Vol. 30 No. 10 情報処理学会論文誌 Oct. 1989

実時間パターン照合によるデータ圧縮の高性能実用算法†

横尾 英俊††

実時間データ圧縮用のふたつの高性能実用算法を提案する．ともに，既に実用化されているデータ圧縮法の中で最も定評のある LZW 法の圧縮力を改善したもので，入力記号列上のパターン照合による入力済み記号列の参照に基礎を置いている．LZW 法では，入力済み記号列を分解して変換表に登録し，それを参照して新たな入力の符号化を行うが，本論文では，変換表が成長するに従って，圧縮力が向上する事実に着目している．提案する算法の顕著な特長は，変換表成長のオーダを上げているにもかかわらず，圧縮速度のオーダをLZW 法と同じ線形時間に抑えた点にある．種々のファイルに対する圧縮実験の結果，提案した算法の圧縮力は，従来のいくつかの標準的な手法のそれを多くの場合で上回ることが確認できる．提案した算法は，ファイル圧縮のほか，ネットワーク通信におけるデータ伝送の高速化にも有効であると思われる．

1. はじめに

データ圧縮の実用上の意義が，ハードウェアの急速な発達と低廉化に伴って低下しても，これまでに開発された種々のデータ圧縮法がソフトウェアの基本的な手法の蓄積に直接的または間接的に重要な役割を果してきたように，データ圧縮法の研究は単にそれ自身にとどまらない意義を有している．たとえば，動的ハフマン符号の Vitter[1] による実現は，ファイル圧縮という観点からは，それ以前の版[2),3)] の性能を本質的に上回るものではないが，そこで提案されたデータ構造やプログラミング技法は基礎的で重要な概念を含んでいて，ACM の CALGO[4] にも収められている．

汎用のファイル圧縮の手法としては，動的ハフマン符号のほか，算術符号に基礎を置く Langdon と Rissanen[5] の DAFC (Double-Adaptive File Compression) 法，自己組織リストを利用した Bentley ら[6] の方法，そして，Ziv-Lempel 符号[7] に基礎を置く Welch[8] の LZW 法などが代表的である．これらは，いずれも，ハフマン符号をはじめとする（情報）理論的な基礎と，それを実用上の算法として実現するための新たな修正という二つの側面を有している．したがって，データ圧縮法の研究は，基礎となる新概念の考案およびその実現法，またさらに，それらを目的に応じて種々組み合せるための研究とに分類することができる．組み合せの最近の例としては，Ziv-Lempel 符号のアイデアと算術符号とを組み合せた朴らの方式[9] がある．

このような分類に従えば，本論文で提案する二つの新算法は，基礎として Welch の LZW 法に類似した概念を採用し，それを単純かつ高速に実現して高い圧縮力を得るようにしたものである．LZW 法は，入力記号列上のパターン照合により，入力中の記号列を既に処理の済んだ部分へのポインタによって符号化して圧縮を行うもので，実用化されているデータ圧縮法の中では，圧縮力，速度の両面で現在最も強力とされる算法のひとつである．特に，算法の著しい単純さが特長となっている．提案する算法は，LZW 法の圧縮力をさらに改善するために開発したもので，速度では LZW 法に劣るものの，圧縮力では，中規模以上の長さのファイルに対しては，動的ハフマン符号，DAFC 法，LZW 法のいずれよりも高性能である．本論文では，算法を詳述したあと，このことをシミュレーションによって示す．

2. Welch の LZW 法

データ圧縮の対象として，有限アルファベット A 上の有限系列のみを考える．集合 A の大きさを α で表し，A 上の長さ 0 以上の記号列の集合を A^* で，自然数（0 も含む）および整数の集合をそれぞれ $\boldsymbol{N}$ と $\boldsymbol{Z}$ で表す．本論文を通して，A の要素を x, y などの小文字で，A 上の記号列を X, Y などの大文字で表す．

LZW も提案する新算法も，変換表 (string table) を使用し，それに既に登録済みの記号列を使った最長一致法によって入力記号列を分解し，分解した記号列を変換表の番地によって符号化するものである．変換表は次の意味での語頭性を常に満たしている．

（R 1）変換表に記号列 Xy ($X \in A^*$, $y \in A$) が存在するなら，記号列 X も変換表に存在する．

† High-Performance Data Compression Algorithms via String Matching in Real Time by HIDETOSHI YOKOO (Department of Computer Science, Gunma University).

