データベースシステム 107－6
（1996．3．11）

圧縮ファイルへの直接検索を可能にする符号化法の考案

松本 光崇† 角田 達彦† 松本 裕治‡
matumoto@pine.kyoto-u.ac.jp
† 京都大学 工学研究科
〒606-01 京都市左京区吉田本町
‡ 奈良先端科学技術大学院大学 情報科学研究科

本論文では，$2^n(k)$ 符号語を提案し，この符号語を用いた新しい文書ファイル圧縮方法を提案する．この手法の目的は，ファイルを圧縮したままの形で直接データを検索出来，しかもそれを効率的に行なうことを可能にすることである．効率的な直接検索を実現する上で解決すべき問題として，符号語区切りの処理の高速化がある．本論文では，符号語に制限を加えることによってこの処理が高速化され，単純なパターン照合で直接検索が可能になることを示す．この符号語を用いて日本語文書，英語文書を圧縮して実験を行なった結果，ファイル容量が約 60% に削減され，約 1.8 倍の照合高速化が実現された．

A coding scheme which enables a fast direct search in the compressed files

Mitsutaka Matsumoto† Tatsuhiko Tsunoda† Yuji Matsumoto‡
† Kyoto University
Yoshida, Sakyo, Kyoto, 606-01 Japan
‡ Nara Institute of Science and Technology

We propose the $2^n(k)$ coding scheme using our original $2^n(k)$ codeword sets. The purpose of this scheme is to enable a fast direct search in the compressed files. One of the problems which prevent an efficient direct search is the process of partitioning a codeword string. We show that some restrictions to the codewords accelerate the process and enable the search by a simple pattern-matching. As a result of experiments, both Japanese and English text files were compressed to about 60% and searching was acceleraed about 1.8 times.

1 概要

近年，大容量記憶媒体の普及，情報通信網の整備，出版工程の電子化に伴い大量の文書データが電子化され利用されている．しかし大量のデータがオンライン化される一方で，その処理能力が追い付かないという問題が生じつつある．それらのデータは何らかの形で圧縮されて保存されているが，オンライン化された情報が溢れその整理が課題となっている今日では，高い圧縮率や圧縮に要する時間を短縮すること以上に新たな要望が生まれていると言える．すなわちもしデータを圧縮したままの形で直接データを検索出来，しかもそれが効率的に行なえるのならば非常に都合が良い．本研究では文書ファイルの圧縮ファイルに対して，データをそのまま復号せずに文字列照合を行なうことを可能にする符号化法を提案する．

圧縮ファイルに対して直接検索を効率的に行なうことを考えた場合の問題点は次の三つにまとめられる．2章で詳しく説明する．

1. 文字列符号化の曖昧性の問題

2. ビット単位データ扱いの問題

3. 符号語の区切り処理の問題

本論文では，特に三番目の問題の解決法を中心に議論する．照合にあたって，符号語の区切りをファイルの最初から与えていく処理が必要であり，この処理が効率的な検索の妨げになるという問題である．特に可変長符号語を用いた符号化法の場合，圧縮ファイルへの検索が単純なパターン照合では実現出来ないことがほとんどで，区切り処理も一般的に重い．3章では本研究で提案する $2^n(k)$ 符号語について述べる．この符号語は可変長符号語でありながら単純なパターン照合で検索が可能であり，符号語区切りの処理も高速化される．4章では $2^n(k)$ 符号語を用いて文書ファイルを符号化する手順と，圧縮ファイルへの直接検索の手順を示す．5章で本手法を採用した場合の圧縮率および検索速度を実験し評価する．上述の一つ目と二つ目の問題について幾つかの方法を比較検討することにする．そして実際に日本語文書ファイルと英語文書ファイルに適用してその実用性についても検討する．

符号化表

単位文字列	符号語
aa	t_1
abc	t_2
cde	t_3
de	t_4
fab	t_5
:	:

文書ファイル：
$| \underline{abc} | \underline{de} | fab | aa | \underline{fab} | \underline{cde} | de \cdots$
圧縮ファイル：　$t_2\ t_4\ t_5\ t_1\ t_5\ t_3 \cdots$
検索パターン：　"$abcde$"
照合パターン：　"$t_2\ t_4$", "$t_5\ t_3$"

