データベースシステム 120－6
（2000．1．24）

コンパクトなグラムベース索引
—可変長グラムの固定長コーディング—

佐藤隆士　杉原和郎　八田直 平岩浩司

大阪教育大学大学院総合基礎科学専攻数理情報コース
大阪府柏原市旭ヶ丘 4-698-1

Abstract

高速全文検索システムに使用するグラムに基づくの索引を構成するグラムのコード化方法に関する研究である．検索時の高速性を優先するため，コード化グラムは固定長とし，ビット操作など高負荷な処理を避けている．文書を形態素解析し，必要と思われるグラムのみを索引化することにより，索引の大きさを抑えている．グラムを構成する各文字に割り当てるビット数は文字の出現頻度をもとに可変とするため，グラムを構成する文字数も可変となる．コンパクト且つ高選択度の索引となることをモデルと解析で示している．日本語の文字を対象とした実験では，記憶オーバヘッド 47.3%で平均時間 50msec，しかも実用的に誤検索なしの検索を達成できた．

Coding Grams of Variable Length into Fexied Bytes
—For Fast Full Text Retrieval with Compact Indices—

Takashi Sato, Kazuo Sugihara, Nao Hatta, Kouzi Hiraiwa

Course of Mathematical and Information Science, Division of Pure and Applied Science,
Graduate School of Education, Osaka Kyoiku University,
4-698-1 Asahigaoka, Kashiwara 582-8582, Japan

Abstract

This paper describes how to code grams of variable number of characters into fixed byte length for fast full text retrieval with compact indices. In this method, the number of bits assigned to a character is variable according the frequency of appearance in general texts. Using models and analisys, we show that indices made by the proposed method are compact and highly selective. In experiments using Japanese newspapers as object text, we obtained an idex of 47.3% space overhead with retrieval of arbitrary strings in 50msec on average and practically no false drops.

1　はじめに

インターネットページをはじめ文書，図書，新聞などもオンライン化が進み，コンピュータで直接検索可能なオンラインテキストが増大し続けている．これら膨大なテキストのもつ潜在的な情報を活用するには，目的とする情報に効率的にアクセスする手段が必要である．

単一の文字列検索のみでは，しばしば検索結果が大量となるため，希望する結果を得るためには絞り込む必要がある．また，指定された検索語に関連する語の検索を含めることもある．検索システムとして用いるには，検索結果を見ながら対話的に質問を調整する機能も持つ必要があるだろう．このように，より利用者の目的に沿った検索結果を得るためには，多数回の文字列検索の結果

を有機的に組み合わせる必要があると考える．このため，文字列検索エンジンは高速なものでなければならない．

大量文書の検索を高速にするためには索引の作成が不可欠である．任意文字列を検索可能とするには，シストリングに基づくもの[1, 2]もあるが，索引作成コストが低い，検索時に複雑なビット処理など高負荷な処理を必要としないなどの理由で，グラムに基付く索引が注目されている[3, 4, 5, 6, 7]．

検索の高速性だけでなく，索引のコンパクトさも重要な要素である．日本語を含む東洋系言語では，記号集合のサイズが大きいため，索引が大きくなる問題が指摘されている[3, 7]．逆に索引サイズを抑えると誤検索 (false drop,noise) が増す．このため，低容量オーバヘッドかつ低誤検索を可能とする索引構造の研究が必要である．

以上の要求を満たすため，著者らは次のようなの特徴を有するグラムに基付く索引構造の研究を行ってきた[8, 9]．

[長いグラムの使用] 従来法では，グラム長以下の文字列検索ができないため，長さ $L(\geq 3)$ のグラムを索引に用いる場合，1および2グラムの索引も用意していた．しかし，著者の方法では，木構造の索引を用い順探索可能としたため，別に短いグラムを用意する必要はない．

[位置情報不要] ふつう，グラムの現れる文書番号と文書内でのオフセット (位置情報という) の組を索引で管理する．検索キーが分解されてできたグラム集合と同じものを文書が持っていてもそれらが文書中で隣接していなければ誤検索である．グラムの隣接性を調べるため位置情報が必要である．しかし，著者の方法では，位置情報は記憶しない．これは，(1) 長いグラムの使用で，検索キーがグラム長以下になり，誤検索は生じない場合が多い．(2) 検索キーが長くてグラム集合に分解された場合でも，グラムが長いため十分検索候補が絞られており実用上，位置情報がなくても誤検索が生じない．(3) 必要なら原テキストで誤検索か否かを確認できることによる．