†† 群馬大学工学部情報工学科

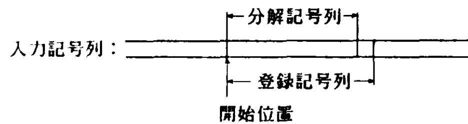


図 1　LZW 法による記号列の分解と登録
Fig. 1 Parsing in LZW method.

LZW 法では，符号語の出力後，**図 1** に示すように，分解記号列の次の 1 文字による拡張を変換表に登録する点に特徴がある．これは，変換表に登録する記号列を変換表の最小の拡張として定義したとき，それを見い出す入力記号列上の開始位置を指定するひとつの規則を与えているとみることができる．処理の済んだ入力記号列長を n としたときの変換表の大きさ，すなわち，変換表に登録済みの記号列の個数を $T(n)$ で表すと，LZW 法では，$T(n)=O(n/\log n)$ が成立する[10]．これは，登録記号列を決定するときの開始位置を指定する規則が与えた結果であり，開始位置を指定する規則を修正すれば，それに伴って変換表の成長の速度も変わることになる．ところで，一般に，LZW 法やその基礎となっている Ziv-Lempel 符号などは，入力記号列長の増加に従って，その圧縮力が向上することが知られている．これは，処理の済んだ入力系列長を n としたとき，次に分解される長さ Δn の記号列に対する理想的な圧縮比 $\log T(n)/\Delta n$ が $T(n)$ の増加とともに減少することを意味している．したがって，変換表の成長を加速することで圧縮力の向上をはかることができる．提案する算法のうちのひとつは，登録記号列を決定するときの開始位置を入力記号列上の各位置にとり，その結果，$T(n)=n+o(n)$ となるようにしたものである．以下の目的のひとつは，変換表の成長速度をこのようにオーダの意味で増しても，それを LZW 法と同じ線形時間で実現できることを具体的な算法によって示す点にある．

3. 新　算　法

提案する算法は，変換表の構成を高速に行うために，条件（R 1）とともに

（R 2）変換表に記号列 yX ($y\in A$, $X\in A^*$) が存在するなら，記号列 X も変換表に存在する．

を満たすように変換表を構成するものである．変換表中の記号列 X には，変換表の中での X の番地である固有の識別子 $c(X)\in \boldsymbol{N}$ が与えられる．記号列 $X=Yz=yZ$ (Y, $Z\in A^*$, z, $y\in A$) に対し，

$$\pi(c(X))=c(Y),$$
$$fin(c(X))=z,$$
$$\sigma(c(X))=c(Z)$$

を満たす写像 $\pi:\boldsymbol{N}\to\boldsymbol{N}$, $fin:\boldsymbol{N}\to A$ そして $\sigma:\boldsymbol{N}\to\boldsymbol{N}$ を導入する．記号列 X が変換表の $c=c(X)$ 番地に登録されているとき，その中で X は，$\pi(c)$, $fin(c)$, $\sigma(c)$ の 3 要素によって表現される．また，本質的に必要なものではないが，復号を容易にするために，X の長さ $|X|$ を $len(c)$ として同時に記録する．ここで，関数 $h:\boldsymbol{N}\times A\to\boldsymbol{Z}$ を

$$h(k,z)=\begin{cases} c & k=\pi(c) \text{ かつ } z=fin(c) \text{ のとき}, \\ -1 & \text{上記以外}, \end{cases}$$

によって定義する．すると，任意の識別子 c について，

$$\sigma(c)=h(\sigma(\pi(c)),fin(c))$$

が成立する．$h(k,z)=-1$ となるのは，変換表に $\pi(c)=k$ かつ $fin(c)=z$ となる番地が生成されていないときであり，次に述べる算法では，これを $h(k,z)<0$ として判定している．h の実現に当たっては，主記憶に十分の容量がある場合や小規模の試作実験を行う場合には，2 次元配列を使って h を直接表現することも可能であるが，一般的には，適当なハッシング技法を使用することになる．これは，Welch が LZW 法にも採用している手法である．

