# N.M-gram：ハッシュ値付きN-gram索引による全文検索の一手法

平林幹雄[†] 江渡浩一郎[††]

全文検索システムの転置索引を実現するにあたり，テキストデータからN-gram法によって切り出したトークンを検索キーにする手法が広く用いられている．この手法には，言語中立性や再現率の完全性という利点がある反面，索引ファイルのサイズが肥大化して空間効率が悪化するという欠点がある．検索の際にクエリから切り出した各トークンが対象文書のテキスト内でも連接しているかどうかを判断するためには，索引ファイル内にトークンの文書内での出現位置を記録しておくことが必要となるが，この位置情報が索引ファイルの肥大化の一因となっている．本稿では，N-gram法の欠点である索引ファイルの空間効率を改善する手法として，**N.M-gram法**を提案する．N.M-gram法では，各トークンの文書内での位置情報のかわりに後続のトークンのハッシュ値を用いることによって，N-gram法の利点である言語中立性や再現率の完全性を保持したまま，空間効率を改善することができる．

# N.M-gram: A Method of Full-text Search by N-gram Index with Hash Values

Mikio Hirabayashi[†] and Kouichirou Eto[††]

When constructing inverted index for full-text search system, using N-gram is very popular for tokenizing text data of target documents. Although the method has many advantages like language neutrality and perfect recall ratio, it has also shortage that the index file becomes large. The tokens extracted from documents tend to be enormous. The system needs to record each offset of tokens into the index file because the offset is used for checking adjacency of tokens. The index file tends to be large because of the offset. In this paper, we describe *N.M-gram* method, which improves space efficiency of N-gram. The method uses hash values of succeeding tokens instead of offset in each document. The method can improve space efficiency without losing advantages of N-gram.

## 1. はじめに

近年，電子文書の増大にともない全文検索システムの需要は増大している．大規模な文書群を高速に検索するためには，検索対象の文書群のテキストからトークンを抽出して転置することによって生成される索引ファイルを用いるのが一般的である．この索引ファイルは**転置インデックス**と呼ばれる．テキストからトークンを抽出する主要な手法の1つとして，**N-gram法**があげられる[1),2)]．N-gram法は，文字数のみに基づいてトークンを切り出すため，対象文書の言語に対して中立であり，分かち書き（形態素解析）に見られるような解析ミスが起きず，検索の再現率が完全になるという利点がある．一方，N-gram法には，抽出するトークンの数が膨大になるために，転置インデックスのサイズが肥大化して空間効率が悪化するという欠点がある．N-gram法のインデックスに対して検索を行う際には，クエリから切り出した各トークンが対象文書のテキスト内でも連接しているかどうかを判定する必要がある．そのため，転置インデックス内にトークンの文書内での出現位置を記録しておくことが必要となるが，この位置情報が転置インデックスの肥大化の一因となっている．

本稿では，N-gram法による転置インデックスの肥大化を軽減して空間効率を改善する手法として，**N.M-gram法**を提案する．これは，N-gram法によって切り出したトークンの位置情報の代用としてハッシュ値を用いることで，転置インデックスに記録する情報量の抑制を図る手法である．N.M-gram法を用いることによって，N-gram法の利点である言語中立性や再現率の完全性を保持したまま，空間効率を改善することができる．転置インデックスの空間効率は全文検索システムのスケーラビリティと直結するので，空間効率の改善は，大規模な全文検索システムへの要請が高まる中で意義を持つ．また，本稿では，全文検索システ

† 株式会社ミクシィ
mixi, Inc.
†† 独立行政法人産業技術総合研究所
National Institute of Advanced Industrial Science and Technology (AIST)

ム Hyper Estraier における N.M-gram 法の実装について述べるとともに，N.M-gram 法と N-gram 法の空間効率を比較した評価についても述べる．

## 2. N.M-gram 法のアルゴリズム

本章では，まず最初に N.M-gram 法の基盤となる N-gram 法のモデルについて説明し，そのうえで N.M-gram 法のアルゴリズムおよびその特徴について述べる．

### 2.1 N-gram 法におけるデータ構造

N-gram 法は，全文検索システムにおいて対象文書のテキストからトークンを抽出して転置インデックスを構築するための主要な手法の 1 つである．N-gram 法においては，対象文書のテキストから，連続した一定の文字数の文字列をトークンとして切り出す[3]．テキストを 1 文字ずつ分割する方法を 1-gram（uni-gram）法，2 文字ずつ分割する方法を 2-gram（bi-gram）法，3 文字ずつ分割する方法を 3-gram（tri-gram）法といい，N 文字ずつ分割する方法を総称して N-gram 法という．トークンを構成する文字が何文字であっても，トークンは 1 文字ずつずらしながら重複して切り出される．たとえば「東洲斎写楽」という文字列に 2-gram 法を適用すると，「東洲」「洲斎」「斎写」「写楽」というトークンが得られる．

切り出された各々のトークンには，その位置情報が付与される．位置情報は，そのトークンを含む文書の識別子と，そのトークンが文章中の何番目に出現したかというオフセットからなる．転置インデックスにトークンの情報を記録する際には，文書内における同一パターンのトークンの情報をまとめてレコードを構成する．レコードはトークンの文字列を検索キーとして転置され，ハッシュ表や B+木等のインデックスをともなったデータベースに記録される（**図 1**）．

記録した位置情報は，検索時に，トークンの存在と連続性を調べるために使われる．たとえば「東洲斎」という検索要求に対しては，「東洲」および「洲斎」が同一の文書に含まれるかどうかをまず調べ，次に，「東洲」を N 番目としたときに「洲斎」が N+1 番目にあるものを該当と見なすことになる．