[グラムのコード化] しばしば，グラムはそれを構成する文字のコードをそのまま連接したものが用いられるが，著者の提案する方法では，検索構造をコンパクトで且つフィルターの機能も十分果たすようにするため単なる文字の連接ではなくコード化し用いている．検索時の処理を効率化するため，索引探索時には処理の重いビット処理は避けている．与えられた検索キーから検索用のグラムを一度作成すると，バイトあるいはワード単位の処理で効率よく探索できるようにしている．

[グラムの選択] 検索を行う場合，文書中の単語などの意味のある区切りが検索キーとして使用される．そこで，文書を形態素解析し文書中に意味のある区切りを見つけ，必要と思われるグラムのみを選択し索引に加えることによって索引のサイズを抑えている．

このような特徴を有するグラムに基づく索引の研究の内，本稿ではグラムのコード化に焦点をあてる．検索時の高速性を優先するため，コード化グラムは固定長とし，ビット操作など高負荷な処理を避ける[8]ことは従来の方針と変わらないが，グラムを構成する各文字に割り当てるビット数は文字の出現頻度をもとに可変とする．このためグラムを構成する文字数は可変となる．本稿で提案の手法で，コンパクトで且つ高選択度の索引を作成できることをモデルと解析により示した．日本語の文字を対象とした実験では，記憶オーバヘッド 47.3%で平均検索時間 50msec，しかも実用的に誤検索なしの検索を実現することができた．

2 諸定義

記号集合 (アルファベット) を Σ とする．Σ の大きさは $\sigma(=|\Sigma|)$ で表す．Σ は任意のものでよいが，実験では，日本語 EUC 2byte コードに割りあてられた，$\sigma = 8836$ 文字を対象とした．被検索テキストは，m 個に分けられた文書であり，i 番目の文書の文字数を $n_i(1 \leq i \leq m)$ とする．テキスト全体のサイズ $n(=\sum_{i=1}^{m} n_i)$ は，主記憶

に置くには大きすぎるため，二次記憶に格納されているものとする．

各文書から検索語として必要と思われる部分を選択し，その部分中の各文字を先頭とするグラムと呼ばれる連続した文字列の集合を切り出し，コード化，並べ替えなどの処理を経て索引を作成する．文書と同様，索引も二次記憶に格納されている．グラムを構成する文字数は一定でないが，それをコード化したもの (グラム値と呼ぶ) は，高速な検索を実現するため固定長 w_g[byte] とする．索引も文書と同様二次記憶に格納されているため，索引中のポインタはファイルの先頭からのオフセットであるが，そのサイズを w_p[byte] とする．検索結果は文書番号で得られるが，文書番号は w_d[byte] で表現可能とする．

検索キー k はグラムと同様な手法で，コード化されるものとする．コード化されたサイズを l_k[bit] とする．

テキストと索引を二次記憶に置くため，計算の際，主記憶，二次記憶間のデータ転送が生じる．転送はブロック単位に行われ，サイズは B[byte] とする．

3 木構造グラムベース索引

提案の索引は，根部 (root)，葉部 (leaf) および位置付部 (locator) からなる (図 1)．葉部の各

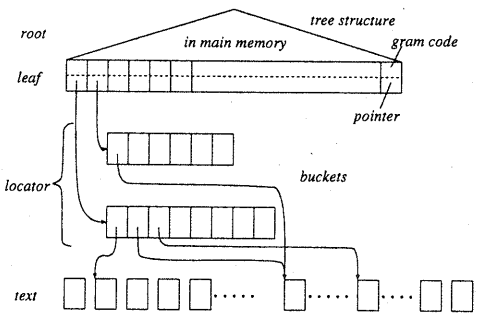

図 1: グラムに基づく索引

要素をスロットと呼ぶ．スロットにはグラム値と位置付部のバケットへのポインタが入っている．更にバケットには，被検索テキストの文書番号が入っている．あるスロットが指すバケット内には，そのスロットに対応する文字列が現れる文書番号を記憶している．

