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

2006－ＦＩ－82 (2)
2006－ＤＤ－54 (2)
2006／3／22

N.M-gram: ハッシュ値付きN-gram法による空間効率の改善

平林幹雄† 江渡浩一郎††

全文検索システムの転置インデックスを生成するための主要な手法であるN-gram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，検索対象の文書群から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．本稿では，N-gram法の欠点である転置インデックス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オフセット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: Improving Space Efficiency by Using N-gram with Hash Values

MIKIO HIRABAYASHI† and KOUICHIROU ETO ††

N-gram is a major method to construct inverted index for full-text search system. Although N-gram has advantages of language neutrality and perfect recall ratio, it has shortage that the inverted index becomes large because tokens extracted from documents are enormous. In this paper, we describe *N.M-gram* method, which improves space efficiency of N-gram. N.M-gram uses hash values of succeeding tokens instead of offset data in each document. N.M-gram can improve space efficiency without losing advantages of language neutrality and perfect recall ratio.

1. はじめに

近年，電子文書の増大に伴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転置インデックスを用いるのが一般的である．テキストからトークンを抽出する主要な手法のひとつとして，**N-gram法**が挙げられる[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[3)]．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．

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．

† 富士ゼロックス株式会社
Fuji Xerox Co., Ltd.
†† 独立行政法人 産業技術総合研究所
National Institute of Advanced Industrial Science and Technology (AIST)

2. N.M-gram法のアルゴリズム

本節では，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，その上で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ひとつである．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[4)]．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

図 1　N-gram 法の転置インデックスの構造
Fig. 1　Structure of Inverted Index of N-gram

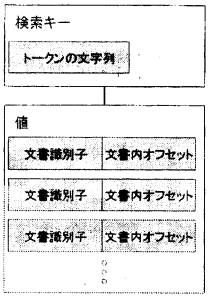
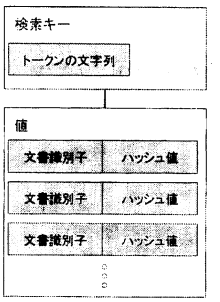


図 2　N.M-gram 法の転置インデックスの構造
Fig. 2　Structure of Inverted Index of N.M-gram



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）法と言い，N 文字ずつ分割する方法を総称して N-gram 法と言う．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．例えば「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+木などのインデックスを伴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．例えば「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みなすことになる．

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[5][6]．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全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 文字を越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 文字のトークンを 1 文字ずつずらしながら切り出すところは N-gram 法と同様だが，各トークンに対して文書の識別子と後続の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ところが異なる．文書内オフセットの代用として，より短いビット長で表現できる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 法などと呼び，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．例えば「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．例えば「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みなせばよい．

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**）．例えば 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じ組み合わせが同一文書内で複数回現れた場合はそれをひとつに代表させることができるため，組み合わせが重複したトークン群の情報量が節約できる．例えば，「アメリカ」という文字列が 2 回現われた場合，「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．例えば 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，トークンの文字数を小さ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格納した際の情報量が少なくて済む．

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，計算量は転置インデックスの規模に対して $O(1)$ で済む．

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[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全てが対象文書において立っていれば，その文書を該当とみなす．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ビット長あたりの実質的な情報量が少なく，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言える．対象文書の全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．

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 $O(1)$ 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ーヘッドは小さいと考えられる．

3. N.M-gram 法の実装

本節では，全文検索システム **Hyper Estraier**[8] 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．

トークンは全て 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上で，その値の 1 バイトをレコードを置く．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前にあるものと全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．

表 1 トークン毎に整理されたレコード群
Table 1 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

以上の処理を「ファイルとファイルの保存」という文書に上述の処理を行ったとすると，**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わかるので，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 法の解析をした上で，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全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．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 個毎でよく，N+M-gram 法では N+M 個毎でよい．例えば，「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

図 3　検索時の処理

Fig. 3　Processing of Search

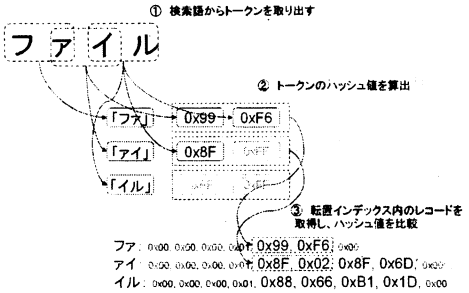


表 2　テストプログラムの実行環境

Table 2　Running Environment of the Test Program

プロセッサ	1.70GHz（Pentium M）
主記憶装置	1G バイト（PC-2700）
ハードディスク	30GB（ATA-100，4200rpm/9.5mm）
OS	Linux 2.4.31（ext2 ファイルシステム）
ライブラリ	GLIBC 2.3.3，QDBM 1.8.45，ZLIB 1.2.3