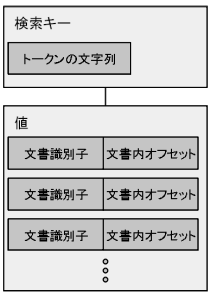


**図 1** N-gram 法の転置インデックスの構造
Fig. 1 The structure of the inverted index of N-gram.

文書内オフセットにおいて，変域を制限せずに効率的に値を表現するためには，可変長のデータ型を用いるのが一般的である[4),5)]．可変長のデータ型においては，小さい値は短いビット長で表現できるが，大きい値を表現するには長いビット長が必要となる．したがって，個々の文書のテキストが長くなるほど文書内オフセットのビット長は長くなる．N-gram 法によるトークンの数は文書内の文字数とほぼ等しくなるので，文書識別子の情報量よりも文書内オフセットの情報量の方が支配的になる．したがって，N-gram 法の転置インデックスにおいては，対象文書のテキストが長くなるほど，転置インデックスのサイズが肥大化し，すなわち空間効率は悪化する．なお，文書内オフセットのリストは単調増加列であるため，より値の低い差分列に可逆変換することが可能であり，それによって情報量を抑制することもできる．文書内オフセットを固定長のデータ型で表現した場合は，転置インデックスの空間効率は対象文書のテキストの長さに依存せずに済む．しかし，固定長データ型において十分な変域のビット長を設定すると，可変長のデータ型を用いるよりも空間効率が悪くなる．

N-gram 法の転置インデックスを使った検索においては，検索語の文字数が N 文字でない場合に時間効率が悪化するという問題がある．N 文字未満の検索語で検索を行う場合には，その検索語に前方一致するすべてのトークンによる検索結果の和集合をとることになる．N 文字を超える検索語で検索を行う場合には，検索語を N 文字からなる複数のトークンに分割し，その各々の検索結果の積集合をとり，さらに位置情報を参照してトークンの連続性を確認する必要がある．

### 2.2 N.M-gram 法におけるデータ構造

本稿で提案する N.M-gram 法とは，言語中立性や再現率の完全性という N-gram 法の特徴を保持しつつ，空間効率を改善する方式である．N.M-gram 法においても，N-gram 法と同様に N 文字のトークンを 1 文字ずつずらしながら切り出し，各トークンに対して文書の識別子と，文書内オフセットの代わりに後続トークンのハッシュ値を付与して転置インデックスに記録する．ただし，文書内オフセットの代用として，より短いビット長のハッシュ値を用いることにより，記憶領域の節約を図る．ここで，後続 1 個のトークンのハッシュ値を付与するものを N.1-gram 法，後続 2 個のトークンのハッシュ値を付与するものを N.2-gram

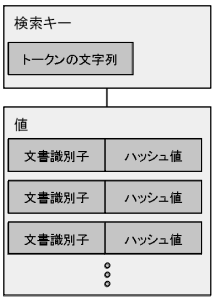


**図 2** N.M-gram 法の転置インデックスの構造
Fig. 2 The structure of the inverted index of N.M-gram.

法等と呼び，後続 M 個のトークンのハッシュ値を付与する手法を総称して N.M-gram 法と呼ぶことにする．たとえば「東洲斎写楽」に 2.2-gram 法を適用すると，「東洲」には「洲斎」のハッシュ値と「斎写」のハッシュ値が付与され,「洲斎」には「斎写」のハッシュ値と「写楽」のハッシュ値が付与され,「斎写」には「写楽」のハッシュ値が付与される．

N-gram 法による文書内オフセットと同様に，後続のトークンのハッシュ値を用いてもトークンの連続性を調べることができる．たとえば「東洲斎」という検索要求に対しては,「東洲」および「洲斎」が同一の文書に含まれ,「東洲」の直後のトークンのハッシュ値が「洲斎」のハッシュ値と一致するものを該当と見なせばよい．

転置インデックスにトークンの情報を記録する際には，N-gram 法と同様に，文書内における同一パターンのトークンの情報をまとめてレコードを構成する（**図 2**）．たとえば 2.2-gram 法において同一文書内に「アメリカ」および「アメヨコ」という文字列が現れた場合,「アメ」を検索キーとして，文書の識別子および「メリ」「リカ」「メヨ」「ヨコ」の各々のハッシュ値を連結したレコードが記録される．

後続のトークンのハッシュ値は固定長のデータ型で表現するため，対象文書のテキストの長さにかかわらず一定の長さになる．また，変域を狭めて短いビット長にすることができるため，N.M-gram 法におけるハッシュ値の情報量は N-gram 法における文書内オフセットの情報量に比べて小さくなる．さらに，後続のハッシュ値の同一の組合せが同一文書内で複数回現れた場合はそれを 1 つに代表させることができるため，組合せが重複したトークン群の情報量が節約できる．たとえば,「アメリカ」という文字列が 2 回現れた場合,「アメ」という検索キーには,「メリ」「リカ」「メリ」「リカ」の各々のハッシュ値を連結したレコードではなく,「メリ」「リカ」の各々のハッシュ値を連結したレコードが記録される．以上の性質により，N.M-gram 法の転置インデックスの空間効率は，N-gram 法の転置インデックスの空間効率よりも良くなる．

