索結果は AND 型のスコア合成法と同じになる。

- 順序入れ替え法と頻度推定法の組み合わせでは、文書内頻度の求め方に応じて文書頻度が決定されるので、組み合わせ不可能あるいは意味がない場合がある。

  (1). 文書内頻度を推定しない場合、順序入れ替えにより文書頻度は正確に求まるので、文書頻度を推定する意味がない。

  (2). 文書内頻度を推定する場合、順序入れ替えにより AND 方式による文書頻度推定値が求まるので、文書頻度推定なし・MIN 方式による文書頻度推定と組み合わせる意味がない。

以上の検討に基づいて組み合わせ可能性を整理すると表 5.1 のようになる。ここでは、組み合わせを 3 文字のアルファベットで表現しており、1 文字目は順序入れ替えを適用する（R）しない（N）を、2 文字目は文書頻度推定を AND 方式でする（A）MIN 方式でする（M）しない（N）を、3 文字目は文書内頻度推定を MIN 方式でする（M）しない（N）を示す。基本方式（NNN; 全てが×となっているもの）も含めて、組み合わせは 12 通りである。ここから上述した組み合わせ不可能な場合— (1) による場合が RAN, RMN、(2) による場合が RNM, RMM — を除くと 8 通りとなる。なお、最後の欄の検索件数はランキング検索結果として返される件数であり、○は基本方式と同じ値になること、AND は検索語から選択された n-gram を AND 結合した結果（AND 型スコア合成法の結果）と同じになることを表している。

上記手法の検索精度への影響を考える。検索文字列の頻度情報の推定誤差が大きいほど検索精度も悪くなるとすると[3]、文書頻度の推定は AND 方式、MIN 方式の順で精度が悪化す

表 5.2: 順序入れ替え法と擬似頻度法の組み合わせごと位置検査回数

	第 1 ステップ	第 2 ステップ	合計
NNN	1	K	K+1
RNN	—	K	K
NAN	0	K	K
NMN	—	K	K
NNM	1	0	1
NAM	0	0	0
RAM	—	0	0
NMM	—	0	0

る。文書頻度の推定と文書内頻度の推定のいずれが影響が大きいかは不明だが、文書頻度/文書内頻度の両者を組み合わせたものは単独の場合より精度が悪化すると考えられる。ただし、文書頻度/文書内頻度組み合わせであっても、検索文字列に対する頻度に基づいてスコア計算を行うため、スコア合成法と比較すれば検索精度の低下は小さいと期待できる。なお、順序入れ替え方はスコア計算に用いる値を変えるものではないので、検索精度への影響はない。

ここまでの考察から、以下の右から左の順に検索精度が高いと考えられる（$Prec(\mathtt{X})$ で X の検索精度を表す）。

$$\begin{aligned}(Prec(\text{NNN}) = Prec(\text{RNN})) > (Prec(\text{NAN}) > Prec(\text{NMN})) \sim \\ Prec(\text{NNM}) > (Prec(\text{NAM}) = Prec(\text{RAM})) > Prec(\text{NMM})\end{aligned} \tag{5.4}$$

一方、検索速度への影響は以下のようにまとめられる。各手法の位置検査回数をまとめると表 5.2 のようになる。ここで、第 1 ステップは文書頻度を求めるための処理であり、第 2 ステップは文書内頻度を求める処理である。順序入れ替え法および文書頻度を MIN 方式で推定する場合には第 1 ステップは不要である。この表において K/1/0 は、それぞれ、検索文字列の文書内頻度を得るために全ての出現位置を求めること、検索文字列の出現を確認するために 1 つの出現位置を求めること、出現位置を求めないことを意味している。

検索は表 5.2 の合計欄が少ないほど高速と考えられる。合計欄が同じ場合については、以下のことを考慮する必要がある。

- 位置検査が 0 の場合でも、候補文書を決定するために文書 ID にアクセスする、あるいは n-gram の出現回数にアクセスする必要があるため、索引に全くアクセスしないわけではない。したがって、合計の位置検査回数が同じ場合であっても、第 1 ステップに 0 と書かれている組み合わせはそうでない組み合わせよりも処理量は多く、検索が遅いと考えられる（例えば、NAN と RNN, NMN）。
- RNN と NMN、あるいは RAM と NMM のように第 2 ステップの値が同じ場合、順序入れ替え法では、第 2 ステップの後にメモリ上でスコア計算を行う処理が必要とな

[3]検索文字列の頻度情報の推定誤差が大きいほど検索精度も悪いということは必ずしも正しいとは限らない。それは現状のランキング検索が文書がユーザの検索要求を満足するか否かを検索文字列の頻度という単純なものだけで算出するモデルを採用しており、そのモデル自体も多くの仮定に基づいているからである。

表 5.3: ランキング検索における論理演算子の動作

	結果集合	スコア計算
OR	子ノードのいずれかを満足する文書	子ノードのスコアの合計
AND	子ノードの全てを満足する文書	子ノードのスコアの合計
ANDNOT	第1子ノードを満足し、第2子ノードを満足しない文書	第1子ノードのスコア

る。したがって、順序入れ換え法を組み合わせていない方式（NMN, NMM）の方が高速であると考えられる。

ここまでの考察をまとめると、以下の右から左の順に検索が高速であると考えられる（$Speed$(X) で X の検索速度を表す）。

$$\begin{aligned} &Speed(\mathrm{NMM}) \geq Speed(\mathrm{RAM}) > Speed(\mathrm{NAM}) > Speed(\mathrm{NNM}) > \\ &\quad Speed(\mathrm{NMN}) \geq Speed(\mathrm{RNN}) > Speed(\mathrm{NAN}) > Speed(\mathrm{NNN}) \end{aligned} \tag{5.5}$$

なお、位置検査が全くなくて1つの処理ステップで検索が実現される NMM, RAM は伸長処理・位置検査数の観点では AND 型スコア合成法と同等である。しかし、スコア計算回数で比較すると、スコア合成法では n-gram ごとにスコア計算をするのに対し、NMM, RAM は検索文字列ごとにスコア計算するため、後者のほうが少ない。

5.6 論理演算子処理

ランキング検索を適用する場面では、自然言語による検索要求文を受け付け、システムは検索要求文から適切な検索条件を生成し、その条件に対する検索を実施するのが一般的である [Sal83b, Fra92, Tok99]。検索要求文から複数の検索文字列（検索語）を抽出した後は検索文字列の独立性を仮定して以下のように扱われることが多い。

- いずれかの検索文字列が出現する文書を検索結果とする
- 検索文書のスコアは、その文書における各検索文字列のスコアの合計とする

検索結果集合の決め方はブーリアン検索における OR 演算子と同じ動作であり、子ノードのスコアを合計する機能を追加することで、OR 演算子がランキング検索にも使用できるようになる。AND 演算子、ANDNOT 演算子についても同様の拡張すれば、ランキング検索において論理演算子を陽に使用することが可能となる。各演算子の動作をまとめると表 5.3 のようになる。

ランキング検索において演算子を取り込んだモデルとしては、ファジィモデル [Rad81]、拡張ブーリアンモデル [Sal83a] などがある。代表的なファジィモデルでは、結果集合の決め方は本論文と同じであるが、スコア計算に OR 演算子では最大値、AND 演算子では最小値を用いる。一方、拡張ブーリアンモデルでは、AND/OR ともにいずれかが該当する文書を結果集合とし、スコア計算に OR 演算子では p-conorm、AND 演算子では p-norm という距離計算を用いる。しかし、これらの方式に対しても、スコア合成方法を切り替え可能な形

```
void OrNode::rankingRetrieve(RankingResult& result)
{
  // 最初の子ノードについて検索結果を求める
  child[0].rankingRetrieve(result);
  // 2番目以降の子ノードを処理する
  for (int i = 1; i < child.size(); ++i) {
    RankingResult tmp;
    // 子ノードのランキング検索を行う
    docId = child[i].rankingRetrieve(tmp);
    // 結果をマージする
    result += tmp;
  }
}
```

図 5.8: OR 演算子のランキング検索処理

式で実装しておけば、本論文の方式で対応可能である。スコア合成方法として何が適切かは各モデルの適格性をテストコレクションを用いた評価する必要があり、本論文の範疇を超えるので、ここではこれ以上議論しない（次章の評価実験では、もっとも一般的な合成法である算術和を使用する)。

本節の残りでは、論理演算子のランキング検索処理について、ブーリアン検索のために3.4 節で提案した位置検査省略による高速化が適用できるかを検討する。

5.6.1 OR 演算子

ブーリアン検索では、あるノードについて満足すると判断された文書については、それ以降のノードに関する位置検査を省略することで高速化を実現した。しかし、ランキング検索では、その文書を満足する全てのノードのスコアを計算する必要がある。したがって、位置検査省略による高速化は適用できない。

処理アルゴリズムは図 5.8 のように、子ノードごとにランキング検索し、その結果をマージ（コード上は+=演算子）する。マージでは、結果集合に含まれている文書ではスコアを加算し、含まれていない文書はスコアをそのままに結果集合に追加する。なお、本章で提案したランキング検索の高速化手法は各子ノードの検索処理に適用可能である。

5.6.2 AND 演算子

ランキング検索の基本アルゴリズムは、子ノードごとにランキング検索を行ない、それまでの検索結果との集合積（コード上は*=演算子）を順次求めればよい（図 5.9)。*=演算子では結果集合に含まれている文書に対してはスコアを加算し、含まれていない文書は結果集合から除外する。

ブーリアン検索では、候補特定用ノードを用いて特定される全ての子ノードについて候補文書となる文書以外では位置検査を省略することで、検索を高速化した。一方、ランキング検索では、AND 演算子を満足する文書が存在する場合、全ての子ノードについて文書頻度

```
void AndNode::rankingRetrieve(RankingResult& result)
{
  // 最初のノードをランキング検索する
  child[0].rankingRetrieve(result);
  if (result.size() == 0) {
    // 該当文書がなければ終了
    return;
  }

  // 残りのノードをランキング検索する
  for (int i = 1; i < child.size(); ++i) {
    RankingResult tmp;
    child[i].rankingRetrieve(tmp);
    result *= tmp;
    if (result.size() == 0) {
      // 該当文書がなければ終了
      return;
    }
  }
}
```

図 5.9: AND 演算子のランキング検索の基本アルゴリズム

が必要となるのでブーリアン検索の位置検査省略をそのまま適用することはできない。ただし、ランキング検索においてスコアを計算しなければならない文書、すなわち文書内頻度を求めるために全ての出現位置を求める必要がある文書は全ての子ノードの検索文字列が出現する文書だけである。したがって、それ以外の文書については検索文字列の有無だけを確認すればよく、基本アルゴリズムと比較すると、文書内頻度を求めるための位置検査を省略することが可能である。

位置検査省略を適用したランキング検索の拡張アルゴリズムを図 5.10 に示す。ここでは、ノードごとにブーリアン検索を行い、それらの集合積（*=演算子）を求めて検索結果集合を決定している。その際、ノードごとの検索結果数を文書頻度として記録しておき、結果集合が決まった後に文書ごとのスコアを計算する際に利用する。

拡張アルゴリズムでは、AND 条件を満たしている文書については、文書頻度を求めるためのブーリアン検索で検索文字列の有無を確認するために位置検査した上で、文書内頻度を求めるために再度位置検査を実施している。したがって、拡張アルゴリズムが基本アルゴリズムより常に高速であるとは限らないが、AND 条件を満足する文書の割合が低いほど、あるいは検索文字列の文書内頻度が大きいほど拡張アルゴリズムが有利となる。

5.6.3 ANDNOT 演算子

ランキング検索の基本アルゴリズムは、第 1 子ノードのランキング検索結果から第 2 子ノードの該当文書を除去すればよく、図 5.11 のようになる。

ブーリアン検索の高速化は、第 2 子ノードを満足する文書においては第 1 子ノードの位置

```
void AndNode::rankingRetrieve(RankingResult& result)
{
  vector<int> df;
  BooleanResult tmp1, tmp2;

  // 最初のノードをブーリアン検索する
  child[0].booleanRetrieve(tmp1);
  if (tmp1.size() == 0) {
    // 該当文書がなければ終了
    return;
  }
  // 文書頻度を記録する
  df.push(tmp1.size());

  // 残りのノードをブーリアン検索する
  for (int i = 1; i < child.size(); ++i) {
    child[i].booleanRetrieve(tmp2);
    tmp1 *= tmp2;
    if (tmp1.size() == 0) {
      // 該当文書がなければ終了
      return;
    }
    // 文書頻度を記録する
    df.push(tmp2.size());
  }

  // 該当文書についてスコア計算を行う
  for (int j = 0; j < tmp1.size(); ++j) {
    int docId = tmp1[j];
    float score = 0;
    for (int i = 1; i < child.size(); ++i) {
      score += calculate(df[i], child[i].getTF(docId));
    }
    Entry entry;
    entry.docId = docId;
    entry.score = score;
    result.push(entry);
  }
}
```

図 5.10: AND 演算子のランキング検索の拡張アルゴリズム

```
void AndnotNode::rankingRetrieve(RankingResult& result)
{
  // 1番目の子ノードをランキング検索する
  RankingResult tmp1;
  child[0].rankingRetrieve(tmp1);
  // 2番目の子ノードをブーリアン検索する
  BooleanResult tmp2;
  child[1].booleanRetrieve(tmp2);
  // 1番目の結果から2番目の結果を除外する
  for (int i = 0; i < tmp1.size(); ++i) {
    if (tmp2.find(tmp1[i].docid) == false) {
      // 2番目の結果に含まれていないので ANDNOT としての該当文書
      result.push(tmp1[i]);
    }
  }
}
```

図 5.11: ANDNOT 演算子のランキング検索の基本アルゴリズム

検査を省略する、あるいは、第1子ノードを満足しない文書においては第2子ノードの位置検査を省略する、のいずれかによって検索を高速化した。ランキング検索では、スコア計算が必要なのは第1子ノードだけなので、後者の方法であれば、位置省略による高速化を適用できる。また、第1子ノードについて、文書頻度を求めるためには検索文字列を含む全ての文書を特定する必要はあるが、第2子ノードが該当する文書については文書内頻度を求める必要はなく、この点からも高速化が可能である。

前記観点から位置検査省略を適用したランキング検索の拡張アルゴリズムを図 5.12 に示す。これはブーリアン検索の拡張アルゴリズム（図 3.24）を基本としているものの、候補文書の検索を行うのは2番目のノードだけであり、1番目のノードは基本アルゴリズム（図 3.22）のように毎回位置検査を行なう。ただし、1番目のノードに順序入れ替え法を適用しており、最初のフェーズでは第1子ノードの結果が見つかるごとに文書頻度をカウントしておき、全ての結果が見つかった後、文書ごとのスコアを計算することにしている。

```
void AndnotNode::rankingRetrieve(RankingResult& result)
{
  int docId0 = 1, docId1 = 0;
  int df = 0;
  while (1) {
    // 1番目の子ノードの該当文書を探す
    docId0 = child[0].findNext(docId0);
    if (docId0 == maxId) {
      // これ以上候補文書がないので、検索終了
      break;
    } else if (docId0 > docId1) {
      // 2番目の子ノードの候補文書を探す
      docId1 = child[1].findNextCandidate(docId0);
    }
    // 1番目の子ノードを満足する文書をカウントする
    ++df;
    if (docId0 < docId1)  {
      // 2番目の子ノードが追い越したので、1番目の子ノードの
      // 該当文書は ANDNOT としても該当文書
      Entry entry;
      entry.docId = docId0;
      entry.score = child[0].getTF(docId0);
      result.push(entry);
    } else {
      // 2番目の子ノードの候補文書と1番目の子ノードの
      // 該当文書が同じ
      if (child[1].checkCandidate(docId1) == false) {
        // docId1 が本当に出現していなければ該当文書
        Entry entry;
        entry.docId = docId0;
        entry.score = child[0].getTF(docId0);
        result.push(entry);
      }
    }
    ++docId0;
  }
  // スコアを計算する
  for (int i = 0; i < df; ++i) {
    result[i].score = calculate(df, result[i].score);
  }
}
```

図 5.12: ANDNOT 演算子のランキング検索の拡張アルゴリズム

第6章　ランキング検索高速化手法の評価

前章で提案したランキング検索の高速化手法— 順序入れ替え法および頻度推定法 — を評価する。ランキング検索の評価では、検索速度（時間）だけでなく、ランキング結果の妥当性の指標である検索精度も評価する必要がある。検索精度を求めるためには、文書検索の評価用データであるテストコレクションを使用するのが一般的である [Kis00, Bae99]。以下では、本研究の時点で最新の日本語向けテストコレクションであった NTCIR-1 を用いた評価について述べる。

6.1　評価データ

6.1.1　テストコレクション NTCIR-1

検索精度を評価するためには、検索対象・検索要求およびその組み合わせに対する正解文書を定めた評価用データが必要である。文書検索の分野では、検索対象・検索要求・正解文書から成る評価用データをテストコレクションと呼ぶ [Kis00, Bae99]。本研究で使用した NTCIR-1（予備版）[Sek00] は、学術情報センター（現、国立情報学研究所）が作成したものであり、評価実験を行った 1999 年時点で最新の日本語向けテストコレクションであった。

6.1.2　検索対象

検索対象には日英混在の学会データベースの論文要旨から構成される JE コレクションを用いた。各論文要旨は SGML 形式であり、タイトル・著者・会議名・日付・要旨・キーワード・学会名から構成される。各フィールドは日本語・英語の両方で記述されることがあるが、実際には日本語だけで記述されている場合も多い。図 6.1 に論文要旨の例を示す。評価実験においては、文書内容に基づく検索には直接関係ないと思われる著者・会議名・日付・学会名は除外し、タイトル・要旨・キーワードを抜き出して 1 文書とした。文書数は 339501 件、文書サイズは 267 MB（ 1 件当たり 825 B）であった。

6.1.3　検索条件

テストコレクションが評価対象としているのはランキング検索であり、ブーリアン形式の検索条件ではなく、検索システムが検索すべき文書を記述した検索要求が与えられる。NTCIR-1（予備版）の検索要求はその内容を 2 ～ 3 語の名詞句の形式で簡潔に表現した「タイトル」、内容を 1 文で表現した「検索要求」、内容を詳細に数文で表現した「検索要求説明」、内容に関連した 10 ～ 20 個のキーワードで表現した「概念」、内容の技術分野を表現する「分野」の 5 つのフィールドから構成され、SGML 形式で表現されている。図 6.2 に検

```
<REC>
<ACCN>gakkai-0000000001</ACCN>
<TITL TYPE="kanji">電気回路演習用CAIとその改良</TITL>
<TITE TYPE="alpha">CAI for Exercise in Electrical Circuits and Its
Improvement</TITE>
<AUPK TYPE="kanji">小野 敏夫 / 大川原 宣夫 / 栗原 秀行 / 小林 一郎
</AUPK>
<AUPE TYPE="alpha">Ono,Toshio / Ookawara,Norio / Kurihara,Hideyuki
/ Kobayashi,Ichiro</AUPE>
<CONF TYPE="kanji">昭和63年電気学会全国大会一般講演</CONF>
<CNFD>1988. 03. 29 - 1988. 03. 31</CNFD>
<ABST TYPE="kanji"><ABST.P>大学等での基礎的な電気回路演習を支援するCAI
ソフトウェアとその改良について述べている。本CAIはコンピュータが出題され
る回路を学習者各人のレベルに応じて自動的に作成すること、解答を数式で入力
することが大きな特長である。また、誤った解答に対しては、原因の検討を容易
にするメッセージが表示されるなど、効果的な個別学習が限られた設備・要員で
実施可能であるよう配慮した。昨年度の学生による本CAIの使用結果のアンケー
ト、および発表の場における質疑等を参考に、操作を容易とし、効果を上げるた
めの改良を行った。</ABST.P></ABST>
<KYWD TYPE="kanji">電気回路 // 演習</KYWD>
<KYWE TYPE="alpha">Electrical Circuit // Exercise // CAI</KYWE>
<SOCN TYPE="kanji">電気学会</SOCN>
<SOCE TYPE="alpha">The Institute of Electrical Engineers of Japan
</SOCE>
</REC>
```

図 6.1: NTICR-1 の検索対象文書の例

索要求の例を示す。本実験では「検索要求」フィールドのみを使用した。なお、検索要求は30個である。

実験で用いる「検索要求」フィールド（以下、検索要求文と呼ぶ）は自然言語で記述されている。したがって、検索要求文をそのまま検索文字列とすることはできず、そこから適切な検索語を抽出して検索条件を作成する必要がある。

本実験では、以下の手順で検索条件を生成した。

(1). 検索要求文を形態素解析し、単語に分割する

(2). あらかじめ用意した不要語を除く名詞類を検索文字列として選択する

(3). 選択された検索文字列をOR演算子で結合し、検索条件とする

例えば、図6.2の検索課題では、検索要求文「自律移動ロボットについて」は「自律（サ変名詞），移動（サ変名詞），ロボット（一般名詞），に（格助詞），つい（動詞)、て（接続助詞)」のように形態素解析され、動詞・助詞類を除いた「自律」「移動」「ロボット」が検索文字列として選択され、最終的には「#or(自律, 移動, ロボット)」という検索条件が生成される。

前述の手順によって生成された検索条件に関して、検索要求当たりの検索文字列の個数と長さ（文字数）の分布を表6.1に示す。この表から、検索要求のなかでは単語数が4のもの

```
<検索課題 q=0001>
<タイトル>
ロボット
</タイトル>
<検索要求>
自律移動ロボットについて
</検索要求>
<検索要求説明>
自律移動ロボット自体の設計、開発、評価などが総合的に書かれた文献、または、
自律移動ロボットにおける部分的なシステム(経路制御、物体認識など)の設計
について書かれた文献が検索要求を満たす。自律移動はするがロボットではない
ものの設計、開発に関する論文も部分的に検索要求を満たす。自律分散システム
などの自律移動ロボットを応用したシステム、自律移動しないロボットに関する
文献は検索要求を満たさない。
</検索要求説明>
<概念>
自律移動，自律走行，ロボット，画像処理，物体認識，ファジィ制御，カメ
ラ，センサ，ロボットビジョン，ステレオビジョン，経路地図，認知地図，自
律分散システム，分散アルゴリズム，協調，設計，開発，評価
</概念>
<分野>
1．電子・情報・制御
</分野>
</検索課題>
```

図 6.2: NTICR-1 の検索要求の例

が最多で 12 個、検索語の長さは 2 文字のものが最多で 81 個であることがわかる。

なお、形態素解析には社内の言語処理グループが開発した CJP(Compact Japanese Parser) を利用した。CJP の特徴は以下の通りである [Ito91, Koj91, Moc91]。

- 階層的品詞体系と、品詞よりも微小な文法属性を記述する素性を用い、多様なレベルでの文法記述を可能にすることにより、規則数を抑えつつ解析精度を高めている。
- トライ構造を用いた辞書引き、動的計画法を用いたパス探索等により、高速な形態素解析を実現している。

6.2 評価方法

本章の最初で述べたように検索精度と検索速度の両面から評価を行った。

6.2.1 検索精度の評価

検索精度は NTCIR の標準的な手順に従って評価した。以下、その手順を説明する。

検索精度の基本的な指標に再現率（recall）・適合率（precision）がある [Sal83b, Wit94, Bae99]。再現率は正解文書を洩れなく検索できる能力、適合率は正解でない文書を検索しな

表 6.1: 検索要求当りの検索文字列の個数と長さの分布

	個数	長さ
1	1	0
2	2	81
3	6	11
4	12	10
5	4	6
6	3	8
7	1	4
8	0	3
9	1	0
10 以上	0	2
平均	4.1	3.1

い能力を表すもので、以下の式で計算される。

$$\text{再現率} = \frac{\text{検索された正解文書数}}{\text{正解文書数}} \tag{6.1}$$

$$\text{適合率} = \frac{\text{検索された正解文書数}}{\text{検索文書数}} \tag{6.2}$$

ランキング検索においては、どのランクまでの文書を検索文書とするかによって再現率・適合率が異なる。NTCIR で最も標準的な指標として用いられる平均適合率は、正解文書が検索された時点での適合率の平均値である（この際、検索されなかった正解文書に対する適合率は 0 とする）。検索手法による検索結果数の違いの影響を少なくするため、ランキングの上位 1000 位までを用いて平均適合率を求める。このようにして検索要求ごとに求めた平均適合率の平均値が最終的な検索精度指標としての平均適合率である。

平均適合率はランキング全体に対する検索精度を測定するものなので、平均適合率だけではランキングの上位・下位における検索特性の違いを把握することができない。この問題点を補うために使用されるものとして再現率・適合率グラフがある。検索要求ごとに、ランキングごとの再現率・適合率から再現率が $0.0, 0.1, \cdots, 1.0$ における適合率を計算し、さらに全検索要求での平均値を求める。これをグラフにしたものが再現率・適合率グラフである。再現率が 0.0, 0.1 等の低い場合の適合率がランキング上位、再現率が 0.9, 1.0 等の高い場合の適合率がランキング下位での検索精度を表す。

なお、NTCIR では A（完全に適合する）と B（部分的に適合する）の 2 段階の正解判定を行っているが、本実験では A/B いずれと判定されていても正解として扱うこととした。

6.2.2 検索速度の評価

性能評価のため、4 章と同じく、検索対象文書に関する bi-gram（索引を作成した。使用したマシンも同じ Sun Microsystems 社の Ultra 10（CPU: Ultra SAPRC II 300MHz, メインメモリ 128 MB)、外付けローカルディスク（18GB, 7200rpm, Ultra-SCSI 接続)、OS は

Solaris 2.5.1 である。索引作成に要した時間は 3.8 時間で、索引サイズは 490.1 MB（もとテキストの 1.9 倍）であった。

検索時間の測定状況は、4 章と同く、COLD, WARM, HOT の 3 通りについて 3 回の測定結果の平均値を用いた（単位は秒）。実装に依存しない処理量を測るものとして、位置検査数、伸長処理数、スコア計算数、検索文書数を測定した。位置検査数、伸長処理数は 4.2 節でも測定したものであるが、ランキング検索では文書内頻度を知るために検索文字列の全ての出現を求める必要があるので、位置検索数は検索文字列がある文書に n 回出現している場合には n 回とカウントする。スコア計算数は、`calculate()` の呼び出し回数である。これらの評価値は、検索条件ごとの測定値を全検索条件について平均したものを測定結果とした。

6.3 ランキング検索のスコア計算手法

ランキング検索のスコア計算には以下の式を用いた [Oga00c]。

$$score(d,t) = \log(k_t \cdot \frac{N}{f_t} + 1) \cdot \frac{f_{d,t}}{k_d + f_{d,t}} \cdot \frac{f_{q,t}}{k_q + f_{q,t}} \tag{6.3}$$

これは確率モデルの代表的な計算式である式 (2.1) を修正した式である（修正内容は付録 D において詳しく説明する)。パラメータは $k_t = 1, k_d = 1, k_q = 0$ に固定した。

スコア合成法における n-gram スコアの合成には算術和を使用した。算術和を用いたのは、スコア合成法の比較検討を行なった別の評価実験において bi-gram 索引に対する検索精度が良かったからである [Oga97c]。

6.4 測定結果

6.4.1 既存手法の比較

基本方式と従来の高速化手法であるスコア合成法について比較した結果を表 6.2 に示す。一番左の NNN は基本方式であり、それ以外のものがスコア合成法である。スコア合成法では、スコア計算対象文書の決定法に関して OR 型、AND 型、PROX 型の 3 種類と、使用する n-gram の選択法に関して全 n-gram、最小頻度パスの 2 種類の 6 通りの組み合わせを評価した。表 5.1 と同じくアルファベット 3 文字で表現することとし、1 文字目はスコア合成法であることを示すため S に固定し、2 文字目は計算対象決定法を OR 型（O）AND 型（A）PROX 型（P）を、3 文字目は n-gram 選択法を全 n-gram 法（A）最小頻度（M）を示す。表 6.2 の各列で、ブーリアン検索と共通部分はこれまでと同じである。新たに追加された平均適合率、検索文書数、位置検査数、スコア計算数は検索要求ごとの値を平均値を示している。