提案する算法は算法 1 と算法 2 の 2 種であり，LZW 法と比較した場合の圧縮力では算法 1 が最も高い性能を有し，速度では LZW 法が最も高速である．算法 2 はいろいろな意味において，算法 1 と LZW 法との中間に位置するものと見なすことができる．

3.1 算　法　1

まず，変換表の 0 番地に長さ 0 の記号列 λ を対応させ，1 番地から α 番地までは A の要素を順に対応させる．すなわち，A の第 c 要素を x_c としたとき，$c=1,2,\cdots,\alpha$ に対し，

$$\pi(c)=0,$$
$$fin(c)=x_c,$$
$$\sigma(c)=0,$$
$$len(c)=1$$

と初期化する．次に，データ圧縮の対象となる入力記号列から，番地 $\alpha+i$ には，第 i 番目の記号から始まり，条件（R 1）を満足する最長の記号列を対応させる．すると，結果的に条件（R 2）も満たされることになる．たとえば，$A=\{a,b,c\}$ 上の入力 $ababcbabaa\cdots$ からは，**表 1** に示す変換表が得られる．ここで，任意の番地 c において，$\sigma(c)<c$ または $\sigma(c)=$

表1 算法1の変換表
Table 1 String table in algorithm 1.

番地 c	部分列	$\pi(c)$	$fin(c)$	$\sigma(c)$	$len(c)$
1	a	0	a	0	1
2	b	0	b	0	1
3	c	0	c	0	1
4	$a\ b$	1	b	2	2
5	$b\ a$	2	a	1	2
6	$a\ b\ c$	4	c	7	3
7	$b\ c$	2	c	3	2
8	$c\ b$	3	b	2	2
9	$b\ a\ b$	5	b	4	3
10	$a\ b\ a$	4	a	5	3
11	$b\ a\ a$	5	a	12	3
12	$a\ a$	1	a	1	2

$c+1$ が成立していることに注意する.

算法1による符号化の過程を具体的に示すと次のようになる.

```
Initialize;
T:=α; P:=0; K:=0; M:=T;
while there are more characters to encode do
    begin
      Let x be the next input character;
      Encode(x);
      TableUpdate(x)
    end;
encode P with fixed-length;
```

手続き *Initialize* は上述の変換表の初期化を行うものである. 整数変数 T は, 変換表の最上位番地を表し, P と K とは, それぞれ, 符号化する記号列を入力から分解するため, および, 変換表に新たに登録する記号列を決定するために使用するポインタである. 変数 M は算法1では全く使用しないが, 後述の算法2で必要になるものである. 手続き *Encode* は符号化すべき記号列を決定し, それを固定長の符号語に変換するものである:

```
procedure Encode(x : char);
begin
if h(P, x)<0 then begin
    encode P with fixed-length;
    P:=h(0, x)
    end
else P:=h(P, x)
end;
```

手続き *Encode* において, **if** 文の判定条件 $h(P, x)<0$ は, $\pi(c)=P$ かつ $fin(c)=x$ となる番地 c が変換表にまだ作られていないことに対応する. この場合, 既に部分列の存在する番地である P の値が適当な固定長符号化によって出力される. これは, 入力記号列から P 番地に対応する部分列を分解し, それを整数値 P として符号化することに等しい. P の符号長は, $\lceil a \rceil$ で a より小さくない最小の整数を表すと, $\lceil \log_2 T \rceil$ ビットあれば十分であるが, 実用的には, $T=2^{12}$ までは 12 ビットで, それ以上 $T=2^{16}$ までは 16 ビットで, などとするのが現実的であろう.

手続き *TableUpdate* は, 各算法を特徴づけるもので, 算法1では, 新記号列の変換表への登録を次のようにして行う.

```
procedure TableUpdate(x : char);
begin
if h(K, x)>0 then K:=h(K, x)
else begin
    T:=T+1;
    π(T):=K;
    fin(T):=x;
    len(T):=len(K)+1;
    while h(σ(K), x)<0 do begin
        σ(T):=T+1;
        K:=σ(K);
        T:=T+1;
        π(T):=K;
        fin(T):=x;
        len(T):=len(K)+1;
        end;
    σ(T):=h(σ(K), x);
    K:=σ(T)
    end
end;
```

以上の手続きを組み合せると, 先にあげた入力記号列例 *ababcbabaa*… は, a, b, ab, c, ba, ba, a… と分解され, 番地の整数列 1, 2, 4, 3, 5, 5, … として符号化される.