まれることは検査しなくてもわかる．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わかるので，その時点で検索処理を完了できる．

4. 評　　価

本節では，N.M-gram 法の典型例として 2.2-gram 法を取り上げ，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，単一の文書での空間効率について述べ，さらに，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述べる．

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** の通りである．転置インデックスはデータベースライブラリ QDBM[9] 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 個毎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，同じトークンの出現数を数えた上で，その値の 1 バイトをレコードを置く．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全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．

表 3　単一文書おける 2.2-gram のトークンの出現傾向

Table 3　Appearance Tendency of 2.2-gram Tokens in a Document

文書のサイズ	6028.01 バイト
トークンの数	2387.83
異なり語数	1136.10
レコードのキーの総サイズ	6165.79 バイト
レコードの値の総サイズ	9934.90 バイト

N-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，トークンの文書内での出現位置のリストを差分列に変換した上で，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する．さらに，各レコードに番兵として 0x00 を末尾に加える．

検索対象文書群のコーパスとして，WikiPedia 日本語版[10] から無作為に抽出した 10000 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 57.48M バイトであり，ファイルシステム上では 75.70M バイトの領域を占める．

4.2 単一の文書における評価

単一の文書における 2.2-gram 法のトークンの出現傾向を調べたところ，**表 3** の結果が得られた．なお，10000 件の各々の値の相加平均を代表値とした．

この結果により，一般的な日本語の文書では，2-gram 法によるトークンの数の 40%程度が異なり語数になることが予想される．その場合，レコードのキーの総サイズは対象文書のサイズとほぼ同等になり，値のサイズは対象文書のサイズの 165%程度になる．なお，同一文書内で同じパターンのトークンが複数回出現する確率は，テキストが長くなるほど大きくなる．したがって，N.M-gram 法の転置インデックスの空間効率は，対象文書群を扱のテキストが長くなるほど向上する．また，N-gram 法と N.M-gram 法の双方において，多数の文書を対象として転置インデックスを作成した場合は，レコードのキーのサイズが全体に占める割合は小さくなり，レコードの値のサイズが支配的となる．

4.3 文書群における評価

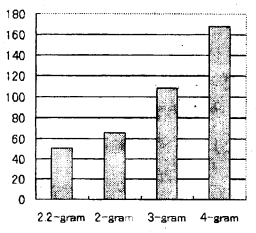
多数の文書群を対象として，2.2-gram 法と N-gram 法のインデックスの空間効率を比較する実験を行った．本実験の結果を**表 4** および**図 4** に示す．

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

表 4　2.2-gram 法と N-gram の空間効率の比較
Table 4　Space Efficiency of 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	3,604,461	8,393,111
レコードサイズ合計	97.8 MB	91.9 MB	141.2 MB	213.6 MB
キーサイズ小計	3.0 MB	3.0 MB	28.8 MB	87.6 MB
値サイズ小計	94.7 MB	88.8 MB	112.3 MB	125.9 MB
圧縮 B+木のサイズ	50.8 MB	65.8 MB	108.4 MB	167.4 MB
圧縮 B+木の圧縮率	0.519	0.716	0.767	0.783
構築時間	164 秒	173 秒	1,638 秒	8,062 秒

図 4　転置インデックスのサイズのグラフ
Fig. 4　Graph of Size of Inverted Indexes



は，双方でほぼ同じになることがわかる．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，77%ほどになっていることがわかる．その理由としては，N-gram 法のレコードは差分列に byte aligned 符号化をして既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わかった．

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わかる．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 以下になる．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事が確認できる．2.2-gram 法と 2-gram 法の異なり語数は等しいので，2.2-gram 法が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伴うので，時間効率が悪化してしまう．

5. 考　察

本節では，前節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 法を実装する際に留意すべきことについて述べる．

5.1 N と M の変化

N.M-gram 法の N と M を変化させると性能特性も変化する．前節で行った実験と同じ条件で，N の値を 1 から 4 まで変え，M の値を 0 から 3 まで変えて組み合わせた結果を**表 5** および**図 5** に示す．なお，M が 0 の場合はトークンにハッシュ値を付与しないので，位置情報を保存しない N-gram 法の転置インデックスを構築するのと同じことになる．

この結果から，N の値にかかわらず，M を 2 以下にした場合は N.M-gram 法の転置インデックスのサイズが N-gram 法よりも小さくなることがわかる．また，N.M-gram 法においては，N を変えずに M を増やすと転置インデックスのサイズがほぼ線形に増大していくことがわかる．