転置インデックスの空間効率が良いということは，大規模な文書群を対象にした転置インデックスを作成した場合にも，そのサイズを小さく抑えられるということである．転置インデックスのサイズが小さいことはハードディスク等の補助記憶装置の節約になるだけでなく，主記憶装置上にキャッシュできる情報量を増やすことができるため，検索速度や転置インデックスの更新速度の高速化にも寄与する．

N.M-gram 法では，N 文字から N+M 文字までのトークンによる検索を効率的に行うことができる．たとえば 2.2-gram 法では，2-gram 法でトークンを分割しながらも，4-gram 法と同等の絞り込みを行う能力を持つ．4-gram 法だと，3 文字以下の検索語では検索できないか，検索処理に多大な時間を要することになるが，2-gram 法で効率が悪くなるのは検索語が 1 文字の場合のみである．すなわち，N.M-gram 法では，N-gram の N を小さくしても効率的に検索できる．N-gram 法においても，N.M-gram 法においても，転置インデックスの構築にかかる時間計算量および領域計算量は，対象文書の規模に対して $O(n)$ であるが，N を減らしてトークンの文字数を少なくするとトークンの種類（異なり語数）が少なくなるため，転置インデックスにおける検索キーの情報量が少なくて済む．検索キーが少ないほど，各トークンの出現情報を参照および更新する際の時間効率は向上する．

### 2.3 N.M-gram 法の特性

後続のトークンについてはハッシュ値で表現するので，異なるパターンのトークンのハッシュ値が衝突する可能性があり，その場合は意図せぬ文書が検索結果に現れてしまう．ただし，その確率は，ハッシュ値を 1 バイト（オクテット）で表現するならば $1/256^M$ でしかないので，実用的には無視することもできると考えられる．意図せぬ結果が許容できない場合は，該当した文書の実データを参照して，検索語が実際に含まれているかどうかを調べればよい．この検査は結果の表示件数の分だけ行えばよいので，時間計算量は転置インデックスの規模に対して $O(1)$，表示件数に対して $O(n)$ で済む．

利便性のために検索結果において該当文書のスニペット（紹介文）を表示することが考えられる．スニペットは該当文書のテキストの中から検索語の周辺部分を抽出することによって生成されるが，位置情報を保持しない N.M-gram 法の実装ではその処理を効

率的に行うことはできない．しかし，トークンの連続性の検査と同様に，スニペット生成の計算量は転置インデックスの規模に対して $O(1)$，表示件数に対して $O(n)$ で済む．

検索結果において該当文書のスコアリング（順位付け）を行うことも考えられる．N-gram 法ではトークンの位置情報を用いてスコアの重み付けを行うことができるが，位置情報を保持しない N.M-gram 法の実装ではそれは不可能である．この問題に対しては，スコア情報をトークンに別途付与することで対処できる．スコア値の情報量は 8 ビット程度でも十分な精度が出るため，文書内オフセットを保持するよりはオーバヘッドは小さいと考えられる．

### 2.4 関 連 研 究

不確定ではあるが高い確率で検索対象文書の絞り込みができるという点においては，N.M-gram 法を**シグネチャファイル**の延長ととらえることもできる．シグネチャファイルにおいては，文書内に出現した各トークンに対してハッシュ値を算出し，unary 符号で表現したハッシュ値のビット論理和を文書のシグネチャとして記録する[6),7)]．検索時には，検索クエリのシグネチャと各対象文書のシグネチャを比較し，検索クエリにおいて立っているビットのすべてが対象文書において立っていれば，その文書を該当と見なす．ただし，unary 符号のハッシュ値は binary 符号のハッシュ値に比べて記憶領域あたりのエントロピーが低く，ハッシュ値の衝突率が高くなるため，十分な精度を出すためにはシグネチャのサイズを大きくする必要がある．N.M-gram 法は，転置インデックスのレコードに binary 符号のシグネチャを分散させて記録する手法だともいえる．対象文書のすべてのシグネチャを比較する必要はなく，検索クエリの先頭のトークンで絞り込んだ候補のシグネチャのみを比較するので，N.M-gram 法はシグネチャファイルに比べて検索精度が高く，時間効率も優れている．

N-gram 法の転置インデックスの空間効率を向上させる手法として，**n-gram/2L 法**が提案されている[8)]．これは転置インデックスを 2 つに分割することによって位置情報の冗長性を削減する手法である．転置インデックスを構築する際には，まず，N すなわちトークンの長さを大きくした転置インデックス（バックエンドインデックス）を構築する．そのうえで，バックエンドインデックスに出現した各トークンを対象とした，より短いトークン長の転置インデックス（フロントエンドインデックス）を構築する．たとえば 4-gram のバックエンドインデックスと 2-gram のフロントエンドインデックスを用いる．検索時には先にフロントエンドインデックスを参照してそのトークンを含むバックエンドインデックスのレコードを特定し，次にバックエンドインデックスから該当のレコードを取得して位置情報を特定する．n-gram/2L 法は DNA 情報等の冗長性の高い文字列を対象とした転置インデックスを構築する際には高い空間効率を実現することができるが，自然文等の比較的冗長性の低い文字列を対象とした場合には空間効率の向上は見込めない．

## 3. N.M-gram 法の実装

本章では，全文検索システム **Hyper Estraier** [9)] における N.M-gram 法の実装について詳述する．

### 3.1 転置インデックスの構築

Hyper Estraier は 2.2-gram 法による転置インデックスを実装している．転置インデックスを構築する際には，対象文書から取り出したテキストから，2-gram 法に基づいてトークンを切り出す．すなわち，連続する 2 文字のパターンを切り出す処理を，先頭から 1 文字ずつずらしながら末尾まで行う．ただし，末尾のみは 1 文字のトークンとする．