B 木[11]の葉にはキーのみならずポインタも格納する．同様に提案の索引でも B 木のキーをグラム値に対応すると，位置付部に格納している文書番号を葉部に埋め込むことは可能である．敢えて分離したのは，グラム値の取出しの際，文書番号の不必要な読込みを防ぐためである．索引を圧縮した場合は，不必要な文書番号のデコードも防止されることになる．

根部は，調べたいグラム値をもつ葉部のスロットに 1 回のブロックアクセスで到達できるよう葉部にあるグラム値を一定間隔で集めたものである．根部は比較的小さく，被検索テキストが数十ギガバイト程度であれば，問題なく主記憶の作業領域に格納できる．それ以上大きくなる場合は，2 階層に分け，葉部と合わせて 3 階層構造にすればよい．

3.1 記憶コスト

葉部のスロットはひとつ当り，$w_g + w_p$[byte] である．文書中に同じ文字列が存在すれば当然同じグラム値が得られるが，葉部には同一グラム値は複数回書かない．この重複の除去をユニーク化と呼びその割合を $u_s (0 < u_s \leq 1)$ とする．葉部のスロット中にあるグラム値は昇順に並んでいる．位置付部のバケットは対応するグラム値の順に連接しているので，それへのポインタも昇順に並んでいる．順に並んだ数列はランレング法[12]で圧縮きるので，この圧縮率を α_s とする．従って，葉部は，$\alpha_s u_s n(w_g + w_p)$[byte] となる．

位置付部については，同一文書中にある同一グラムは重複して記憶しないことにしているので，u_d でユニーク化されるとする．文書全体のグラム値の重複より多くないので，$0 < u_s \leq u_d \leq 1$ である．また，同一バケット内に含まれる文書番号は昇順に並べているので，ランレング法で圧縮

率 α_d の圧縮ができるとする．以上から，位置付部は，$\alpha_d u_d n w_d$[byte] となる．

根部のグラム値も昇順に並ぶためランレング法で圧縮できるが，葉部に比べ十分小さいので記憶コストからは無視できるサイズである．

以上のことから，記憶コストを

$$n\{\alpha_s u_s(w_g + w_p) + \alpha_d u_d w_d\} \text{ [byte]} \quad (1)$$

と見積もることができる．

3.2 作成コスト

作成コストを索引作成のために必要なブロックアクセス数で見積もる．索引作成は，まず主記憶に置くことができるだけのグラム値と文書番号の組を一回分のバッチとして作成し，その後これらのバッチをマージして全体の索引を作成する[10]．バッチもソート，ユニーク化しランレング法を適用している．

まず，バッチ作成のコストを見積もる．1 文字 2byte として，検索対象テキストの読込みに $2n/B$ 回のブロックアクセスを必要とする．バッチは主記憶上で作られ，バッチごとに二次記憶に書かれる．バッチは，検索に使用しないので根部はないが，葉部，位置付部は索引と同じ構造なので，ブロックアクセス数は，

$$n\{\alpha'_s u'_s(w_g + w_p) + \alpha_d u_d w_d\}/B \quad (2)$$

となる．但し，α'_s, u'_s は，それぞれバッチ分のスロットのユニーク化率とランレングス法による圧縮率であり，$0 < \alpha_s \leq \alpha'_s \leq 1$ および，$0 < u_s \leq u'_s \leq 1$ である．

バッチのマージコストは，バッチを読込むためのコストと索引を書出すためのコストの和なので，式 (1) を B で割りブロックアクセス数としたものに (2) を加えればよい．

作成コスト全体は，

$$n\{2 + (\alpha_s u_s + 2\alpha'_s u'_s)(w_g + w_p) + 3\alpha_d u_d w_d\}/B \quad (3)$$

となる．

3.3 検索コスト

木構造索引を用いた検索アルゴリズムおよび検索コストは，文献[8]に詳しい．本稿のグラムのコード化で定義した記号を用いて場合分けし，式を書き換えたものを書いておく．

[A] l_k が w_g にちょうどコード化されるとき

$$1 + \lceil M w_d/B \rceil \quad (4)$$

ここで，M はマッチ数である．

[B] l_k が w_g より短くコード化されるとき

$$\lceil \sum_i (w_g + w_P)/B \rceil + \lceil \sum_i M_i w_d/B \rceil \quad (5)$$