ここで, 手続き *TableUpdate* を説明するために, 入力記号列中の長さ $k+2$ の部分 $y_0y_1y_2\cdots y_kx$, y_i, $x\in A$ について, 次のような状況(1)～(4)が同時に成立している場合について考えてみる.

(1) k 個の記号列 $y_iy_{i+1}\cdots y_k$, $i=1,\cdots,k$ が変換表に登録済みである.

(2) $k-j$ 個の記号列 $y_iy_{i+1}\cdots y_kx$, $i=j+1,\cdots,k$

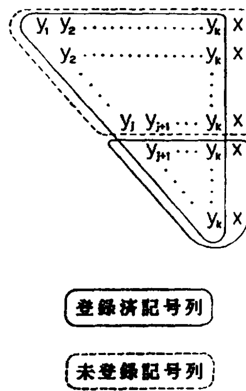


図 2 仮定(1)～(3)の下での登録済み記号列と未登録記号列
Fig. 2 Strings under the assumptions (1), (2) and (3).

が変換表に登録済みである．

（3） j 個の記号列 $y_i y_{i+1} \cdots y_k x$, $i=1, \cdots, j$ が変換表に存在しない．

（4） ある m $(1 \leqq m < k)$ について，記号列 $y_0 y_1 \cdots y_m$ が変換表に存在しない．

特に，(1)～(3)を図式的に表すと，**図 2** に示すようになる．

この場合，*TableUpdate*(y_m) が実行されて，記号列 $y_0 y_1 \cdots y_m$ が変換表に登録された直後の変数 K の値は，記号列 $y_0 y_1 \cdots y_m$ の存在する番地になる．その後，*TableUpdate*(y_i), $i=m+1, \cdots, k$ が呼び出されると，記号列 $y_1 y_2 \cdots y_i$, $m+1 \leqq i \leqq k$ が既に変換表に存在するので，K の値は手続き中の3行目で必ず与えられ，**else** 節は実行されない．そして，*TableUpdate*(y_k) を実行した後の K の値は，記号列 $y_1 y_2 \cdots y_k$ が存在する番地になる．

つぎに，*TableUpdate*(x) を実行すると，3行目の **if** 文で，記号列 $y_1 y_2 \cdots y_k x$ が変換表に登録済みか否かが調べられ，結果として，**else** 節が実行されることになる．そこでは，まず，記号列 $y_1 y_2 \cdots y_k x$ を変換表に登録し，つぎに，**while** ループに制御が移って，記号列 $y_i \cdots y_k x$, $i=2, \cdots, j$ が変換表に登録され，記号列 $y_{j+1} \cdots y_k x$ が変換表に登録済みであることを知って，**while** ループを脱け出すことになる．すなわち，$j-1$ 回 **while** ループを反復することになる．

以上の例で理解できるように，**while** ループ1回の実行で記号列1個が登録されるので，**while** ループの延べ反復回数は入力記号列長を越えることはない．こうして，算法1による符号化の手間が $O(n)$ であることが知れる．

表 2 算法2の変換表
Table 2 String table in algorithm 2.

番地 c	部分列	$\pi(c)$	$fin(c)$	$\sigma(c)$
1	a	0	a	0
2	b	0	b	0
3	c	0	c	0
4	$a\ b$	1	b	2
5	$b\ a$	2	a	1
6	$b\ c$	2	c	3
7	$c\ b$	3	b	2
8	$b\ a\ b$	5	b	4
9	$a\ b\ a$	4	a	5
10	$a\ a$	1	a	1

算法1の復号算法は上述の符号化算法を逆にたどる過程であるので容易に構成することができる．構造的には符号化算法よりやや複雑であるが，復号の本体部分 (*Decode*) で h の計算を必要としないので，その分，符号化よりも高速に実行することができる．これも，LZW 法の場合と同様である．復号算法の全体を付録に示した．