表 6.2 から、5.1.3 節で議論したように、検索精度では基本方式が、検索時間ではスコア合成法が優れているということが確認できる。スコア合成法の中では、最小コスト法と組み合わせた AND 型である SAM が検索精度・検索時間ともに優れていることがわかった。

手法ごとの特性を分析すると以下のようにまとめられる。

表 6.2: スコア合成法の評価結果

	NNN	SOA	SOM	SAA	SAM	SPA	SPM
平均適合率	0.3827	0.3214	0.3526	0.3179	0.3526	0.3215	0.3509
	—	−16.0%	−7.9%	−16.9%	−7.9%	−16.0%	−8.3%
検索時間 COLD	1.291	1.581	1.249	1.104	0.935	1.260	1.103
	—	+54.5%	−3.3%	−14.5%	−27.6%	−2.4%	−14.6%
検索時間 WARM	1.023	1.308	1.012	0.830	0.700	0.986	0.837
	—	+27.9%	−1.1%	−18.9%	−31.6%	−3.6%	−18.2%
検索時間 HOT	0.663	0.829	0.639	0.389	0.349	0.516	0.484
	—	+25.0%	−3.6%	−41.3%	−47.4%	−22.2%	−27.0%
ページ数 全体	101.3	144.1	102.3	129.5	100.5	129.4	104.2
	—	+42.2%	+1.0%	+27.7%	−0.8%	+27.7%	+2.9%
ページ数 評価	84.5	127.2	88.8	112.6	86.9	112.5	87.9
	—	+50.5%	+5.1%	+33.3%	+2.8%	+33.3%	+4.0%
位置検査数	67369	0	0	0	0	15851	16786
	—	−100%	−100%	−100%	−100%	−76.5%	−75.1%
伸長処理数	606270	722937	510879	456170	369730	445811	404691
	—	+19.2%	−15.7%	−24.8%	−39.0%	−26.5%	−33.2%
スコア計算数	48595	242453	171354	78644	69893	76294	65298
	—	+399%	+253%	+61.8%	+43.8%	+57.0%	+34.4%
検索文書数	43077	92387	87028	43630	44246	43077	43077
	—	+114%	+102%	+1.3%	+2.7%	±0.0%	±0.0%

検索精度の分析

- 平均適合率を比較すると、スコア合成法のなかで最も良い SOM, SAM でも基本方式に対して 7.9 % 低下している。t-student 検定による統計的検定 [Hul93] でも、「基本方式と SAM の平均適合率が等しい」という仮説の検定統計量は 2.401 となり、有意水準 2.5%で SAM は基本方式に劣っていると判断できる。

- スコア合成法のバリエーション間の検索精度の違いを見る。スコア計算に使用する n-gram 選択法（全 n-gram 法, 最小頻度パス法）の影響を調べると、検索対象決定法（OR 型, AND 型, PROX 型）によらず最小頻度パスの方が検索精度が高い。これは、全 n-gram 法では文書頻度の大きい n-gram もスコア計算に使用されるため、検索語と n-gram の文書頻度のずれが大きくなるからと考えられる。一方、検索対象決定法の違いが検索精度に与える影響はそれ程大きくない。これは、検索対象決定法の違いは検索文書数に影響するものの、スコア計算方法は同じなので PROX 型で検索される文書のスコアは OR 型, AND 型についても等しい値となるからと考えられる。

- 再現率・適合率グラフを用いて詳細に比較する。n-gram 選択法による相違は小さくグラフ上では差異を確認しにくいので、スコア合成法のなかでは検索精度の良い SOA,

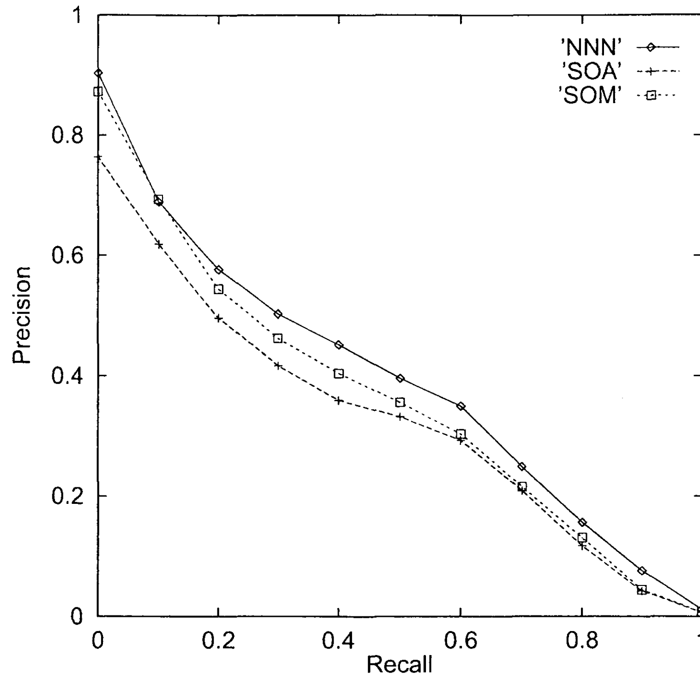

図 6.3: 再現率・適合率グラフ

SOM を基本方式 NNN とともにプロットしたものが図 6.3 である。図 6.3 から、SOM では再現率が 0.2 以上で適合率が低下しており、ランキング中位以降の検索漏れが多いことがわかる。SOA では再現率が低いところでも適合率が大きく低下しており、ランキング上位からも正解が抜け落ちていることがわかる。

検索時間の分析

- 検索時間を比較すると、SOA 以外では、基本方式よりもスコア合成法の方が検索時間が短い。これは基本方式の位置検査回数が多いことに起因していると考えられる。また、検索時間の差異は HOT の方が COLD よりも大きいが、その原因は、ブーリアン検索のときと同じく、位置検査回数の差がディスクアクセスの減少に直接は結び付かないからである。
- スコア合成法での検索時間の違いを見る。まず、対象文書決定法による影響を調べる。OR 型では、検索文書数が多いことが検索回数/スコア計算回数を増加させ、検索時間の増大につながっている。AND 型は検索文書数が少なく、かつ位置検査が不要であることから、検索時間は最短である。PROX 型は検索文書数は基本方式と同じでスコア合成法の中では最も少ないが、位置検査が必要なため、検索時間では AND 型よりも検索時間が増大している。一方、n-gram 選択法では、n-gram 数が少ない最小頻度パス法の方が検索時間も短い。なお、検索要求当たりの n-gram 数は、全 n-gram 法が 8.9、最小頻度パス法が 6.9 であった。

表 6.3: 順序入れ替え法・頻度推定法の評価結果

	NNN	RNN	NAN	NMN	NNM	NAM	RAM	NMM
平均適合率	0.3827	0.3827	0.3810	0.3751	0.3813	0.3850	0.3850	0.3791
	—	±0.0%	−0.4%	−1.9%	−0.5%	+0.6%	+0.6%	−0.9%
COLD	1.291	1.115	1.189	1.112	1.077	1.020	0.955	0.948
検	—	−13.6%	−7.9%	−13.9%	−16.6%	−20.1%	−26.0%	−26.6%
索 WARM	1.023	0.858	0.931	0.857	0.821	0.763	0.696	0.696
時	—	−16.1%	−9.0%	−16.2%	−19.7%	−25.4%	−32.0%	−32.0%
間 HOT	0.663	0.502	0.548	0.502	0.467	0.388	0.347	0.346
	—	−24.3%	−17.3%	−24.3%	−29.6%	−41.5%	−47.6%	−47.8%
ペ 全体	101.3	101.3	101.3	101.3	101.3	104.2	104.2	104.2
ー	—	±0.0%	±0.0%	±0.0%	±0.0%	+2.9%	+2.9%	+2.9%
ジ 評価	84.5	84.5	84.5	84.5	84.5	87.9	87.9	87.9
数	—	±0.0%	±0.0%	±0.0%	±0.0%	+4.0%	+4.0%	+4.0%
位置検査数	67369	50483	50483	50483	16384	0	0	0
	—	−25.1%	−25.1%	−25.1%	−75.7%	−100%	−100%	−100%
伸長処理数	606270	409832	493284	409832	372298	453182	369730	369730
	—	−32.4%	−18.6	−32.4%	−38.6%	−25.3%	−39.0%	−39.0%
スコア計算数	48595	48595	48595	48595	48595	49923	49923	49923
	—	±0.0%	±0.0%	±0.0%	±0.0%	+2.7%	+2.7%	+2.7%
検索文書数	43077	43077	43077	43077	43077	44246	44246	44246
	—	±0.0%	±0.0%	±0.0%	±0.0%	+2.7%	+2.7%	+2.7%

6.4.2 提案手法の比較

本研究で提案した高速化手法である順序入れ替え法・頻度推定法の評価結果を表 6.3 に示す。表の各列に与えられているアルファベット 3 文字が順序入れ替え法・頻度推定法の組み合わせを識別するもので、その意味は表 5.1 の通りである。一番左の NNN は基本方式で、数値は表 6.2 と同じであるがベースラインとしてこの表にも掲載した。なお、各行の意味は表 6.2 と同じである。

表 6.3 において順序入れ替え法を適用しているのは、NNN に対する RNN と NAM に対する RAM の 2 つである。それらの組み合わせを比較することで、順序入れ替え法は検索精度に影響することなく、検索速度を向上できることが確認できる。一方、頻度推定法（NNN に対する NAN, NMN, NNM）は検索精度をほとんど低下させることなく検索時間を短縮しており、その有効性が確認できた。

手法ごとの特性を分析すると以下のようにまとめられる。

検索精度の分析

検索精度の差は全体に小さく、5.5 節の予想の式 (5.5) とは順序入れ替え法が検索精度に影響しないことを除いては必ずしも一致しなかった。これは 5.2 節でも触れたように、頻度情報の正確さと検索精度の関連がそれほど高くないことに起因していると考えられる。以下、頻度推定法について考察する。

- 文書頻度の推定による検索精度の低下は、MIN 方式（表中では NMN）でも 1.9 % とわずかである。t-student 検定による検定統計量は 1.016 で、有意水準 10%でも有意でない低下であり、検索精度は実質的に維持されたと判断できる。AND 方式（表中では NAN）の方が MIN 方式よりも低下の割合が小さいが、5.5 節で検討したように文書頻度の見積もりの精度が高いからであろう。
- 文書内頻度の推定（表中では NNM）による検索精度の低下は 0.5% であり、文書頻度の推定の場合よりも小さい。
- 文書頻度と文書内頻度の両方を推定した場合（表中の NAM, NMM）においても、検索精度はほぼ同じであった。NAM では若干ではあるものの検索精度は向上しているが、この向上は統計的には有意水準 10%でも有意ではない。

なお、各手法とも、平均適合率の差異が少ないことから想像される通り再現率ごとの適合率の差もわずかであったため、再現率・適合率グラフは特に示さない。

検索時間の分析

検索時間は検索速度の予想である式 (5.5) の通りとなっており、5.5 節での検討が正しかったことがわかる。5.5 節で検討していない点を補足すると以下のようになる。

- 順序入れ替え法による検索速度向上の原因を確認する。アクセスページ数は順序入れ替え法によって変化していない。基本方式の検索においては、文書頻度を求めるためと文書内頻度を求めるために 2 回転置リストを走査することになるが、順序入れ替え法ならば 1 回で済む。ただし、今回の規模であれば 1 回目にアクセスしたページが全てキャッシュに残るため、ページアクセス数は変化していない。検索速度向上は、転置リストのデータの伸長処理と、検索文字列の出現位置を求めるための位置検査数の減少が原因となっている。
- 文書頻度推定と文書内頻度推定の時間短縮の割合に大きな相違はないが、組み合わせにより相乗効果が得られることがわかる。組み合わせにより検索語の位置検査が全く行われなくなり、スコア合成法でもっとも高速である AND 型に近い検索速度を達成できた。なお、RAM、NAM と SAM では、検索回数・位置検査回数が全く同じであるのに、スコア計算回数の多い SAM の方が検索時間が短い。これは、SAM の方が処理フローが簡単であるためと考えられる。実装の改良により、差異を小さくし、さらには RAM, NAM が SAM を上回るようにすることが可能であろう。

最後に、基本的な検索手法である頻度合成法 (NNN) に対する検索精度（平均適合率）と検索時間の増減比をグラフにして各手法の特徴を再確認する。スコア合成法の代表として

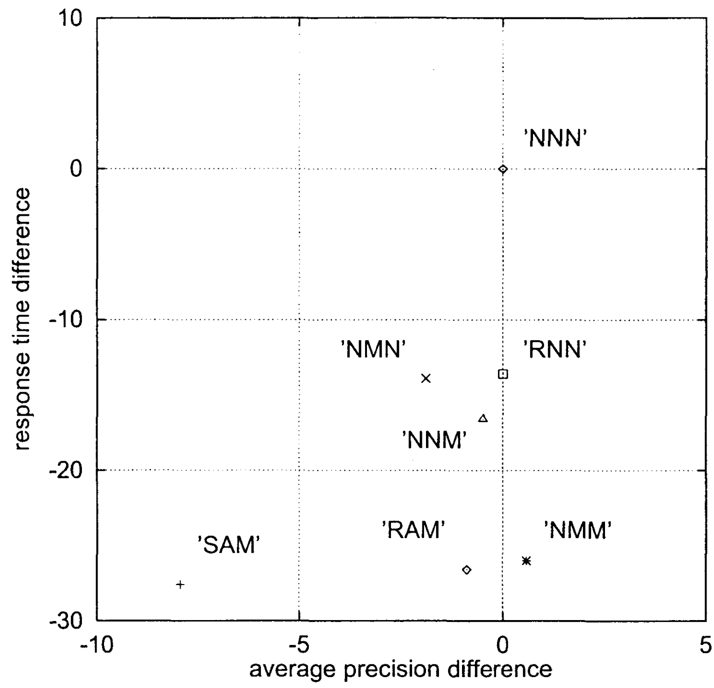

図 6.4: 平均適合率・検索時間 (COLD) の増減比 (%)

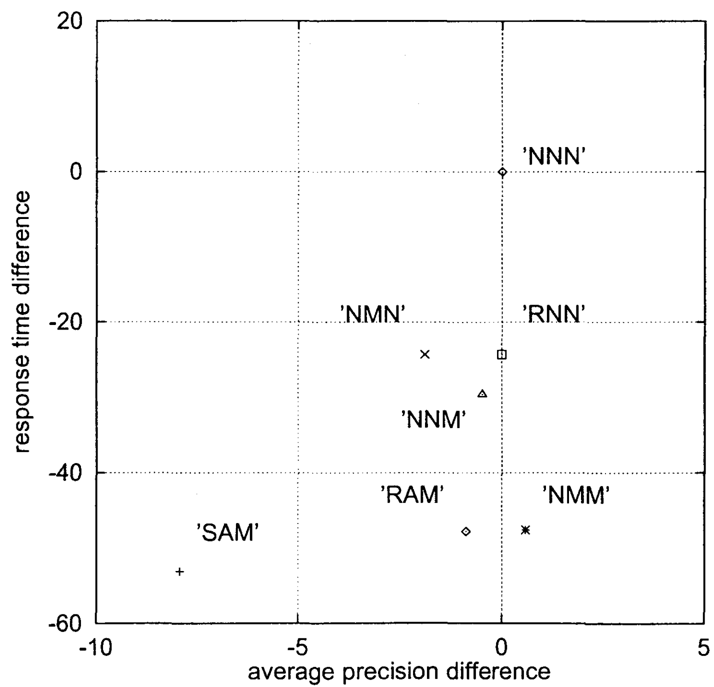

図 6.5: 平均適合率・検索時間 (HOT) の増減比 (%)

表 6.4: 単語索引と n-gram 索引の登録性能比較

	単語索引	n-gram 索引
登録時間 (sec)	15246	13668 (−10.4%)
ファイルサイズ (MB)	196.8	490.1 (+149.0%)

SAM、順序入れ替え法・推定頻度法の代表として RNN, NMN, NNM, RAM, NMM を取り上げた。COLD, HOT のグラフを図 6.4、図 6.5 に示す。これらの図においては、x 軸は 0 に留まったまま y 軸に沿ってマイナス側にプロットされる点が検索精度に影響を与えることなく高速化されることを意味しており、提案手法のなかでも RAM, NMM が優れていることが確認できる。これらの図からも、本研究で提案した順序入れ替え法・推定頻度法は検索精度を低下させることなく検索を高速化できるという特徴が読み取れる。

6.5 単語索引との性能比較

これまでの評価実験により、n-gram 索引におけるランキング検索について従来方式と本研究の提案手法の比較を行い、提案手法の有効性を確認することができた。以下では、n-gram 索引と単語索引の性能比較を行い、1.4 節の定性的な分析の妥当性を検証する。

6.5.1 単語索引の評価方法

この比較実験を行った 1999 年時点では、文書検索用の転置ファイルには英語に限定した単語索引機能しか実装していなかった。すなわち、空白・カンマ・ピリオド等のあらかじめ定められた区切り文字に基づいてテキストを単語に分割する機能しか実装されていなかったため、単語区切りが明示的でない日本語に対する単語索引の直接的な性能測定はできない。そこで、登録対象文書をあらかじめ日本語形態素解析プログラムにより単語分割して英語のように単語間にスペースを挿入したデータを作成し、それを対象文書とすることで単語索引の評価を行うこととした。

形態素解析には検索条件生成と同様に社内開発の CJP を利用した。なお、この実験を行った 1999 年当時の形態素解析ライブラリとしては代表的なフリーソフトである奈良先端大学院大学開発の Chasen と比較しても CJP は高速であった。

6.5.2 登録性能の比較結果

登録性能を表 6.4 に示す。単語索引と比較して n-gram 索引は登録時間は 10%程度短いもののファイルサイズは約 2.5 倍に大型化している。単語索引の登録時間には CJP による形態素解析時間（2894 秒）が含まれている。

この結果は 1.4.3 節の表 1.2 の定性的な分析と一致しており、形態素解析による登録時間の増大、文書あたりの索引単位数の多さによるファイルサイズの増大という両索引手法の特性が確認できる。

表 6.5: 単語索引と n-gram 索引の検索性能比較

		単語	NNN	SAM	RNN	NNM	NMM
平均適合率		0.3699	0.3827	0.3526	0.3827	0.3813	0.3791
		—	+3.5%	−4.7%	+3.5%	+3.1%	+2.5%
	COLD	0.698	1.291	0.935	1.115	1.077	0.948
検		—	+85.0%	+34.0%	+59.7%	+54.3%	+35.8%
索	WARM	0.410	1.023	0.700	0.858	0.821	0.696
時		—	+149.5%	+70.7%	+109.3%	+100.2%	+69.8%
間	HOT	0.319	0.663	0.349	0.502	0.467	0.346
		—	+107.8%	+9.4%	+57.4%	+46.4%	+8.5%
ペ	全体	28.9	101.3	100.5	101.3	101.3	104.2
ー		—	+250.1%	+247.8%	+250.1%	+250.1%	+260.6%
ジ	評価	21.3	84.5	86.9	84.5	84.5	87.9
数		—	+296.7%	+308.0%	+296.7%	+296.7%	+312.7%
位置検査数		0	67369	0	50483	16384	0
		—	—	±0.0%	—	—	±0.0%
伸長処理数		126158	606270	369730	409832	372298	369730
		—	+380.6%	+193.1%	+224.9%	+195.1%	+193.1%
スコア計算数		42356	48595	69893	48595	48595	49923
		—	+14.7%	+65.0%	+14.7%	+14.7%	+17.9%
検索文書数		37735	43077	44246	43077	43077	44246
		—	+14.2%	+17.3%	+14.2%	+14.2%	+17.3%

6.5.3 検索性能の比較結果

検索性能比較は n-gram 索引の検索手法の比較と同様に行った。n-gram 索引に関しては、ベースライン（NNN）・スコア合成法で最速のもの（SAM）・順序入れ替え法（RNN）・頻度推定法単体で最速のもの（NNM）・順序入れ替え法と頻度推定法の組み合わせで最速のもの（NMM）を取り上げた。評価結果を表 6.5 に示す。表 6.2・表 6.3 と同様にページ数・位置検査回数等の基本的な指標と、ベースラインに対する増減比も示している。ただし、これまでとは異なりベースラインが単語索引の結果となっているので注意が必要である。

表 6.5 から、検索精度は n-gram 索引のほうが優れているものの、検索時間が大幅に増大することがわかる。

検索精度についてまとめると以下の通りである。

- スコア合成法以外の n-gram 索引の検索精度は単語索引より約 3 〜 5%良い。これは複合語と構成語の問題、形態素解析誤り等による検索漏れが原因と考えられ、単語索引の問題点を確認することができた。実際、単語索引の検索文書数は 37735 件と n-gram 索引より 15%近く少なく、検索漏れが生じていることがわかる[1]。

[1] n-gram 索引では文字列上の一致だけから検索される過剰検索が起きているので、検索文書数の減少分の全てが形態素解析による検索漏れというわけではない。

- n-gram 索引のなかでは SAM（スコア合成法）だけが単語索引より検索精度が低い。単語索引とスコア合成法は索引単位の頻度情報に基づいてスコア計算するという点では同じである。両者を比較した場合には単語索引の検索精度が優れているということは、単語単位の頻度情報に基づくスコア計算が有効であるという 5.1.3 節の考察の正しさを実験的に確認できたと言える。

一方、検索時間の傾向をその他の評価指標と同時に検討すると以下のようにまとめられる。

- n-gram 索引のなかでは基本方式である NNN が最も遅く、SAM, NMM が高速である。ただし、SAM, NMM であっても単語索引に対しては COLD の検索時間が約 35%増大しており、単語索引の高速性が確認できる。

- 単語索引の高速性は、IO 処理の指標であるページ数、CPU 処理の指標である位置検査数、伸長処理数、スコア計算数の全てが n-gram 索引より優れていることの結果であると確認できる。

- 単語索引におけるアクセスページ数は全体で 28.9 であり、100 ページを超える n-gram 索引の 1/3 以下と非常に少ない。このように差が大きい理由としては、n-gram 索引では検索文字列の長さに応じてアクセスする n-gram 数が増大すること、同一の索引単位であっても単語索引では単語としての出現だけであるのに対し n-gram 索引では単語であるか否かに関わらず全ての出現を記録するためにデータ量が増大すること、の２つが考えられる。前者に関しては、検索時にアクセスする索引単位の個数で確認できる。実際、単語索引では 3.8 個[2]であるの対し n-gram 索引では 10.6 個であり、大きな差が存在している。

  一方、アクセスページ数は約 250%増大しているのに対して検索時間は最大でも 85%しか増大しておらず、ページ数の差と比べて検索時間の差は相対的に小さい。この理由としては、単語索引では索引単位あたりのデータ量が少ないために索引単位ごとにランダムアクセスが必要となるのに対し、n-gram 索引では索引単位あたりのデータ量が多いために１つの索引単位のデータが複数ページにまたがり[3]、シーケンシャルアクセスの割合が相対的に高いからと考えられる。

- 単語索引による検索では位置検査が不要なので、位置検査数は 0 である。一方、n-gram 索引では、高速化のために位置検査を完全に省略した SAM, NMM を除いては、位置検査は発生する。

- 伸長処理数の差は 190 〜 380%と非常に大きい。位置検査が不要である SAM, NMM でも約 190%と大きな差がついているが、これはページ数のところでも説明したように１つの検索文字列の処理に複数の n-gram が必要であることと索引単位ごとのデータ量が増大していることの２点が原因と考えられる。さらに、NNN, RNN, NNM では、位置検査が必要であるため、単語索引との差異がさらに大きくなっている。

[2]表 6.1 から検索要求ごとの検索文字列数は 4.1 個である。単語索引では索引単位数と検索文字列数が一致するはずであるが、実際にアクセスした索引単位数が 3.8 個と少ないのは、検索文字列に一致する索引単位が索引中に存在していないことがあるからである。

[3]次章の転置ファイルの物理編成法において説明するが、各索引単位の格納形式には単一ページに収まるショートリストと複数ページにまたがるロングリストの２種類がある。n-gram 索引においてはロングリストの個数が増大する。

- スコア計算数の差は、検索文書数の差とほぼ一致している。両手法で用いる検索文字列は共通であり、SAM以外は単語単位にヒットと判定された文書においてスコア計算を行うため、スコア計算数は検索文書数に比例すると考えられるからである。SAMでは検索文字列を構成するn-gramごとにスコア計算を行うため、スコア計算の差（65.0%）は検索文書数の差（17.3%）よりも増大している。

- 検索時間のCOLD, WARM, HOTでの違いを検討する。単語索引に対するn-gram索引の検索時間の増大は、COLDで35〜85%、WARMで70〜150%、HOTで8〜110%である。n-gram索引の高速化手法の比較ではCOLD, WARM, HOTの順に効果があったのと比較して、WARMでの影響が最も大きい点でこれまでとは異なる傾向が見られる。これは検索時間に占める割合が大きいIO処理が、単語索引とn-gram索引の比較では、COLDよりもWARMにおいて差が増大することの影響と考えられる。実際、COLDにおけるページ数である「ページ数：全体」の増減比が250%程度であるのに対し、EVALにおける「ページ数：評価」では300%程度になっている。この増減比は、CPU処理である伸長処理数およびスコア計算数の増減比よりも大きいため、HOTよりもWARMの方が単語索引とn-gram索引の検索時間の差も大きくなったと考えられる。

なお、ここでの比較ではn-gram索引はbi-gram($n = 2$)としている。1.5.1節に示したようなn-gram抽出法の改良を施すことで、n-gram索引の検索性能を向上させることができる。すなわち、この評価実験は検索速度に関してn-gram索引が不利な状況であり、この結果だけから単語索引とn-gram索引の優劣を決められるものではない。

第7章　高速検索手法向きの転置ファイルの物理編成法

前章まででは、長い検索文字列処理に伴う位置検査を省略するという考えに基づく高速検索手法を提案してきた。しかし、検索処理時間は転置ファイルの物理編成にも依存しているため、物理編成が位置検査省略にふさわしいか否かに応じて高速化の効果も大きく左右される。そこで、本章では位置検査省略向きの転置ファイルの物理編成法ついて検討する。

7.1　従来の転置ファイルの物理編成法

7.1.1　転置ファイルの基本的な物理編成法

転置ファイルは登録文書から抽出された索引単位の出現情報を索引単位ごとに記録するものである。索引単位（n-gram 索引では n-gram）ごとに出現情報を記録するデータ構造が転置リストで、索引文字列・文書頻度（索引単位が出現する文書数）等の転置リスト全体に関わる情報を記録するヘッダー部と文書 ID・その文書における n-gram の出現回数・その文書における出現位置等の文書ごとの出現情報を記録するデータ部から構成される。転置リストのデータ部を模式的に示すと以下の通りである。

< 文書 ID=1, 頻度=3, 位置={6, 18, 42}>< 文書 ID=5, 頻度=1, 位置={30}>
...