但し，$\sum_i$ は検索キーが接頭辞となるグラム全てについて和をとる．M_i は i 番目にマッチするグラムのマッチ数である．

[C] l_k が w_g より長くコード化されるとき

$$\sum_j 1 + \sum_j \lceil M_j w_d/B \rceil \quad (6)$$

但し，$\sum_j$ は検索キーが分解された全てのグラムについて和をとる．M_j は分解された j 番目グラムのマッチ数である．

4 グラムのコード化

出現頻度の高い文字列は検索時にフィルターとしてあまり有効でないと考えられる．従って，グラムコード化時には，短いビット長のコードに対応させ，グラムにより多くの文字がコード化されるようにする．逆に出現頻度の小さい文字はフィルターとして有効なので，長いビット長のコードを与える．高速な検索を行うためグラムは固定長にコーディングするため，結果的にグラムを構成する文字数は可変となる．提案の方法では，従来の手法に比べ，グラム値の長さ w_g が同一の場合，誤検索の確率を下げることができる．逆に一定の誤検索率を許すなら，より小さい w_g を用いることができる．即ち，よりコンパクトな索引で誤検索のない検索が可能になると期待できる．

記憶，作成，検索コストは3に示したものと変わりない．但し，コーディングにより，$u_s, u_d, \alpha_s, \alpha_d$ が異なることと，検索キーによって w_g にはいる文字数が異なる点に注意する必要がある．

5 モデルと選択度の解析

4のグラムコード化による索引を用いた検索の誤検索率を見積もるため，モデルを作り選択度の解析を行う．

5.1 文字および文字列のモデル

テキストおよび検索キーを構成する文字間に相関はないと仮定する．まず，アルファベットを構成する文字のコードテーブルを作成する．コーディングは各文字のテキスト中での出現頻度を参考にハフマンコーディングする．ハフマンコーディングは，ほぼ最適に行えたとする．即ち，文字 c_i の出現頻度を $p(c_i)(\leq 1)$ とするとき，長さ $-\log_2 p(c_i)$[bit] のコードにコード化できたとする．

テキスト全体でのコード化された文字の長さの平均値を $l_c = \overline{-\log_2 p(c_i)}$ で表す．このとき，一文字あたりの選択度の平均値は $1/2^{l_c}$ となる．仮定からこの値は，$1/\sigma$ にほぼ等しくなる．即ち，$l_c \simeq \log_2 \sigma$ である．

5.2 解析

解析を容易にするため，コード化された文字の長さに平均値 l_c を用いる．

5.2.1 グラムの選択度

グラムは $8w_g$[bit] まで詰めることが可能だが，コード化された長さ l_c の文字を詰めると，端数が平均 $l_c/2$[bit] できるので，有効なビット数は平均 $8w_g - l_c/2$[bit] である．これを l_g とおく．文字間に相関はないとしたので，グラムの選択度は平均

$$1/2^{l_g} \tag{7}$$

となる．

5.2.2 検索キーの選択度

[A]$l_k \leq l_g$ のとき

コード化された検索キーはグラムに納まるので，選択度は l_g によらず直接 l_k から計算される．

$$1/2^{l_k} \tag{8}$$

誤検索は生じない．

[B]$l_k > l_g$ のとき

検索キーは平均 $n_k = l_k/l_c$ 文字，グラムは平均 $n_g = l_g/l_c$ 文字をコード化したものになる．従って，検索キーは平均 $n_k - n_g + 1$ 個のグラム集合に分解され検索される．

例として，$n_k = 5, n_g = 3$ の場合について説明する[1]．検索キーを $k = c_1c_2c_3c_4c_5$ とするとき，グラム集合は，$\{c_1c_2c_3, c_2c_3c_4, c_3c_4c_5\}$ となる．簡単のため，c_1, c_2, c_3, c_4, c_5 は互いに異なる文字とする．

索引からグラムに対する文書番号が得られる．位置情報を記憶していないので，(0)$c_1c_2c_3c_4c_5$ だけでなく，(1-1)$c_1c_2c_3$ + $c_2c_3c_4c_5$, (1-2)$c_1c_2c_3c_4$ + $c_3c_4c_5$ や (2)$c_1c_2c_3$ + $c_2c_3c_4$ + $c_3c_4c_5$ をもつ文書も検索される[2]．言うまでもなく (1-1),(1-2),(2) は誤検索である．(1-1)(1-2) は検索キーが隣接する $n_g - 1$ 文字を重複しながら1箇所で切断された文字列，(2) は同様に2箇所で切断された文字列である．選択度はそれぞれ，(0)$1/2^{5l_c}$, (1-1)(1-2)$1/2^{7l_c}$, (2)$1/2^{9l_c}$ となる．