図 1: 文字列符号化の曖昧性の例

2 圧縮ファイルの直接検索

2.1 定義

符号化とは単位文字列から符号語への写像である．単位文字列とは符号化すべき系列（文書ファイル）が分解される基本単位のことをいう．

圧縮比は，圧縮ファイルの容量と元ファイルの容量との比で定義される．

2.2 直接検索の問題

文字列符号化の曖昧性

図 1 に符号化変換表の例を示す．文書ファイルを符号化する場合，符号化表中に登録されている文字列との最長一致を調べながら符号語をあてていく．このようにして符号化された圧縮ファイルに対して文字列照合を行なうことを考える．"$abcde$"という文字列を照合するものとする．符号化表では文字列"abc"，"de" に対応する符号語が割り当てられている．したがって文字列"$abcde$" は "$|abc|de|$" という形で区切られ，t_2t_4 と符号化されるであろう．しかし符号化ファイルの中では必ずしもそのように符号化されているとは限らない．図1の例では "fab"，"cde" という文字列に対応する符号語も与えられており，文字列 "$abcde$" は "$|fab|cde|$" という形で区切られ，t_5t_3 とも符号化されている．すなわち同一文字列の符号化に複数のパターンがある可能性があり，符号化表を探索して全てのパターンを見つけ出す処理が必要になる．ここでは二通りのパターンのみを示したが，符号化表が大きくなり登録文字列が

符号化表

単位文字列	符号語
a	00
b	01
c	10
d	110
e	111

検索パターン： “c (10) ”
圧縮ファイル： “… 11100 …”
区切りの曖昧性： “… 111 | 10 | 0 …”
“… | 111 | 00 | …”
“… 1 | 110 | 0 …”

図 2: 符号語区切り処理の例

多くなるにつれ，また検索パターンが長くなるにつれその数は莫大になり，処理が重くなり有効な検索の妨げになる．また符号化パターンをすべて見つけ出したとしても，複数パターンの照合処理が必要になる．

ビット単位データの扱い

現在の汎用的な計算機は 1 バイト（8 ビット）単位のデータ扱いを中心としている．しかしほとんどの圧縮アルゴリズムはビット単位の符号語に変換するので圧縮ファイルにアクセスするためにはビット単位のデータ扱いをしなければならない．ビット単位のデータを扱う場合，バイト単位でデータを取り込みそこからビット列を切り出すという操作が必要になる．その操作には計算と時間を要する．バイト単位の元ファイルに照合操作を行なうことと比較すれば高速化は期待出来ないであろう．

また仮に専用計算機を用いるにしても，一回のデータ読み込みごとに必要な内部処理を行なうので，ビット単位のデータ扱いでは読み込み回数が多くなり処理速度を遅くする．

符号語の区切り処理

ファイル検索には，符号語の区切りをファイルの最初から与えていく処理が必要になる．等長符号語であればこの処理は軽く，単純なパターン照合で検索が可能である．しかし可変長符号語の場合，検索は単純なパターン照合では成功せず，この処理も重いのが一般的である．

例として図 2 で考える．例で挙げた符号化表の符号語は語頭条件を満たしている．しかし符号化ファイル中で文字 “c” に対応する部分を検索する場合に，単純にその符号語 “10” をパターン照合するだけでは実現出来ない．例えば符号化ファイル中に “…11100…” なる部分があったとしても，ここに現れる “10” は “c” に対応するとは限らない．図 2 に示す通り，”…|111|00|…” で “ea” に対応する一部分であるかもしれないし，文字 “d” の符号語”110” の一部分に過ぎないかもしれない．符号語の区切りをファイルの始めから与えていくことが必要で，その処理は軽いものではない．

2.3 関連研究

圧縮ファイルへの直接検索について，まず理論的な側面から幾つかの研究が行なわれている ([1][2] など)．しかしこれらの研究では実際的な検索時間についての言及はない．上述の問題などについても計算時間のオーダーに影響しないという立場である．

一方実用的な側面からは，深町ら [3] がハフマン法圧縮ファイルへの直接検索の高速化を試みており，ビット単位のデータ扱いの問題についても議論している．また Manber [4] は英語文書に対し文字列符号化の曖昧性の問題を考慮した符号化法を提案している．英語文書ファイルに対してそれぞれ約 60%，70% の圧縮比を実現しており，検索速度は両者共に約 1.4 倍の高速化を実現している．