転置ファイルは、図 7.1（図 2.14 の再掲）のように索引文字列に対応する転置リストを特定するための語彙ファイル（dictionary file）と転置リストの集合体である位置ファイル（posting file）から構成される。

検索高速化には、検索時のディスクアクセスを少なくすることが重要である。そのためには、各転置リストは単一の連続領域に配置することが望ましい。しかし、転置リストを常に単一連続領域に配置するには文書登録時のデータ移動量が増大し、登録コストは高くなる。すなわち、登録・検索性能は相反する傾向があるので、物理編成を検討する際にも対象アプリケーションに応じて両者のバランスをとる必要がある [Fal82]。例えば、WWW のサーチエンジンでは、一定期間ごとに索引を再作成するので検索性能中心に考えればよい。一方、オフィスにおける文書管理のための全文検索であれば、業務上作成された文書が随時登録されるので、登録性能も十分に考慮しなければならない。本研究の適用先には文書管理等も含まれているので、本章の議論では登録性能にも配慮する。

登録・検索の両性能に配慮した転置リストの物理配置法として、転置リストの大きさがディスクアクセスの単位であるページを超える場合には転置リストをリンクされた複数ページに分割する方法がある [Fal82]。この場合、転置リストは大きさに応じてショートリストとロングリストの２つの分類できる [Bro95a, Tom94]。

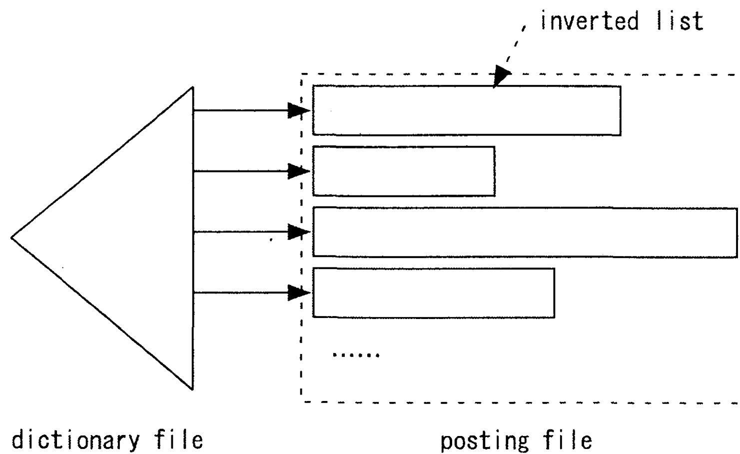

図 7.1: 転置ファイルの論理構成

- ショートリスト

  ショートリストとは大きさが 1 ページ以下で、1 ページに格納される転置リストである。転置ファイルを小さくするためには全てのショートリストができるだけ少ないページに収まることが望ましいので、1 つのページには詰められるだけのショートリストを格納する[1]。ショートリストの大きさはそれぞれ異なるので、ショートリストは図 7.2 のようなヒープ形式でページに格納するの [Zob93]。

  ショートリストはヘッダー部とデータ部から構成されており、両者が隣接するようにページ内に配置する。ヘッダー部には、索引単位、文書頻度等を記録する。データ部（図 7.2 の id/frequency/location の部分）には、文書 ID・文書内頻度・文書内出現位置を記録する。

- ロングリスト

  ロングリストとは大きさが 1 ページを超え、複数ページにまたがって格納される転置リストである。ロングリストの構成は図 7.3 のようになり、先頭のヘッダーページにその転置リスト全体の情報を記録するヘッダー部は先頭のヘッダーページに配置し、データ部はヘッダーページの残りの部分と残りのページに格納する。検索速度の点からは、ロングリストを構成するページはできる限り隣接した位置に配置することが望ましい [Zob93]。

7.1.2 転置ファイルの圧縮

ファイルサイズの小型化およびそれに伴う登録・検索の高速化のため、2.4.2 節で述べたように転置ファイルは圧縮するのが一般的である。通常、転置リスト単位に以下のように圧縮する [Wit94, Zob93]。

- 文書 ID は前回の出現との差分を可変長符号化により圧縮する

[1]ページ管理を簡単にするため、ショートリストとロングリストの一部が混在するページは考えない。

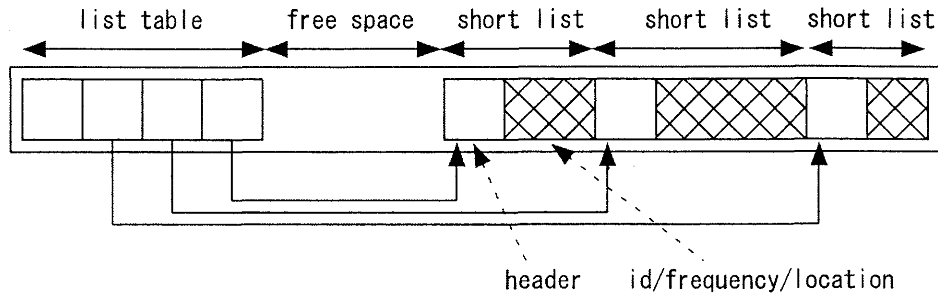

図 7.2: ショートリストのページでの格納

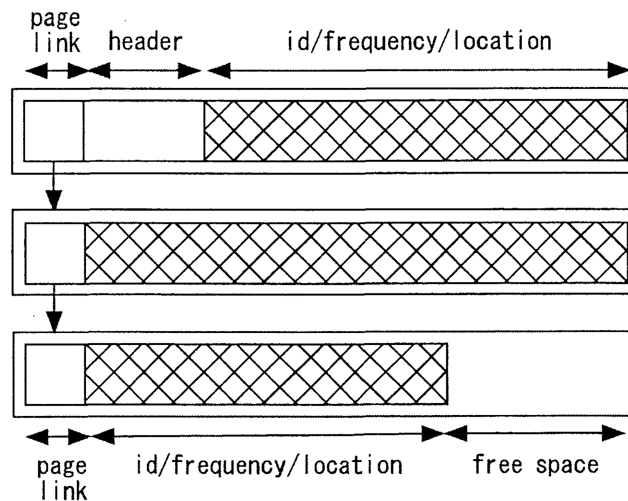

図 7.3: ロングリスト

- 出現頻度はそのままの値を可変長符号化により圧縮する
- 出現位置は前回の出現位置との差分を可変長符号化により圧縮する

転置リストの圧縮を模式的に示すと以下のようになる。

< 文書 ID=1, 頻度=3, 位置={6, 18, 42}>< 文書 ID=5, 頻度=1, 位置={30}>
···

↓（差分計算）

< 文書 ID 差分=1, 頻度=3, 位置差分={6, 12, 24}>< 文書 ID 差分=4, 頻度=1, 位置差分={30}> ···

↓（符号化）

< 文書 ID 差分='1', 頻度='011', 位置差分={'00110', '0001100', '000011000'}>< 文書 ID 差分='00100', 頻度='1', 位置差分={'000011110'}> ···

ここでは、2.4.2 節に示した符号化方式の 1 つである γ 符号 [Wit94] を用いた。

実際には、文書 ID 差分、頻度、出現位置差分によって値の分布が異なるため、それぞれの分布に応じた符号化手法を用いることが望ましい。適切な圧縮を施すことにより、文書内位置を記録しない文書レベルの単語索引であればもと文書の 10%程度、文書内位置を記録する単語

レベルの単語索引でもと文書の25%程度の大きさにまで索引を小型化できる [Wit94, Zob92]。ただし、これらの数値は不要語（stopword）を除いた場合の結果である。n-gram 索引の場合、文書から抽出される索引単位の個数が多いこと、文書内位置も記録する必要があることから、圧縮を施してももと文書と同等から2倍程度の大きさになる [Aka96b, Mat97a, Sug96]。それでも、圧縮しない場合の 20 〜 30%程度にまで小型化することが可能である [Mat97a]。

7.2 従来の物理編成の改良手法

7.2.1 基本的な物理編成法の問題点

前節に示した転置ファイルの基本的な物理編成法と圧縮法の問題点は十分な検索性能が得られないことである。例えば、1つの転置リストの文書IDだけを参照すればよい検索（n-gram 索引であればnに等しい検索文字列、単語索引であれば複合語ではない単純な単語に対するブーリアン検索）は以下の理由から効率的ではない。

- 転置リストがショートリストの場合、データ部には文書ID・文書内頻度・文書内出現位置が出現文書順に圧縮して記録されているので、文書IDだけを得るためにも文書内頻度・文書内出現位置を伸張しなければならない。すなわち、本来は伸長不要なデータが無駄に伸長されてしまい、検索が遅くなる。

- 転置リストがロングリストの場合、文書ID・文書内頻度・文書内出現位置が連続して記録されているので文書IDにアクセスする際には文書内頻度・文書内出現位置も同時に読み出さなければならない。すなわち、前述した伸長の問題に加え、本来はアクセス不要なデータに対して無駄にアクセスすることも問題となる。

一方、複数の転置リストのANDをとる検索では、前の値との差分を符号化するという文書IDの単純な圧縮法のために検索が非効率的になる。単純な圧縮法では、各転置リストの全ての文書IDを伸長した上でANDを求めなければならないが、転置リストに記録されている文書数が多い場合には全てを伸長するだけでもかなりの処理量となるからである。n-gram 索引の長い検索文字列に対する候補文書特定処理は転置リストの AND 検索として実現されるので、この問題は特に重要である。

これら問題を解決して検索を高速化するため、転置ファイルに関する様々な改良がこれまでにも提案されている。主な改良提案は以下の通りである。

- 文書IDと文書内出現位置の格納ページの分離

- 文書内出現位置の圧縮データ長の記録

- 圧縮のブロック化

- 圧縮符号化手法の改良

符号化手法の改良は転置ファイルの物理編成とは関係ない項目であるので除外し、それ以外の項目について次節以降で説明する。

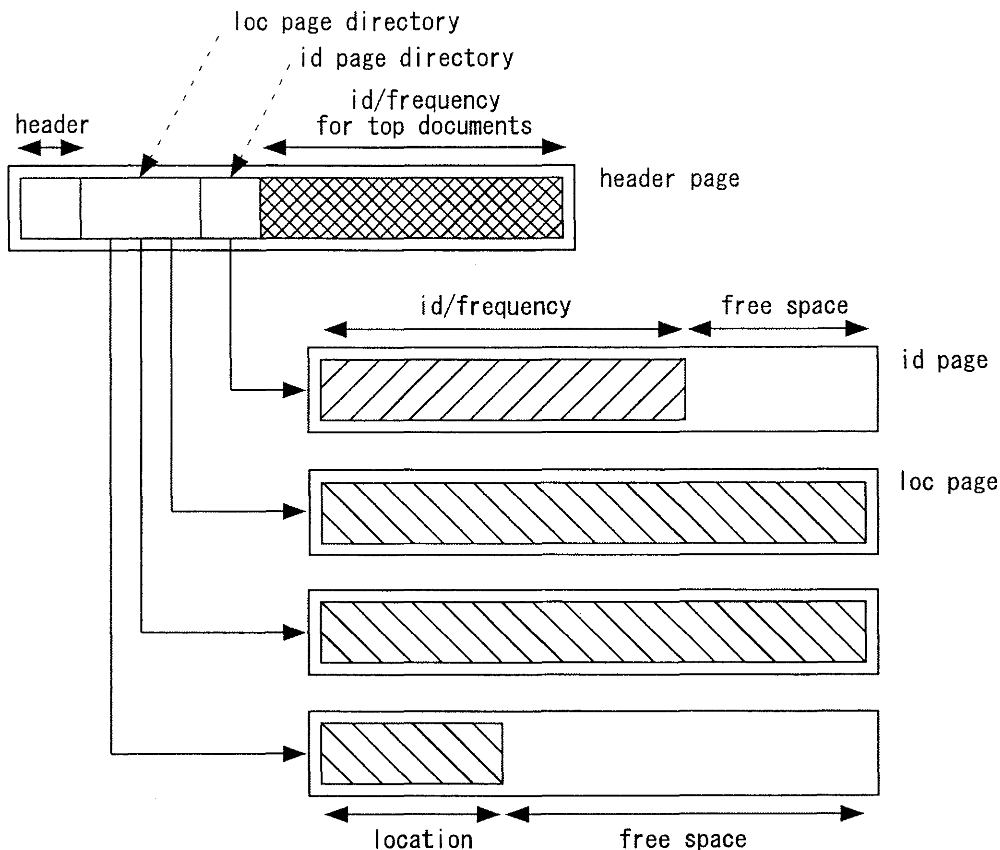

図 7.4: 分離型ロングリスト

7.2.2 文書 ID と文書内出現位置の格納ページの分離

文書 ID にアクセスする際に文書内出現位置等も必ずアクセスされる問題は、文書 ID と文書内出現位置を格納するページを分けることで解決できる [Bro95b]。Brown らによって提案された方法では、ロングリストを以下の 3 種類のページに分割して格納する。

- ID ページ：文書 ID と文書内頻度を格納する
- LOC ページ：文書内出現位置を格納する
- ヘッダーページ：ヘッダー部、ID ページを管理する ID ページディレクトリ、LOC ページを管理する LOC ページディレクトリを格納する

ここで、Brown が文書内頻度を文書 ID 側に配置したのは、文書内頻度が必要なランキング検索の高速化を目的としているからである。

Brown の提案するロングリストの構成を図 7.4 に示す。なお、ヘッダーページの余った部分には、文書 ID と文書内頻度のうち文書内頻度の大きいものを収まりきれるだけ ID ページと重複して格納する。ヘッダーページに格納される文書 ID と文書内頻度は、Pruning [Har90] による検索高速化のために用意したもので、Pruning の際に ID ページへのアクセスが不要となるため、検索高速化の効果が大きくなる。

文書内出現位置を分離した転置リストのデータ部はつぎのようになる。

< 文書 ID=1, 頻度=3, 位置={6, 18, 42}>< 文書 ID=5, 頻度=1, 位置={30}>
...

↓（位置情報の分離）

ID・頻度：< 文書 ID=1, 頻度=3>< 文書 ID=5, 頻度=1> ...

出現位置：< 位置={6, 18, 42}>< 位置={30}> ...

このように出現位置を分離した転置リストを分離型転置リストと呼ぶ。分離型転置リストにおいても通常の転置リストと同様に圧縮する可能である。前述の例であれば、圧縮結果は以下のようになる。

ID・頻度：< 文書 ID 差分='1', 頻度='011'>< 文書 ID 差分='00100', 頻度='1'>
...

出現位置:<位置差分={'00110', '0001100', '000011000'}><位置差分={'000011110'}>
...

分離型転置リストでは、出現位置が不要な検索においては LOC ページへのアクセスを省略できるだけでなく、出現位置の無駄な伸長処理も削減できるという利点もある。

なお、ショートリストに対しても文書内出現位置の分離は適用可能であり、その結果不要な伸長処理を削減できる。しかし、もともと 1 ページに収まるショートリストにおいて文書 ID と文書内出現位置を分離しても、文書 ID のみを必要とする検索においてはアクセスするページ数を削減することはできず、逆に出現位置が必要な場合には 2 ページにアクセスしなければならないという理由から、Brown らは出現位置の分離をショートリストには適用していない。

7.2.3 出現位置圧縮長の記録

出現位置情報の伸長を行うことなく次の文書に移動するには、各文書ごとに出現位置情報の圧縮したビット長（以下、出現位置圧縮長と呼ぶ）を記録しておけばよい [Mat97a]。出現位置情報が不要な場合にはビット長だけスキップすることで次の文書 ID の記録位置に移動できるので、不要な出現位置を伸長しなくても済ませられるからである。ファイルサイズの増大をできるだけ少なくするため、ビット長自身を可変長符号化して圧縮記録する。

先に示した転置リストに出現位置圧縮長を追加すると以下のようになる。

< 文書 ID=1, 頻度='011', 位置差分={'00110', '0001100', '000011000'}>< 文書 ID 差分='00100', 頻度='1', 位置差分={'000011110'}> ...

↓（圧縮長の追加）

< 文書 ID=1, 頻度='011', 圧縮長=21, 位置差分={'00110', '0001100', '000011000'}>< 文書 ID 差分='00100', 頻度='1', 圧縮長=9, 位置差分={'000011110'}> ...

↓（圧縮長の符号化）

< 文書 ID=1, 頻度='011', 圧縮長='000010101', 位置差分={'00110', '0001100', '000011000'}>< 文書 ID 差分='00100', 頻度='1', 圧縮長='0001001', 位置差分={'000011110'}> ...

なお、前節でも論じたように検索によっては出現位置だけでなく文書内頻度も不要な場合があるので、文書内出現頻度を含めた圧縮長を記録することも考えられる。しかし、ランキ

ング検索では文書内出現頻度は必ず必要となるので、出現位置の圧縮長を記録するほうがよい。

7.2.4 圧縮のブロック化

これまでに説明した2つの提案は出現位置のアクセス・伸長に関する効率化であるのに対し、圧縮のブロック化は文書IDの伸長を効率化するものである。具体的には、文書IDを先頭から連続して圧縮するのではなく、文書IDを適当にブロック化し、各ブロックの先頭の文書IDは直前の値との差分をとらずに記録し、残りの文書IDはこれまで通り直前の値との差分を可変長符号により圧縮する [Anh98, Mof94b, Mof96b, Mof95]。一方、検索は以下の2段階で実施する。

- ブロックの先頭の文書IDを用いて所望の文書IDが含まれる可能性があるブロックを特定する
- ブロック内部を順に伸長しながら探索する

この結果、先頭から全ての値を伸長する必要がなくなり、検索に要する伸長処理を大幅に削減できる。

ブロック化の方法はつぎの2つに大別できる。

- 固定数ブロック法：ブロックに格納する値の個数を固定とする方法
- 固定長ブロック法：ブロックの大きさを固定とする方法

以下、それぞれの方法について詳細に説明する。

固定数ブロック法

固定数ブロック法はブロックに格納する文書数を固定とする方法である。文書ID等の圧縮データサイズは文書ごとに異なるので、ブロックの大きさもブロックごとに異なる。

Moffatらの提案 [Mof94b, Mof96b] によれば、各ブロックには以下のようにデータを記録する。

- 先頭の文書IDは、前のブロックの先頭の文書IDとの差分を可変長符号により圧縮する。
- 先頭文書IDに続いて、次のブロックまでのビット長を圧縮して記録する。ただし、ビット長を伸長した時点では、先頭文書IDとビット長の大きさは既知となるので、ビット長として記録するのは現在のブロックの大きさではなく先頭を除いた残りの文書IDの圧縮ビット長の合計とする。
- 残りの文書IDは前の文書IDとの差分を可変長符号により圧縮する。

例えば、ブロック当たりの文書数を4とすると圧縮結果は以下のようになる（オリジナルは文書IDと文書内頻度を記録する転置リストに適用しているが、見やすさのために頻度は省略する）。

< 文書 ID=1>< 文書 ID=5>< 文書 ID=8>< 文書 ID=11>< 文書 ID=12>< 文書 ID=16>< 文書 ID=18>< 文書 ID=20>< 文書 ID=23> ···

↓（ブロック化）

[< 文書 ID=1>< 文書 ID=5>< 文書 ID=8>< 文書 ID=11>][< 文書 ID=12>< 文書 ID=16>< 文書 ID=18>< 文書 ID=20>][< 文書 ID=23> ···

↓（差分計算）

[< 文書 ID 差分=1>< 文書 ID 差分=4>< 文書 ID 差分=3>< 文書 ID 差分=3>][< 文書 ID 差分=11>< 文書 ID 差分=4>< 文書 ID 差分=2>< 文書 ID 差分=2>][< 文書 ID 差分=11> ···

↓（符号化）

[< 文書 ID 差分='1'>< 文書 ID 差分='00100'>< 文書 ID 差分='011'>< 文書 ID 差分='011'>][< 文書 ID 差分='0001011'>< 文書 ID 差分='00100'>< 文書 ID 差分='010'>< 文書 ID 差分='010'>][< 文書 ID 差分='0001011'> ···

↓（圧縮長の追加）

[< 文書 ID 差分='1', 圧縮長=11>< 文書 ID 差分='00100'>< 文書 ID 差分='011'>< 文書 ID 差分='011'>][< 文書 ID 差分='0001011', 圧縮長=11>< 文書 ID 差分='00100'>< 文書 ID 差分='010'>< 文書 ID 差分='010'>][< 文書 ID 差分='0001011', 圧縮長=10> ···

↓（圧縮長の符号化）

[< 文書 ID 差分='1', 圧縮長='0001011'>< 文書 ID 差分='00100'>< 文書 ID 差分='011'>< 文書 ID 差分='011'>][< 文書 ID 差分='0001011', 圧縮長='0001011'>< 文書 ID 差分='00100'>< 文書 ID 差分='010'>< 文書 ID 差分='010'>][< 文書 ID 差分='0001011', 圧縮長='0001010'> ···

固定長ブロック方法

固定長ブロック法はブロックの大きさを固定とする方法である。固定数ブロック法とは逆に、ブロックに格納される値の個数がブロックごとに異なる。

Anh らの提案 [Anh98, Mof95] によれば、各ブロックには以下のようにデータを記録する。

- 先頭の文書 ID は差分・圧縮をせずにそのまま記録する。
- 残りの文書 ID は前の文書 ID との差分を可変長符号により圧縮する。

固定数ブロック法とは異なり、ブロックの大きさが固定であるため、圧縮長をブロックごとに記録する必要はない。また、この方法では、先頭文書 ID の記録法が Moffat らの固定数ブロック法とは異なっている。固定長ブロック法でも Moffat らの方法を採用することも可能であるが、Anh らはブロックの決定時に伸長が不要で高速検索可能であることを鑑み、先頭文書 ID をそのまま記録している。

この方法による圧縮結果を以下に示す。

< 文書 ID=1>< 文書 ID=5>< 文書 ID=8>< 文書 ID=11>< 文書 ID=12>< 文書 ID=16>< 文書 ID=18>< 文書 ID=20>< 文書 ID=23> ···

↓（ブロック化）

[<文書 ID=1><文書 ID=5><文書 ID=8><文書 ID=11><文書 ID=12>][<文書 ID=16><文書 ID=18><文書 ID=20><文書 ID=23> ···

↓（差分計算）

[<文書 ID=1><文書 ID 差分=4><文書 ID 差分=3><文書 ID 差分=3><文書 ID 差分=1>][<文書 ID=16><文書差分 ID=2><文書 ID 差分=2><文書 ID 差分=3> ···

↓（符号化）

[<文書 ID=1><文書 ID 差分='00100'><文書 ID 差分='011'><文書 ID 差分='011'><文書 ID 差分='1'>][<文書 ID=16><文書差分 ID='010'><文書 ID 差分='010'><文書 ID 差分='011'> ···

ここでは、データ全体をブロック化・差分計算・符合化の順に処理するように図示しているが、実際にはデータの先頭から差分の計算、符号化を行ないながらブロックの範囲を決定するように処理は進む。なぜなら、各ブロックに詰められるデータの個数は値を圧縮しなければ決定できないからである。上の例では、ブロックの圧縮部分の大きさを 12 ビットとしており、5 個目の文書 ID=12 までであればビット長は 12 であるが、6 個目の文書 ID=16 までであればビット長は 17（12 に差分 4 を圧縮ビット数 5 を加えた値）となるため、最初のブロックには文書 ID=12 までを格納している。2 番目のブロックについては、先頭のブロックにどの文書 ID までが入るのかが決まってから値を詰めることとなる。

固定数ブロック法と固定長ブロック法の比較

固定数ブロック法は、領域に無駄が生じないのでファイルサイズを小型化できるという長所がある一方、ブロックサイズが一定ではないのでランダムアクセスは不可能であり、検索処理が複雑で速度の点でも不利となる。固定長ブロック法は、未使用領域ができることがあるため小型化の点では不利であるが、ランダムアクセスできるので高速化に向いている。ただし、ファイルサイズへの影響を考えた場合、実際にはブロックサイズは数 10 〜数 100 バイトであるのに対して未使用領域はせいぜい数バイトと小さいため、両者のサイズ差はわずかである。したがって、検索が高速である固定長ブロック法の方が望ましいと考えられる。実際、両者とも同一グループが提案しているものであるが、最近のシステムでは固定長ブロック法を採用している [Anh98]。

7.3 高速検索手法向きの転置ファイルの検討

本研究で提案した n-gram 索引の高速検索手法の基本アイデアは、長い検索文字列の処理で必要になる位置検査を可能な限り省略するというものである。以下では、位置検査省略向きの転置ファイルの物理編成を検討する。

n-gram の位置検査省略という観点からは、転置ファイルの構造に関して以下の項目を考慮すべきと考えられる。

- 検索文字列が n-gram に等しいか短い場合のブーリアン検索には文書内頻度・文書内出現位置は不要である。したがって、文書 ID だけにアクセスできる必要がある。

- 検索文字列が長い場合のブーリアン検索には、候補文書特定のために検索文字列から抽出された複数のn-gramによるAND演算子が多用される[2]。また、ランキング検索のAND方式による文書頻度の推定でも同様である。これら場合のAND処理では全てのn-gramが出現する文書IDを特定するため、転置リスト内を文書IDで高速検索できる必要がある。

- 候補文書が特定された場合には、その文書における出現位置情報に基づいて検索文字列の位置検査を行う。したがって、特定された文書の出現位置を高速に読み出せることが必要である。

これらの要請に対応するため、以下の方針を採用する。

- 転置リストをデータサイズに応じてショートリストとロングリストに分類する。

  1ページに収まるか否かに応じてショートリスト、ロングリストに分類する。この点では従来の方式と変わりはない。

- ショートリスト、ロングリストとも分離型転置リストとする。

  高速化のために分離型転置リストを採用する。ただし、ロングリストのみを分離型としたBrownらとは異なり、ショートリストも分離型とする。これは、ショートリストにおいても文書IDによる検索には不要な出現位置情報を分離することで、伸長処理のコストを最大限に削減するためである。ただし、ショートリストの2つの領域は、出現位置情報が必要な場合に新たなページアクセスが発生しないよう、同一ページに配置することとした。

- 分離型の構成では、文書IDと文書内頻度／文書内出現位置に分離する形式とする。

  文書IDと文書内出現位置を分離するという点では、Brownらと同じであるが、文書内頻度を文書内出現位置側に配置した点が異なる。このような配置としたのはn-gram索引の第1の要請事項を考慮して、文書IDによる転置リスト内の検索を最大限に高速化するためである。

- ロングリストでは、文書IDを固定長ブロック法によってブロック化する。

  文書IDを分離させるだけでは高速化に十分ではないので、伸長処理をさらに削減するためにブロック化を導入し、その方法にはブロック間の移動が高速に行える固定長ブロック化法を採用する。さらに、候補文書を特定した際にその文書に対応する出現位置に高速にアクセス可能とするため、文書IDのブロック化に同期するように文書内頻度／文書内出現位置もブロック化する。

  なお、ショートリストについては、1ページ内に格納できる文書数はそれほど多くないこと、ロングリストに比べると個数が多いためにブロック化によるデータサイズ増大の影響が大きいことを考慮し、ブロック化は行なわない。

- 出現位置圧縮長を記録する。ただし、文書内頻度が1である場合は例外として圧縮長は記録しない。

[2]n-gramに等しいか短い検索文字列のAND条件でも、n-gramのAND処理が行われる。

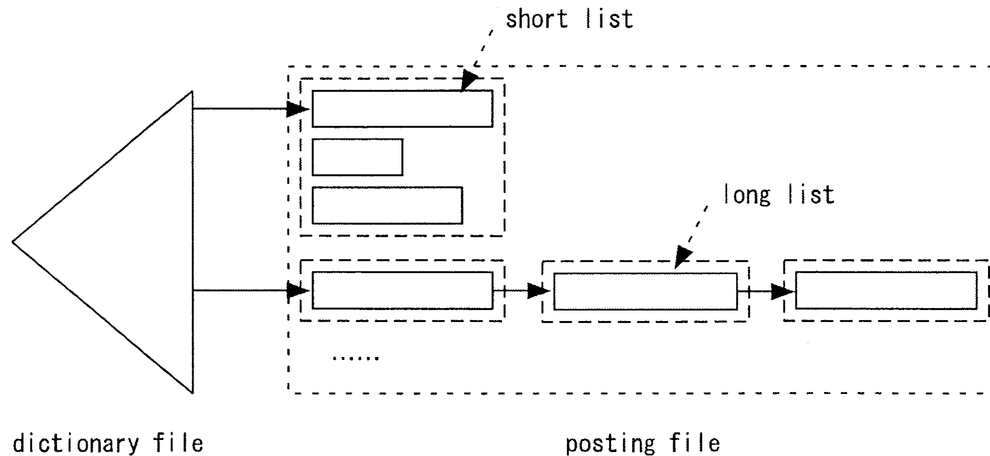

図 7.5: 高速検索手法向き転置ファイルの構成

所望の文書 ID に対する文書内頻度／文書内出現位置に高速にアクセスするために、文書ごとの出現位置圧縮長を記録する。ただし、7.2.3 節のように全ての文書について圧縮長を記録するのではなく、文書内頻度が 2 以上のものについてのみ記録することとした。この方式を圧縮長の適応型記録方式と呼び、詳細は 7.4.1 節で述べる。