3.2 算 法 2

先にあげた例の表1において，

（R 3） 任意の c について，$\sigma(c)<c$ である．

を満足しない行を削除して，必要ならば値の変更を行って残った行のみで表を再構成すると**表 2** を得る．表2では，後述する理由から部分列長 *len* が省略されている．入力記号列の分解と符号化にこのような変換表を使うのが算法2であるが，条件（R 3）を満足しない行を単に削除したのでは，それを π や σ で参照している番地が残る可能性がある．算法2では，条件(R3) を満足しない行を，算法1による変換表を作成した後に削除するのではなく，3条件（R 1），（R 2）かつ（R 3）を満足する変換表を，条件（R 3）による判定を陽的に行うことなく，逐次的に直接構成することができる．

算法2のメイン・ループは算法1と同一であるが，既に述べたように，大域変数 M が必要になる．算法1では，入力記号列の第 i 番目の記号から始まる記号列が常に変換表の第 $\alpha+i$ 番目に登録されるのに対し，算法2では，変換表に登録される記号列が入力の何番目の記号から開始した部分列であるかをあらかじめ知ることはできない．このため，変換表の最高位番地 T とは別に，変換表の利用可能な上限を制限する必要が生じる．これを表現するのが変数 M であるが，この

点を除けば，次に示すように，符号化手続きは算法1のものとほとんど等しい．

```
procedure Encode(x : char);
begin
if h(P, x)<0 or h(P, x)>M then begin
    encode P with fixed-length;
    P:=h(0, x);
    M:=T
    end
else P:=h(P, x)
end;
```

変換表への記号列の登録は，算法1の *TableUpdate* の **while** ループ内での記号列の登録を省いたときに相当し，算法1に比較してより単純かつ高速に実行することができる．

```
procedure TableUpdate(x : char);
begin
if h(K, x)>0 then K:=h(K, x)
else begin
    while h(σ(K), x)<0 do K:=σ(K);
    T:=T+1;
    π(T):=K;
    fin(T):=x;
    σ(T):=h(σ(K), x);
    K:=σ(T)
    end
end;
```

算法2の復号は，付録に示した算法1の復号算法をそのまま使うことによっても実行することができる．ただし，変換表の使用可能な番地に上限を設定したので，手続き *Decode* の中で $p>T$ が成立することはなく，本質的には 21 行目以降の **else** 節しか必要としない．したがって，上述の *TableUpdate* では，登録記号列長を *len* として変換表に記録することをしていない．

変換表への登録記号列の開始位置を入力記号列の各記号に取ったものが算法1であるのに対し，算法2や LZW 法は開始位置をより疎に与えたものであり，開始位置の指定法によって，より一般の方式が設計できることを示唆している．算法2は，そのような中にあって，開始位置判定のための特別の手間を必要としない特異な方法となっている．また，実用的には，性能と計算量の上で，算法1と LZW 法との中間に位置するという意義も有している．

4. シミュレーションによる性能評価

ハフマン符号でデータ圧縮が行えることの理由は，記号の出現頻度と符号長の可変性とを対応づけている仕組から直観的にも理解される．これに対し，本論文で扱った方法のいずれを考えても，それがなぜ圧縮を行うのかを簡単に理解することは容易でない．実際，LZW 法を提案した Welch 自身も，これについての定性的説明や理論的説明には何も言及していない．基本的には，ハフマン符号が源記号と符号語との間に固定長-可変長の対応を与えているのに対し，LZW 法や提案した算法では，可変長-固定長の対応を与え，入力記号列上の出現頻度の高い記号列ほど，より長いブロックとして符号化されるようにし，提案した算法は，入力記号列から出現頻度のより高い記号列を抽出

表3 シミュレーション用ファイル
Table 3 Tested files.