3 $2^n(k)$ 符号語

3.1 概要

本研究では 2.2で述べた符号語区切り処理の問題を中心に議論し，この処理の高速化を検討する．本章では $2^n(k)$ 符号語を定義し，この符号語を用いた符号化法を提案する．本符号化法では単位文字列から符号語への写像を次の二段階に分けて行なう．

1. 単位文字列から整数への写像
2. 整数から符号語への写像

写像を二段階に分けるのは，$2^n(k)$ 符号語を考案するにあたり「整数のユニヴァーサル符号化」の一手法として考案された符号語区切りビット列の考え方 ([5] など) を導入しているためである．整数と符

表 1: $2^2(k)$ 符号語 (c_0=00,c_1=01,c_2=10,c_3=11)

(単位文字列)	整数 p	$2^2(1)$ 符号語	$2^2(2)$ 符号語	$2^2(3)$ 符号語
a	1	c_0	c_0	c_0
b	2	c_1c_0	c_1	c_1
c	3	c_2c_0	c_2c_0	c_2
d	4	c_3c_0	c_2c_1	c_3c_0
e	5	$c_1c_1c_0$	c_3c_0	c_3c_1
:	6	$c_1c_2c_0$	c_3c_1	c_3c_2
	7	$c_1c_3c_0$	$c_2c_2c_0$	$c_3c_3c_0$
	8	$c_2c_1c_0$	$c_2c_2c_1$	$c_3c_3c_1$
	9	$c_2c_2c_0$	$c_2c_3c_0$	$c_3c_3c_2$
	10	$c_2c_3c_0$	$c_2c_3c_1$	$c_3c_3c_3c_0$
	11	$c_3c_1c_0$	$c_3c_2c_0$	$c_3c_3c_3c_1$
	:	:	:	:

号語とは後述の写像関数で相互に変換可能であり，n と k の値が与えられれば数値計算で変換が可能になる．したがって符号化表には n,k の値と単位文字列・整数間の写像のみを記述すればよいので，単位文字列・符号語間の写像を記述することに比べ記述量が少なくて済む利点がある．

以下では $2^n(k)$ 法を説明し，実際の符号化の方法は 4 章で説明する．

3.2 $2^n(k)$ 符号語

$2^n(k)$ 符号語は次の二つの制約によって定義される．

- 符号語は n ビットの符号語記号 (2^n 種類) から構成される．
- 符号語記号のうち k 個を符号語区切り記号に定める．符号語は必ず符号語区切り記号を語尾とし，かつ符号語区切り記号は語尾以外に現れることはない．

最初の制限は符号語長に対する制限でもある．符号語長は n ビットの倍数しかとらない．圧縮ファイルをアクセスする単位ビット数（n の値）は検索効率を大きく左右する．二番目の制限は符号語区切りビット列の考え方に基づいている．符号語区切り記号の数（k の値）は圧縮率を大きく左右する．k の値を調整することによって，圧縮率を高めることが可能になる．

例で示す．n=2 として，$2^2(1)$，$2^2(2)$，$2^2(3)$ 符号語（それぞれ k=1,2,3 に対応）の整数との対応を表 1 に挙げる．符号語記号は c_0=00, c_1=01, c_2=10, c_3=11 の四種類である．一つ目の制限に従い，符号語は 2 ビット単位の長さしかとらないことが分かる．また二つ目の制限に従って，四つのうち k 個を符号語区切り記号として設定する．例えば k=1 の $2^2(1)$ 符号語では c_0 が区切り記号になる．表から分かる通り語尾には必ず c_0 が用いられている．c_0 が語尾以外につくことはない．同様に k=2 の場合には c_0 と c_1 の二つが符号語区切り記号に，k=3 の場合には c_0, c_1, c_2 の三つが符号語区切り記号になる．

