Vol. 35 No. 7　　情　報　処　理　　July 1994

解　説

情報理論の計算機システムへの応用

1. ユニバーサルデータ圧縮アルゴリズム：原理と手法†

山　本　博　資††

1. はじめに

通信や記録を効率よく行うためにデータ圧縮技術が用いられているが，同一のアルゴリズムで広範囲な情報源出力を効率よく圧縮できる**ユニバーサル符号**が最近特に注目され，いろいろなところで実用化され始めている．ユニバーサル符号が盛んに研究されるきっかけを作ったのは，Ziv と Lempel がいわゆる **Lempel-Ziv**（あるいは Ziv-Lempel）**符号**[17)~19)]を提案したことによるが，かれらの符号は符号化定理を証明するために考案されたものであり，効率のよいものではなかった．その後，Welch[21)] が Lempel-Ziv 符号を改良したいわゆる **LZW 符号**を提案し，それが高速でかつ効率のよい圧縮方法であることを示した．また LZW 符号を少し改良した符号が UNIX の **compress コマンド**としてインプリメントされたことにより，実用的な符号としてユニバーサルデータ圧縮符号が広く知られるようになった．

その後，より効率がよく，より高速な圧縮を実現するために，多くのユニバーサルデータ圧縮アルゴリズムや改良アルゴリズムが提案され，またすでに多くの解説書（解説論文）[1)~6)]も出版されている．しかし，それらの解説書は個々のアルゴリズムの紹介に留まっているものが多く，ユニバーサル符号全体を統一的に解説したものは少ない．本稿では，個別のアルゴリズムを紹介することよりも，ユニバーサル符号全体に共通する基本的な概念および効率のよい符号を構成するための共通的なテクニックを紹介することに主眼をおいて解説する．

† Universal Data Compression Algorithms: Principles and Techniques by Hirosuke YAMAMOTO (Department of Mathematical Engineering and Information Physics, Faculty of Engineering, University of Tokyo).

†† 東京大学工学部計数工学科

なお，本稿では Lempel-Ziv 符号を始めとする**辞書（Dictionary）法**（あるいは Textual Substitution 法，Macro 法）と呼ばれているユニバーサル符号について解説を行うが，ユニバーサル符号には辞書法以外に，**算術符号**と **Context-Modeling**（あるいは，MDL[9)]などを用いた **Universal-Modeling**）を用いて構成する方法などがある[7)]．紙面の都合上後者については省略するが，辞書法の各符号と等価な文脈モデルを作ることにより，「算術符号＋文脈モデル」で辞書法の符号と等価な符号を作ることもできる[3),8)]．しかし，一般にそのように構成した符号は辞書法に比べて，多くのメモリが必要であり符号化復号化速度が遅い欠点がある☆．

性能のよいユニバーサル符号を作るためには，その性能評価方法を正しく理解しておく必要がある．各種のユニバーサル符号の性能を公正に比較するためには次のような項目に関して性能を比較しなければならない．

(**a 1**) 圧縮率
(**a 2**) 符号化速度，復号化速度
(**a 3**) 符号化メモリ量，復号化メモリ量
(**a 4**) そのユニバーサル符号が適用可能な情報源クラスの広さ

これらの項目すべてに優れているユニバーサル符号は存在せず，一方の項目の性能を上げると他方の性能が悪くなるという trade-off の関係が一般に成り立っている．

どの項目の性能をよくすべきかは，その使用目的により異なる．たとえば，「性能の低い端末から高性能計算機へのデータ通信」では「符号化速度，符号化メモリ量の性能」が，また逆に「高性能計算機から性能の低い端末へのデータ通信」では「復号化速度，復号化メモリ量の性能」が重要

☆ 辞書法の一部に算術符号を利用することもある（2.(b1)参照）．

になる．また，計算機のディスク上に記録された頻繁に使用するファイルを圧縮するような場合は，その圧縮されたファイルを瞬時に復号する必要があるが，符号化は計算機が使用されていないとき（あるいは負荷の低いとき）に時間をかけて圧縮することができる．したがって，そのような場合は復号時間の短い符号が必要となる．