File #	type	original size [bytes]
1	C	2938
2	Pascal	8284
3	C	10122
4	Fortran	15727
5	English	18965
6	Fortran	47181

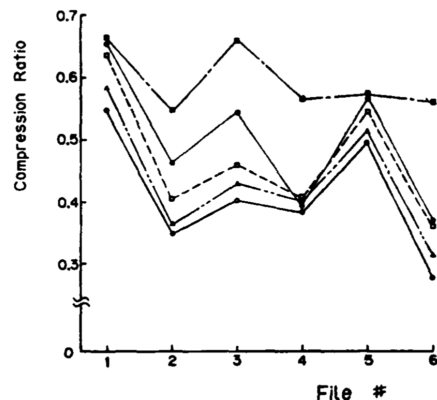


図3 圧縮比の比較
Fig. 3 Compression comparisons.

する作業を LZW 法よりも精細に行ったものと考えることができる．これを厳密に議論し，定量的に評価することは今後の課題であるが，本論文では，LZW 法が経験的に高く評価されている事実に着目し，比較のための実験をいくつか試みた．

LZW 法と提案した算法の出力符号語長としては，変換表の大きさ T が $T \leqq 2^{12}$ では 12 ビット，$2^{12} < T \leqq 2^{16}$ では 16 ビットを選び，出力符号語の総長をバイト（8ビット）単位で測ったものを入力記号列長で割ったものを圧縮比 (Compression Ratio) としている．比較のために，動的ハフマン符号を実現した *compact* 命令[11]および DAFC 法による測定も併せて行った．DAFC 法は，パラメータの設定に任意性があるが，原論文[5]で使用している値を採用し，いわゆるランモードの利用はしていない．

表3に，入力記号列として取り上げたファイルの例を示す．ファイル1と3とはC言語の，ファイル2は Pascal の，ファイル4と6とは Fortran のソースプログラムで，ファイル5は英文のテキストである．Fortran プログラムのファイル6は，ファイル4を3個単に接続しただけのものである．いずれもバイト・データで，$\alpha=256$ とした．

圧縮比の測定結果を**図3**に示す．いずれのファイルに対しても，算法1が最も効率的であり，算法2は，それと LZW 法との中間の結果を示し，3算法の相互関係を反映している．ファイル4に対しては，DAFC 法，LZW 法，算法1そして算法2とも大きな差はないが，これを3個接続したファイル6になるとその差が歴然とし，特徴的である．しかし，入力長が長くなければ提案した算法の効果が現れないわけではなく，10K バイト以下のファイルに対しても算法1の顕著な優位性は変わらない．*compact* 命令のマニュアル[11]では，ファイルの種類と圧縮比の関係について，C言語よりは Pascal のソースプログラムが圧縮しやすく，英文のテキストは両者の中間にあることが指摘されている．これと同様の傾向が，算法1および算法2にも認められる．しかし，*compact* 命令の圧縮力がファイル長の比較的初期に飽和してしまうのに対し，LZW 法や提案した算法では，ファイルの長さとともに圧縮力が向上することが観察できる．

処理時間については，LZW 法と比較すると算法1で3倍強，算法2で約3倍程度必要とするが，それでも *compact* 命令や DAFC 法よりははるかに高速である．重要なことは，算法1，算法2そして LZW 法の3方法とも，同一のデータ構造上に定義できるということである．したがって，処理時間と圧縮力とのトレードオフから，入力記号列の途中で相互に方法を移行することも容易であり，定常なデータであれば，初期には算法1を用い，後に LZW 法に移行することなどが妥当な方法として考えられる．

いずれにしても，これだけ単純な算法でしかも高速に，このように高い圧縮結果が得られる汎用算法はほかに例がなく，今後は Welch と同様，より効率的な実現法やハードウェア化について検討を加える必要がある．

5. おわりに

本論文では，実時間データ圧縮用の高性能実用算法として，入力記号列上のパターン照合に基づく2種の新算法を提案した．いずれも，非常に単純な論理で実現できるうえに，従来の汎用算法の性能に劣らない圧縮力を有している．本論文では，このことをシミュレーションによって確認したが，これを理論的に説明する適切なモデルの構築は今後の課題である．特に，算法内で使用している変換表の大きさと性能との関係を定量的に評価することができれば，提案した算法の性能の理論的な裏づけとなろう．また，変換表に記号列を登録するときの，入力記号列上の開始位置を指定する規則の中で，線形時間で実現可能な自明な方法以外の方法がほかに存在するかについても検討する必要がある．