トークンはすべて UTF-8 のバイト列として扱う．抽出した各々のトークンに対し，後続の 2 つのトークンに別々のハッシュ関数を適用し，各 1 バイトずつのハッシュ値を算出する．ただし，変域は 0x01 から 0xFC までとする．また，後続がない場合は 0xFF とする．

次に，トークンを整理して，同じ文字列のトークンに連なるハッシュ値を単一のレコードにまとめる．レコードの先頭には，対象文書の識別子を 4 バイトの値として置く．その後には，同じトークンの出現数を数えたうえで，その値を 1 バイトに丸めておく．この値は検索時のスコアリングに用いられる．スコア値の後には出現したハッシュ値のリストを連結する．ただし，連結するハッシュ値がすでに出現したものとまったく同じ場合は連結しない．さらに，各レコードの末尾に番兵として 0x00 を加え，複数の文書のデータを番兵の後に次々と連結できるようにする．

スコア値は文書内のトークンの数の平方で割ることによって調整される．その理由は，テキストが長くなるほどそれに含まれるトークンの数は多くなり，検索語に含まれるトークンが出現する確率も高まるが，検索語とは関係ない内容の文章の量も多くなる傾向があるため，その文書を検索結果の上位に提示すると検索精度が下がると考えられるからである．ただし，トークン数が一定以下の場合は調整は行わない．

**表 1** トークンごとに整理されたレコード群の例
Table 1 An example of records organized by tokens.

| キー | 値 |
|---|---|
| ファ | 0x00, 0x00, 0x00, 0x01, 0x02, 0x99, 0xF6, 0x00 |
| ァイ | 0x00, 0x00, 0x00, 0x01, 0x02, 0x8F, 0x02, 0x8F, 0x6D, 0x00 |
| イル | 0x00, 0x00, 0x00, 0x01, 0x02, 0x88, 0x66, 0xB1, 0x1D, 0x00 |
| ルと | 0x00, 0x00, 0x00, 0x01, 0x01, 0x42, 0x77, 0x00 |
| とフ | 0x00, 0x00, 0x00, 0x01, 0x01, 0xF2, 0x48, 0x00 |
| ル保 | 0x00, 0x00, 0x00, 0x01, 0x01, 0x20, 0x84, 0x00 |
| 保存 | 0x00, 0x00, 0x00, 0x01, 0x01, 0x1D, 0xFF, 0x00 |
| 存 | 0x00, 0x00, 0x00, 0x01, 0x01, 0xFF, 0xFF, 0x00 |

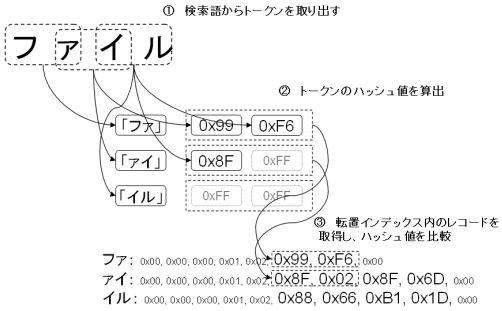


**図 3** 検索のアルゴリズム
Fig. 3 The algorithm of search.

以上の処理を「ファイルとファイルの保存」という文書に行ったとすると，**表 1** に示すレコード群が得られる．

トークンをレコードのキーとし，出現数とハッシュ値と番兵を連結したデータをレコードの値として，B+木構造のデータベースに保存する．このデータベースが転置インデックスとなる．なお，B+木の実装にはリーフノードを圧縮してファイルに書き込む**圧縮 B+木**を用いる．圧縮アルゴリズムには deflate（LZ77 およびハフマン符号）を用いる．複数の対象文書を転置インデックスに登録する際には，番兵によって区切りが分かるので，追加するレコードの値を既存のレコードの値の末尾に連結していけばよい．

### 3.2 検　　索

検索時には，検索語に対しても転置インデックス作成時と同様の N-gram 法の解析をしたうえで，各トークンのハッシュ値の算出を行い，各トークンに対応するレコードを転置インデックスから取り出し，ハッシュ値に検索クエリの条件と一致する部分があるか検査する（**図 3**）．

ハッシュ値が一致するすべてのトークンに含まれる文書識別子があれば，その文書には検索語が存在することになる．検索語に複数のトークンが含まれる場合はそれらのスコアは加算され，そのうえで TF-IDF 法に基づいて調整した値が最終的な文書のスコアとなる．

**表 2** テストプログラムの実行環境
Table 2 The testing environment for the performance measurement.

| | |
|---|---|
| プロセッサ | 1.70 GHz（Pentium M） |
| 主記憶装置 | 1 G バイト（PC-2700） |
| ハードディスク | 30 GB（ATA-100，4200 rpm/9.5 mm） |
| OS | Linux 2.4.31（ext2 ファイルシステム） |
| ライブラリ | GLIBC 2.3.3，QDBM 1.8.60，ZLIB 1.2.3 |

該当の文書群はスコア値を用いてソートされて，検索結果として提示される．

なお，検索語に含まれるトークンで実際に検査を行うものは，N-gram 法では N 個ごとでよく，N.M-gram 法では N+M 個ごとでよい．たとえば，「ファイル」という検索語を 2-gram 法で考えると，「ファ」が 1 番目に含まれ，「イル」が 3 番目に含まれるならば，「ァイ」が 2 番目に含まれることは検査しなくても分かる．同じく「ファイル」を 2.2-gram 法で考えると，「ファ」のレコードを取り出せば「ァイ」と「イル」が後続するかどうかも分かるので，その時点で検索処理を完了できる．