$2^n(k)$ 符号語よって単純なパターン照合によって圧縮ファイルへの直接検索が可能になる．この例で，例えば $2^2(2)$ 符号語による圧縮ファイルで d を検索するときには “$[c_0|c_1]c_2c_1$” を単純に照合すればよい．d の符号語として c_2c_1 が用いられる場合には必ず c_0 または c_1（前の符号語の語尾）の後に出現するからである．$2^2(1)$ 法，$2^2(3)$ 法の場合にはそれぞれ “$c_0c_3c_0$”，“$[c_0 - c_2]c_3c_0$” で照合すればよい．

k の値は圧縮率に影響を及ぼす．高い圧縮率を実現する k の値は，単位文字列の生起確率の分布に依存するのであるが，大まかには単位文字列の個数に依存する．単位文字列の数が非常に多い場合は，一般に k が小さいほど高い圧縮率を実現する．大きな整数に対応する符号語の長さは，k の値が小さいほど短いためである．n =2 の例でも，大きな整数に対しては，k =1 の符号語（$2^2(1)$ 符号語）が最も短い符号語を与えることが分かる．逆に単位文字列の数が少なければ，k の値が小さい程，高い圧縮率を実現する．例えば a, b, c の三つのみであれば，k =3 の符号語が平均符号長 2 ビットで最も良い圧縮率となる．k の値は検索効率にはほとんど影響しないため圧縮率を考慮して設定する．

一方，n の値は圧縮率に影響を及ぼすと同時に検索効率にも大きな影響を及ぼす．5 章の実験において，圧縮率を最適化する n を用いる場合と，検索速度を高める n (=8) を用いる場合を，圧縮率，検索速度の両面で比較する．

3.3 写像関数

3.3.1 整数から符号語への写像関数

整数 p に対応する符号語 $codeword(p)$ を求める写像関数を示す．

$$codeword(p) = c_{w_1}c_{w_2}\cdots c_{w_{len}} \qquad (1)$$

上の定義から明らかに $c_{w_{len}}$ は符号語区切り記号であり，その他の記号は区切り記号以外の符号語記号になる．符号語のこの表記は，例えば len=2, w_1=3, w_2=0 は $codeword(p)$ =c_3 c_0 を意味する．

整数 p が与えられたら，まず符号語の長さ len を求めることから始まる．

$$len = \min\{s \mid p \leq \sum_{i=0}^{s-1} k(2^n - k)^i\} \qquad (2)$$

もし $len = 1$ であれば c_{p-1} が符号語である．

$$codeword(p) = c_{p-1} \qquad (3)$$

$len > 1$ のときには以下のように与えられる．まず次の q の値を求める．

$$q = \left\lfloor \frac{p-1}{k} \right\rfloor - \sum_{i=0}^{len-2} (2^n - k)^i \qquad (4)$$

得られた q の値から，p に対応する符号語が以下のように求められる．

$$\begin{cases} w_1 & = \left\lfloor \frac{q}{(2^n-k)^{len-2}} \right\rfloor + k \\ w_i & = \left\lfloor \frac{q \bmod (2^n-k)^{len-i}}{(2^n-k)^{len-i-1}} \right\rfloor + k \qquad i=2,3,\cdots len-1 \\ w_{len} & = (p-1) \bmod k \end{cases} \qquad (5)$$

3.3.2 符号語から整数への写像関数

符号語 $c_{w_1} c_{w_2} \cdots c_{w_{len}}$ に対応する対応する整数 p を求める．

まず符号アルファベット番号 $w_1, w_2, \cdots, w_{len-1}$ と次の漸化式から a_{len} を求める．

$$a_0 = 0, \quad a_i = a_{i-1} \times (2^n - k) + w_i - k + 1 \qquad (6)$$

これよって得られる a_{len} と記号 w_{len} から整数 p は以下のように求められる．

$$p = a_{len} \times k + w_{len} + 1 \qquad (7)$$

4 符号化と検索の手順

4.1 符号化手順

符号化とは単位文字列から符号語への変換であり，本手法では写像を二段階に分けて行うことを前章で述べた．写像の決定過程を表 2の例に基づき説明する．

表 2: 符号化表の作成過程

単位文字列	頻度	確率	整数	(符号語)
の	111,679	0.040	1	
、	92,266	0.033	2	
。	74,969	0.027	3	
ret	74,969	0.027	4	
に	68,334	0.025	5	
い	66,061	0.024	6	
:	:	:	:	

単位文字列から整数への写像

まず単位文字列を決定する．これは圧縮比を左右すると同時に，2.2で述べた文字列区切りの曖昧性が生じるか否かも決める．表 2は日本語文字を単位文字列とした例である．