さらに n-gram 索引向けということでは、本論文の高速化の対象として取り上げてはいない n-gram より短い検索文字列についても考慮する必要がある。短い検索文字列の検索では、検索文字列と先頭が一致する n-gram を OR 条件として処理するので、転置リストを辞書順にソートし、処理対象の転置リストをできるだけ少ない物理的に隣接したページに配置しなければならない。

以上の検討に基づき、図 7.5 のように転置ファイルは転置リストに対する B^+-tree ファイルとして実現した[3]。ショートリストを含むページには辞書順に連続するショートリストを配置し、辞書ファイルからはそのページの先頭の転置リストのみを指す。ロングリストはヘッダーページのみが辞書ファイルから指されており、それ以外のページはヘッダーページからのリンクによって辿る。

7.4 転置リストの構造

この節では、転置リストの構造をショートリスト・ロングリスト（以下では改良型ショートリスト・ロングリストと呼ぶ）にわけて詳しく説明する。ただし、まず両リストに共通である出現位置情報の圧縮長の適応型記録方式を説明する。

[3]辞書ファイルの形式としては B^+-tree 以外にも hash [Fol92] 等が候補となるが、辞書順にソートした索引単位を扱い易いことから B^+-tree が最適である。さらに、(1) 文書が追加（および削除）される動的環境にも向いていること、(2) 索引文字列の出現の偏りは大きい [Bro95a, Zip49] ので、hash の場合には適切な hash 関数を選択するのが難しいこと、の 2 点からも B^+-tree は辞書ファイルに向いている。

7.4.1 出現位置圧縮長の適応型記録方式

文書内頻度が 1 である場合には、出現位置圧縮長を記録しても次の文書の出現位置への移動を高速化できるわけではない。圧縮長も圧縮して記録しているため、圧縮長を得るにはそれ自体を伸長しなければならず、1 つの出現位置を伸長するのと同じことになってしまうからである。そこで、適応型記録方式では、出現位置圧縮長を文書内頻度が 2 以上の文書についてのみ記録する。その結果、全ての文書について圧縮長を記録するのと比べて、若干ではあるが転置ファイルを小型化できる。

この方式の転置リストは以下のような構成となり、文書頻度に応じて圧縮長が記録される場合（最初の文書)、記録されない場合（2 番目の文書）が混在する。

< 文書 ID=1, 頻度='011', 位置差分={'00110', '0001100', '000011000'}>< 文書 ID 差分='00100', 頻度='1', 位置差分={'000011110'}> ···

↓（圧縮長の追加：頻度 2 以上の場合のみ）

< 文書 ID=1, 頻度='011', 圧縮長=21, 位置差分={'00110', '0001100', '000011000'}>< 文書 ID 差分='00100', 頻度='1', 位置差分={'000011110'}> ···

↓（圧縮長の可変長コーディング）

< 文書 ID=1, 頻度='011', 圧縮長='000010101', 位置差分={'00110', '0001100', '000011000'}>< 文書 ID 差分='00100', 頻度='1', 位置差分={'000011110'}> ···

今回の検索処理の実装では、検索文字列の位置検査を行う際に、はじめにその文書に対応する出現位置を一括して伸長するのではなく、位置検査の進行に応じて必要な出現位置を逐次的に伸長する。したがって、検索文字列の出現（あるいは出現していないこと）が確定した場合には、その文書に関する残りの出現位置の伸長する必要はない。文書ごとに出現位置圧縮長を記録しておくことで、不要な出現位置を伸長することなくつぎの文書の先頭に伸長位置を位置付けることが可能となり、位置検査を行う場合の処理高速化にも貢献する。

7.4.2 改良型ショートリストの構造

改良型ショートリストは文書 ID と文書内頻度／出現位置を分離して記録する分離型転置リストである。単純には、ショートリストのヘッダーを除いたデータ領域を文書 ID 用と文書内頻度／出現位置用に分割すればよい。しかし、このような単純な方法では、それぞれの末尾に空き領域が生じるため、データ領域の使用率が下がるという問題点がある。この問題を解決するため、単一のデータ領域の両端からデータを詰める 2 端方式（double-ended method）[Mof95] を採用した。

改良型ショートリストを図 7.6 に示す。この図のように、文書内頻度／出現位置はデータ領域の前側から後に向かって、文書 ID は後側から前に向かって詰める。空き領域は中央部に 1 つだけ存在するので、単純な方法で生じる無駄はなくなり、分離しない場合と同じ大きさとなる。それぞれの領域ごとにデータの末尾位置を記録する必要があるので、従来型の末尾位置が 1 つである場合よりヘッダーサイズは大きくなるが、増加分は 2 バイトで十分であり、影響は小さい。

後ろ向きにデータを圧縮するためには、圧縮・伸長方法を後ろ向き用にも実装する必要がある。しかし、前向きが実装されていれば、後ろ向きの実装は簡単である [Mof95]。例えば、2.4.2 節で紹介した符合化方法に関しては、接頭部のビットの順序を反転させ、接頭部、区

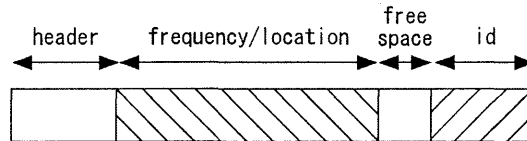

図 7.6: 改良型ショートリスト

値	unary	γ	Exponential Golomb ($k=2$)
1	1,	,1,	00,1,
2	1,0	0,1,0	01,1,
3	1,00	1,1,0	10,1,
4	1,000	00,1,00	11,1,
5	1,0000	01,1,00	000,1,0
6	1,00000	10,1,00	001,1,0
7	1,000000	11,1,00	010,1,0
8	1,0000000	000,1,000	011,1,0
9	1,00000000	001,1,000	100,1,0
10	1,000000000	010,1,000	101,1,0

図 7.7: さまざまな符合化手法の後ろ向きの圧縮結果

切り子、接尾部の順序を入れ替えれば後ろ向きが実現できる。図 2.15 に対応する後ろ向きのビットパターンを図 7.7 に示す。

7.4.3 改良型ロングリスト の構造

改良型ロングリストは、Brown らの提案した分離型のロングリストとは以下の点が異なっている。

- 文書 ID と文書内頻度／出現位置を分離した。

  改良型ショートリストで説明したように、n-gram 索引向きということで文書内頻度は出現位置側に配置した。

- 文書 ID を固定長ブロック法によってブロック化し、これとあわせて文書内頻度／文書内出現位置もブロック化した。

  分離した文書 ID を効率的に検索するため、固定長ブロック法にブロック化した。さらに、候補文書を特定した際にその文書に対応する出現位置に高速にアクセス可能とするため、文書 ID のブロック化に同期するように文書内頻度／文書内出現位置もブロック化した。

  これら 2 種類のブロック — 文書 ID を格納するブロックを**ID** ブロック、文書内頻度／出現位置を格納するブロックを**LOC** ブロックと呼ぶ — の関係を図 7.8 に示す。こ

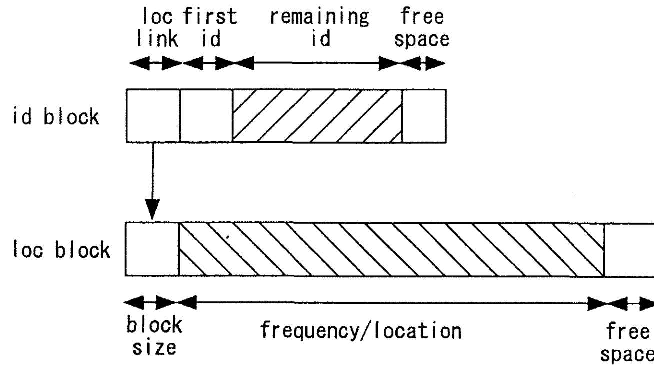

図 7.8: ロングリストにおける2種類のブロック

の図のように ID ブロックに LOC ブロックへのリンクを持たせたことで、文書 ID が特定された文書の文書内頻度／出現位置を高速に取得することができる。LOC ブロック内で特定された文書の情報を得るには、特定された文書が ID ブロックの何番目かを覚えておき、LOC ブロックを読み込んだ後、その数だけ文書内頻度／出現位置を読み飛ばす。

ID ブロックは固定長であるのに対し、文書内頻度／出現位置の圧縮長は文書ごとに異なるため LOC ブロックは可変長となる。したがって、図 7.8 に示したとおり、LOC ブロックの先頭部分にはブロックサイズを記録する。

- ID ブロックを記録するページを工夫した。

文書 ID だけにアクセス可能とするため、ID ブロックと LOC ブロックは原則として異なるページに配置する。LOC ブロックを格納する LOC ページを用意する点では Brown らと同じ方法をとったが、ID ブロックの格納ページは n-gram 索引向きに改良した。

Brown らの提案は単語索引におけるランキング検索の高速化を目的としているため、辞書ファイルから指されるヘッダーページのヘッダーおよびディレクトリ以外の部分を文書内頻度の大きい文書の文書 ID ／文書内頻度を記録するのに用い、全ての文書の文書 ID はヘッダーページとは別の ID ページに記録していた。しかし、n-gram 索引では、ブーリアン検索はもとより、ランキング検索する際にも検索文字列が n-gram と異なる長さの場合には文書 ID ／文書内頻度を直接的には利用することができない。そこで、改良型ロングリストでは、ヘッダーページの余った部分を ID ブロックの記録に用いる。ヘッダーページを ID ブロックの記録に用いることで、ブーリアン検索の候補文書特定ステップあるいはランキング検索であれば AND 方式による文書頻度の推定にはヘッダーページだけにアクセスすればよく、検索を高速化できるからである。

図 7.9 にブロックの配置を示す。この図で id と書かれている部分に ID ブロック、frequency/location の部分に LOC ブロックを配置する。ID ブロックはヘッダーページのみにあり、ヘッダーページのあいている部分は空き領域として残してある。一方、LOC ブロックは LOC ページに置かれ、最終の LOC ページには空き領域がある。なお、LOC ブロックは ID ブロックからリンクされているので、ヘッダーページに LOC

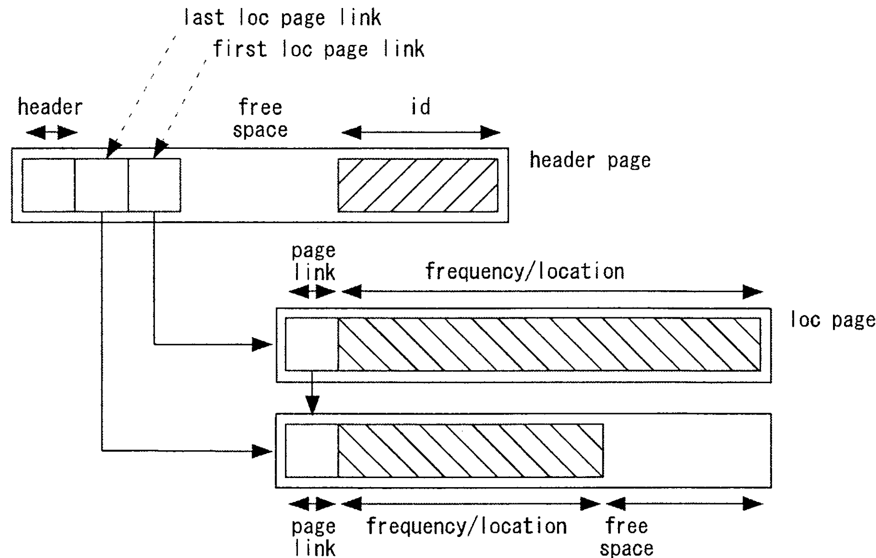

図 7.9: ロングリスト (LOC ページのみ)

ページディレクトリは必要なく、ヘッダーページからは先頭と末尾の LOC ページだけをリンクするようにした。

登録文書数が増大して ID ブロックがヘッダーページに収まらなくなった場合、あふれた ID ブロックはまず LOC ページに空き領域がある際には LOC ページに配置する。このように LOC ページの空き領域を有効活用することで、転置ファイルの小型化をはかっている。LOC ブロックと ID ブロックのページ内の管理を簡単とするため、LOC ブロックはページの前側から後ろ向きに、ID ブロックはページの後ろ側から前向きに配置する 2 端方式とした。この様子を図 7.10 に示す。この時点で、ヘッダーページには ID ブロックが含まれているページを管理するための ID ページディレクトリができ、ID/LOC 共通ページを指している。

登録文書数がさらに増大して ID ブロックが ID/LOC 共通ページに収まらなくなった場合、ID ブロックのみを含む ID ページに置かれるようになる。この様子を図 7.11 に示す。ID ページディレクトリのエントリは増加し、各 ID ページを指している。

ID ページディレクトリがヘッダーページに収まりきらない場合を考慮すると、ID ページディレクトリの一部を管理する形式が必要となる。しかし、ヘッダーページの大きさが 4K バイトとして約 1000 個の ID ページを管理でき、1 つの文書 ID を圧縮して平均 4 ビットであるとすると、4K バイトの ID ページあたりで 16384 個の文書を記録できるので、図 7.11 の形式で 1000 万個を越える文書まで管理できる。したがって、今回は図 7.11 の形式までを実装した。

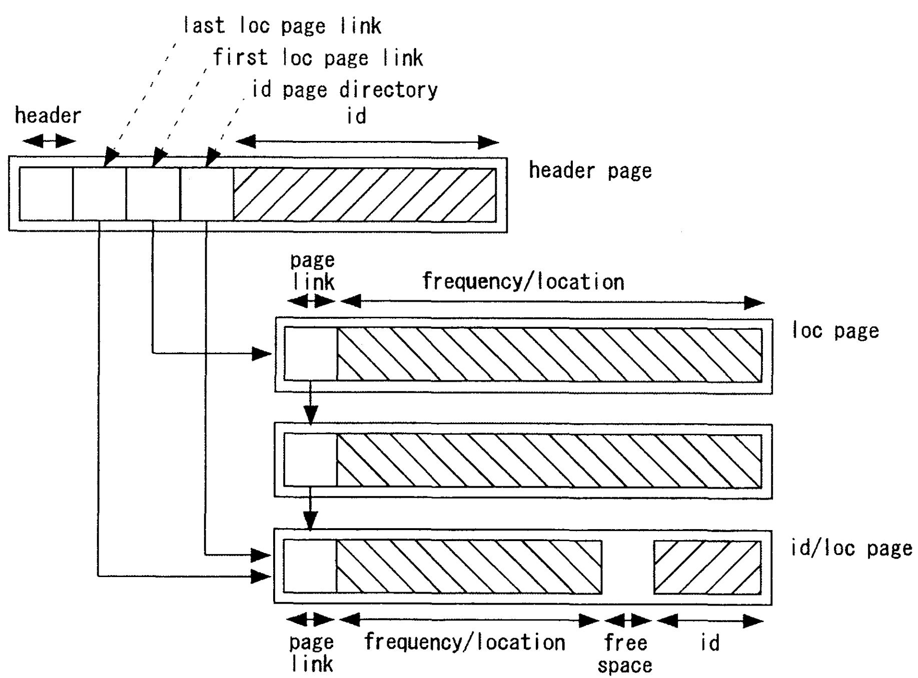

図 7.10: ロングリスト (ID/LOC ページあり)

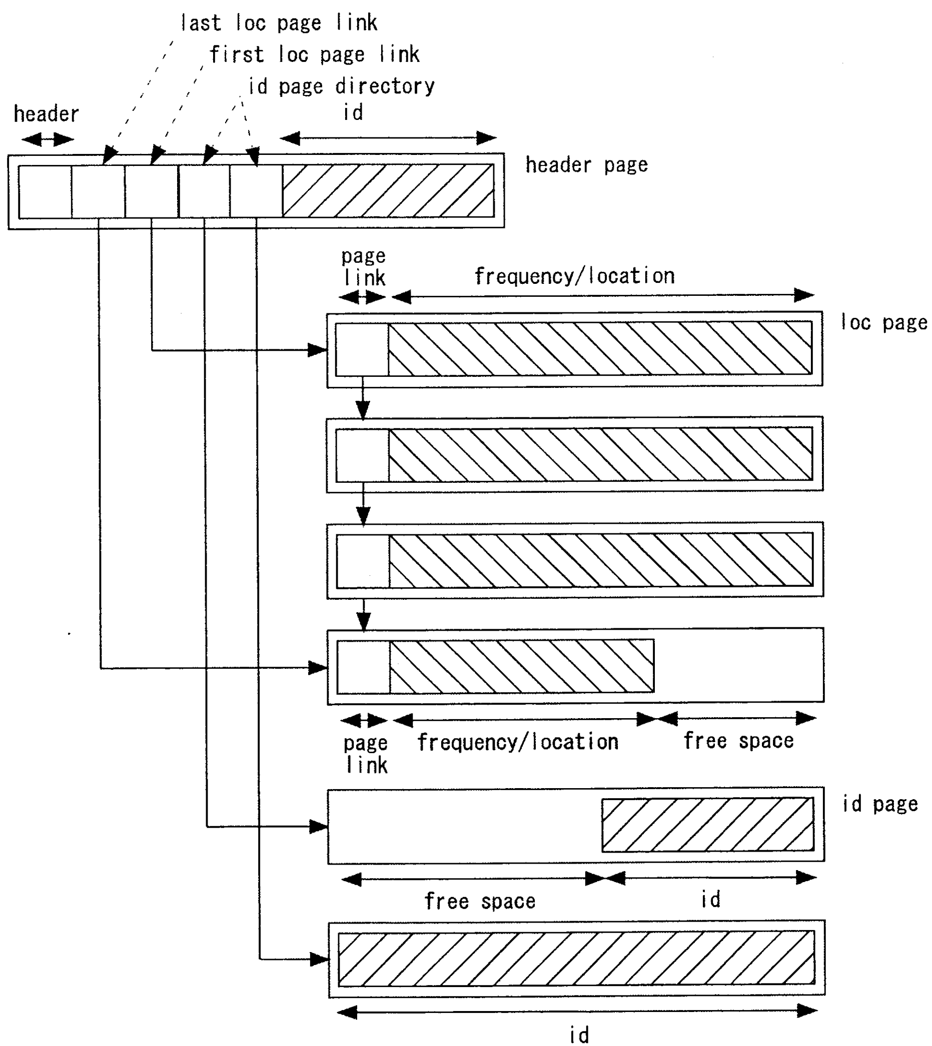

図 7.11: ロングリスト (ID ページあり)

7.5 高速検索手法向き転置ファイルの評価

7.5.1 実験概要

本章で提案した n-gram 索引のための高速検索手法向きの改良型転置ファイルの有効性を評価する。改良型転置ファイルが有効であることを実証するためには逐次的に圧縮する単純な転置ファイル（以下、逐次圧縮型転置ファイル）、Brown らの転置ファイル等を実装し、性能比較する必要がある。しかし、それら全てを実装するのは作業量的に困難であるため、ここでは改良型転置ファイルを用いてそれ以外の従来型の場合をシミュレートすることで性能比較する。ここでの比較結果はシミュレートであるため必ずしも正確とは言えないが、性能差異が大きければ改良型の有効性は確認できると考えられる。

評価実験では、まず、改良型転置ファイルの３つの特徴を ON/OFF することで、その有効性を検証する。

- 文書内出現位置の適応型圧縮長記録

  改良型転置ファイルには文書内出現位置の圧縮長を文書内頻度に応じて記録しており、検索処理時に不要な出現位置は伸長することなく次の文書の処理を進むことができる。この機能を OFF するには、圧縮長が記録してある場合にも全ての出現位置を伸長することで次の文書に進むようにすればよい。

  ベースラインとなる転置ファイルを実装した場合と比較すると、(1) 圧縮長を記録する分だけファイルアクセスが増大すること、(2) 圧縮長を伸長する分だけ CPU 処理が増大することの２点からシミュレーションのほうが検索時間も多めになると考えられる。

- 位置情報の分離

  改良型転置ファイルは文書内頻度・圧縮長・出現位置を文書 ID とは分離して格納している。したがって、検索処理時に出現情報が不要な場合には全く伸長する必要がなく、さらにロングリストであればファイルアクセスも不要となる。この機能を OFF するには、ある文書の文書 ID を伸長する場合には位置情報が不要であっても必ず文書内頻度・圧縮長・出現位置を伸長すればよい。ロングリストの場合、ID ブロックにアクセスする際には LOC ブロックにも必ずアクセスすればよい。

  位置情報の分離だけを実現する転置ファイルと比較すると、ブロック化ごとに生じる空き領域、ID ブロックの先頭文書 ID を圧縮しないことによる増加分、および ID ブロック中の LOC ブロックへのポインタの分だけファイルサイズが増大する。この増大が検索処理時のファイルアクセスを増大させる可能性があり、検索時間が多めになると考えられる。ただし、ファイルアクセスに差が生じる可能性があるのはロングリストの場合だけであり、その比率はそれほど大きくないと考えられる。

- 圧縮のブロック化

  改良型転置ファイルは文書 ID および位置情報をそれぞれブロック化して圧縮しておき、検索時にはブロックごとの先頭文書 ID を使って不要なブロックをスキップして伸長処理回数を削減している。この機能を OFF するには、転置リスト内を文書 ID で検索する際にブロックを考慮せずに全ての値を伸長すればよい。

実際には、3つの特徴のON/OFFの全ての組み合わせを比較するのではなく、以下の4つの場合のみ比較した。

- ベースライン

  文書ID・文書内頻度・出現位置が逐次的に圧縮・記録されている最も単純な転置ファイルである。全ての3つの機能をOFFすることでシミュレートできる。

- 適応型圧縮長

  文書内頻度の圧縮長を適応的に記録する転置ファイル。この機能だけONとし残り2つをOFFすることでシミュレートできる。

- 位置情報分離

  適応型圧縮長記録に加えて、文書内頻度・出現位置を文書IDと分離して格納する転置ファイル。適応型圧縮長・位置情報分離をONとし残りのブロック化だけをOFFすることでシミュレートできる。

- 圧縮ブロック化

  位置情報に加えて、圧縮をブロック化する転置ファイル。全ての機能をONとした改良型転置ファイルそのものである。

つぎに改良型転置ファイルの圧縮のブロック化におけるブロックサイズの影響を評価する。ブロックサイズは、ロングリストにおける文書IDの検索速度に影響する。IDブロックのデフォルトの大きさは64バイトであり、これまでの性能評価は全てこの場合の性能結果である。以下では、IDブロックを16, 32、128, 256バイトについても測定を行い、デフォルト値と比較する。

これら2つの評価では、単一検索文字列のブーリアン検索（4章）によって性能測定を行った。検索手法には4章の評価によって最も高速であった選択n-gram法を用いた。改良型転置ファイルの物理編成の有効性の検証のためには、4章で作成した改良型転置ファイルによるbi-gram索引を用い、他の構成の転置ファイルをシミュレートした。ブロックサイズの影響調査のためには、様々なブロックサイズの転置ファイルを作成し、検索性能の測定を行った。

7.5.2 物理編成法の相違に関する評価結果

物理編成の相違に関する評価結果を表7.2に示す。この表において各行の下段はベースラインに対する削減率を表す。

表7.2からベースラインに対し文書内出現位置の適応型圧縮長記録、位置情報の分離、圧縮のブロック化と改良を加えることにより検索時間が大幅に短縮されており、改良型転置ファイルの有効性が確認できる。ベースラインに対して最左欄の改良型転置ファイルは、COLDで検索時間が約半分に、HOTであれば1/10以下になっている。ただし、これは改良型転置ファイルを用いてベースライン等での検索をシミュレーションした結果であるので、高速化の効果が実際よりも大きくなっていると思われる点に注意が必要である。

改良ごとの特性は以下のようにまとめることができる。

表 7.1: 高速化手法の比較

		ベースライン	＋適応型圧縮長	＋位置情報分離	＋圧縮ブロック化
検索時間	COLD	1.886	1.779	1.001	0.911
		—	−5.7%	−46.9%	−51.7%
	WARM	1.656	1.534	0.768	0.671
		—	−7.4%	−53.6%	−59.5%
	HOT	0.925	0.815	0.167	0.083
		—	−11.9%	−81.9%	−91.0%
ページ数	全体	83.1	83.1	64.0	63.4
		—	±0.0%	−22.9%	−23.7%
	評価	69.9	69.9	50.9	50.3
		—	±0.0%	−27.2%	−28.1%
伸長処理数		697332	591387	197405	94865
		—	−15.2%	−71.7%	−86.4%

- 適応型圧縮長は伸長処理数を15%削減するが、シミュレーション実験であるためアクセスページ数は全く変化していない。したがって、検索時間短縮への効果はHOTの場合に最も大きく、処理時間の中にファイルアクセスが含まれるWARM/COLDではHOTよりも短縮の割合は小さくなっている。

- 位置情報分離はアクセスページ数・伸長処理数の両方を大幅に改善する。ページ数は20%以上、伸長処理数はベースラインに対し70%以上、適応型圧縮長に対してでも約66%削減している。ページ数の減少が相対的に少ないのは、ページ数が減少するのはロングリストでLOCページに候補文書が全く含まれていない場合に限定されるからである。HOTの方がCOLDよりも検索時間が短縮されているのは圧縮長の場合と同じであるが、COLDでもベースラインの約半分になっており、位置情報分離が検索高速化に非常に有効であることがわかる。

- 圧縮のブロック化は主に伸長処理数の削減に有効であり、位置情報分離したものに対して約半分（削減率51.9%）、ベースラインに対しては86.4%もの削減効果がある。アクセスページ数も減ってはいるものの、その割合はごくわずかである。検索時間削減への効果はこれまでと同じくHOTの場合が大きく、位置情報分離したものに対して約半分（削減率50.3%）、ベースラインに対しては1/10以上（91.0%）に短縮されている。COLD/WARMにおいても、検索時間をベースラインに対して半分以下に短縮しており、十分有効である。

なお、4章の評価では、検索文字列ごとに位置検査した回数の合計である位置検査数も測定・比較していた。しかし、位置検査数は物理編成法にはよらずに一定値となるので、表7.2には掲載しなかった。

表 7.2: ブロックサイズによる登録時間等の比較

	16B	32B	64B	128B	256B
登録時間	15.5	14.7	14.3	14.2	14.2
	+8.4%	+2.8%	—	−0.7%	−0.7%
ファイルサイズ	1917	1651	1574	1546	1534
	+21.8%	+4.9%	—	−1.8%	−2.5%

表 7.3: ブロックサイズによる検索時間等の比較

	16B	32B	64B	128B	256B
| COLD	1.030	0.955	0.911	0.917	0.952
検 |	+13.1%	+4.8%	—	+0.7%	+4.5.%
索 | WARM	0.774	0.698	0.671	0.674	0.696
時 |	+15.4%	+4.0%	—	+0.4%	+3.7.%
間 | HOT	0.077	0.079	0.083	0.092	0.103
|	−3.6%	−4.8%	—	+10.8%	+24.1%
ペ | 全体	76.1	66.4	63.4	62.7	62.5
ー |	+20.1%	+4.7%	—	−1.1%	−1.5%
ジ | 評価	63.0	53.3	50.3	49.6	49.4
数 |	+25.4%	+5.9%	—	−1.4%	−1.8%
伸長処理数	39235	66513	94784	125344	157620
	−58.6%	−29.8%	—	+32.2%	+66.3%