## 4. 評　　価

本章では，N.M-gram 法の典型例として 2.2-gram 法を取り上げ，N-gram 法との比較実験によってその性能を評価する．まず最初に実験環境について述べる．次に，多数の文書群の転置インデックスを生成した際の空間効率および時間効率について N-gram 法と比較し，さらに，n-gram/2L 法との比較を行う．

### 4.1 実 験 環 境

本実験で用いたテストプログラムの実行環境は，**表 2** のとおりである．転置インデックスはデータベースライブラリ QDBM [10] の圧縮 B+木の API を用いて実装した．圧縮アルゴリズムには deflate 圧縮を用い，B+木の論理ページは異なり語 99 個ごとに割り当てるようにした．

N.M-gram 法の転置インデックスでは，レコードの先頭には，対象文書の識別子を 4 バイトの値として置き，その後には出現したハッシュ値のリストを連結する．ただし，連結するハッシュ値が前にあるものとまったく同じ場合は連結しない．ハッシュ値は 1 バイトで表現する．さらに，各レコードに番兵として 0x00 を末尾に加える．N-gram 法の転置インデックスも同様の構造をとるが，ただしハッシュ値の代わりにトークンの位置のリストを記録する．出現位置のリストは差分列に変換したうえで，byte aligned 符号による 128 進数の可変長数値として表現したものを連結して表現する．

**表 3** 2.2-gram 法と N-gram の比較
Table 3 The comparison between 2.2-gram and N-gram.

| 方式 | 2.2-gram | 2-gram | 3-gram | 4-gram |
|---|---|---|---|---|
| 異なり語数 | 560,149 | 560,149 | 3,596,486 | 8,370,344 |
| レコードサイズ合計 | 91.72 MB | 91.92 MB | 140.83 MB | 212.95 MB |
| キーサイズ小計 | 3.06 MB | 3.06 MB | 28.82 MB | 87.42 MB |
| 値サイズ小計 | 88.65 MB | 88.86 MB | 112.01 MB | 125.53 MB |
| 圧縮 B+木のサイズ | 60.44 MB | 65.85 MB | 108.08 MB | 166.91 MB |
| 圧縮 B+木の圧縮率 | 0.659 | 0.716 | 0.767 | 0.783 |
| 構築時間 | 158 秒 | 165 秒 | 1,551 秒 | 7,510 秒 |

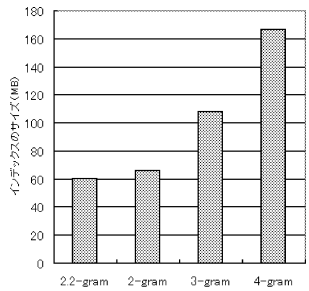


**図 4** 2.2-gram 法と N-gram 法の転置インデックスのサイズ
Fig. 4 The inverted index size of 2.2-gram and N-gram.

検索対象文書群のコーパスとして，Wikipedia 日本語版[11] から無作為に抽出した 10,000 件の記事のデータを用意した．UTF-8 としてのデータサイズの合計は 57.48 M バイトであり，ファイルシステム上では 75.70 M バイトの領域を占める．

### 4.2 N-gram 法との比較

2.2-gram 法と N-gram 法のインデックスの空間効率を比較する実験を行った．本実験の結果を**表 3** および**図 4** に示す．

異なり語数が同じである 2.2-gram 法と 2-gram 法を比較して考える．この結果から，レコードの値のサイズの合計は，双方でほぼ同じになることが分かる．一方，圧縮 B+木のサイズを見ると，2.2-gram 法の転置インデックスの方が小さく，90%ほどになっていることが分かる．その理由としては，N-gram 法のレコードは差分列に byte aligned 符号化をしてすでに圧縮されているのに対して，N.M-gram 法のレコード内のハッシュ値はトークンの出現傾向の偏りを保持しているために，エントロピーが低いことが考えられる．転置インデックスの構築にかかる時間は，2.2-gram 法と 2-gram 法では大差ないことが分かった．

検索においてトークンを評価する回数が同じである 2.2-gram 法と 4-gram 法を比較すると，2.2-gram 法の空間効率および時間効率が非常に優れていることが分かる．4-gram 法に対して 2.2-gram 法は，転置インデックスのサイズが 3 分の 1 ほどになる．また，転置インデックスの構築にかかる時間は 50 分の 1 程度になる．

転置インデックスのサイズはレコードのサイズの合計と論理ページの圧縮率によって決まる．レコードのサイズはキーのサイズと値のサイズに分けて考えられる．レコードのキーの総サイズは異なり語数に比例し，異なり語数は N-gram 法の N の増加にともなって等比級数的に増加する．2-gram 法でのキーのサイズを基準とすると，3-gram 法では 6.5 倍程度，4-gram 法では 15 倍程度になっていることからそのことが確認できる．2.2-gram 法と 2-gram 法の異なり語数は等しいので，検索においてトークンを評価する回数が同じである 4-gram 法に比べて 2.2-gram 法は有利である．なお，転置インデックスを trie 構造で表現して検索キーの領域を節約することも考えられるが，trie の探索はランダムアクセスをともなうので，時間効率が悪化してしまう．