5.2 実質空間効率

ここで，対象文書のサイズの合計と転置インデックスのサイズとの比率を**索引比率**と呼ぶことにする．N.M-gram 法の転置インデックスに対する 1 回の問い合わせによって対象文書の絞り込みを行う能力は N+M 文字分であるが，絞り込み能力を空間比率で割った値を**実質空間効率**と呼ぶことにする．実質空間効率を尺度とすれば，N.M-gram 法の空間効率を定量的に比較することができる．前節および本節で取り上げた実験結果を実質空間効率の降順で並べると，**表 6** のようになる．

N.M-gram 法の中では 1.M-gram 法のバリエーションの実質空間効率が高い．ただし，N が小さい場合は各レコードのサイズが大きくなるので，検索処理には時間がかかることに注意すべきである．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毎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．

表 5　N.M-gram 法のパラメータを変えた比較

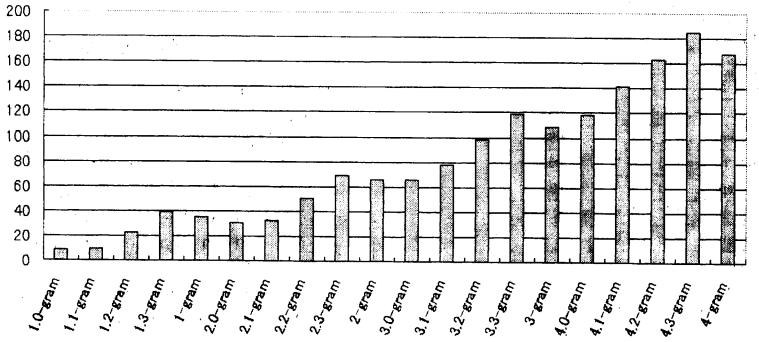
Table 5　Comparison of Variants of N.M-gram

方式	1.0-gram	1.1-gram	1.2-gram	1.3-gram	1-gram	2.0-gram	2.1-gram	2.2-gram	2.3-gram	2-gram
異なり語数	8,661	8,661	8,661	8,661	8,661	560,149	560,149	560,149	560,149	560,149
レコードサイズ合計	20.7 MB	24.2 MB	41.8 MB	65.3 MB	46.0 MB	68.1 MB	78.9 MB	97.8 MB	119.8 MB	91.9 MB
キーサイズ小計	0.02 MB	0.02 MB	0.02 MB	0.02 MB	0.02 MB	3.0 MB	3.06 MB	3.06 MB	3.06 MB	3.06 MB
値サイズ小計	20.6 MB	24.1 MB	41.7 MB	65.3 MB	45.9 MB	65.0 MB	75.8 MB	94.7 MB	116.8 MB	88.9 MB
圧縮 B+木のサイズ	8.9 MB	9.1 MB	22.4 MB	39.1 MB	35.2 MB	30.3 MB	32.5 MB	50.8 MB	69.2 MB	65.9 MB
圧縮 B+木の圧縮率	0.430	0.376	0.536	0.597	0.765	0.444	0.412	0.519	0.577	0.716
構築時間	37 秒	38 秒	52 秒	63 秒	64 秒	136 秒	144 秒	164 秒	182 秒	173 秒

方式	3.0-gram	3.1-gram	3.2-gram	3.3-gram	3-gram	4.0-gram	4.1-gram	4.2-gram	4.3-gram	4-gram
異なり語数	3,604,461	3,604,461	3,604,461	3,604,461	3,604,461	8,393,111	8,393,111	8,393,111	8,393,111	8,393,111
レコードサイズ合計	118.6 MB	133.6 MB	153.1 MB	174.9 MB	141.3 MB	191.5 MB	208.8 MB	228.9 MB	250.6 MB	213.6 MB
キーサイズ小計	28.9 MB	28.9 MB	28.9 MB	28.9 MB	28.9 MB	87.7 MB	87.7 MB	87.7 MB	87.7 MB	87.7 MB
値サイズ小計	89.7 MB	104.7 MB	124.3 MB	146.0 MB	112.4 MB	103.8 MB	121.1 MB	141.2 MB	162.9 MB	126.0 MB
圧縮 B+木のサイズ	65.8 MB	77.6 MB	98.2 MB	119.1 MB	108.4 MB	118.4 MB	141.2 MB	162.6 MB	184.8 MB	167.4 MB
圧縮 B+木の圧縮率	0.555	0.581	0.640	0.680	0.767	0.618	0.676	0.710	0.7373	0.783
構築時間	1,638 秒	1,680 秒	1,688 秒	1,779 秒	1,639 秒	7,839 秒	7,852 秒	8,009 秒	8,063 秒	8,062 秒