一般的には，検索キーが文書中にある選択度は (0)$1/2^{n_k l_c}$，1箇所切断されたものがある選択度は (1)$1/2^{n_k l_c} \times 1/2^{(n_g-1)l_c}$ で ${}_{n_k-n_g}C_1$ 個の場合がある．$i(\leq n_k - n_g)$ 箇所切断の選択度は $(i)1/2^{n_k l_c} \times 1/2^{i(n_g-1)l_c}$ で，${}_{n_k-n_g}C_i$ 個の場合がある．従って，これらの和をとったものが，全

[1] 説明を簡単にするため，意識的に n_g を小さくしている．

[2] '+' はそれで結ばれた文字列が同じ文書に存在することを示す．

体の選択度になる．

$$1/2^{n_k l_c} \times \sum_{i=0}^{n_k-n_g} {}_{n_k-n_g}C_i / 2^{i(n_g-1)l_c} =$$

$$1/2^{n_k l_c} \times \{1 + 1/2^{(n_g-1)l_c}\}^{n_k-n_g} \quad (9)$$

和の計算に2項定理を用いている．上記(0)以外が誤検索なのでその確率は，

$$1/2^{n_k l_c} \times \{(1 + 1/2^{(n_g-1)l_c})^{n_k-n_g} - 1\} \quad (10)$$

つなる．従って，$1 \leq i \leq m$ に対して，上式 $\times n_i$ が1に比べ十分小さければ，このモデル上では誤検索は無視できることになる．グラムを長くとると $(n_g-1)l_c = 8w_g - (3/2)l_c$ が大きくなり，誤検索は実用上無視できることが分かる．

5.3 モデルの限界

解析を容易にするため，5.1ではテキストおよび検索キーを構成する文字間に相関はないと仮定したが，実際は，文字間に相関がある．このため，解析結果が定量的に現実のキー検索の際にそのまま反映される分けでわない．しかし，定量的には合わなくとも，検索キーやグラム長が検索キーの選択度や誤検索に与える定性的なふるまいが分かる．文字に与えるコードは基本的には出現頻度の高い文字には短いものを与えるが，スペースや句読点など文章の区切りとなる文字はそれを挟む前後の文字列との相関が少ないので，長めのコードを割り当てると有利になることが分かっている．なお，文字間の相関に関する考察およびコード化方法に付いては[13]を参照されたい．

6 実験結果

使用コンピュータはCPU:Compaq Alpha 21264 500MHz, 主記憶:512Mbyte, OS:Tru64, 二次記憶は平均シーク時間5msec，回転待ち時間2mecのハードディスクである．被検索テキストとして「CD －毎日新聞94版」をもとに，キーワード，読みなどをはずし，ID，見出し，日付，前文，本文のみの137Mbyte分を8つ繋げあわせて1.1Gbyteにしたものを使用した．

図2にグラムのバイト数 w_g と索引の記憶容量の関係を示す．w_g = 5byteで47.3%のオーバヘッドである．

図3，図4は索引作成時間の測定値である．

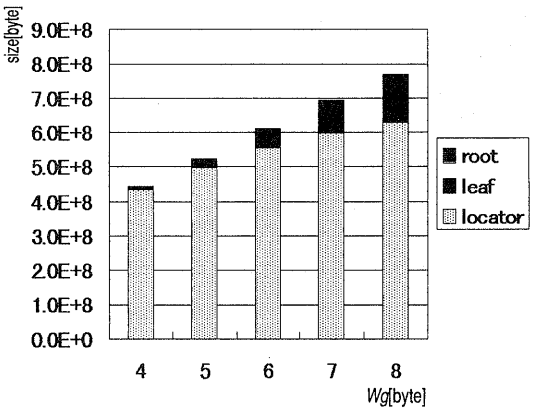

図 2: 索引容量

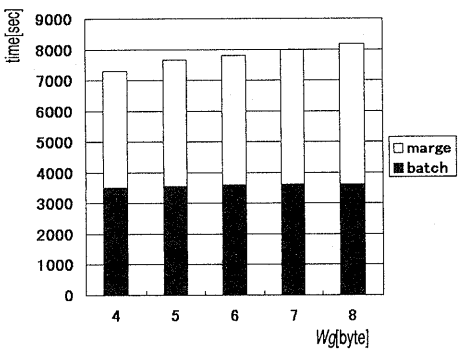

図 3: 索引作成時間 (主記憶に 10^6 個グラム)