### 4.3 n-gram/2L 法との比較

2.2-gram 法と，n-gram/2L 法のインデックスの空間効率を比較する実験を行った．n-gram/2L 法の一実装として，4-gram のバックエンドインデックスと 2-gram のバックエンドインデックスからなる転置インデックスを比較対象とした．

n-gram/2L 法のインデックスの構築にかかった時間は 1321 秒であり，圧縮 B+木のサイズは 109.58 MB（フロントエンドインデックス 47.71 MB およびバックエンドインデックス 61.86 MB）であった．このことから，一般的な日本語の文書のインデックスを構築する際には，4-gram と 2-gram からなる n-gram/2L 法よりも 2.2-gram 法の方が効率的であるといえる．

## 5. 考　　察

本章では，前章における実験および性能評価の過程で得られた各種の事象について考察し，N.M-gram 法の転置インデックスを実装する際に留意すべきことについて述べる．

### 5.1 エントロピーと圧縮率

各トークンに付与されるデータの性質および圧縮アルゴリズムの性質によって，レコードを圧縮してデータベースに記録する際の圧縮率は変化する．圧縮率の理論的な限界はエントロピーに基づいて求めることができる．たとえば，Wikipedia 日本語版の全文書（214,358 件）をコーパスとして 2-gram 法の転置インデックスを作成すると，各文書のトークン数の平均は 1173.48 となり，文書内オフセットのエントロピーは $\log_2 1173.48 = 10.19$ ビットとなる．なお，可変長データ型を配列の要素にする場合，各要素のサイズを

**表 4** N-gram 法および N.M-gram 法の各トークンのエントロピー
Table 4 The entropy of each token of N-gram and N.M-gram.

| 方式 | エントロピー | レコード総計 | 圧縮 B+木 | 圧縮率 |
|---|---|---|---|---|
| 2-gram | 11.46 ビット | 996.90 MB | 723.56 MB | 0.725 |
| 2.1-gram | 4.06 ビット | 799.13 MB | 475.89 MB | 0.595 |
| 2.2-gram | 6.94 ビット | 1008.45 MB | 655.45 MB | 0.649 |
| 2.3-gram | 8.59 ビット | 1237.40 MB | 853.84 MB | 0.690 |
| 3-gram | 11.83 ビット | 1336.31 MB | 1141.09 MB | 0.853 |
| 3.1-gram | 3.60 ビット | 1132.43 MB | 883.42 MB | 0.780 |
| 3.2-gram | 6.22 ビット | 1347.44 MB | 1090.23 MB | 0.809 |
| 3.3-gram | 7.74 ビット | 1575.97 MB | 1316.71 MB | 0.835 |

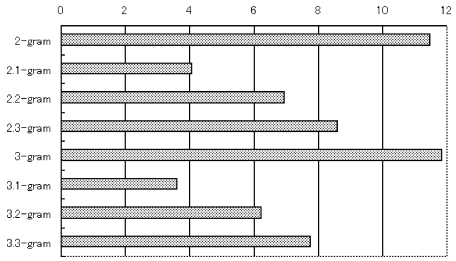


**図 5** N-gram 法および N.M-gram 法の各トークンのエントロピー
Fig. 5 The entropy of each token of N-gram and N.M-gram.

符号中に表現する必要があるので，最終的なエントロピーは 10.19 よりも大きくなる．一方で N.M-gram 法のハッシュ値は固定長データ型を要素とした配列で表現されるので，各要素のサイズを記録するためのオーバヘッドはかからない．

N.M-gram 法におけるエントロピーは，コーパスの各文書のトークン数だけでなく，語彙の偏りにも依存する．また，N-gram 法における文書オフセットも，差分列への変換を行った際のエントロピーはコーパスの語彙の偏りに依存することになる．そこで，同じく Wikipedia 日本語版の全文書に対して各トークンに付与される文書内オフセットもしくはハッシュ値のエントロピーを集計してみたところ，**表 4** および**図 5** に示す結果が得られた．N-gram 法の文書内オフセットのエントロピーは，byte aligned 符号化した文書内オフセットのバイト単位のエントロピーの平均に符号の長さの平均を掛けたものとした．N.M-gram 法のハッシュ値のエントロピーは，後続トークンの M 個のハッシュ値を M バイト単位の数値として扱って算出した．圧縮率は，レコード全体を deflate 圧縮した際の圧縮率である．

2.2-gram 法のレコードに含まれるハッシュ値のエントロピーは 2-gram 法のレコードに含まれる文書内オフセットのエントロピーの 60%ほどであり，2.2-gram 法のレコードの方が圧縮しやすいデータだといえる．2.2-gram 法ではエントロピーが 6.94 のデータを 16 ビットの領域で表現しているので，圧縮率の限界は $6.94/16 = 0.43$ となり，同様に，2.3-gram 法の圧縮率の限界は $8.59/24 = 0.35$ となる．ただし，M 個のハッシュ値を単位として符号を割り当てるという知識を用いることによってはじめてこのような低いエントロピーになるのであり，そのような知識を持たない deflate のような汎用の圧縮アルゴリズムで圧縮率の限界に近付くのは困難である．圧縮率を向上させるには，トークンごとに辞書を切り替える等，N.M-gram 法の特性を利用した符号化を行うことが求められる．

### 5.2 時間効率

ハードディスク等の大容量の補助記憶装置は以前より安価に入手できるようになっているため，全文検索システムのスケーラビリティを決めるのは空間効率ではなく時間効率だといえる．時間効率は，検索処理と更新処理に分けて考えることができるが，そのどちらがスケーラビリティを決定するかは全文検索システムの運用の仕方によって異なる．対象文書が頻繁に更新あるいは追加される場合，スケーラビリティは更新処理の時間によって制限される．検索要求が頻繁にある場合，スケーラビリティは検索処理の時間によって制限される．