7.5.3 ブロックサイズの相違に関する評価結果

ここではIDブロックのブロックサイズの影響に関する評価結果を示す。表7.2はブロックサイズごとの登録時間・ファイルサイズをまとめたものである。ブロックサイズに伴って登録時間・ファイルサイズが小さくなっている。これは、ブロックサイズを小さくすることでその個数が増大すると、各ブロックの先頭文書IDは圧縮せずに記録するために文書IDに関する圧縮率が下がることと、ブロック数の増大に伴い空き領域が増大するためにファイルサイズも大きくなるからと考えられる。また、登録時間が増大するのは、ファイルサイズが増大することで書き込みに要する時間が増大するためである。

ブロックサイズごとの検索性能をまとめたものが表7.3である。COLD/WARMでは、検索時間はブロックサイズ64Bにおいて最小となっており、今回の改良型転置ファイルの実装においてはデフォルト値に設定した64Bが最適値であることが確認できた。一方、HOTの場合は16Bにおいて最小となっており、COLD/WARMとは若干傾向が異なっている。このようにCOLD/WARMとHOTで最適値が異なる理由は、ブロックサイズとアクセスページ数・伸長処理数の関係から説明できる。すなわち、アクセスページ数は転置ファイル全体のサイズ（表7.2参照）に応じて変化するが、転置ファイルはブロックサイズが大きいほど小さ

くなるのに対し、伸長処理数はブロックサイズが小さくなる程ブロック単位にスキップできる場合が増大するため、ブロックサイズが小さいほど伸長処理数も少ない。COLD/WARMの検索時間は主にアクセスページ数に決定されるが、ページ数の差が小さい場合には伸長処理数によっても影響されるため、ブロックサイズ64Bが最適値となっている。一方、HOTの検索時間は主に伸長処理数によって決定されるため、今回の実験において16Bの場合に最小となった。

第8章　実システムにおける評価

前章までに提案したn-gram索引による高速検索手法はテキストの登録・検索を行う全文検索エンジンであるFTS (Full-text Search) サーバとして実用化した。

全文検索エンジン単体ではさまざまなユーザニーズを実現するアプリケーションプログラムにはなりえないので、FTSは拡張関係モデルに基づくデータベース管理システムG-BASE [Hir82] のサブモジュールとして動作し、G-BASEの全文検索機能の強化に貢献している。G-BASEは単体で販売されるとともに、図書管理システムLIMEDIO等の業務アプリケーションとしても販売されている。さらに、FTSはオフィス向けの文書管理システムRidoc Document Serverシリーズの中でも全文検索エンジンとして使用されている。

本章では、FTSについて紹介するとともに、他の商用文書検索システムとの性能比較結果を示す。

8.1　FTSの概要

FTSは本研究で提案した高速文書検索手法に基づいて筆者の所属する (株) リコー・画像システム事業本部ソフトウェア研究所[1]において開発されたサーバ・クライアント型の文書検索システムである。FTSはWindows、Solaris (SUNのUNIX) およびLinux上で動作する。本節では、FTSの概要を紹介する。

8.1.1　データモデル

FTSは文書の登録・検索等のデータ操作の枠組みとしてリレーショナルデータモデル [Cod70] を採用した。ただし、実装を簡単にするため、文書の登録・検索に必要最小限な機能として、以下の非常に限定した仕様とした。

- スキーマは固定であり、ただ1つのスキーマに従ったテーブルのみ作成できる
- データベースはシステムに1つであり、全てのテーブルは同一のデータベースに属する
- 1つの問合せは1つのテーブルのみを対象とし、複数のテーブルにまたがった問合せは実行できない

FTSが提供するスキーマは以下の形式である。

- ID:

  登録した文書を識別するための識別子。4バイトの符号あり整数値。前章までで文書IDと呼んでいたものであり、文書登録時にFTSによって付与される。なお、テーブ

[1]現在は (株) リコー・ソフトウェア研究開発本部である。

ルから削除された行の ID は欠番となり、再度別のテキストデータに与えられることはない。

- DATA:

  文書のテキストデータ。符号なし文字列。文字コードは UNICODE[JIS95d] であり、クライアントライブラリからは UTF-8 エンコーディングで受け渡しする。

- SCORE:

  文書スコア。8 バイトの浮動小数点値。仮想的なフィールドであり、ランキング検索結果の文書スコアが置かれる。

DATA フィールドに対する検索により文書検索が実施できる。単にテーブルを作成した場合には、DATA フィールドの文書データを 1 件ずつ読出して文字列照合を行う逐次処理で文書検索が実施される。一方、テーブルの DATA フィールドに対して索引を作成することが可能であり、索引作成指示に基づいて本研究の成果である n-gram 索引が作成される[2]。索引のあるテーブルに対する検索は n-gram 索引によって処理される。

FTS が実施可能な主なデータ操作は以下の通りである。

- テーブルの作成
- テーブルの破棄
- インデックスの作成
- インデックスの破棄
- テーブルへの文書登録
- テーブルからの文書削除
- テーブルからの文書検索
- テーブルのバックアップ
- テーブルのリストア

データ操作は SQL[JIS95a] に準拠した構文で行う。FTS は機能が限定されているため、サポートする SQL も極めて限定されているが、文書検索の観点からはランキング検索や近傍演算等の拡張を施した。

[2]開発当初は FTS としては n-gram 索引のみが作成可能であった。しかし、6.5 節の単語索引との性能評価実験用に転置ファイルを英語に対する単語索引も作成できるように機能拡張した。その後、日本語に対する単語索引もサポートできるように改良し、現在では FTS としても日本語・英語に対する単語索引を正式にサポートしている。

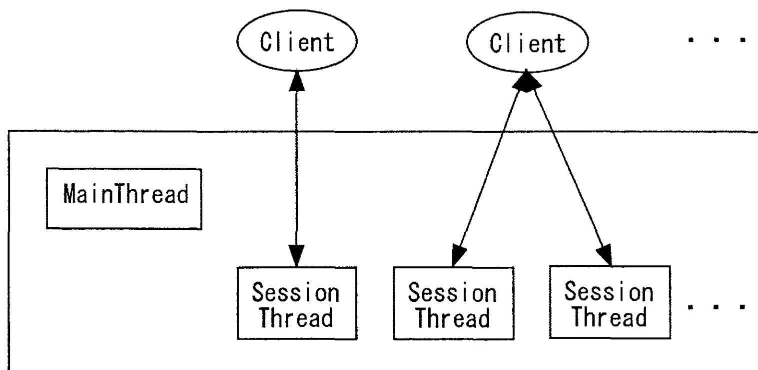

図 8.1: FTS のプロセス構成

8.1.2 システム構成

プロセス構成

FTS は 1 プロセスでマルチスレッド型のサーバ・プログラムであり、Windows ではサービスとして、Solaris/Linux ではデーモンプログラムとして起動される。図 8.1 に FTS のプロセス構成を示す。FTS の MainThread はクライアントからの要求に応じて、クライアントとのセッションを確立し、セッションごとにそれ以降の処理の実行を行う SessionThread を生成する。1 つのクライアントが複数のセッションを有することも可能である。

モジュール構成

FTS は MOD ライブラリを使用して作成されている。MOD (Multimedia On Demand) とはマルチメディアデータに対応したトランザクション制御、同時実行制御、障害回復機構等のデータベース機能を提供する C++ で書かれたクラスライブラリ群である。ファイル上でのデータ管理機能は論理ファイルドライバとして実装され、クライアントからの要求に対して FTS のサーバ・メイン部が適切な機能を呼び出す構成となっている。

FTS のモジュール構成を図 8.2 に示す。

- MOD 共通ライブラリー

  MOD、アプリケーションで利用できる汎用ライブラリ。メモリ管理、スレッド、シリアライズ、仮想 OS、文字列、正規表現照合、STL(Standard Template Library) 等のクラスがある。

- 論理ファイルドライバ

  論理ファイルドライバはデータの永続的な管理を実現するソフトウェアモジュールである。現状、可変長レコードの管理を行うためのヒープファイル、固定長レコードの管理を行うためのベクタファイル、文書検索用 n-gram 索引を実現する転置ファイルの 3 種類のドライバーが存在している。

- MOD マネージャ

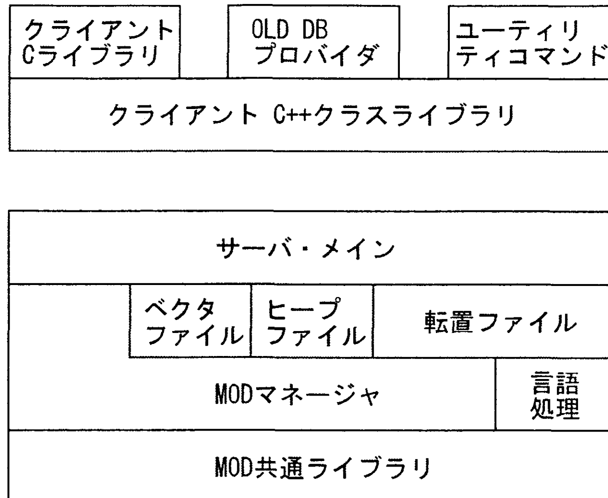

図 8.2: FTS のモジュール構成

バッファ管理、論理/物理ログ処理、論理ファイル、物理ファイル、リクエスト処理、トランザクション、システム管理のマネージャがある。

- 言語処理

  言語処理を行うライブラリ。本研究を実施した FTS の開発当初には、文字列の表記を統一する異表記正規化機能のみを提供していた。その後、自然文検索および単語索引を FTS に組み込むにあたってはテキストを単語に分割する形態素解析機能も提供するようになった。

- サーバ・メイン

  FTS のメイン部分で、前述のライブラリ・モジュール群を使用してデータモデルを実現する。

- クライアント C++ クラスライブラリ

  クライアントプログラムに、サーバとの接続、データ操作を提供するための C++のクラスライブラリである。FTS とクライアント間のプロセス通信にはソケットを用いる。

- クライアント C ライブラリ

  クラスライブラリの上位に作成した C ライブラリ。C 言語で開発されているクライアントのためのライブラリである。

- OLE DB プロバイダ

  Windows 版 FTS に固有のモジュール。クラスライブラリの上位に作成した、Microsoft が提唱するデータアクセスのための API である OLE DB (V1.1)[Mic97] に準拠した COM ライブラリである。

- ユーティリティコマンド

  FTS の運用管理用のコマンドで、以下のものがある。

- ftsdown: サーバの停止
- ftsstat: クライアント情報の取得
- ftsinfo: テーブル情報の取得
- ftscptable: テーブルのコピーの作成
- ftsmvtable: テーブルの格納場所の移動
- ftscheck: 整合性検査
- ftsdump: システムあるいはテーブルのダンプ
- ftsrestore: システムあるいはテーブルのリストア

G-BASEはクライアントCライブラリを、Ridoc Document SystemはOLE DBプロバイダを介してFTSにアクセスしている。

なお、4章・6章の評価実験における検索時間はクライアントライブラリで測定したのではなく、転置ファイルドライバのAPIを用いて測定した。これは、サーバ・クライアント通信等のオーバーヘッドを排除するためである。

8.1.3 n-gram索引（転置ファイル）のパラメータ

FTSに組み込まれたn-gram索引には、さまざまなチューニングパラメータが用意されており、登録文書および検索文字列の特性に応じてn-gram索引の動作を調整することができる。チューニングパラメータ（とそのデフォルト値）は以下の通りである。

- n-gram抽出方式：

  文字種に関係しない単純n-gram抽出法、文字種に応じて切り出し長を調整可能な文字種依存型n-gram抽出法のいずれかを選択することができる。単純n-gram抽出法ではn-gramの長さ、文字種依存型n-gram抽出法では文字種ごとのn-gramの長さと文字種を跨いで切り出しを行うか否かを文字種の組み合わせごとに指定することができる。

- ページサイズ：

  転置ファイルの物理的アクセスの単位であるページの大きさ。キロバイト単位で指定できる。

  指定は省略可能であり、その場合のデフォルト値は4Kバイトである。

- IDブロックサイズ：

  改良型ロングリストのIDブロック（7.4.3節参照）の大きさ。バイト単位で指定できる。

  指定は省略可能であり、その場合のデフォルト値は64バイトである。

- 圧縮方式：

  転置ファイルの圧縮方式を調整するものである。本論文の転置ファイルでは、7章で述べたように文書ID、文書内頻度、出現位置の圧縮長、出現位置を圧縮しており、それぞれの圧縮方式を指定できる。指定可能な圧縮方式には、2.4.2節にあげたunary符号、γ符号、Exponential Golomb符号（パラメータも指定可能）等がある。

指定は省略可能であり、その場合は Exponential Golomb 符号が選択される。なお、Exponential Golomb 符号のパラメータは圧縮対象に応じて異なっており、文書 ID では $\lambda = 2$、文書内頻度では $\lambda = 3$、出現位置の圧縮長では $\lambda = 6$、出現位置では $\lambda = 6$ である。

これらの値はテーブルごとに調整可能であり、インデックス作成時に指定する。また、インデックス作成時には、利用目的に応じて異表記正規化を行うか否かを指定することができる。異表記正規化時には語幹処理を行うか否かも指定することができる。

4 章・6 章の評価実験では、n-gram 抽出方式としては単純 n-gram 抽出法で抽出長さを 2 とし、ページサイズ、ID ブロックサイズ、圧縮法はデフォルト値を使用した。次節に示す FTS レベルでの性能比較でも同様である。

8.1.4 DBMS との連携

FTS のテーブルのスキーマは 8.1.1 節で述べたような非常に単純な形式に固定されており、作成者・作成日等の書誌事項を含むアプリケーションに必要とされる文書以外のデータ管理には不十分である。したがって、本格的な文書管理等のアプリケーション作成には他のデータベース管理システム（DBMS）との連携が不可欠である。

FTS と DBMS との連携には 2 つの方法が考えられる。

- アプリケーションが連携させる方法

  文書内容以外のデータは DBMS、文書内容のテキストは FTS というように役割分担してデータ管理を行う。この場合、アプリケーションプログラマが両者を別個に呼び出す必要があり、書誌検索と文書検索を組み合わせた場合の最終的な結果を得るための処理もコーディングしなければならないため、プログラマへの負担が大きい。その一方で、文書検索だけであればオーバーヘッドが一切ないために高速であること、DBMS を自由に選択できること等の利点もある。

  リコーのオフィス向け文書管理システム Ridoc Document Server はこの方法によって DBMS と FTS を連携させている。DBMS にはマイクロソフト社の SQL Server を採用して、文書の書誌項目、ユーザ情報等を管理する。なお、FTS はマイクロソフト社のデータアクセス API である OLD DB をサポートすることで、プログラマの連携のための負担を小さくするよう工夫している。

- DBMS が FTS を呼び出す方法

  文書検索を DBMS の機能として組み込み、例えば可変長テキストデータ型のフィールドの管理を FTS を代行させる方法である。そのフィールドへのデータ挿入は FTS へのデータ挿入となり、検索は FTS への検索となる。DBMS に組み込むことにより、アプリケーションプログラマは FTS を意識することなく高速文書検索を利用することができる。しかし、FTS と DBMS 間の通信や DBMS 側の処理がオーバヘッドとなり、FTS 単体で使用する場合よりは処理速度が低下する。

  リコー・ソフトウェア研究所では G-BASE[Hir82] という DBMS を研究開発してきており、FTS の開発にあわせて G-BASE と FTS を連携させるよう G-BASE も改良し

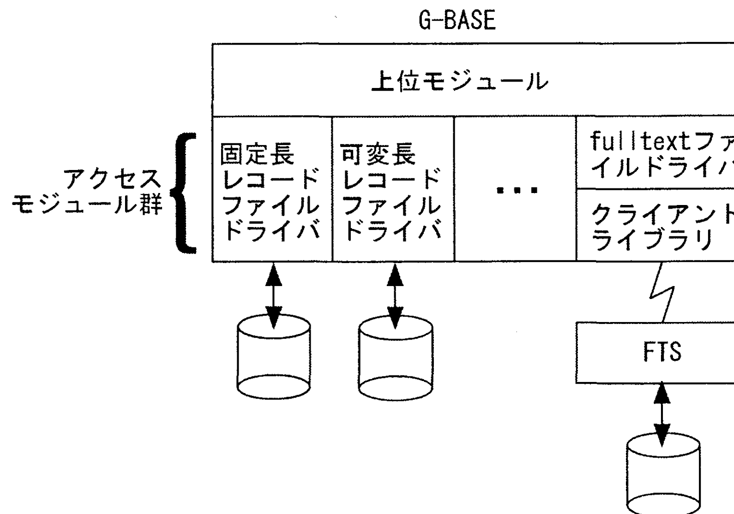

図 8.3: FTS と G-BASE の連携

た。FTS/G-BASE の連携にあたっては、fulltext 型というテキスト用の新しいデータ型を導入し、fulltext 型のフィールドを用いることで FTS の機能を簡単に呼び出せるようにした。FTS と G-BASE の連携の様子を図 8.3 に示す。G-BASE にはレコード型に応じたファイルアクセスモジュール群が用意されているが、fulltext 型のアクセスモジュールは FTS の C クライアントライブラリを用いて FTS と通信し、テキストデータの登録・検索等を行う。

図書管理システム LIMEDIO 等では G-BASE を用いてアプリケーションを構築している。LIMEDIO の開発には FTS ができる以前から G-BASE を使用してきたが、G-BASE から FTS を呼び出せるようにしたことで、FTS 開発後もアプリケーションである LIMEDIO 側のコード書き換えを最小限に抑えながら高速文書検索をエンドユーザに提供することができた。

8.2 性能比較

4 章・6 章では n-gram 索引の検索高速化という観点から性能評価を行ってきたが、ここでは本論文で提案する n-gram 索引を組み込んだ FTS の有効性を検証する。そのため、FTS を他の商用文書検索システムと性能比較する。

8.2.1 評価データ

評価には、特許明細書 20 万件を使用した。特許庁が配布している特許 CD-ROM には書式に従って SGML でタグ付けされているが、そのうちの発明の説明部分を SGML のタグを除去してテキストだけを抜き出したものを評価に使用した。テキストサイズは 3.6GB（1 件当たり約 20KB）であった。

索引付けには、4・6 章の bi-gram 索引ではなく、文字種に応じて切り出し長を調整する文字種依存型 n-gram 抽出法を用いた。文字種ごとの切り出し長は以下の設定とした[3]。

- カタカナ・アルファベットに対しては tri-gram($n = 3$)
- それ以外の文字種に対しては bi-gram($n = 2$)
- 漢字・ひらがなの連続部分の bi-gram は抽出する
- 漢字・ひらがな以外の異なる文字種連続は抽出しない

8.2.2 検索条件

検索性能を測定するために以下のような検索条件を用意した。

- 単一検索文字列

  検索文字列として良く使用されるものとして、漢字だけから構成されるもの（1〜5文字の計100個）、カタカナだけから構成されるもの（2〜10文字の計140個）、異なる文字種から構成されるもの（2〜5文字の計80個）の合計320個。

- AND 演算子のみを含む検索条件

  検索文字列を2〜5個まで組み合わせた合計60個。検索文字列の長さ、構成文字種については特別の制約は加えていない（後述の OR、混在の場合も同様）。

- OR 演算子のみを含む検索条件

  検索文字列を2〜5個まで組み合わせた合計60個。

- AND,OR 演算子を含む検索条件

  AND, OR 演算子が少なくとも1つは含まれるようなもの30個。

検索条件はブーリアン検索の評価で使用したものとは異なっているが、同一の社内情報検索システムのログから選択した点は共通である。

検索時間の測定は、サーバ・クライアント環境ということで4.2節とは若干異なる以下の3つの状況で測定した。

- COLD：

  検索条件ごとにサーバを起動した直後の検索時間である。検索条件ごとの検索時間を3回測定した場合の平均値（1つの検索条件の検索時間）を全検索条件で平均した値を測定結果として示す。

- HOT：

  COLD の測定後に同一の検索条件を再度検索した場合の検索時間である。測定結果の求め方は COLD と同じである。

[3]この設定は、社内の特許システムの検索ログから検索文字列長の分布を調べ、その分布に対する平均検索時間と索引サイズのバランスを考慮して決定したものであり、FTS のデフォルト値として採用されているものである。

表 8.1: 性能比較結果

	X	FTS	
登録時間 (時間)	25:17	51:42	(+104%)
ファイルサイズ (GB)	13.30	5.39	(−59.5%)
COLD 検索時間 (秒)	43.75#	1.45	(−96.7%)
	5.75		(−74.8%)
HOT 検索時間 (秒)	31.33#	0.53	(−98.3%)
	1.36		(−61.0%)
連続検索時間 (秒)	27.37#	0.52	(−98.1%)

- 連続：

  サーバを起動した後に 200 個の検索条件を連続して処理した場合の、検索条件あたりの検索時間である。200 個は COLD/HOT の測定に使用した 470 個の検索条件から、検索ログでの検索条件タイプ（単一文字列の場合は文字種と検索文字列長、論理演算子を含む場合は論理演算子の種類と個数）は出現頻度に基づいて 200 個を並べたものである[4]。運用状態の平均的な検索時間を表すものと考えられる。

8.2.3 比較対象

本実験では、他社の商用文書検索システム（以下、X と呼ぶ）との性能比較をおこなった。比較対象としたものは、この比較実験を行った 1998 年において検索の高速性で市場の評判が高かったものである。その当時で既に 200 セット以上の販売実績があるとのことであった。

比較対象はブーリアン検索のみが可能であることから検索性能の比較はブーリアン検索で行った。

評価実験に使用したマシンは、Microtron 製の PC サーバ（CPU: Pentium Pro 200MHz x 2, メインメモリ 512MB）で、OS は Windows NT 4.0 である。データは Ultra Wide SCSI 接続の外付けディスク（126GB）に置いた。

なお、FTS と X とではクライアント API の設計が異なっている。FTS の API では検索実施によって文書 ID もクライアント側に送信するのに対し、X の API では検索関数では検索件数のみが得られ、検索結果の文書 ID は検索とは別の関数で 20 件ずつ取得しなければならない。したがって、X については検索関数の実行時間と結果取得まで行なった時間の両方を測定した（ただし、連続に関しては結果取得まで行った場合の時間のみを測定した)。

8.2.4 評価結果

登録時間・ファイルサイズ・検索時間（COLD/HOT/連続）の測定結果を表 8.1 に示す。なお、X の検索時間において# が付いている数字は検索結果と結果取得を行った処理時間

[4] 470 個のなかから選択したため、漢字 2 文字の検索文字列等の出現頻度の高い検索条件タイプの検索条件は 200 個の中には 2 回検索が行われるものがある。一方、選択されなかった検索条件も当然ながら存在している。

表 8.2: 検索条件タイプごとの検索時間比較結果

	COLD			HOT		
	X	FTS		X	FTS	
単一文字列	3.77	1.56	(−58.6%)	0.65	0.63	(−3.1%)
AND 演算子	11.68	1.22	(−89.6%)	4.32	0.30	(−93.1%)
OR 演算子	8.27	1.68	(−79.7%)	1.82	0.58	(−68.1%)
複合条件	10.59	1.15	(−89.1%)	2.96	0.34	(−88.5%)
平均	5.75	1.45	(−74.8%)	1.36	0.53	(−61.0%)

であり、そうでないものは検索結果を求めるのに要した処理時間である。

この表からわかるように、登録時間を除いて FTS が X を上回った。登録時間については、FTS は通常の動作環境にあわせるということで論理ログ・物理ログをとる状態での測定を行った[5]。FTS では論理ログ・物理ログを取らない設定にして文書登録を行うことも可能であり、その場合には登録時間は短縮できる。X には登録等の更新処理のロールバック等のトランザクション機能はなく、ログも取っていないと考えられる。したがって、表 8.1 の結果だけから FTS（本研究の成果である n-gram 索引）の登録性能が低いとは必ずしも言えないと思われる。

検索時間に関しては、以下の傾向がわかる。

- X では結果取得に膨大な時間がかかることがわかる。X の検索時間と結果取得を含めた処理時間を比較すると、COLD で約 8 倍、HOT では 20 倍以上も後者が時間がかかっている。この結果から、X は検索処理では検索結果をディスクに一時書き込み、結果取得においてディスクから検索結果を読み出していると予想される。

- FTS は X と比較して非常に高速である。結果取得を伴わない検索処理だけで比較しても、検索時間は COLD で 1/4 程度、HOT では半分程度である。結果取得を行う場合では、X との速度差はさらに広がり、検索時間は COLD で 1/30 程度、HOT で 1/60 程度である。

  以上のことから、FTS と X との速度差は結果取得を伴うか否かで傾向が異なることがわかる。すなわち、結果取得を伴う場合には COLD の方が差が小さいのに対し、結果取得を伴わない場合には HOT の方が小さくなっている。これは、検索処理においては索引がキャッシュされることによって処理時間が短縮されるのに対し、結果取得においては検索結果を常に書き込むためキャッシュが効かないからと考えられる。

- 運用時の検索時間を示すと考えられる連続条件での検索時間は HOT に近い値を示している（X については結果取得を含む場合しか測定していないので、その場合の COLD/HOT と比較した）。この結果からも FTS の高速性が検証できた。

検索処理の特性の違いを調べるため、検索時間についてさらに詳細に検討する。まず、表 8.2 に検索条件タイプごとに検索時間を示す。全てのケースにおいて FTS の方が高速であるが、詳細に見ると以下の傾向がわかる。

[5] 前章までの実験では、論理ログ・物理ログをとらない状態での測定であった。

- 単一検索文字列の場合に FTS と X の検索時間の差が小さい。特に HOT の場合に差が小さくなることがわかる。

- 論理演算子を含む場合には、FTS と X との差が単一検索文字列の場合よりも大きく、X は集合演算処理が遅いということがわかる。検索時間に関する考察から、X は検索結果をファイルに保存していることが予想されており、検索文字列を演算子で結合した場合には、検索文字列ごとに保存した検索結果の集合演算を行っていると考えられる。そのため、演算子を含む場合に、X と FTS の差が拡大しているのであろう。なお、単一検索文字列とは異なり、HOT/COLD による差異はほとんどない。

単一文字列の検索時間をさらに分析するため、検索時間に対する文字数の影響をカタカナを対象に詳細に調べる。検索文字列の長さに対する検索時間をプロットしたグラフを図 8.4・図 8.5 に示す。図 8.4 が COLD、図 8.5 が HOT の結果である。

FTS はすべてのケースについて X よりも高速であるが、検索文字列の長さと検索時間の関係も以下のように異なった傾向がある。

- FTS はカタカナに対しては tri-gram（3 文字組）でインデキシングしているので、3 文字の検索文字列の場合に最も高速である。3 文字以上のケースでは、4 文字が最も検索時間がかかり、それ以降は検索時間は短縮されている。この傾向は COLD/HOT に関係ない。

- X は COLD/HOT で若干挙動が異なる。COLD では文字数に応じて検索時間がほぼ線形に増大するのに対し、HOT では 2,3,4 文字と検索時間が増大した後、10 文字までほぼ同じ検索時間となる。