単位文字列を決定したら，与えられたファイル中での各単位文字列の頻度を求める．頻度の高いものから $1, 2, 3, \cdots$ と整数を割り当てることで整数への写像が決定される．

整数から符号語への写像

整数から符号語への写像は $2^n(k)$ 符号語の n と k の値を設定することで決定される．圧縮率と検索効率の両面を考慮するべきであるため，n と k の値を設定する方法は幾つか考えられる．

最も良い圧縮率を実現する n_{opt}, k_{opt} を求める場合を考える．基本的には n, k の全組合せに対して平均符号長を計算して求める．実際には n_{opt} には上限があるため無限の計算量になることはない．単位文字列の数を m として，n_{opt} の最大値は $\lceil \log_2 m \rceil$ である．n がこの値以上であれば長さ n の等長符号語が割り当てられることになり，平均符号長は単調増加するからである．各 n につき k の値は $2^n - 1$ 種類存在するので，n,k の組合せは最大 $4m$ である．m=10,000 程度であれば，現在の計算機で数分程度で計算出来る．

4.2 直接検索の手順

検索は次の三つのステップで行なわれる．

1. 検索文字列の符号化

表 3: EDR 日本語コーパス評価版

容量 (KB)	5,493	文数	74,969
文字種類	3,031	形態素種類	63,715
のべ文字数	2,709,007	のべ形態素数	1,746,388
平均長 (bits)	15.8	平均長 (bits)	24.1
entropy(bits)	8.23	entropy(bits)	9.06

2. 符号化された検索文字列を圧縮ファイル中で照合（単純なパターン照合）

3. 適合部分を復号して出力

5 実験と評価

日本語文書ファイル，英語文書ファイルを対象にして，圧縮ファイルへの直接検索の実験を行なった．効率的な直接検索を実現するためには，2章で述べた三つの問題点に対処する必要がある．それぞれ以下に示す各実験で比較検討の対象とした．

文字列符号化の曖昧性： 実験 4

ビット単位データの扱い： （実験 1），実験 2

符号語の区切り処理： 実験 1

実験は全てワークステーション“Sun Sparc Station Classic”で行なった．検索時間は UNIX の“time”コマンドで得た．バイト単位の文字列照合には照合 UNIX ツール“egrep”を用い，ビット単位の文字列照合には KMP アルゴリズムに基づくプログラムを用いた．

“pack”，“compress”，“gzip” はファイル圧縮 UNIX ツールであり，比較対象として圧縮比を示す．“pack”は動的ハフマン法，後の二つはレンペル・ジブ動的辞書法に基づいている．“zgrep”とは“compress”で圧縮したファイルを復号しながら文字列照合するツールであり，検索時間の比較対象とした．

5.1 日本語文書ファイルに対する実験

表 3に示す EDR 日本語コーパス評価版を文書ファイルとして用いた．新聞記事を中心としたコーパスであり，約 5.5 メガバイトの容量を持つ．

表 4: EDR 日本語コーパスの文字単位符号化

符号化法	圧縮比 (%)	検索時間 (秒)
元テキスト	(100.0)	5.2
Huffman	52.5	43.1
$2^5(22)$	57.0	28.5
$2^8(246)$	61.9	2.7
cf.“pack”	67.5	–
cf.“compress”	55.1	16.2
cf.“gzip”	50.1	–

英語アルファベット一文字につき 1 バイト (8 ビット) 割り当てられているのに対して，日本語文字の標準文字コードは一文字につき 2 バイト (16 ビット) ずつ割り当てられている．

実験 1： $2^n(k)$ 符号語の有効性

圧縮ファイルのデータをビット単位として，符号語区切り処理の方法を比較する実験を行なった．文字列符号化の曖昧性については，文書ファイルへの検索要求の最小単位は文字単位であるので，単位文字列を日本語文字と設定することによりこの問題は生じない．

符号語区切り処理方法の比較対象として，以下の三つの符号化方法で圧縮したファイルへの検索を比較した．

- ハフマン法 ： 区切りを与えながらの照合
- フィボナッチ法 ： 1 ビット単位の単純なパターン照合
- $2^n(k)$ 法 ： n ビット単位の単純なパターン照合