Wikipedia 日本語版の全文書をコーパスとして作成した転置インデックスを用いて検索処理時間を測定した．Wikipedia 日本語版の全文検索機能を提供するデモサイト[12] に一般のユーザが入力した 38,321 件の検索語で 4 文字以上のものから無作為に抽出した語のリストと，38,321 件のすべてから無作為に抽出した語のリストを検索条件として用いた．検索語の完全一致条件で検索する処理を 10,000 回行って時間を測定し，処理時間の相加平均を算出した結果を**表 5** および**図 6** に示す．

トークンが短いほどインデックスのレコードを特定するための時間は小さくなると考えられるが，4 文字以上の語の検索時間の平均を見ると，2-gram 法や 2.M-gram 法よりも 3-gram 法や 3.M-gram 法の方が検索の時間効率が良いことが分かる．これは，逆に各レコードのサイズが大きくなるので，トークンが短いほどレコードの読み込みに時間がかかるからである．また，N-gram 法よりも N.M-gram 法の方が検索の時間効率が良いことも分かる．これは，N-gram 法では N 個のトークンのレコードを取り出さなければならないのに対し，N.M-gram 法では N+M 個ごとのトークンを取り出せばよいので，N.M-gram 法の方がレコー

**表 5** 検索性能の比較
Table 5 The comparison of the search performance.

| 方式 | 4 文字以上平均 | 全検索語平均 |
|---|---|---|
| 2-gram | 82.3 ミリ秒 | 59.3 ミリ秒 |
| 2.1-gram | 31.3 ミリ秒 | 24.8 ミリ秒 |
| 2.2-gram | 32.3 ミリ秒 | 27.7 ミリ秒 |
| 2.3-gram | 36.8 ミリ秒 | 30.3 ミリ秒 |
| 3-gram | 26.7 ミリ秒 | 22.5 ミリ秒 |
| 3.1-gram | 5.7 ミリ秒 | 15.5 ミリ秒 |
| 3.2-gram | 6.0 ミリ秒 | 17.8 ミリ秒 |
| 3.3-gram | 6.5 ミリ秒 | 20.8 ミリ秒 |

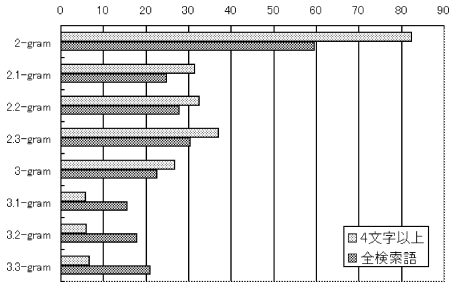


**図 6** 検索性能の比較
Fig. 6 The comparison of the search performance.

ドを読み込む回数が少ないからである．

N.M-gram 法で N が同一の場合，M が小さい方が速度が速いことが分かる．これは，M を大きくすることで減るレコードの読み込み回数の影響よりも，M を大きくすることでレコードのサイズが増大する分のオーバヘッドの方が大きいことを示している．転置インデックスをファイルシステムに置く場合，レコードサイズの平均とデータベース内におけるページごとのレコード数がファイルシステムのブロックサイズに等しくなるように N と M を設定すると検索処理の時間効率を最大にできるが，具体的な最適値は文書群の内容や規模によって変わる．

文字数を制限しない全検索語の検索時間の平均を見ると，2-gram 法や 2.M-gram 法の検索性能と 3-gram 法や 3.M 法の検索性能の差は 4 文字以上に制限した場合に比べて小さくなっていることが分かる．N-gram 法や N.M-gram 法の性能は検索語の長さに強く依存するので，実際の検索性能をとらえるには実際に入力される検索語の傾向を考慮する必要がある．そこで，一般のユーザが入力した検索語の各々の文字数を集計したところ，**表 6** および**図 7** に示す頻度分布が得られた．平均は 4.41 文字であった．検索語の長さがトークンの長さを下回る確率は，2-gram 法や 2.M-gram 法の場合は 3.10%ほど，3-gram 法や 3.M-gram 法の場合は 20.84%ほど，4-gram 法や 4.M-gram 法の場合は 38.94%ほどになる．

**表 6** 検索語の文字数の頻度分布
Table 6 The frequency of search word length.

| 文字数 | 頻度 | 割合 | 累積割合 |
|---|---|---|---|
| 1 文字 | 1189 | 0.031 | 0.031 |
| 2 文字 | 6800 | 0.177 | 0.208 |
| 3 文字 | 6936 | 0.180 | 0.389 |
| 4 文字 | 8634 | 0.225 | 0.614 |
| 5 文字 | 5197 | 0.135 | 0.750 |
| 6 文字 | 3474 | 0.090 | 0.841 |
| 7 文字 | 2428 | 0.063 | 0.904 |
| 8 文字 | 1443 | 0.037 | 0.942 |
| 9 文字 | 913 | 0.023 | 0.965 |
| 10 文字 | 591 | 0.015 | 0.981 |