以上の結果から、X の索引構成・検索アルゴリズムは公開されていないが、FTS に採用されている本研究の高速検索方式の有効性が確認できる。

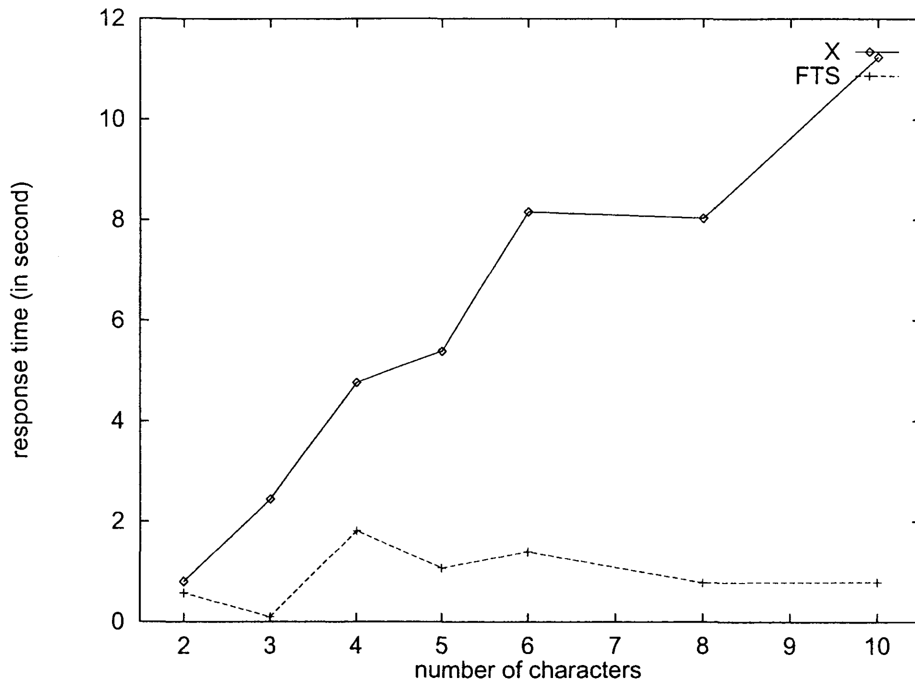

図 8.4: COLD での検索時間に対する文字数による影響

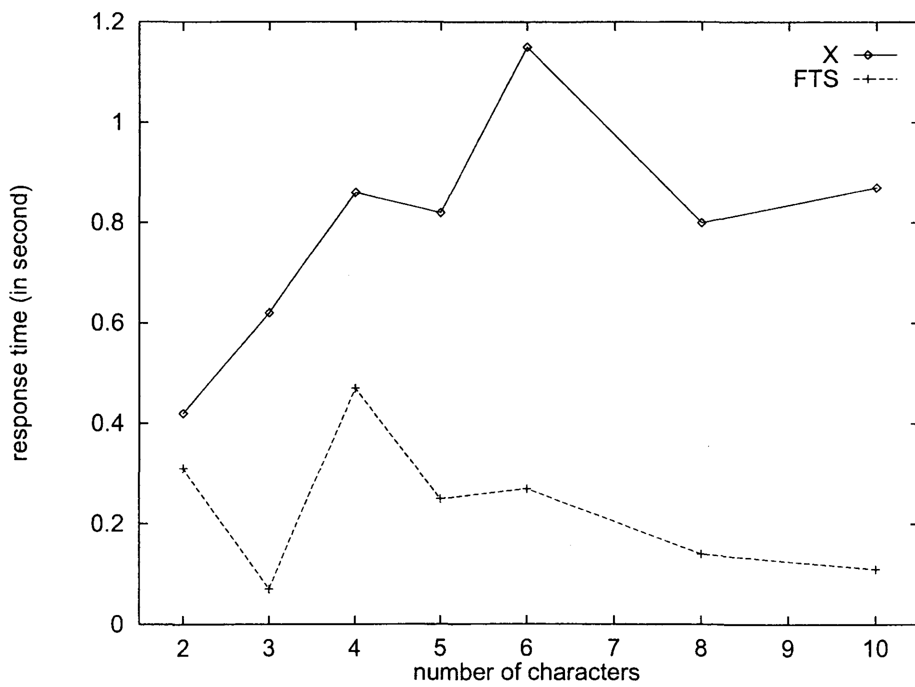

図 8.5: HOT での検索時間に対する文字数による影響

第9章　結論

9.1　本研究の成果

本研究では、日本語文書を対象としたn-gram索引の高速検索手法を提案した。基本的なアイデアは、n-gram索引において検索コスト増大の主原因である文書内でのn-gramの位置検査のコストを低減するというものである。位置検査コストの低減という考え方自体は従来から存在していたが、本研究では位置検索コスト低減をさらに発展させた。

3章ではブーリアン検索の高速化手法を提案した。基本となる単一検索文字列に対しては、処理フェーズに合わせて使用するn-gramを選択し分けることにより、位置検査を省略するとともに位置検査自体のコストを低減し、検索を高速化する選択n-gram法を提案した。選択n-gram法では、検索に使用する2つのn-gram群— 位置検査用n-gram群と候補文書特定用n-gram群— を使用状況にあわせて調整し、位置検査用n-gram群には最小頻度パス、候補文書特定用n-gram群には最小頻度パスにそれらの前後にある文書頻度の少ないものを加えた集合を選択する。一方、論理演算子の検索処理では、複数の検索文字列の関係を考慮して本来は不要な位置検査を省略する拡張省略法を提案した。AND, OR, ANDNOTの3種類の演算子について、この考えに基づく検索アルゴリズムを提示するとともに、改良アルゴリズムが演算子が入れ子になった検索条件にも対応できることを示した。

5章ではランキング検索の高速化手法を提案した。ランキング検索では、文書頻度・文書内頻度という検索文字列に関する2種類の頻度情報が文書スコア計算に必要であり、両頻度を求めるたびに位置検査が発生していたので、検索コストが増大していた。この問題に対し、本研究では2つの高速化手法を提案した。1つ目の順序入れ替え法は、検索文字列が出現する文書ごとに文書内頻度を最初に求め、それをメモリ上に保存しておき、その結果から文書頻度および文書ごとのスコアを求めることで、文書頻度を単独で求める処理を省略するというものである。もう1つの頻度推定法は、文書頻度・文書内頻度を検索文字列を構成するn-gramの頻度情報をもとに近似的に求めることで位置検査を省略するというものである。文書頻度の推定と文書内頻度の推定は独立なものであり、両者の組み合わせが可能である。なお、推定された頻度は必ずしも正しい値と一致するとは限らないため、頻度推定法はランキング検索結果にも影響する。

各提案手法が有効となる検索状況を表9.1にまとめた。選択n-gram法は単一文字列のブーリアン検索を高速化するために提案した手法であるが、論理演算子を含む検索条件およびランキング検索に対しても有効である。論理演算子のブーリアン検索の高速化手法である拡張省略法はランキング検索には限定的にしか適用できない。ランキング検索においては、検索文字列ごとに文書頻度を求めなければならないため、検索文字列の共起に基づいて位置検査を省略することはできないからである。ただし、AND演算子・ANDNOT演算子では、論理条件を満足しない文書において各検索文字列の文書内頻度を求める必要がなく、拡張省略法の考え方はランキング検索に一部適用である。順序入れ替え法・頻度推定法はランキング

表 9.1: 本研究の提案手法が有効な検索状況

	ブーリアン		ランキング
	単一文字列	論理演算子	
選択 n-gram 法	○	○	○
拡張省略法	—	○	△
順序入れ替え法	—	—	○
頻度推定法	—	—	○

表 9.2: 高速化効果のまとめ

	COLD	WARM	HOT
単一文字列	8.3%	20.8%	73.0%
論理演算子	(10.7%)	(21.9%)	(110%)
AND 演算子	21.1%	40.1%	252%
OR 演算子	4.0%	11.0%	41.3%
ANDNOT 演算子	5.7%	14.5%	35.1%
ランキング	36.2%	40.7%	91.7%

検索にのみ適用可能である。

提案高速化手法の有効性を検証するため、4 章ではブーリアン検索、6 章ではランキング検索について評価を行った。表 9.2 に評価結果をまとめた。ブーリアン検索における単一文字列・論理演算子（AND, OR, ANDNOT 演算子単独の場合とそれらの単純平均値）、およびランキング検索[1]に関し、全データを読み込む COLD、条件評価フェーズにおけるデータ読み込みのみの WARM、データ読み込みが全く不要な HOT の 3 つの状況おいて、従来手法に対する提案手法の検索速度向上の比率を表にしたものである。この表から、提案手法により検索を大幅に高速化できたことが確認できる。特に、AND 演算子およびランキング検索における効果が大きいことがわかる。また、COLD, WARM, HOT となるにしたがって高速化効果が大きく、データ読み込みを伴わない HOT の効果は 73 ～ 250%にも達していることから、提案手法はディスクアクセスよりも CPU 処理の削減に効果が大きいことがわかる。

表 9.3 には単語索引・n-gram 索引の性能比較を示す。この結果は 6.5 節のランキング検索における性能比較の結果をまとめたものである。この表で、n-gram 索引というのは基本方式（NNN）、改良 n-gram 索引は提案方式の中で最も高速であった頻度推定法の組み合わせ（NMM）の結果である。改良 n-gram 索引では検索精度をほとんど低下させることなく検索が高速化され、単語索引との検索時間の差異が小さくなることが確認できた。特に HOT においては検索時間の差は約 10%と小さい。6.5 節の最後に述べたように n-gram 索引にとって不利な状況での測定結果ということを考慮すると、検索が遅いという n-gram 索引の問題点は提案手法により概ね克服できた。その結果、文書検索アプリケーションの設計者に対す

[1]ランキング検索については 6 章の評価結果で最も高速であった文書頻度・文書内頻度ともに MIN 方式で推定する NMM の結果を示している。

表 9.3: 単語索引と n-gram 索引の比較（実測値）

	単語索引	n-gram 索引	改良 **n-gram** 索引
索引サイズ	197MB	490MB	490MB
登録時間	15246sec	13668sec	13668sec
検索時間			
COLD	0.698sec	1.291sec	**0.948sec**
WARM	0.410sec	1.023sec	**0.696sec**
HOT	0.319sec	0.663sec	**0.346sec**
検索精度	0.3699	0.3827	**0.3791**

る索引手法選択の自由度を大きく向上させることができたと言えよう。

7 章では、高速化検索処理向き転置ファイルの物理編成法を提案した。検索高速化のためにページアクセスと伸長処理の削減を図った。特に位置検査の削減という検索処理の特徴を最大限に生かすため、文書 ID と出現位置情報を分離と文書 ID を主体とするブロック化を組み合わせた物理編成法を提案した。評価実験により、両者の組み合わせという提案編成法により 81%(COLD), 128%(WARM), 860%(HOT) の高速化が得られることを確認した。

本研究の成果は、8 章に説明したように、FTS サーバとして製品化した。FTS の開発を行った当時に市場で高速であるという評判が高かった他社の商用文書検索システムとの性能比較を行い、FTS が 2.5 倍（HOT）から 4.0 倍（COLD）高速であることがわかった。FTS は当社の図書管理システムやオフィス向け文書管理システムなどで広く採用されている。

9.2　今後の課題

本研究で提案した高速化検索手法により n-gram 索引の検索速度を大幅に改善することができた。しかし、n-gram 索引に関しては以下にような問題点が依然として残されており、今後の課題としたい。

- 日本語以外の言語における有効性の検証

  本研究で提案した高速検索手法では言語の特性を直接的には利用していないので、言語によらず検索を高速化できるはずである。その一方で、高速化の効果は、対象とする言語で標準的に用いる文字の異なり数、単語の平均的な長さなどに依存している。

  1.3 節で検討したように、文字の異なり数が多く、単語境界が明示的に示されないという特徴を有する東アジアの言語等が n-gram 索引向きである。実際、n-gram 索引を中国語に適用した研究には [Che97, Kwo97, Raj97, Wil97]、韓国語に適用した研究には [Lee96] がある。

  本研究では、日本語文書を対象として評価実験を行い、提案手法の有効性を検証した。今後は潜在的に適用可能と考えられる中国語あるいは韓国語に対しても 4 章・6 章と同様の評価実験を行い、提案手法の有効性を検証したい。さらには、英語などの単語索引が標準的に使用されている言語に対する高速化の効果も評価したい。

- 複数索引への対応

  本研究では、登録文書の全てを単一の索引に登録する場合を対象としていた。しかし、索引への登録には時間がかかるため、複数の索引を用意しておき、ダブルバッファのようにそれらを交互に用いることで登録を高速化する手法等が提案されている。また、Web 検索のように膨大な文書を扱う場合には、ファイルサイズの制限等から単一の索引では対処できないこと、検索時間の短縮等の理由から、索引を複数に分割することがある。

  複数に分割して索引付けする場合、各索引は登録文書群の一部分を受け持つことになるので、検索文字列の登録文書全体に対する文書頻度を知ることができず、ランキング検索のための文書スコア計算に支障をきたす。この問題に対しては大域的な文書頻度の表を持つという対応策が考えられるものの、n-gram 索引では検索文字列が複数の索引単位に分割されることがあるため、この方法を実際に適用することは難しい。一方、索引ごとに得られた検索文字列の文書頻度を合計することで大域的な文書頻度を求め、それに基づいて索引ごとにランキング結果を求め、最後にマージするという手順であれば、複数索引にも対応可能である。しかし、このような単純な手順では、文書頻度を求める処理とランキング検索結果を求める処理の 2 回、索引を利用するため、検索速度の低下を招く。これは、5.1 節で指摘した n-gram 索引の問題点と同様の問題が複数索引の場合にも生じることを意味している。

  この問題を解決するには、単一索引用に提案した順序入れ替え法および頻度推定法を複数索引向けに拡張すればよいと考えられる [Oga02]。

- 質問拡張

  文書検索における検索精度低下の原因の 1 つとして、同一概念の表現に対象文書と検索条件で異なる単語（あるいは表現）が用いられるという語彙のミスマッチがある。語彙ミスマッチを解決する方法として、検索システムが検索条件に適切な単語を補って検索条件を拡張する質問拡張（query expansion）がある [Fra92, Sal83b]。質問拡張は同義語辞書・シソーラス等の言語リソースを用いる辞書拡張法と、検索対象のなかから検索条件に関連する単語を選択する適合フィードバック（relevance feedback）に大別できる。しかし、いずれの方法であっても通常は拡張する単位は単語であることが一般的である。

  n-gram 索引を用いた場合、拡張単位と索引単位が一致しないことが、ランキング検索において検索文字列と索引単位が一致しなかったのと同様に問題となる。辞書拡張法の場合には、言語リソースが提供されれば、検索文字列と文字列照合することによって拡張単語を選択可能であるので、拡張単位と索引単位の相違が大きな問題となるわけではない[2]。

  一方、適合フィードバックの場合には対象文書から単語を選択するので、n-gram 索引に適用する場合には対象文書を形態素解析する必要がある。単語索引の場合には、文書登録時に形態素解析しているので、拡張時にあらためて形態素解析する必要がないのに対し、処理速度上の問題となる。

[2]もちろん、拡張された単語が索引単位と一致するわけではないことから、5.1 節に示したような問題はあるものの、質問拡張すること自体には大きな問題はない。

表 9.4: 順序入れ替え法と擬似頻度法の組み合わせ

	単語拡張＋ 文書頻度推定	n-gram 拡張
処理速度	×	○
検索精度	○	×
対話性	○	×

適合フィードバックのもう1つの問題は、拡張単語を選択するためには、文書から抽出された単語について文書頻度等を利用して有用度を算出し、適切と思われる単語のみを検索条件に追加しなければならないが、候補単語ごとの文書頻度を得るための処理コストが高い点である。すなわち、単語索引であれば単語候補の文書頻度は単純に転置ファイルのヘッダー部分を参照するだけで簡単に入手することができる。これに対し、n-gram 索引の場合、単語候補が必ずしも索引単位と一致するわけではないため、ランキング検索において検索文字列の文書頻度を求める場合と同じく索引単位に一致しない単語候補についてはブーリアン検索によって文書頻度を求める必要がある。しかし、この方法では、拡張すべき単語を決定するだけでも相当な処理量となり、適合フィードバックを実用に供することは難しい。

これらの問題を解決するには、以下のような2つのアプローチが考えられる。

- 単語拡張＋文書頻度推定方式
  拡張単位は単語とする。この場合、対象文書を形態素解析し、得られた単語の品詞や不要語辞書を用いて候補を限定し、得られた候補ごとに有用度を計算する。文書頻度の取得に関する問題を解決するため、ランキング検索と同様にして文書頻度を推定する。推定方法は 5.4.1 節で提案した AND 方式・MIN 方式のいずれも適用可能であるが、処理精度では AND 方式、処理速度では MIN 方式が優れていると考えられる。
- n-gram 拡張方式
  単語ではなく n-gram を拡張単位とする [Oga01]。この場合、対象文書を n-gram に分割し、可能ならば n-gram レベルの不要語辞書を用いて候補を限定し、得られた候補ごとに有用度を計算する。候補は n-gram であるので、n-gram 索引を参照するだけで正確な文書頻度を高速に取得できる。

両者を比較すると、拡張単語の品質では候補限定の際に品詞を用いることができるので単語単位＋文書頻度推定方式、処理速度では拡張単位が索引単位に一致している n-gram 方式が優れていると考えられる。適合フィードバックでは、システムが選択した拡張単位をユーザが取捨選択することが検索精度向上に役立つと言われている。このような拡張単位の対話的な操作を考慮した場合、システムが選択した拡張単位の適切さの判断には単語であるほうがユーザ向きであると考えられる。

両アプローチの特徴をまとめると表 9.4 のようになる。n-gram 索引を用いた場合でも、自然文検索における検索文字列の選択法としては単語の方が望ましいかったように、

対話性まで考慮すると単語拡張に文書頻度推定を組み合わせるのが望ましいと考えられる。しかし、検索速度・検索精度の両面から評価を行い、定量的な分析に基づく妥当性の検証が不可欠である。

- パッセージ検索

  文書検索は検索条件を満たす文書群を検索することで終わりではなく、ユーザは検索結果から要求に満足した文書を選択し、さらにはその文書中の必要な部分を特定する必要がある。この点を考慮すると、文書中の検索要求に最もマッチする部分を提示することはユーザにとって大きな助けとなる。このように、文書中の適切な部分（パッセージ）を特定すること（このような検索をパッセージ検索と呼ぶ）は文書検索の基本機能の 1 つと考えられる [Sal98, Cal94]。また、近年研究が盛んに行われている質問応答（question answering）はパッセージ検索に高度な言語処理を組み合わせることで実現される。

  パッセージ検索に関しては、索引に格納されている索引単位ごとの文書内出現位置から検索文字列の出現位置を求めることができる。その上で、複数の出現位置がある場合には、それらの相対位置関係、パッセージごとに含まれる検索文字列の異なり数などから最適なパッセージを判断すればよい。このような検索文字列の出現位置に基づくパッセージの特定では位置検査処理が発生するので、処理速度の低下が懸念される。これに対し、検索文字列を構成する n-gram の出現位置の相対位置関係などを用いてパッセージを判断する方法も考えられる。パッセージの適切さと処理速度の観点から両手法の性能比較を行いたい。

付 録A　文字成分表の検索高速化

シグネチャファイル形式のn-gram索引である文字成分表の検索高速化に関しては、文字成分抽出の改良 [Oga95b, Oga96b] および文字成分表向けの高速ランキング検索手法 [Oga95a, Oga96a, Oga96b] の２つの研究を行った。以下、それぞれについて簡単に説明する。

A.1　文字成分抽出の改良

A.1.1　基本的な文字成分抽出法

文字成分表の場合、文書からn-gramを抽出するが、そのn-gramをそのまま索引単位とするのではなく、n-gramに対してハッシュ関数を施した結果を索引単位とする。n-gramの抽出と抽出n-gramに対するハッシュ関数の適用を合わせて文字成分の抽出と呼ぶ。

文字成分表では、各文字成分は登録済みの文書ごとに存在しているか否かを表す1, 0の2値で記録される。ハッシュ関数を適用しているため、転置ファイル形式のようにn-gramの存在そのものは保障されないので、文字成分表による検索結果には誤検索（false drop）が含まれる。この誤検索を除去するには文字成分表で検索された文書ごと文字列照合を行って検索文字列の存在を確認する必要があり、検索時間が増大する。すなわち、文字成分表を用いた検索の場合、文字成分表検索の高速化だけでなく、誤検索率（文字成分表による検索結果に含まれる誤検索の割合）を低くすることが、文字列照合を含めたトータルな検索高速化に必要である。

誤検索は文字成分抽出法によって左右される。n-gram抽出に関しては、1.4.2節で説明したように$n = 1, 2$とすることが一般的である。一方、ハッシュ関数に関しては、以下のような方法が提案されている。

- 文字コードハッシュ

  文字コードに単一のハッシュ関数を作用させる [Fur94, Tam95]。

- 文字種適応型文字コードハッシュ

  漢字・平仮名・片仮名などの文字種に応じて文字の使われ方が異なるのに、文字コードハッシュではこのような特性の相違を無視しているので、索引サイズの増大と誤検索率の悪化を引き起こす。これに対し、文字種ごとの特性を考慮してハッシュ関数を選択することで、性能向上をはかることができる [Fuj94, Iwa93]。

  なお、文字種適応という考え方は転置ファイル形式のn-gram索引にも見られるものであるが、転置ファイル形式においては文字種に応じてn-gram抽出法（nの値）を変化させるのに対し、文字成分表ではハッシュ関数を変化させる点で異なっている。

- 出現頻度ハッシュ

n-gram の出現頻度のばらつきは大きい [Zip49]。しかし、文字コードハッシュでは出現頻度とは無関係の文字コードに基づいてハッシュ値を計算するので、索引単位の出現頻度（ハッシュでまとめられる n-gram の出現頻度の総和）のばらつきも大きく、誤検索率が高くなる。これに対し、索引単位の出現頻度が平均化するようにハッシュ値を決定すれば、頻度の高い n-gram が特定の索引単位に集中することを防ぎ、性能向上をはかることができる [Fuk93, Kaw92]。

A.1.2　文字成分抽出法の改良

改良手法の概要

前述の基本的な文字成分抽出法に対し、[Oga95b, Oga96b] では以下の抽出法を提案した[1]。

- uni-gram

  文書中の uni-gram（単一文字）を抽出し、そのまま索引単位とする。

  ハッシュ関数を用いないことで、単一文字が存在するか否かを文書そのものを参照することなく正確に判断でき、誤検索除去は不要である。検索文字列が単一文字の場合、検索文書が多くなる傾向があるので、誤検索除去が不要であることはトータルの検索時間削減に非常に有効である。

- bi-gram

  文書中の bi-gram（２文字組）を抽出し、後述する文字種適用型頻度ハッシュを施し、索引単位とする。複数文字からなる検索文字列に対して uni-gram だけでは誤検索が増大するので、この誤検索を抑えるために bi-gram が必要である。

- n-gram($n > 2$)

  あらかじめ索引単位として抽出すべき $n > 2$ である n-gram のセットを決定しておき、このセットに含まれる n-gram が出現した場合にはそのまま索引単位とする。uni-gram 同様にハッシュ関数は用いないので、このセットに含まれる文字列の存在は文書を参照することなく判断できる。抽出すべき n-gram を拡張文字列と呼ぶ。

  長い検索文字列に対してアクセスする文字成分の個数を抑えるために拡張文字列を導入するものである。静的な文字列集合である点で転置ファイルの語彙ファイル（索引単位の集合）とは異なっている。

この文字成分抽出法においては、文字種適用型頻度ハッシュと拡張文字列が検索高速化に大きく貢献する。以下、両者について説明する。

文字種適応型頻度ハッシュ

文字種適応型頻度ハッシュとは、文字種ごとのハッシュ法に頻度ハッシュを適用するものである [Oga95b, Oga96b]。文字種適応型ハッシュと頻度ハッシュを組み合わせることで、誤検索を低減させる。

[1] 本研究に際しては文書は日本語文書だけを対象としており、入力される文書は JIS コード [JIS95b, JIS95c] でコーディングされており、使用される文字数は約 7000 に限定されるものとしていた。

前述の方針にしたがって bi-gram に頻度ハッシュを適用する場合、単純には文字の異なり数の二乗のサイズの頻度テーブルが必要となる。文字種で分割しても JIS コードの漢字は約 6300 文字以上あるため頻度テーブルが大きくなるという問題がある。これに対し、まず文字ごとに頻度ハッシュを適用し、ハッシュ値の組から最終的なハッシュ値を計算する方法を提案した。この方式では頻度テーブルのサイズは文字の異なり数の大きさとなり、コンパクトである。

文字種適応型頻度ハッシュは検索高速化にも有効である。文字の頻度のばらつきが大きいことから、ハッシュエントリ数をある程度大きくした場合、単一文字でハッシュエントリを専有する文字が現われる。このような文字を専有文字、専有されるハッシュエントリを専有ハッシュエントリと呼ぶ。提案手法で文字成分を抽出する場合、検索文字列からは uni-gram と bi-gram が抽出される。この際、bi-gram を構成する文字が専有文字であれば、bi-gram から算出する索引単位がその専有文字の存在を確実に示すことになるので、uni-gram を検索処理に使用する必要はない。すなわち、検索に使用される文字成分が減少するので、検索を高速化できる。

例えば、「電話」という検索文字列が与えられたとする。ここで頻度を用いない文字種適応型ハッシュを用いた場合、「電」「話」の２つの uni-gram と、「電話」という bi-gram にハッシュ関数を適用結果の計３つの索引単位を検索処理で使用する。これに対し、文字種適応型頻度ハッシュを用い、「話」が専有文字であるとした場合、「電話」から算出される索引単位が「話」の存在を示すので、従来方式では必要であった「話」の uni-gram が不要となる。すなわち、検索に必要な索引単位は２つとなり、検索も高速化される。

拡張文字列

高速化のもうひとつの工夫が拡張文字列の利用である。拡張文字列はハッシュを適用せずにそのまま索引単位となるので、検索文字列に拡張文字列が含まれていればその内部に含まれる uni-gram/bi-gram の索引単位は不要となり、検索を高速化できる。

拡張文字列が高速化に貢献する度合いは、検索文字列あるいはその一部分として使用される頻度が高いもの、文字列長が大きいものほど大きくなる。[Oga95b, Oga96b] では、uni-gram, bi-gram に対応する索引単位も使用していることを考慮し、長さ 3 以上の n-gram で出現頻度の高いものを選択し、拡張文字列辞書として用意することとした。頻度の計数方法であるが、検索文字列に使用されることが多いのは漢字・片仮名であることから、登録文書あるいは適当なコーパスから漢字の連続部分および片仮名の連続部分を切り出して頻度を計数することとした。

なお、転置ファイル形式の n-gram 索引においても拡張文字列の利用は検索高速化に有効であり、フレキシブル文字列インバージョン法 [Aka96a] では拡張文字列を使用している。

A.1.3 性能評価

提案手法の有効性を評価するための実験を行った。ここでは評価実験の概略のみを報告する（詳細は [Oga96b] を参照）。

検索対象には特許要約文 100,000 件、文書サイズは 14 MB（１件当たり 140 B）を使用した。検索文字列としては、長さが 2,4,6,8,10 の漢字および片仮名の文字列を各 30 個、合

表 A.1: 文字種適応型頻度ハッシュの評価

	漢字		片仮名	
	頻度なし	頻度あり	頻度なし	頻度あり
検索時間	0.518	0.314 (−39.4%)	0.588	0.209 (−64.5%)
誤検索率	5.62×10^{-5}	4.27×10^{-5} (−24.0%)	1.48×10^{-4}	1.52×10^{-5} (−89.7%)

表 A.2: 拡張文字列の評価

	漢字		片仮名	
	拡張なし	拡張あり	拡張なし	拡張あり
検索時間	0.496	0.344 (−30.6%)	0.323	0.156 (−51.7%)
誤検索率	2.07×10^{-6}	1.71×10^{-6} (−17.4%)	4.93×10^{-5}	3.71×10^{-5} (−24.7%)

計 600 個を用いた。測定には SUN Microsystem 社のワークステーション SPARC Station20（CPU: SUltra PARC II 70MHz, メインメモリ 32 MB）を用いた。システム起動直後のデータがキャッシュされていない状態である COLD の検索時間を測定した（単位は秒)。ここでは文字成分表の検索時間のみを測定しており、その後に行われるべき文字列照合時間は含んでいない点に注意が必要である。また、以下の式で計算される誤検索率も測定した。

$$\text{誤検索率} = \frac{\text{検索文字列を含まないのに検索された文書数}}{\text{検索文字列を含まない文書数}} \tag{A.1}$$