謝辞 本論文作成に協力いただいた，山形大学青木ミヨ子技官に深謝する．

参 考 文 献

1) Vitter, J. S.: Design and Analysis of Dynamic Huffman Codes, *J. ACM*, Vol. 34, No. 4, pp. 825-845 (1987).
2) Gallager, R. G.: Variations on a Theme by Huffman, *IEEE Trans. Inf. Theory*, Vol. IT-24, No. 6, pp. 668-674 (1978).
3) Knuth, D. E.: Dynamic Huffman Coding, *J. Algorithms*, Vol. 6, No. 2, pp. 163-180 (1985).
4) Vitter, J. S.: Dynamic Huffman Coding, in *Collected Algorithms of ACM*, Association for Computing Machinery (1986).
5) Langdon, G. G. Jr. and Rissanen, J. J.: A Double-Adaptive File Compression Algorithm, *IEEE Trans. Commun.*, Vol. COM-31, No. 11, pp. 1253-1255 (1983).
6) Bentley, J. L., Sleator, D. D., Tarjan, R. E. and

Wei, V. K.: A Locally Adaptive Data Compression Scheme, *Comm. ACM*, Vol. 29, No. 4, pp. 320-330 (1986).
7) Ziv, J. and Lempel, A.: Compression of Individual Sequences via Variable-Rate Coding, *IEEE Trans. Inf. Theory*, Vol. IT-24, No. 5, pp. 530-536 (1978).
8) Welch, T. A.: A Technique for High-Performance Data Compression, *IEEE Comput.*, Vol. 17, No. 6, pp. 8-19 (1984).
9) 朴 志煥, 今井秀樹, 山森文範, 伊藤秀一, 石井正博：パターンマッチングと算術符号を用いた実用的なデータ圧縮法, 電子情報通信学会論文誌, Vol. J 71-A, No. 8, pp. 1615-1623 (1988).
10) 横尾英俊：Welch 型データ圧縮法の新算法, 第11回情報理論とその応用シンポジウム資料, pp. 735-740 (1988).
11) McMaster, C. L.: Documentation of the Compact Command, in UNIX User's Manual, 4.2 BSD, Virtual VAX-11 Version, Univ. of California, Berkeley (1984).

付録 復号算法

下記のメイン・ループの中で使用する手続き *ReceiveAndDecode* は，符号化算法において固定長で符号化された整数値を復号するものである．手続き *Decode* の中では，長さ L の一次元配列 *buffer* を使用するが，L は変換表に登録される記号列の最大長だけ必要で，100 K バイト程度までのファイルを対象とする場合には，数百もあれば十分である．

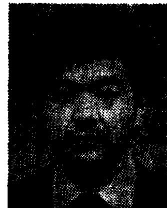
メイン・ループ：

```
Initialize;
T:=α; K:=0;
while there is more to decode do begin
    p:=ReceiveAndDecode;
    Decode(p);
    end;
```

手続き *Decode*：

```
procedure Decode(p: integer);
var i, j, k, q: integer;
    x, buffer[L]: char;
begin
j:=0;
if p>T then begin
    q:=K;
    for i:=T+len(K) downto p do begin
        j:=j+1;
        buffer[j]:=fin(q);
        q:=π(q)
        end;
    k:=j; i:=1;
    while p>T or i≠len(p) do begin
        x:=buffer[k];
        Output(x);
        TableUpdate(x);
        if k=1 then k:=j else k:=k-1
        end
    end
else begin
    while p>0 do begin
        j:=j+1;
        buffer[j]:=fin(p);
        p:=π(p)
        end;
    while j>0 do begin
        x:=buffer[j];
        Output(x);
        TableUpdate(x);
        j:=j-1
        end
    end
end;
```

(昭和63年9月 9 日受付)
(平成元年7月18日採録)



横尾 英俊（正会員）
1954年生．1978年東京大学工学部計数工学科卒業．1980年同大学院情報工学専攻修士課程修了．同年山形大学工学部電気工学科助手．1989年群馬大学工学部情報工学科講師．工学博士．データ圧縮，記号処理およびそれらの応用に関する研究に従事．電子情報通信学会，情報理論とその応用学会各会員．