図3はバッチ作成時の主記憶におくグラム数を 10^6 に固定し，w_g との関係を示したものであり，逆に図4は $w_g = 5$ 固定して，グラム数との関係を見たものである．w_g が大きくなると作成時間も緩やかに増加すること，主記憶におくグラム数にはあまり依存しないことが分かる．

図5は被検索テキストから生成される全グラムに入る文字数の分布を集計したものである．棒中

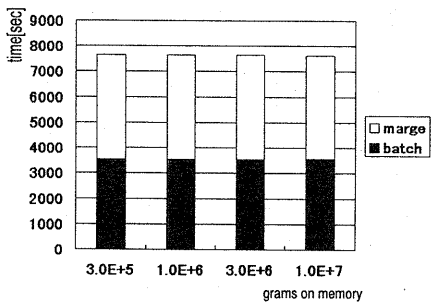

図 4: 索引作成時間 ($w_g = 5$)

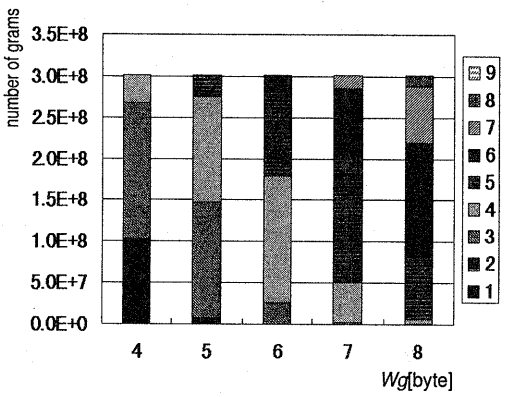

図 5: テキストグラムの構成文字数の分布

の数字が文字数である．$w_g = 5$ 以上でほぼ 3 文字以上になることが分かる．これは図 7の誤検索がほとんどないことと対応している．

検索実験は適当に選んだ 105 個の検索キーについて行った．図 6は w_g とキーが何グラムに分割されて検索されるかの分布を示している．棒中の数字がグラム数である．グラム長が長くなると分割数が少なくなる様子がよく分かる．

図 7は検索された総数と，そのうち検索キーがひとつのグラムに入り，誤検索が生じないことが保証されている件数を表している． 全部で 331,408 件検索されたが $w_g = 5$byte で 24，6byte で 16 個の誤検索，7 と 8byte では語検索は生じなかった．全検索数から見ると，5byte 以上では実用上，誤検索なしと考えてよい結果である．一般的に短いキーの方が検索数が多い傾向にあるので，図 6で全体のうち分割なし (グラム数が 1)

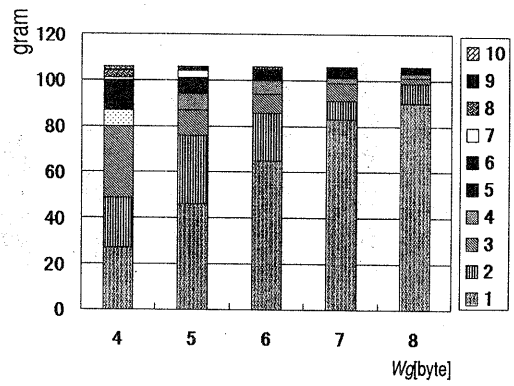

図 6: キー中のグラム数の分布

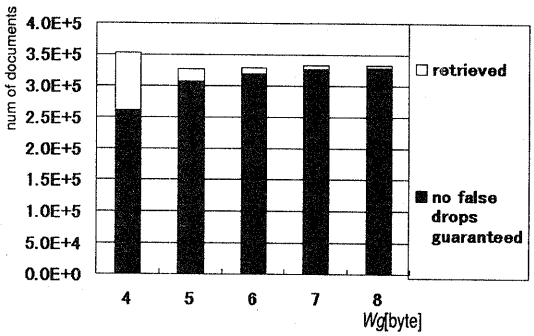

図 7: 検索された文書数

が占める割合に比べ，図 7の誤検索なしが占める割合の方が多くなっている．

図 8は w_g と平均の検索時間の関係を示す．索引のうち根部は主記憶に置き，葉部および位置付部は部分的にも主記憶にない状態から連続してキー検索した測定値である．引き続き同じキー集合で検索すると，索引がキャッシュされているので，検索時間はほぼ 1 桁小さくなることが観測されている．w_g が長くなるほど少しづつ速くなるのは，キーが分けられるグラム数が減るためと考えられる．