文字種適応型頻度ハッシュの評価結果を表 A.1 に示す。この表では、頻度なしが従来の文字種適応型ハッシュ、頻度ありが文字種適応型頻度ハッシュの結果を表している。また、ハッシュエントリ数を漢字は 256、片仮名は 64 とした[2]。この表から判るように、頻度を用いることで検索時間が短縮されるとともに誤検索も大きく低下することが確認できる。漢字と片仮名を比較すると片仮名の方が頻度ハッシュが有効に作用している。これは、漢字では文字の異なり数 6353 に対してハッシュエントリ数が 256 と小さいのに対し、片仮名では異なり数 87 に対してハッシュエントリ数が 64 と大きく、片仮名のほうが専有文字の割合も高いことに起因すると考えられる。実際、専有文字数は漢字が 97、片仮名が 57 であった。

つぎに拡張文字列の評価結果を表 A.2 に示す。この表では、拡張なしが文字種適応型頻度ハッシュ（ハッシュエントリ数は漢字は 64、片仮名は 32 とした)、拡張ありが漢字・片仮名それぞれに 512 個の拡張文字列を導入した場合の結果を表している。ここでも、拡張文字列を導入することで、検索時間が短縮され、誤検索も低下することが確認できる。なお、拡張文字列の対応する索引サイズは 0.94MB であり、uni-gram/bi-gram に対応する部分の 14%と小さい。

[2] bi-gram に対する文字成分数はハッシュエントリ数の二乗となる。

A.2 文字成分表向け高速ランキング検索手法

A.2.1 文字成分表を用いたランキング検索の問題点

文字成分表をランキングに採用した場合、ランキングに必要な検索語の文書内頻度・文書頻度が記録されていないことが問題となる [Oga95a, Oga96a]。2.3 で述べた転置ファイル形式の n-gram 索引と共通の問題であるが、転置ファイルでは文書内出現位置が記録されているのに対し文字成分表においては文書そのものにアクセスする必要がある点で問題が深刻である。

文書内頻度は文書における検索文字列の出現頻度であるので、対象文書を文字列照合して、検索文字列の出現回数を計数する必要がある。ランキング検索においては、検索文字列を一つでも含む文書がランキングの対象となるので、それら文書全てについて文書内頻度を計数しなければならない。一方、文書頻度は検索文字列を含む文書数であるので、文字成分表を用いて行った検索結果の文書数を用いればよい。正確な文書頻度を得るには文字成分表検索に伴う誤検索の除去が必要であるが、文書内頻度計数結果が 0 であった文書は誤検索と判断できるので、文書内頻度を計数すれば誤検索除去のために新たな処理は不要となる。

複数個の検索文字列がある場合にも同一文書を複数回処理することがないように、以下の手順でランキングを行う。

(1). 文字成分表検索により少なくとも一つの検索文字列を含む文書を特定し、ランキング候補とする。

(2). 全てのランキング候補に逐次的にアクセスし、各候補における各検索文字列の文書内頻度を計数・記録する。全候補を処理し終えた時点で、各検索文字列の文書頻度も明らかになる。

(3). 記録されている文書頻度・文書内頻度を用いて、ランキング候補ごとに文書スコアを計算する。

(4). ランキング候補を文書スコア順にソートし、検索結果とする。

以下、この処理手順を一括確定法と呼ぶ。

ステップ (2) では、ファイルアクセス・文字列照合というコストの高い処理を全てのランキング候補について実施する。ランキング候補数は登録文書数の 50%以上にもなるという実験結果 [Bro95b, Mof94a] を考慮すると、ステップ (2) はかなりの処理量となり、検索時間を大幅に増大させる原因となる。したがって、検索速度の点から一括確定法は実用的ではない。

A.2.2 逐次確定法による効率的ランキング処理

前節の議論から、ランキング検索の時間短縮には、ファイルアクセス・文字列照合の対象文書を削減することが不可欠であるとわかった。以下では、逐次確定法という手法を採用することで、これらの処理対象文書を削減し、検索を高速化できることを示す。

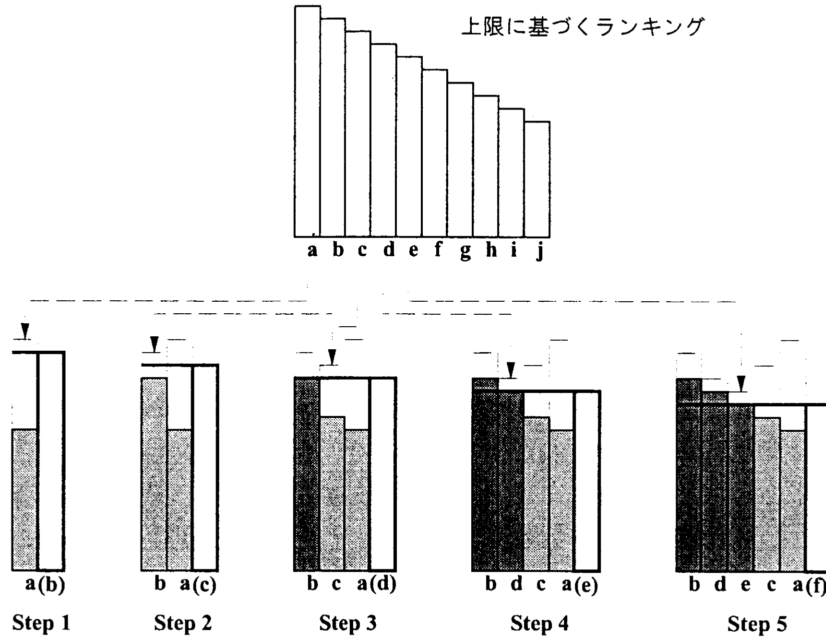

図 A.1: 逐次確定法による上位ランキング文書の決定

逐次確定法

ランキング検索では、ユーザが実際に参照するのは比較的少数の上位にランキングされた文書のみであるので、比較的少数の上位ランキングの文書を高速に決定できればユーザを満足させることができる [Buc85, Won93]。逐次確定法では、文書スコアの上限（upper bound: 文書スコアより大きいという制約を満たすスコアの推定値）を利用することで上位ランキングの文書を高速に決定する [Kna94, Won93]。

検索要求 q に対する文書 d のスコア $score(d,q)$ の上限を $upper(d,q)$ と書く。このとき、"$upper(d,q) \geq score(d,q)$" なので、二つの異なる文書 d_i, d_j に対して以下の関係式が成立する。

$$score(d_i,q) \geq upper(d_j,q) \Rightarrow score(d_i,q) \geq score(d_j,q) \qquad (A.2)$$

ランキングの上位文書を決定するに当たっては、全てのランキング候補について上限を計算し、上限の大きい順にランキング候補のスコア計算を行う [Kna94, Oga95a, Oga96a]。上限の大きい順に l 個の文書のスコアを計算した時点で、残りの文書のなかでは $l+1$ 位の文書がもっとも大きな上限を持っている。したがって、上限によるランキングの第 l 位の文書の ID を $o(l)$ と書くと、式 (A.2) から文書集合 $R_l = \{d_i | score(d_i,q) \geq upper(d_{o(l+1)},q)\}$ に属する文書の最終ランキングを決定できることがわかる。ランキングの上位 k 文書を決定するには、"$|R_l| \geq k$"（これを終了条件と呼ぶ）を満たした時点で処理を終了すれば良い。ここで、$|X|$ は集合 X の要素数とする。

逐次確定法によるランキング決定の様子を図 A.1 に示す。文書はアルファベット（a, b, …）で識別するものとし、上段が上限によるランキング結果を示す。白抜き矩形の高さで上限、網掛け矩形の高さで文書スコアを表している。処理は下段の左から右と進み、ステップ l では l 番目の文書のスコアを計算し、$l+1$ 番目の上限との比較によってランキングが決定

できるかの判定を行う。下段において、濃く網掛けになっているのがランキングの確定した文書であり、例えばステップ 5 では b, d, e の 3 文書のランキングが確定している。$k = 3$ であれば、ここで処理を終了できる。

逐次確定法の文字成分表への適用

逐次確定法を文字成分表を適用した場合、ランキングの処理手順は以下のようになる。

(1). プレランキングフェーズ
文字成分表を用いて、検索文字列を少なくとも一つ含む文書を特定し、ランキング候補とする。さらに、全てのランキング候補について文書スコアの上限を計算し、ランキング候補を上限順にソートする。

(2). 逐次確定フェーズ
プレランキング順に候補文書にアクセスし、その文書における検索文字列の文書内頻度（出現回数）を計数し、それを用いて文書スコアを計算する。終了条件を満たしたら、処理を終了する。

上記手順ではランキング候補全てにアクセスするわけではないので、文字成分表検索で発生する誤検索が完全には除去されない。したがって、ランキング結果に対する誤検索の影響について検討する必要がある。誤検索の影響には、以下に示す二つがある。

一つは文書スコア計算への影響である。逐次確定法では、文字成分表による検索結果の文書数を文書頻度に代用する。ところが、誤検索が残っているため、この値は本来の本来の文書頻度より大きく、文書スコアに影響する。しかし、文書スコア計算では後述のように文書頻度の log をとった値を使用しているので、文字成分表のパラメータ [Oga96b] を調整して誤検索率を低く押さえることが可能であり、誤検索による文書スコアの相違はそれほど大きくない。文書スコアが多少変化してもランキングに影響しない場合もあるので、誤検索を含んだ文書数を使用してもランキング結果への影響は小さいと考えられる。

もう一つの問題は上限計算への影響である。上限の計算はプレランキングフェーズで行うので、文字成分表のみを用いて上限は計算しなければならない。そこで、文字成分表から知ることのできる文書に検索文字列が出現しているか否かに基づいて上限を計算する。しかし、誤検索のために、実際には検索文字列が出現していないにもかかわらず出現していると判断され、誤検索がない場合よりも上限が大きく計算されることがある。この結果、処理ステップが増加し、検索が遅くなる可能性がある。しかし、誤検索率が低ければ、速度低下への影響も小さいと考えられる。

以上の議論から、誤検索がランキング結果に若干影響する懸念はあるが、全候補を誤検索除去する一括確定法と比べて検索時間の大幅な短縮が見込まれる。したがって、逐次確定法を文字成分表に適用して得られるメリットは大きい。

A.2.3 性能評価

文字成分表の高速ランキング検索手法である逐次確定法を検索精度と検索速度の両面から比較する。検索精度の評価にはテストコレクションが必要であるが、本実験を行った 1996 年当時入手可能であった唯一のテストコレクションである BMIR-J1[3] [Kes96] を利用した。検

表 A.3: 一括確定法・逐次確定法の評価

	一括確定法	逐次確定法
検索時間	66.4	6.98 (−89.5%)
平均適合率	0.2756	0.2752 (−0.1%)

索要求文は 47 個で、検索対象は 600 文書、872 KB（1 件当り 1406B）である。評価指標には平均適合率を使用した。

検索時間の評価には、日本経済新聞 CD-ROM 93 年版を利用した。これは、BMIR-J1 の対象文書数が 600 と処理時間の測定には少ないからである。本 CD-ROM には 163,110 件、テキストサイズは 159 MB（1 件当り 988 B）である。測定には、前節同様、SUN Microsystem 社のワークステーション SPARC Station20 を用いた。

評価結果を表 A.3 に示す。ここで検索時間はランキングの上位 20 件を決定するのに要する時間であり、単位は秒である。この表から判るように、検索時間は約 1/10 に短縮されているのに対し、平均適合率への影響は 0.1%と極めてわずかである。このことから、逐次確定法が文字成分表を用いたランキング検索手法として有効であることが確認できる。

A.3 転置ファイル形式 n-gram 索引の検索高速化との関連

最後に文字成分表の検索高速化の提案と転置ファイル形式 n-gram 索引との関係を整理する。

- 文字成分抽出に関する提案は n-gram 抽出法の改良と関連している。転置ファイル形式ではハッシュ関数の適用は行なわれていないが、文字種適応型文字コードハッシュは n-gram 抽出における文字種適応に、頻度ハッシュおよび拡張文字列は n-gram 抽出における n の動的調整に対応するものである。ただし、頻度ハッシュおよび拡張文字列は文書登録開始時までに収集した頻度情報に基づく静的な手法であるのに対し、n の動的調整は登録された文書の情報に基づく動的な手法である点で優れている。

- 逐次確定法はランキング検索に関する検索処理の改良と関連している。n-gram 索引においても、検索文字列から抽出される n-gram が全て出現している文書を発見する候補文書特定と、候補文書に検索文字列が実際に含まれているかを確認する位置検査の２つのフェーズから構成されており、文書そのものへのアクセスは不要であるものの文字成分表を用いた文書検索に類似している。したがって、逐次確定法を n-gram 索引のランキング検索の高速化に適用可能である。

[3](株) 日本経済新聞の協力によって、(社) 情報処理学会・データベースシステム研究会・情報検索システム評価用データベース構築ワーキンググループが、1993 年 9 月 1 日から 12 月 31 日の日本経済新聞記事を基に構築した情報検索評価用データベース（テスト版）である。

付 録B　検索アルゴリズムの記述

本論文のなかでは、2 章以降、検索アルゴリズムの記述に C++風の記述形式を採用している。そこでは、記述を簡単にするためにオブジェクト指向プログラミングで一般的なクラス階層を前提にしている。検索条件の内部表現も、クエリノードを表現する抽象クラスをベースとして長さに応じた検索文字列ノード、各種論理演算子を表現する演算子ノード等があるので、ここで簡単にその継承関係を説明する。

継承関係は図 B.1 の通りである（この図において * がついているノードは抽象クラスである)。ここで、各ノードは以下のようなものである。

- QueryNode

  ノードのインタフェースを規定する抽象クラス。

- TermNode

  検索文字列に対応するノードのための抽象クラス。

- LeafNode

  検索条件の内部表現の木構造において末端ノードとなるノードのための抽象クラス。

- InternalNode

  検索条件の内部表現の木構造において中間ノードとなるノードのための抽象クラス。子ノードはノードのベクトル型（`vector<QueryNode>`）のメンバー`child`として保持・管理される。

- GramNode

  n-gram に等しい長さの検索文字列のためのノードで、TermNode と LeafNode を多重継承する。対応する n-gram の文書出現情報を管理する転置リストクラス`InvertedList`のメンバー`invertedList`を持ち、そこから得られる情報に基づいて検索処理を行う。

- ShortTermNode

  n-gram よりも短い検索文字列のためのノードで、TermNode と OrNode を多重継承する。

- LongTermNode

  n-gram よりも短い検索文字列のためのノードで、TermNode と AndNode を多重継承する。複数の n-gram の出現位置検査機能を持つ。

- OrNode

  OR 演算子に対応するノードで、InternalNode を継承する。

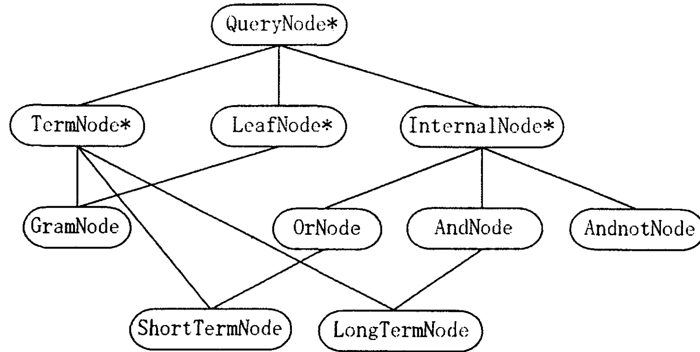

図 B.1: ノードの継承関係

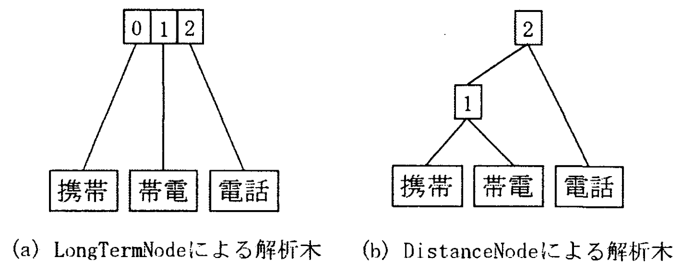

図 B.2: LongTermNode の DistanceNode による表現

- AndNode

  AND 演算子に対応するノードで、InternalNode を継承する。

- AndnotNode

  ANDNOT 演算子に対応するノードで、InternalNode を継承する。

LongTermNode は 2 個以上の任意の個数の子ノードを持つことができる。しかし、実装においては、子ノードを 2 個だけ持ち、同一文書内の出現位置が指定されれた相対距離もを満足するものだけをヒットと判断する DistanceNode を導入し、長い検索文字列は複数の DintanceNode で処理することとした。DistanceNode は LongTermNode と同様に TermNode と AndNode を多重継承するが、これは子ノード数が固定なので位置照合の実装が簡単であるという利点がある。$m(m > 2)$ 個以上の n-gram に分割される LongTermNode は $(m-1)$ 個の DisntaceNode で表現できる。図 B.2 に「携帯電話」に対応する LongTermNode を 2 つの DisntaceNode で表現したものを示している。

付 録C　パスの個数の導出

式 (3.5) の右辺の個数（パスの個数）の算出法について説明する。長さ m の検索文字列に対するパスの個数を p_m と書くとする。

まず、$m = \alpha n, (\alpha > 1)$ の場合、パスは一意になるので

$$p_m = 1 \tag{C.1}$$

つぎに $m = (\alpha+1)n-x, (\alpha > 1, 0 < x < n)$ の場合、3.3.1 節の議論で述べたように、検索文字列の分割法は $(X_{1..\alpha n}, X_{(\alpha n-x+1)..((\alpha+1)n-x)})$, $(X_{1..(\alpha n-1)}, X_{(\alpha n-x+1)..((\alpha+1)n-x)})$, $\cdots$, $(X_{1..(\alpha n-x)}, X_{(\alpha n-x+1)..((\alpha+1)n-x)})$ の $x+1$ 通りある。したがって、以下の式が得られる。

$$p_{(\alpha+1)n-x} = \sum_{s=0}^{x} p_{\alpha n-s} \tag{C.2}$$

この式は以下のように変形できる。

$$\begin{aligned} p_{(\alpha+1)n-x} - p_{\alpha n-x} &= \sum_{s=0}^{x-1} p_{\alpha n-s} \\ &= p_{(\alpha+1)n-(x-1)} \end{aligned} \tag{C.3}$$

n の 2 倍未満の検索文字列の分割法は 1 通りしかないので $p_{2n-x} = 1$ である。そこで、帰納法を用いてパスの個数の計算式が得られる。

$$p_{(\alpha+1)n-x} = \frac{\prod_{s=0}^{x-1}(\alpha+s)}{x!} \tag{C.4}$$

この式が式 (C.2) を満たすことは容易に確認できる。

付 録D　確率モデルとその改良

D.1　確率モデル

確率モデルでは、単語が適切な文書に含まれる確率と不適切な文書に含まれる確率をもとに単語の重み付けを行い、文書のスコアを計算する [Rob76]。ベクトル空間モデルと並んでランキング検索の代表的なモデルの 1 つである。検索精度の高さから近年の TREC, NTCIR 等の検索システムの評価会でも確率モデルを採用したシステムは多く、それらシステムの多くは実際に上位の成績を収めている [Kan02, Voo02]。

確率モデルにおいて、単語 t の重み t_t は以下の式で与えられる。

$$w_t = \log \frac{p_t}{(1-p_t)} - \log \frac{q_t}{(1-q_t)} \tag{D.1}$$

ここで p_t は単語が適切な文書に出現する確率であり、q_t は不適切な文書に出現する確率である。w_t は p_t と q_t の logit 変換（$\log(x/(1-x))$）の差分となっている。

検索時点ではその単語に対して適切・不適切である文書がわからないため、p_t, q_t も不明である。そこで、通常は単語の出現頻度に基づいて p_t, q_t を推定する。古典的な推定法として、Croft らが提案した方法がある [Cro79]。

$$p_t = p_0 \tag{D.2}$$

$$q_t = x_t \tag{D.3}$$

ここで、$x_t = f_t/N$（N は検索対象の全文書数、f_t は t が出現する文書数）、$p_0 (0 < p_0 < 1)$ はチューニング用のパラメータである。式 (D.1) に式 (D.2)・式 (D.3) を代入することで重みの計算式が得られる。

$$w_t = \log \left(\frac{p_0}{1-p_0} \frac{1-x_t}{x_t} \right) = K + \log \frac{N-f_t}{f_t} \tag{D.4}$$

ただし、$K = \log(p_0/(1-p_0))$ である。

D.2　従来の推定方式の問題点と Okapi モデル

ユーザが指定した検索語は、ユーザが検索要求を表す単語として選択したものであり、重みが負の値となるのは不適切である。しかし、式 (D.4) では $f_t > 0.5 \cdot N$ の場合に重みが負となるという問題点がある [Rob97]。そこで、Robertson は p_t の推定式を以下のように修正した。

$$p_t = \frac{p_0}{p_0 + (1-p_0)(1-x_t)} = \frac{p_0}{1-(1-p_0)x_t} \tag{D.5}$$

この場合、重みの計算式はつぎのようになる。

$$w_t = \log\left(\frac{p_0}{1-p_0}\cdot\frac{1}{x_t}\right) = K + \log\frac{N}{f_t} \tag{D.6}$$

この計算式は Okapi モデルと呼ばれ、TREC, NTCIR 等で上位のシステムに多く採用されている。

D.3 改良 Okapi モデル

式 (D.6) を詳細に見ると、重みが正であることが保障されるのは $K > 0(p_0 > 0.5)$ の場合だけであることがわかる。TREC データを用いた実験によれば、パラメータを小さくすることで検索精度が向上するものの、小さくしすぎると検索語によっては重みが負となるものがあらわれるため、検索精度が突然悪化する現象が見られた。この問題に対応するには、パラメータ p_0 によらず重みが常に正となるように確率推定式を修正すればよく、p_t の推定式として以下のものを提案した [Oga00c]。

$$p_t = p_0 + (1-p_0)x_t \tag{D.7}$$

この推定式を用いた場合、以下の計算式が得られる。

$$w_t = \log\left(\frac{p_0}{1-p_0}\cdot\frac{1}{x_t}+1\right) = \log\left(k\cdot\frac{N}{f_t}+1\right) \tag{D.8}$$

ここで、$k = p_0/(1-p_0)$ はチューニングパラメータである。

$0 < p_0 < 1$ であるので $k > 0$ となるため、式 (D.8) では重みが負になることはない。TREC における評価実験 [Oga00c, Oga01] によれば検索精度そのものを比較すると若干良い程度に過ぎないかったが、検索対象の規模の変動に対する頑強性が高まっており、動的に変化する検索対象に有効であると考えられる。

付 録E　異表記正規化

E.1　異表記正規化の必要性

文書検索では、ユーザが与えた検索文字列が含まれる文書を検索する。したがって、検索文字列と同一概念が記述されている文書であっても、文書中では検索文字列と異なる表記が使用されている場合には検索できないという問題がある。例えば、英語の "fuzzy" は日本語では「ファジィ」「フアジイ」「ファジー」等などさまざまに表記されるが、検索文字列「ファジィ」では「フアジイ」を含む文書を検索できない。

異表記の問題を解決するには以下の２つの考え方がある。

- 正規化

  同一概念を表す異表記を同じ表記となるように正規化することで解決する。登録・検索時に入力文字列を同一規則に従って正規化することで、文書と検索文字列の表記が異なる場合にも、検索するか否かの判定は正規化した表記で行うので、検索漏れを防ぐことができる。

  既存の文書検索システムにおける異表記正規化手法としては、大文字・小文字を正規化するといった文字レベルの正規化を行うのが一般的である。単語索引においては、複数表記のある単語に対しては全ての表記と１つの代表的な表記を記録する単語辞書を用意し、形態素解析結果としては代表表記を採用することで異表記正規化を実現することができる。

  例えば、小文字は大文字に変換するという正規化により「ファジィ」で「フアジイ」を検索することが可能である。また、「ファジィ」の辞書エントリに「フアジイ」「ファジー」等を異表記として登録しておけば、これらの表記を代表表記（この場合は「ファジィ」）に統一することが可能である。

- 展開

  登録時には入力文字列をそのままとし、検索時には検索文字列に対し異表記の可能性がある表記を全て生成し、それらの OR 演算子で結合した検索条件により検索を行うことで検索漏れを防ぐものである。

  大文字・小文字の関係にあるものはそれぞれを他方に置き換えた文字列を展開用に生成する。例えば、「ファジィ」に対しては「フアジィ」「ファジイ」「フアジイ」を生成する。

両者を比較すると以下のようにまとめられる。

- 登録速度

正規化の場合には、入力文字列を正規化するための処理時間がかかるものの登録処理自体に比べると無視できるほど小さい。むしろ、正規化することにより索引サイズが若干ではあるが小さくなり、ファイル IO も減少するため、登録速度は向上することもある。展開の場合には入力文字列そのものから索引を作るので、正規化しない場合と同様の処理時間である。

- 検索速度

  正規化の場合には、検索文字列を正規化するための処理時間がかかるものの検索処理自体に比べると無視できるほど小さいが、正規化により検索件数が増加することがあり、その場合には若干ではあるが検索速度が低下する。一方、展開の場合には複数表記の OR 条件を処理する必要があるため、検索速度が低下が大きい。

- 検索精度

  正規化の場合には、本来は同一視すべきではない表記のペアが同一表記に正規化されることがあるので、過剰検索が起こる可能性が問題となる。展開の場合には、展開ルールの不備により本来は同一視すべき表記のペアが展開されないことがあるので、検索漏れが起こる可能性が問題となる。

- ルール変更の影響

  正規化の場合には登録時にも入力文字列を正規化するため、ルールの変更を行った場合には索引を再作成が必要である。一方、展開の場合には登録時には入力文字列そのものから索引を作成するため展開ルールを変更しても索引の再作成は不要である。

このように正規化・展開には長所短所があり、どちらか一方が絶対的に優れているわけではない。しかし、現実には検索時間を考慮して正規化を行うのが一般的である。

なお、英語を対象とした文書検索においては接辞処理（stemming）も異表記正規化の一環として実施されることも多い。接辞処理の例としては、単数形・複数形の統一（例えば、複数形の"systems" を単数形の "system" にする)、動詞類の活用語尾変化の統一（例えば、３人称の "retrieves" 、過去・過去分詞形の "retrieved" 、現在進行形の "retrieving" を原形の "retrieve" にする）等がある。しかし、日本語においては、単数形・複数形の相違がないこと、サ変名詞のように語幹部分と活用部分がもともと別形態素とされること等から接辞処理を導入するしているシステムは限られていると思われる。FTS の異表記正規化も接辞処理は含んでいない。

E.2 処理の概要

検索精度を向上させるため、全文検索エンジン FTS でも異表記正規化処理を導入し、その処理系を独自に開発した。本異表記正規化の特徴は以下の通りである。

- 形態素解析が不要な文字列レベル（文字レベルも含む）の処理による正規化

- 正規化と展開を組み合わせたカタカナ文字列の正規化処理

  カタカナに関して展開を組み合わせるのは、ある文字列の変換する先の文字列として複数のものが存在する場合があるため、正規化だけでは検索漏れを防ぐことが不可能

だからである。例えば、「ディ」の変換先としては「デ」と「ヂ」の２種類が考えられる（前者は「ディジタル」と「デジタル」、後者は「ビルディング」と「ビルヂング」を関連付けるために必要となる）。この問題を防ぐためには、どちらか一方に正規化した上で、検索時にはもう一方の表記も生成するように展開を組み合わせる必要があるためである。例えば、「ディ」は「デ」に正規化するとした上で検索時には「ヂ」の展開を行うこととする。このようにすれば、文書中の「ビルディング」は「ビルデング」に変換され、「ビルヂング」は「ビルヂング」のままである（本当は「ビルデング」になる）が、検索時には「ビルディング」、「ビルヂング」のいずれもが「ビルデング」「ビルヂング」に展開されるため、検索漏れを無くすことができる。