フィボナッチ法とは，$2^n(k)$ 法における整数から符号語への写像を，フィボナッチ符号化法 [5] によって行なったものである．符号語区切り記号の考え方の元となった符号化法であり，単純なパターン照合で検索が可能になる．$2^n(k)$ 法では n=5, k=22 とした $2^5(22)$ 法を採用した．圧縮率を最適化した結果である．

実験結果を表 4に示す．この結果から $2^n(k)$ 法について次のことが分かる．

- 単純なパターン照合で検索が可能なためハフマン法より高速

表 5: 日本語文書ファイルの文字単位符号化

	EDR	文書 A	文書 B
元 (KB)	5,492	259	10,229
照合時間 (秒)	5.2	0.4	8.0
符号語	$2^8(246)$	$2^8(249)$	$2^8(246)$
圧縮比 (%)	61.9	61.3	60.8
直接検索 (秒)	2.7	0.9	5.4
cf."zgrep"(秒)	16.2	1.4	26.6

- 符号語が n ビット単位であるためパターン照合の処理が軽くなりフィボナッチ法より高速

実験 2 : $2^8(k)$ 符号語の実用性の検証

上述の実験で $2^n(k)$ 符号語の有効性が示された. しかし 5 ビット単位でデータを扱うためにはバイト単位のデータからの変換処理が必要であり, バイト単位のデータである元ファイルに照合操作を行なうことに比べれば検索速度は遅いものである. 実際, $2^5(22)$ 法の 28.5 秒の照合に対して, 元の文書ファイルを "egrep" で照合すると 5.2 秒で達成できる (表 4). $2^n(k)$ 法で n=8 の $2^8(k)$ 法を採用し, 符号語をバイト単位にした実験を行なった. この結果, ファイル容量は 62% に圧縮された. この値は他の符号化法に比べ劣るのであるが, 検索時間は 2.7 秒に短縮され, 検索の高速化が達成された (表 4).

実験 3 : $2^8(k)$ 法の有効範囲

直接検索の要する時間には, 検索パターンを符号化する処理, および適合部分を復号する処理に要する時間も含まれている. その処理時間の大部分は, 符号化変換表をディスクからメモリに読み込む時間であり, 日本語文字単位の約 3,000 行の表を読み込むには約 0.6 秒要する.

もし元ファイルの容量が小さく, 照合の時間が短ければ, この時間を無視できない. 表 5の実験結果はこの時間の影響を示している. 容量 259 キロバイトの文書ファイルに対しては, $2^8(k)$ 法によって高速化は実現されていない. 一方, 5.5 メガバイトの EDR コーパスや 10.2 メガバイトの文書の場合はこの時間を無視でき, 検索時間は約 60% に短縮された. つまり $2^8(k)$ 法は容量にしてメガバイト単位以上のファイルに対しては高速化を実現する.

表 6: 日本語形態素単位での符号化の結果

	表 (行)	圧縮比 (%)	検索時間 (秒)
形態素単位	63,715	54.9/45.5	9.3
元テキスト	–	(100.0)	5.2
文字単位	3,031	61.9/61.7	2.7

表 7: Wall Street Journal に対する実験結果

	圧縮比 (%)	検索時間 (秒)
元テキスト	(100.0)	16.1
2-gram	57.6	8.3
3-gram	55.8	9.4
cf. "compress"	43.1	47.9
cf. "gzip"	37.9	–

実験 4 : 形態素単位での符号化

符号化変換表に登録する文字を一文字単位から文字列の単位に拡張すると, 圧縮率の向上を望める反面, 文字列区切りの曖昧性という問題が生じる. この問題に対処する方法は幾つか考えられるが, ここでは一例として検索に制限を加えることで対処する. つまり形態素単位で符号化し, 検索も形態素単位で行なわれることを想定する.

EDR コーパスは形態素解析の情報を持っているので, 形態素区切りは設定されたものを利用した. EDR コーパスは約 64,000 種類の形態素からなる (表 3). この符号化の結果 45.5% に圧縮された (表 6). これは"gzip" の 50.1% に比べても良い圧縮比である. しかし 64,000 に及ぶ形態素をすべて登録すると符号化表の容量が大きくなるという問題が生じる. この容量を合わせると圧縮比は 54.9% に落ちる. また表を読み込む時間も大きくなり, 検索の高速化は実現出来なかった. この結果は形態素単位の符号化が有効となるにはファイル容量 (5.5 メガバイト) が小さ過ぎることを示している.