(a 4) に関しては，より広いクラスを対象にすれば一般に圧縮率が悪くなり，狭いクラスを対象にすればそのクラスに特有な性質を利用して圧縮率のよい符号を作れる．しかし，ある特殊なクラスに対して効率を上げるように工夫すると，一般的に言って，そのクラスを外れた情報源に対して圧縮率が悪くなったり，場合によっては正しく復号できなかったりする．

2. 辞書法

辞書法によるユニバーサル符号では，辞書を動的に更新しながら，その辞書を用いてデータ系列を圧縮していく．簡単のためにデータの部分系列を $x(s,t)=x_s x_{s+1}\cdots x_t$ および $x(s,\cdots)=x_s x_{s+1}\cdots$ で表すと，辞書法の符号化および復号化は，それぞれ次の (Step-A1)-(Step-A3) および (Step-B1)-(Step-B3) を繰り返すことにより行われる．

(**Step-A1**) 現時点でまだ符号化されていない $x(s,\cdots)$ から，その語頭 (prefix) 部 $x(s,t)$ をある規則に基づいて分解 (parsing) する．

(**Step-A2**) $x(s,t)$ を現時点の辞書を用いて符号化する．

(**Step-A3**) $x(s,t)$ を用いて，辞書をある決められた方法で更新する．

(**Step-B1**) 符号語系列*の先頭から，元の部分系列一つ分に対応した符号語を取り出す．

(**Step-B2**) 現時点の辞書を用いて，その符号語から元の部分系列 $x(s,t)$ を復号する．

(**Step-B3**) 復号した $x(s,t)$ を用いて，(Step-A3) と同じ方法で辞書を更新する．

復号側の辞書の初期値が符号化側と同じならば，辞書を送らなくても符号化側と同じ辞書が復号側で逐次的に作成でき，正しく復号できる．

上記の (Step-A2) 部分は，さらに細かく次の2段階の手順で符号化される．

(**Step-A2-1**) 符号化したい部分系列 $x(s,t)$ が辞書内のどの単語と一致しているかを示す正整数値を用いて，$x(s,t)$ を符号化する*.

(**Step-A2-2**) その各正整数値を2値符号化（0と1の系列で表現）する．

したがって，よい圧縮率を達成するためには，上記の各段階においてそれぞれ効率のよいアルゴリズムを用いなければならないが，当然互いに関連しており，整合性のよいアルゴリズムを用いる必要がある．

(Step-A2-2) の正整数の2値符号化法は，次のように大きく2通りの方法がある．

(**b 1**) エントロピー符号化を用いる方法
具体的には，(ⅰ)算術符号，(ⅱ)動的 (Dynamic) Huffman 符号[10)~12)etc.]，(ⅲ) 静的 Huffman 符号，などを用いる．

(**b 2**) 正整数のユニバーサル表現を用いる方法

一般的に言って，(b 1)のほうが(b 2)より圧縮率はよくなるが，逆に(b 2)のほうが処理速度が速く必要なメモリ量も少なくてすむ．したがって，これらのうちのどの方法を用いるかは，(a 1)～(a 3)の trade-off の関係により決めることになる．次章では，これらのうち(b 2)の正整数のユニバーサル表現について少し詳しく説明する．

3. 正整数のユニバーサル表現

多くのユニバーサル符号では，前章の (Step-A 2-1) で用いられる正整数が次のどちらかの性質を満たすように工夫されている．

(**c 1**) 整数 n がある既知の N に対して，$1\leq n\leq N$ の範囲で等確率で生起する．

(**c 2**) 小さい正整数 n ほど大きい確率で生起する．

(c 1)の場合，最も簡単な符号化法は $\lceil\log_2 N\rceil$ ビットの**固定長符号**で n を2進数表現する方法である．しかし，この方法では，理想的な符号長 $\log_2 N$ に比べて，$(\lceil\log_2 N\rceil-\log_2 N)$ ビットのロスが生じる．このロスの最大の原因は，固定長符号