以下では、カタカナとそれ以外に分けて正規化処理の内容を説明する。

E.2.1 カタカナ以外の正規化

カタカナ以外は文字列レベルの処理を行う。処理範囲は以下の通りである。

- 大文字・小文字の正規化

  ラテン文字・ギリシャ文字・キリル文字（ロシア語アルファベット）の大文字は対応する小文字に変換する。ひらがなは小文字を大文字に正規化する。なお、カタカナの大文字・小文字の正規化は後段のカタカナ処理において実施する。

- 全角・半角の正規化

  全角の英数字、記号を半角に変換する。英字については同時に小文字への正規化も行なうため、全角大文字は半角小文字に正規化することとなる。半角カタカナ（半角句読点、かぎ括弧も含む）は全角カタカナに正規化する。

- 漢字異体字の正規化

  漢字の異体字（旧字、略字など）を JIS コードの小さいものに変換する。例えば、「劍」「劒」「剱」は「剣」に統一する。

- 仮名旧字の正規化

  ひらがな・カタカナの旧字を新字に正規化する。例えば、「ゐ」は「い」に、「ヱ」は「エ」に正規化する。

- 仮名の合字

  ひらがな・カタカナの清音と濁点・半濁点の連続を、対応する濁音・半濁音に合成する。対応する濁音・半濁音がない場合は、仮名はそのままで濁点・半濁点を削除する。例えば、「さ゛」は「ざ」に合字し、「さ゜」は「さ」のように半濁点だけが削除される。

- 音標符号付き文字の正規化

  音標符号付きのラテン文字・ギリシャ文字・キリル文字を、音標符号を削除した文字に正規化する。例えば、ウムラウト「¨」付きの「Ü」は、音標符号を削除した上で小文字の「u」に正規化する。

E.2.2　カタカナの正規化

カタカナ表記の統一は、正規化（登録時と検索時に実施）と展開（検索時にのみ実施）の２段階で行われる。

カタカナ表記の正規化は、以下のような正規化ルールによって実施される。

- 大文字・小文字の正規化

  全てのカタカナの小文字は対応する大文字に変換する。例えば、「ァ」は「ア」になる。

- 大文字から大文字への変換

  「ヂ」は「ジ」、「ヅ」は「ズ」、「ヴ」は「ブ」に変換するという３種類の正規化規則がある。

- 「大文字＋小文字」から大文字への変換

  特定の組み合わせの「大文字＋小文字」は大文字に変換する。例えば、「ヴァ」は「バ」、「ティ」は「チ」に変換する。

- 長音から大文字への変換

  先行する文字がエ段の場合は長音を「イ」、オ段の場合は長音を「ウ」に変換する。例えば、「ベー」は「ベイ」、「ボー」は「ボウ」に変換する。

- 長音の削除

  先行する文字がア段、イ段、ウ段の場合には、後続する長音を削除する。例えば、「チー」は「チ」、「ティー」は「チ」に変換する。

- 長音削除の抑制

  長音の削除により無関係な語が同一視されるのを防ぐための規則である。対象文字列が「大文字 (＋小文字) ＋長音＋大文字 (＋小文字)(＋長音)」で、抑制対象の長音が２文字目（２文字目が小文字の場合は３文字目）の場合に限定している。例えば、「サーバ」あるいは「サーバー」は「サーバ」に変換し、２文字目の長音は削除しない。また、「ヴァ」を「バ」に変換する規則があることから、「サーヴァ」あるいは「サーヴァー」も「サーバ」に変換する。

複数の規則が適用可能な場合、最長一致で前側から順次規則を適用する。

一方、カタカナ表記の展開は、以下のような展開ルールによって検索時にのみ実施される。

　「チユイングガム」 → 「チユイングガム」「チユインガム」
　「コンサバ」 → 「コンサバ」「コンサーバ」

なお、展開用ルールは、大量のコーパスからカタカナ文字列を抽出し、異表記関係にあるものが正規化では同一視できないケースを拾い出し、ルール化した。

謝辞

本研究をまとめるにあたり、懇切丁寧なご指導と極めて有効なご助言と格別のご配慮をいただいた東京大学生産技術研究所の喜連川優教授に心から感謝申し上げる次第である。

また、ご指導・ご助言をいただいた東京大学大学院工学系研究科の田中英彦教授、武市正人教授、黒橋禎夫助教授、国立情報学研究所の安達淳教授、ならびに東京大学情報基盤センターの中川裕志教授に深く感謝する。

本研究は、株式会社リコーにおいて 1994 年頃から 2000 年ごろにかけて実施した、シグネチャファイル形式の n-gram 索引である文字成分表の研究開発、転置ファイル形式による n-gram 索引の研究開発、全文検索サーバー FTS の研究開発と製品化の成果である。本研究の機会をいただいき、ご指導頂いた國井秀子氏、小林清彦氏、飯沢篤志氏に深謝する。

特に、ブーリアン検索に関する研究は転置ファイル形式による n-gram 索引の研究着手時期の成果であり、松田透氏に多大な貢献を頂いた。文字成分表の研究開発においては岩崎雅二郎氏、全文検索サーバー FTS の研究開発においては山本研策氏、竹川弘志氏、池田哲也氏、平岡卓也氏、和久利智丈氏ら多くの方々にご協力頂いた。FTS の転置ファイルモジュールのプログラム開発では、リコーシステム開発の橋本信次氏、山田昌寛氏、佐々木利幸氏に拠るところが大きい。ランキング検索の研究は FTS の機能拡張として実施したものであり、真野博子氏、成田真澄氏、伊東秀夫氏に大変お世話になった。異表記正規化・形態素解析などの言語処理関係は、小島裕一氏、本間咲子氏、亀田雅之氏らにご担当頂いた。本論文をまとめるにあたっては江尻公一氏、藤本潤一郎氏からご助言と励ましを頂いた。皆様に感謝申し上げる次第である。

最後に、家族の協力と励ましに深く感謝する。

本研究に関する発表論文

ジャーナル論文

- 小川泰嗣：日本語文書検索のための頻度情報を用いた効率的部分文字列索引の提案, 情報処理学会論文誌, Vol. 37, No. 10, pp. 114–120, 1996.
- 小川泰嗣：文字成分表を用いた効率的文書ランキング検索方式, 情報処理学会論文誌, Vol. 38, No. 11, pp. 2286–2297, 1997.
- 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.
- Y. Ogawa and T. Matsuda: Overlapping statistical segmentation for effective indexing of Japanese text, Information Processing and Management, Vol. 35, No. 4, pp. 463–480, 1999.
- 小川泰嗣, 松田透, 橋本信次：N-gram 索引における複合条件の効率的な処理方法, 情報処理学会論文誌データベース, Vol. 40, No. SIG5 (TOD2), pp. 43–53, 1999.
- 小川泰嗣：擬似頻度法: n-gram 索引のための高速な日本語文書のランキング検索法, 電子情報通信学会論文誌, Vol. J83-D-I, No. 10, pp. 1043–1054, 2000.

国際会議での口頭発表

- Y. Ogawa and M. Iwasaki: A new character-based indexing method using frequency data for Japanese documents, in Proc. of 18th ACM SIGIR Conf., pp. 121–129, 1995.
- Y. Ogawa: Effective and efficient document ranking without using a large lexicon, in Proc. of 22nd VLDB Conf., pp. 192–202, 1996.
- Y. Ogawa, M. Kameda, and T. Matsuda: Inforium: A user-friendly document retrieval system, in Proc. of Int. Workshop on Information Retrieval with Oriental Languages, pp. 143–149, 1996.
- Y. Ogawa and T. Matsuda: Overlapping statistical word indexing: A new indexing method for Japanese documents, in Proc. of 20th ACM SIGIR Conf., pp. 226–234, 1997.

- Y. Ogawa and T. Matsuda: Optimizing query evaluation in n-gram indexing, in Proc. of 21st ACM SIGIR Conf., pp. 367–368, 1998.
- Y. Ogawa: Pseudo-frequency method: an efficient document ranking retrieval method for n-gram indexing, in Proc. of 23rd ACM SIGIR Conf., pp. 321–323, 2000.
- Y. Ogawa, H. Mano, M. Narita, and S. Honma: Structuring and expanding queries in the probabilistic model, in Proc. of 8th TREC, pp. 541–548, 2000.
- Y. Ogawa and H. Mano: RICOH at NTCIR-2, in Proc. of 2nd NTCIR Workshop, pp. 227–229, 2001.

国内会議での口頭発表

- 小川泰嗣：文字成分表を用いた効率的文書ランキング法の提案, アドバンストデータベースシステムシンポジウム'95, pp. 29–38 情報処理学会, 1995.
- 小川泰嗣, 松田透：ランキング文書検索におけるスコア合成法の評価, 研究会報告 FI47, pp. 95–100 情報処理学会, 1997.
- 小川泰嗣, 山本研策, 真野博子, 伊東秀夫：全文検索システムのための複数転置ファイルを用いた登録高速化とランキング検索, in Proc. of DEWS2002, pp. 227–229, 2002.

参考文献

[Aka96a] 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, pp. 35–42 情報処理学会, 1996.

[Aka96b] 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 (2), 第 53 回情報処理学会全国大会 (3), pp. 241–242, 1996.

[Aka97] 赤峯亨, 福島俊一, 清古勇治：日本語全文検索における文字組ベースのランキングの評価, 第 56 回情報処理学会全国大会 (3), pp. 116–117, 1997.

[Anh98] V.N. Anh and A. Moffat: Random access compressed inverted files, in Proc. of 8th Australian Database Conf., pp. 1–12, 1998.

[Bae99] R. Baeza-Yates and B. Ribeiro-Neto: Modern Information Retrieval, ACM Press, 1999.

[Bro95a] E.W. Brown: Execution Performance Issues in Full-Text Information Retrieval, Technical Report 95-81, University of Massachusetts, 1995.

[Bro95b] E.W. Brown: Fast Evaluation of Structured Queries for Information Retrieval, in Proc. of 18th ACM SIGIR Conf., pp. 30–38, 1995.

[Buc85] C. Buckley and A.F. Lewit: Optimization of inverted vector searches, in Proc. of 8th ACM SIGIR Conf., pp. 97–110, 1985.

[Cal94] J.P. Callen: Passage-level evidence in document retrieval, in Proc. of 17th ACM SIGIR Conf., pp. 302–309, 1994.

[Cav95] W.B. Cavnar: Using an n-gram-based document representation with a vector processing retrieval model, in Proc. of 3rd TREC, pp. 269–277, 1995.

[Che97] A. Chen, J. He, and L. Xu: Chinese Text Retrieval Without Using a Dictionary, in Proc. of 20th ACM SIGIR Conf., pp. 42–49, 1997.

[Cod70] E.F. Codd: A Relational Model of Data for Large Shared Data Banks, CACM, Vol. 13, No. 6, pp. 377–387, 1970.

[Cro79] W.B. Croft and D.J. Harper: Using probabilistic models of document retrieval without relevance information, Journal of Documentation, Vol. 35, pp. 285–295, 1979.

[Eli75] P. Elias: Universal codeword sets and representations of the integers, IEEE Transactions on Information Theory, Vol. 21, No. 2, pp. 194–203, 1975.

[Fal82] C. Faloutsos and H.V. Jagadish: On B-tree Indeces for Skewed Distributions, in Proc. of VLDB '92, pp. 363–374, 1982.

[Fol92] M. J. Folk and B. Zoellick: File Structures, Addison-Wesley, 1992.

[Fra92] W.B. Frakes and R. Basza-Yates: Information Retrieval: Data Structures and Algorithms, Prentice-Hall, New Jersey, 1992.

[Fuj94] 藤井洋一, 望月泰行, 鈴木克志, 丸山冬樹：全文検索システムにおける文字成分表の作成手法, 第 48 回情報処理学会全国大会 (4), pp. 159–160, 1994.

[Fuk93] 福島俊一, 田村美保子, 垣原睦治：全文検索用文字成分表の一圧縮方式, 第 47 回情報処理学会全国大会 (4), pp. 83–84, 1993.

[Fuk97] 福島俊一, 赤峯亨：全文検索システム RetrievalExpress の開発と評価, 第 3 回年次大会, pp. 361–364 言語処理学会, 1997.

[Fur94] 古瀬一隆, 浅田一繁, 飯沢篤志：DBMS へのシグネチャファイルの実装について, 信学技報 DE94-58, pp. 23–30 電子情報通信学会, 1994.

[Gol66] S.W. Golomb: Run-length encodings, IEEE Transactions on Information Theory, Vol. 12, No. 3, pp. 399–401, 1966.

[Har90] D. Harman and G. Candela: Retrieving Records from a Gigabyte of Text on a Minicomputer Using Statistical Ranking, Journal of the American Society for Information Science, Vol. 41, No. 8, pp. 581–589, 1990.

[Has81] R. Haskin: Special purpose processors for text retrieval, Database Engineering, Vol. 4, No. 1, pp. 16–29, 1981.

[Hat92] 畠山敦, 浅川悟志, 加藤寛次：ソフトウェアによるテキストサーチマシンの実現, 研究会報告 FI25, pp. 19–26 情報処理学会, 1992.

[Hed01] T. Hedlund, H. Keskustalo, A. Pirkola, E. Airio, and K. Jarvelin: Utaclir@CLEF2001 – Effects of Compound Splitting and N-gram Techniques, in Proc. of CLEF 2001 Workshop, pp. 118–136, 2001.

[Hir82] 平岡昭夫：リンク指向 DBMS G-BASE におけるリンク機能の拡張とその応用, 研究会報告 DBS82, pp. 1–10 情報処理学会, 1982.

[Hul93] D. Hull: Using Statistical Testing in the Evaluation of Retrieval Experiments, in Proc. of 16th ACM SIGIR Conf., pp. 329–338, 1993.

[Ito91] 伊藤篤, 望主雅子, 小島裕一：日本語形態素解析における素性を用いた解析方式, 第 43 回情報処理学会全国大会 (3), pp. 113–114, 1991.

[Iwa93] 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, pp. 1–10 情報処理学会, 1993.

[JIS95a] JIS X 3005: データベース言語 SQL, 1995.

[JIS95b] JIS X0202: 国際符号化文字集合（ＵＣＳ）, 1995.

[JIS95c] JIS X0208: 国際符号化文字集合（ＵＣＳ）, 1995.

[JIS95d] JIS X0221: 国際符号化文字集合（ＵＣＳ）, 1995.

[Jon98] G. Jones, T. Sakai, et al.: Experiments in Japanese Text Retrieval and Routing using the NEAT System, in Proc. of 21st ACM SIGIR Conf., pp. 197–205, 1998.

[Kag96] K. Kageura: Bigram statistics revisited: A comparative examination of some statistical measures in morphological analysis of Japanese kanji sequences, `http://www.dcs.shef.ac.uk/ kyo/llc1.ps`, 1996.

[Kam95] 亀田雅之：軽量・高速な日本語解析ツール「簡易日本語解析系 QJP」, 第 1 回年次大会, pp. 349–352 言語処理学会, 1995.

[Kan98] N. Kando, K. Kageura, M. Yoshioka, and K. Oyama: Phrase processing methods for Japanese text retrieval, ACM SIGIR Forum, Vol. 32, No. 2, pp. 23–28, 1998.

[Kan02] 神門典子, 野末俊比古（編）: NTCIR-3 ワークショップ予稿集, 国立情報学研究所, 2002.

[Kaw92] 川下靖司, 浅川悟志, 坂田順, 畠山敦：フルテキストサーチシステム Bibliotheca/TS の開発 (2), 第 45 回情報処理学会全国大会 (3), pp. 241–242, 1992.

[Kaw96] 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 回全国大会予稿集 (3), pp. 237–238 情報処理学会, 1996.

[Kes96] 芥子育雄他：情報検索システム評価用ベンチマーク Ver.1.0(BMIR-J1) について, 研究会報告 DBS-106, pp. 139–145 情報処理学会, 1996.

[Kik92] 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.

[Kis00] 岸和田和明：情報検索とテストコレクション, 情報処理, Vol. 41, No. 8, pp. 898–901, 2000.

[Kit98] 木谷強：全文データベースの構築技法, 情報管理, Vol. 41, No. 5, pp. 367–380, 1998.

[Kit02] 北研二, 津田和彦, 獅々堀正幹：情報検索アルゴリズム, 共立出版, 2002.

[Kna94] D. Knaus and P. Schäuble: Effective and Efficient Retrieval from Large and Dynamic Document Collections, in Proc. of 2nd TREC, pp. 163–170, 1994.

[Koj91] 小島裕一, 望主雅子：日本語形態素解析における三項関係を扱う一手法, 第43回情報処理学会全国大会 (3), pp. 115–116, 1991.

[Kwo97] K.L. Kwok: Comparing Representations in Chinese Information Retrieval, in Proc. of 20th ACM SIGIR Conf., pp. 34–41, 1997.

[Lee96] J.H. Lee and J.S. Ahn: Using N-Grams for Korean Text Retrieval, in Proc. of 19th ACM SIGIR Conf., pp. 216–224, 1996.

[Lin93] G. Linoff and C. Stanfill: Compression of Indexes with Full Positional Information in Very Large Text Databases, in Proc. of 16th ACM SIGIR Conf., pp. 88–95, 1993.

[Mat97a] 松井くにお, 難波巧, 井形伸之：高速テキスト検索エンジン, 研究会報告, 第DD-7巻, pp. 15–21 情報処理学会, 1997.

[Mat97b] 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.

[Mic96] 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.

[Mic97] Microsoft Corporation: Microsoft OLE DB V1.1 プログラマーズリファレンスマニュアル, アスキー, 1997.

[Miy90] 宮原末治, 小橋史彦：文字連接を用いたフルテキスト検索の高速化, 第40回情報処理学会全国大会, p. 880, 1990.

[Moc91] 望主雅子, 伊藤篤：日本語形態素解析における素性の導入, 第43回情報処理学会全国大会 (3), pp. 111–112, 1991.

[Mof94a] A. Moffat and J. Zobel: Fast Ranking in Limited Space, in Proc. of Int. Conf. on Data Engineering, pp. 428–437, 1994.

[Mof94b] A. Moffat and J. Zobel: Self-indexing inverted files for fast text retrieval, Technical Report CITIR/TR-94-2, RMIT, The University of Melbourne, 1994.

[Mof95] A. Moffat, J. Zobel, and S.T. Klein: Improved inverted file processing for large text databases, Australian Computer Science Communications, Vol. 17, No. 2, pp. 162–171, 1995.

[Mof96a] A. Moffat and L. Stuiver: Exploiting clustering in inverted file compression, in Proc. of Data Compression Conf. '96, pp. 82–91, 1996.

[Mof96b] A. Moffat and J. Zobel: Self-indexing inverted files for fast text retrieval, ACM Transactions on Information Systems, Vol. 14, No. 4, pp. 349–379, 1996.

[Mya96] S.H. Myaeng and D.H. Jang: On Language Dependandy in Indexing, in Proc. of Int. Workshop on Information Retrieval with Oriental Languages, pp. 17–23, 1996.

[Nak95] 中渡瀬秀一：統計的手法によるテキストからのキーワード抽出, 信学技報 DE95-1, pp. 9–16 電子情報通信学会, 1995.

[Oga95a] 小川泰嗣：文字成分表を用いた効率的文書ランキング法の提案, アドバンストデータベースシステムシンポジウム'95, pp. 29–38 情報処理学会, 1995.

[Oga95b] Y. Ogawa and M. Iwasaki: A new character-based indexing method using frequency data for Japanese documents, in Proc. of 18th ACM SIGIR Conf., pp. 121–129, 1995.

[Oga96a] Y. Ogawa: Effective and efficient document ranking without using a large lexicon, in Proc. of 22nd VLDB Conf., pp. 192–202, 1996.

[Oga96b] 小川泰嗣：日本語文書検索のための頻度情報を用いた効率的部分文字列索引の提案, 情報処理学会論文誌, Vol. 37, No. 10, pp. 1839–1849, 1996.

[Oga96c] Y. Ogawa, M. Kameda, and T. Matsuda: Inforium: A user-friendly document retrieval system, in Proc. of Int. Workshop on Information Retrieval with Oriental Languages, pp. 143–149, 1996.

[Oga97a] 小川泰嗣：文字成分表を用いた効率的文書ランキング検索方式, 情報処理学会論文誌, Vol. 38, No. 11, pp. 2286–2297, 1997.

[Oga97b] Y. Ogawa and T. Matsuda: Overlapping statistical word indexing: A new indexing method for Japanese documents, in Proc. of 20th ACM SIGIR Conf., pp. 226–234, 1997.

[Oga97c] 小川泰嗣, 松田透：ランキング文書検索におけるスコア合成法の評価, 研究会報告 FI47, pp. 95–100 情報処理学会, 1997.

[Oga98] Y. Ogawa and T. Matsuda: Optimizing query evaluation in n-gram indexing, in Proc. of 21st ACM SIGIR Conf., pp. 367–368, 1998.

[Oga99a] 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.

[Oga99b] Y. Ogawa and T. Matsuda: Overlapping statistical segmentation for effective indexing of Japanese text, Information Processing and Management, Vol. 35, No. 4, pp. 463–480, 1999.

[Oga99c] 小川泰嗣, 松田透, 橋本信次：N-gram 索引における複合条件の効率的な処理方法, 情報処理学会論文誌データベース, Vol. 40, No. SIG5 (TOD2), pp. 43–53, 1999.

[Oga00a] Y. Ogawa: Pseudo-frequency method: an efficient document ranking retrieval method for n-gram indexing, in Proc. of 23rd ACM SIGIR Conf., pp. 321–323, 2000.

[Oga00b] 小川泰嗣：擬似頻度法: n-gram 索引のための高速な日本語文書のランキング検索法, 電子情報通信学会論文誌, Vol. J83-D-I, No. 10, pp. 1043–1054, 2000.

[Oga00c] Y. Ogawa, H. Mano, M. Narita, and S. Honma: Structuring and expanding queries in the probabilistic model, in Proc. of 8th TREC, pp. 541–548, 2000.

[Oga01] Y. Ogawa and H. Mano: RICOH at NTCIR-2, in Proc. of 2nd NTCIR Workshop, pp. 227–229, 2001.

[Oga02] 小川泰嗣, 山本研策, 真野博子, 伊東秀夫：全文検索システムのための複数転置ファイルを用いた登録高速化とランキング検索, in Proc. of DEWS2002, pp. 227–229, 2002.

[Rad81] T. Radecki: Outline of a fuzzy logic approach to information retrieval, International Journal of Man-Machine Studies, Vol. 14, pp. 169–178, 1981.

[Raj97] K. Rajaraman, K. F. Lai, and Y. Changwen: Experiments on Proximity Based Chinese Text Retrieval in TREC 6, in Text REtrieval Conference, pp. 559–576, 1997.

[Rob76] S.E. Robertson and K. Sparck Jones: Relevance weighting of search terms, Jounal of the American Society for Information Science, Vol. 27, pp. 129–146, 1976.

[Rob77] S.E. Robertson: The probability ranking principle in IR, Jounal of Documentation, Vol. 33, pp. 294–304, 1977.

[Rob94] S.E. Robertson and S. Walker: Some Simple Effective Approximations to the 2-Poisson Model for Probabilistic Weighted Retrieval, in Proc. of 17th ACM SIGIR Conf., pp. 232–241, 1994.

[Rob97] S.E. Robertson and S. Walker: On relevance weights with little relevance information, in Proc. of 20th ACM SIGIR Conf., pp. 16–24, 1997.

[Sal75] G. Salton, A Wong, and C. S. Yang: A vector space model for automatic indexing, Communications of the ACM, Vol. 18, No. 11, pp. 613–620, 1975.

[Sal83a] G. Salton, E.A. Fox, and H. Wu: Extended Boolean Information Retrieval, Communications of the ACM, Vol. 26, No. 12, pp. 1022–1036, 1983.

[Sal83b] G. Salton and M. J. McGill: Introduction to Modern Information Retrieval, McGraw-Hill, New York, 1983.

[Sal98] G. Salton, J. Allan, and C. Buckley: Approaches to passage retrieval in full text information systems, in Proc. of 16th ACM SIGIR Conf., pp. 49–58, 1998.

[Sek00] 関根聡, 井佐原均, 栗山和子：日本におけるテストコレクションと評価の動向, 情報処理, Vol. 41, No. 8, pp. 902–905, 2000.

[Sug96] 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.

[Tam95] 玉置志津, 藤原健史, 西谷紘一：シグニチャ法を用いた日本語文書検索システム, 第50回情報処理学会全国大会 (4), pp. 39–40, 1995.

[Tok98a] 東京外国語大学語学研究所（編）: 世界の言語ガイドブック1（ヨーロッパ・アメリカ地域）, 三省堂, 1998.

[Tok98b] 東京外国語大学語学研究所（編）: 世界の言語ガイドブック2（アジア・アフリカ地域）, 三省堂, 1998.

[Tok99] 徳永健伸：情報検索と言語処理, 言語と計算, 東京大学出版会, 1999.

[Tom94] A. Tomasic, H. Garicia-Molina, and K. Shoens: Incremental Updates of Inverted Lists for Text Document Retrieval, in Proc. of 1994 ACM SIGMOD Conf., pp. 289–300, 1994.

[Tsu01] 土屋信明（編）: C++標準ライブラリ 〜 チュートリアル&リファレンス, アスキー, 2001.

[Tur95] H. Turtle and J. Flood: Query Evaluation: Strategies and Optimizations, Information Processing and Management, Vol. 31, No. 6, pp. 831–850, 1995.

[Umi88] 海野敏：出現頻度情報に基づく単語重みづけの原理, Library and Information Science, No. 26, pp. 67–88, 1988.

[Voo02] E. Voorhees ed.: Proceedings of TREC-10, NIST, 2002.

[Wil97] R. Wilkinson: Chinese Document Retrieval at TREC-6, in Text REtrieval Conference, pp. 25–29, 1997.

[Wit94] I.H. Witten, A. Moffat, and T.C. Bell: Managing Gigabytes: Compressing and Indexing Documents and Images, Van Nostrand Reinhold, 1994.

[Won93] W.Y.P. Wong and D.L. Lee: Implementations of partial document ranking using inverted files, Information Processing and Management, Vol. 29, No. 5, pp. 647–669, 1993.

[Yam98a] M. Yamamoto and K.W. Church: Using suffix arrays to compute term frequency and document frequecy for all substrings in a corpus, in Proc. of 6th Workshop on Very Lage Corpora, pp. 28–37, 1998.

[Yam98b] 山本毅雄, 橋爪宏達, 神門典子, 清水美都子：全文検索〜技術と応用, 丸善, 1998.

[Yok97] 横山昌典：高速全文検索エンジン, FUJITSU, Vol. 48, No. 2, pp. 155–158, 1997.

[Zip49] G.K. Zipf: Human Behavior and the Principle of Least Effort, Addison-Wesley, 1949.

[Zob92] J. Zobel, A. Moffat, and R. Sacks-Davis: An Efficient Indexing Technique for Full-Text Database Systems, in Proc. of 18th VLDB Conf., pp. 356–362, 1992.

[Zob93] J. Zobel, A. Moffat, and R. Sacks-Davis: Storage management for files of dynamic records, in Proc. of 4th Australian Database Conf., pp. 26–38, 1993.

[Zob94] J. Zobel, A. Moffat, and K. Ramamhanrao: Inverted files versus signature files for text indexing, Technical Report CITIR/TR-94-1, RMIT, The University of Melbourne, 1994.