5.2 英語文書ファイルに対する実験

$2^8(k)$ 符号化法を英文の文書ファイルに適用する. 英語文書ファイルは日本語文書ファイルと異なり, 文字にはすべて一バイトのコードが与えられている. したがって文字単位で $2^8(k)$ 法で符号化しても

表 8: Brown corpus に対する実験結果

	圧縮比 (%)	検索時間 (秒)
元テキスト	(100.0)	5.4
2-gram	56.2	3.8
3-gram	55.0	4.0
cf. "compress"	41.9	18.7
cf. "gzip"	37.1	–

圧縮されない．日本語の二バイト文字単位で符号化する代わりに，英文文書ファイルでは n -gram（n 文字連続）単位で符号化した．ところがこの場合，文字列符号化の曖昧性が生じる．例えば文字列 "*streef*" は "$|st|re|et|$" と "$s|tr|ee|t$" の二通りに区切られ符号化されている可能性がある．検索方法は 4 章で示したものに若干の修正を要するので，2-gram の場合を例に説明する．符号化ファイルへの直接検索は二段階の処理に分けて行なう．第一段階では "$|st|re|et|$" と "$|tr|ee|$" を符号化して，二つのパターンを照合する．ただしこのままでは "*streef*" を含まなくても文字列 "*tree*" が出現する部分に適合してしまうため，第二段階として，適合部分を復号してから再度 "*streef*" を照合して結果を出力する．

実験には Wall Street Jounal と Brown corpus を用いた．容量はそれぞれ 16.5 メガバイト，6.1 メガバイトである．実験の結果，日本語文書ファイルと同様 50% から 60% の圧縮比が達成され検索高速化も実現された (表 7, 表 8)．しかし "compress" や "gzip" などのツールが英文文書ファイルに対しては高い圧縮率を実現出来るため，これらと比較すると圧縮比に関しては相対的に劣る．

6 結論と今後の課題

符号語区切り記号の考え方を導入した $2^n(k)$ 符号語を用いることによって，圧縮ファイルへの直接検索を単純なパターン照合で実現でき，符号語区切り処理が高速化された．特に n=8 とした $2^8(k)$ 符号語ではバイト単位のデータ扱いで直接照合が可能になる．$2^8(k)$ 符号語を用いて符号化した結果，日本語文書ファイルと英語文書ファイルで共に約 60% の圧縮比が実現され，メガ単位の容量を持つファイルに対しては検索速度も約 1.8 倍の高速化が実現された．

今後の課題としては大規模な情報検索システムでの利用を検討する予定である．本手法は転置ファイル方式との併用も可能であり，効率的な文字列照合を可能にする本圧縮法を転置ファイル方式と併用することで，より柔軟で多様な検索要求に対応できるシステムが実現されるかもしれない．また検索に有効な単位文字列を設定することで，本手法を文書ファイル以外に適用することも可能になる．オンライン化されるデータが増大，多様化する中で，有効な適用対象を模索していくことも今後の課題である．

参考文献

[1] T.Eilam-Tsoreff and U.Vishkin, "Matching patterns in a string subject to multi-linear transformations," *Proc.of the International Workshop on Sequences, Combinatories, Compression, Security, and Transmission*, pp.45–58, Jun. 1988.

[2] A.Amir, G.Benson and M.Farach, "Let sleeping files lie: pattern matching in Z-compressed files," *Proc.of the 5th Symp.on Discrete Algorithms*, pp.705–714, Jan. 1994.

[3] 深町修一, 篠原武, 竹田正幸, "可変長符号圧縮データのための文字列パターン照合," 1992 年 情報学シンポジウム, pp.95–103, Jan. 1992.

[4] U.Manber, "A text compression scheme that allows fast searching directly in the compressed file," *Proc.of the 5th Symposium on Combinatorial Pattern Matching (LNCS 807)*, pp.112–124, 1994.

[5] A.Apostolico and A.Fraenkel, "Robust Transmission of Unbounded Strings Using Fibonacci Representations," *IEEE Transactions on Information Theory*, vol.33, No.2, pp.238–245, 1987.