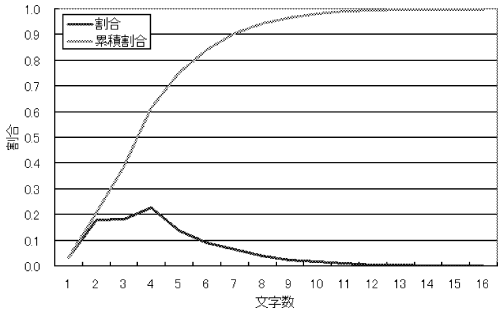


**図 7** 検索語の文字数ごとの割合
Fig. 7 The length ratio of search word.

実際の運用においては，更新性能と検索性能のバランスをとって N と M のパラメータを決定することになる．転置インデックスをファイルシステムに置く場合，更新および検索にかかる時間のほとんどはディスクアクセスに費され，ディスクアクセスにかかる時間のほとんどはシーク時間が占める．したがって，時間効率を改善するためにはシークの回数を減らすことが求められる．N-gram 法の転置インデックスの更新処理においては，シークの回数はレコード数すなわち異なり語数に比例するので，N を小さくした方がシーク回数が少なくなる．N-gram 法の検索においては，シークの回数は検索語の長さを N で割った数となる．すなわち検索処理におけるシーク回数は N に反比例するので，N を大きくした方がシーク回数が少なくなる．一方で，N.M-gram 法においては，更新処理のシーク回数については N に比例するが，検索処理のシーク回数は N+M に反比例する．したがって，N を小さくして更新処理を高速化しても，M を大きくすることで検索性能を保つことができる．

### 5.3 適 合 率

上述の実験で用いた Wikipedia 日本語版のコーパスおよび検索条件のリストを用いて，2.2-gram 法の転置インデックスにおける検索の適合率を測定した．文字数ごとの適合率を**表 7** に示す．ここでいう適合率

**表 7** 検索語の文字数ごとの適合率
Table 7 The precision of each search word length.

| 文字数 | 適合率 |
|---|---|
| 3 文字 | 0.972 |
| 4 文字 | 0.996 |
| 5 文字 | 0.965 |
| 6 文字 | 0.978 |
| 7 文字 | 0.966 |
| 8 文字 | 0.961 |
| 9 文字 | 0.956 |
| 10 文字 | 0.985 |

（検索精度）とは，検索結果の文書に検索語の文字列が含まれている割合のことであり，検索語の文字列が対象文書内で自然文としての単語ないし形態素をなしているかどうかは考慮しないものとする．レコードの検査はトークン 3 つごとに行うものとする．

この結果により，2.2-gram 法の転置インデックスを用いた検索の適合率はおおむね 95%以上になると考えられる．

## 6. おわりに

本稿では，N-gram 法の空間効率を改善する手法である N.M-gram 法について述べた．N-gram 法とともに用いられる位置情報の代わりにハッシュ値を用いることにより，N-gram の利点である言語中立性や再現率の完全性を保ったまま，転置インデックスのサイズを縮小することができることを示した．また，N-gram 法の転置インデックスと N.M-gram 法の転置インデックスの空間効率を比較して，N.M-gram 法が優れていることを示した．

## 参考文献


1) 原田昌紀，風間一洋，佐藤進也：Unicode を用いた N-gram 索引の一実現方式とその評価，情報処理学会研究会報告，2000-NL-136-17, pp.135–142 (2000).
2) 小川泰嗣，松田 透，橋本信次：N-gram 索引における複合検索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.SIG-5(TOD2), pp.43–52 (1999).
3) Shannon, C.E.: Predication and Entropy of Printed English, *The Bell System Technical Journal* (1951).
4) Elias, P.: Gamma Code, Delta Code: Universal codewords sets and representations of the integers, *IEEE Trans. Inf. Theory*, IT-21(2), pp.194–203 (1975).
5) Golumb, S.W.: Golumb Code: Run-length Encodings, *IEEE Trans. Inf. Theory*, IT-12(3), pp.399–401 (July 1966).
6) Faloutsos, C. and Christodoulakis, S.: Description and Performance Analysis of Signature File Methods for Office Filing, *ACM Trans. on Office Information Systems*, pp.237–257 (July 1987).
7) Zobel, J., Moffat, A. and Ramamohanarao, K.: Inverted files versus signature files for text indexing, *ACM Trans. Database Syst.*, pp.453–490 (Dec. 1998).
8) Kim, M-S., Whang, K-Y., Lee, J-G. and Lee, M-J.: n-Gram/2L: A Space and Time Efficient Two-Level n-Gram Inverted Index Structure, *Proc. 31st VLDB Conference*, pp.325–335 (2005).
9) Hyper Estraier. http://hyperestraier.sourceforge.net/
10) Quick Database Manager. http://qdbm.sourceforge.net/
11) Wikipedia 日本語版．http://ja.wikipedia.org/
12) Wikipedia 検索. http://athlon64.fsij.org/mikio/wikipedia/


(平成 18 年 9 月 18 日受付)
(平成 18 年 12 月 28 日採録)

(担当編集委員 橋本 泰一)

**平林 幹雄**

2001 年立命館大学政策科学部政策科学科卒業．同年（株）富士ゼロックス入社．2000 年にデータベースマネージャ「QDBM」，2004 年に全文検索システム「Hyper Estraier」を開発し，オープンソースソフトウェアとして公開している．2006 年より（株）ミクシィに在籍し，検索機能を中心として SNS サイトのシステム開発に従事．

**江渡浩一郎**

1997 年慶應義塾大学大学院政策・メディア研究科修了．2002 年より独立行政法人産業技術総合研究所研究員．ネットワーク上のコミュニケーションをテーマとした研究・作品制作活動を行っている．2003 年より，メーリングリストと Wiki を統合したコミュニケーション環境「qwikWeb」の開発・運用を継続．WikiSym2006 運営委員．2005 年，仮想生物構築環境「Modulobe」を発表．