従来の固定グラム長方式では，誤検索をおさえるため 4 グラム固定を使用していたが，このためにはコード化グラムを 8byte にする必要があった．提案の方法では 5byte のコード化で誤検索をほぼ押さえることができた．高速性を保ったまま

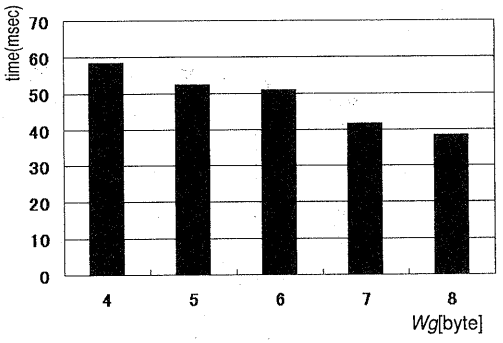

図 8: 平均検索時間

可変長グラムコード化によりコンパクトな索引で実用的に誤検索なしにできることが確認された.

7 まとめ

本稿では, 全文検索を目的としたグラムに基づく索引におけるグラムのコード化について報告した. グラムに入る文字数を可変としながら, 固定長にコード化することにより, コンパクト且つ高選択度の索引となることをモデルと解析で示した. 日本語の文字を対象とした実験では, 記憶オーバヘッド 47.3%で平均時間 50msec, しかも実用的に誤検索なしの検索を達成できた.

今後の研究課題としては, 文字間の相関を考慮したグラムのコード化と解析などが上げられる

参考文献

[1] Gonnet, G., Baeza-Yates, R. and Snider, T., New Indices for Text: Pat Trees, in *Information Retrieval: Data Structure & Algorithms* chapter 5, Frakes, W. and Baeza-Yates, R. Ed., pp. 66–82 (1992).

[2] Ferragina, P. and Grossi, R., Fast string searching in secondary storage: Theoretical developments and experimental results, *Proc. ACM-SIAM Symposia on Discrete Algorithms*, Vol. 7, pp. 373–382 (1996).

[3] Ogawa, Y. and Iwasaki, M., A new character-based indexing method using frequency data for Japanese documents, *In Proc. 18th ACM SIGIR Conf.*, pp. 121–129 (1995).

[4] 菅谷他: *n*-gram 型大規模全文検索方式の開発, 情処 53 全大:5T–2,3 (1996).

[5] 赤峯, 福島: 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, *ADBS '96*(1996).

[6] 松井, 難波, 井形:大容量情報全文検索システム, 信学総大:D–4–6 (1997).

[7] 菊地忠一: 日本語文書用高速全文検索の一手法, 電学論 (D–I), Vol. J75–D–I, No. 9, pp. 836–846 (1992).

[8] Sato, T., Fast full test retrieval using gram based tree structure, *Proc. ICCPOL '97*, Vol. 2, pp. 572–577 (1997).

[9] 佐藤, 杉山, 三木田, 遠藤, 須山, 野田: グラムベース全文検索システムの高速化と応用, 情処研報, DBS-114-2 (1998).

[10] 野田, 遠藤, 佐藤: グラムベース全文検索システムに対する索引作成及び更新, *DEWS'98* (1998).

[11] Knuth, D., *The Art of Computer Programming: Vol.3 Sorting and Searching, 2nd Ed.*, AddisonWesley, Reading, Mass., pp. 481-489(1998).

[12] Zobel, J., Moffat, A. and Sacks-Davis, R., An efficient indexing technique for full-text database, *Proc. 18th Int. Conf. on VLDB*, 1992, pp. 352–362.

[13] Noda, J. and Sato, T., Efficient Variable *n*-gram Coding for Full-text Retrieval - Automatic detection of punctual characters using their frequency and correlation - *Proc. ICCPOL '99*, Vol. 2, pp. 381-384 (1999).