図 5　N.M-gram 法による転置インデックスのサイズのグラフ

Fig. 5　Graph of Size of Inverted Indexes of N.M-gram



5.3 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言える．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．

本稿で取り上げた実験で生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 日本語版の全文書に対して MeCab[11] を用いて形態素解析を行って取り出した 4 文字以上の長さの単語の中から無作為に選択した語の完全一致条件で検索する処理を 10000 回行って時間を測定し，処理時間の相加平均を算出したものである．その結果を昇順で並べると表 7 のようになる．インデックスの構築にかかった時間も併記する．

N または N+M の値が小さいと検索時間が長くなることがわかる．これは実質時間効率の項で述べた性質により説明できる．また，N.M-gram 法において N が同じ場合は，M が小さい方が検索時間が短くなることがある．これは，M が小さい方が転置インデックスのサイズが小さくなるので，ファイル入出力のデータ量が少なくて済むからである．なお，本実験では 4 文字以上の検索語のみを用いているが，4 文字未満の検索語も含めた場合は 4-gram 法の性能は悪化する．3-gram 法や 2-gram 法においても同様の制約がある．

表 6　実質空間効率による順位付け
Table 6　Ranking of Virtual Space Efficiency

方式	実質空間効率	絞込能力	索引比率
1.1-gram	12.658	2 文字	0.158
1.2-gram	7.692	3 文字	0.390
1.0-gram	6.329	1 文字	0.155
1.3-gram	5.891	4 文字	0.679
2.1-gram	5.309	3 文字	0.565
2.2-gram	4.524	4 文字	0.884
2.3-gram	4.156	5 文字	1.203
2.0-gram	3.802	2 文字	0.526
3.1-gram	2.962	4 文字	1.350
3.2-gram	2.929	5 文字	1.707
3.3-gram	2.897	6 文字	2.071
3.0-gram	2.620	3 文字	1.145
4.3-gram	2.177	7 文字	3.214
4.2-gram	2.121	6 文字	2.828
4.1-gram	2.035	5 文字	2.456
4.0-gram	1.942	4 文字	2.059

表 7　検索時間による順位付け
Table 7　Ranking of Search Performance

方式	検索時間	構築時間
4.1-gram	0.64 ミリ秒	7,852 秒
4.2-gram	0.65 ミリ秒	8,009 秒
4.0-gram	0.67 ミリ秒	7,839 秒
4.3-gram	0.67 ミリ秒	8,063 秒
3.1-gram	0.72 ミリ秒	1,680 秒
4-gram	0.73 ミリ秒	8,062 秒
3.0-gram	0.75 ミリ秒	1,638 秒
3.2-gram	0.75 ミリ秒	1,688 秒
3.3-gram	0.76 ミリ秒	1,779 秒
3-gram	0.99 ミリ秒	1,639 秒
2.1-gram	2.14 ミリ秒	144 秒
2.2-gram	2.59 ミリ秒	164 秒
2.3-gram	2.73 ミリ秒	182 秒
2.0-gram	2.76 ミリ秒	136 秒
2-gram	5.48 ミリ秒	173 秒
1.1-gram	5.87 ミリ秒	38 秒
1.0-gram	9.75 ミリ秒	37 秒
1.2-gram	11.09 ミリ秒	52 秒
1.3-gram	12.36 ミリ秒	63 秒
1-gram	33.14 ミリ秒	64 秒

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．

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 Vol. 40 No SIG-5(TOD2), pp.43-52, 1999.
3) Min-Soo Kim, Kyu-Young Whang, Jae-Gil Lee, Min-Jae Lee: n-Gram/2L: A Space and Time Efficient Two-Level n-Gram Inverted Index Structure”, Proceedings of the 31st VLDB Conference, pp.325-335, 2005.
4) C.E. Shannon: “Predication and Entropy of Printed English”, The Bell System Technical Journal, 1951.
5) P. Elias: “Gamma Code, Delta Code: Universal codewords sets and representations of the integers”, IEEE Trans. on Information Theory, IT-21(2), pp.194-203, 1975.
6) S.W. Golumb: “Golumb Code: Run-length Encodings”, IEEE Trans. on Information Theory, IT-12(3), pp.399-401, July 1966.
7) C. Faloutsos, S. Christodoulakis: “Description and Performance Analysis of Signature File Methods for Office Filing , ACM Transactions on Office Information Systems, pp.237-257, July 1987.
8) “Hyper Estraier”, http://hyperestraier.sourceforge.net/
9) “Quick Database Manager”, http://qdbm.sourceforge.net/
10) “WikiPedia 日本語版”, http://ja.wikipedia.org/
11) “MeCab: Yet Another Part-of-Speech and Morphological Analyzer”, http://mecab.sourceforge.jp/