☆ $x(s,t)$ を圧縮符号化したものを符号語と呼び，それらの連なったものを符号語系列と呼ぶことにする．

☆ たとえば，$x(s,t)$="information" が辞書の9ページ目の7項目に載っているとき，"information" を〈9, 7〉で表現するようなことを考えればよい．詳しくは4.を参照せよ．また，辞書内に一致する単語が存在しない場合の処理は，6.の非圧縮モードを参照せよ．

を符号木☆ で表したとき，完全木にならないためであるが，この欠点を改良して完全木になるようにした **CBT 符号** (Complete Binary Tree Code)[22),31)etc.] が知られている．固定長符号に比べて CBT 符号を用いることにより，圧縮率で数パーセント程度の改善効果が得られる場合が多い．

(c 2)の場合は，$P(n) \geq P(n+1)$, $n \geq 1$, を満たすような正整数 n を効率よく 2 値符号化する場合であり，小さい正整数ほど短いビット数で表現できるように工夫した 2 値符号化法である．(c 2)はさらに次の 2 通りの場合に分けて考えることができる．

(**c 2-1**) $1 \leq n \leq N$ (あらかじめ n の最大値 N が分かっている場合)

(**c 2-2**) $1 \leq n$ (あらかじめ最大値が分からない場合や，分かっていても最大値が非常に大きい場合)

(c 2-1)の方式の中で，簡単で効率のよい方式として (**start, step, stop**) **符号**[31)]がある．

(c 2-2)に対する 2 値符号化法は，語頭条件を満たす（つまり符号語の区切りを示す）方法により，大きく次の二つに分割できる[16)etc.]．

(**d 1**) Flag 方式[15),16)etc.]．

(**d 2**) メッセージ長方式[13),14)etc.]．

(d 1)は，データ部（数値を表している部分）の後ろに，区切りを示す特殊な Flag パターンを Suffix として付加し，そのパターンがデータ部に現れないように工夫したものであり，(d 2) はデータ部のビット長を示す符号を Prefix としてデータ部の前に付加する方式である．

これらの方式の多くは漸近的に（つまり，確率 $P(n)$ によって定まるエントロピー $H(P(n))$ が十分大きい場合に）最適であることが示されている．特に(d 2)に属する Elias の二重化 prefix 方式[13)]は，ユニバーサル符号の性能を理論的に評価する場合によく使用される．また，実用的には文献14)～16)などの方式が性能的に優れている．

4. データ系列の分解と符号化

2. で述べた (Step-A1)(Step-A2-1) のように，データを部分系列に分解し，その部分系列が辞書内のどの単語と一致しているかを整数値を用いて示すことにより符号化が行われる．このとき，参照できる単語(すなわち部分系列)の長さにより，

(**e 1**) 固定長方式[27),30),33)etc.]．

(**e 2**) 可変長方式

に分類することができるが，実用的なユニバーサル符号の大半は可変長方式となっている．また，辞書内の単語の参照方法により，次の 2 通りの方式に分類することができる．

(**f 1**) 単語の場所とその長さで指定する方式

(**f 2**) 単語の場所を指定するとその長さが一意に定まる方式

(e 1)の固定長方式は当然(f 2)方式となるが，(e 2)の可変長方式でも，単語長の増加がある一定の規則にしたがって変化している場合は，(f 2)の方式が可能となる[19),21)etc.]．

(f 1)の方式は，一個の部分系列を辞書に登録すれば，その長さを変えて参照できるため，辞書の利用効率がよい．他方，(f 2)の方式は場所のパラメータだけで辞書内の単語を参照できるため，一致する単語が辞書内に存在するときは，短いビット数で符号化できる．しかし，符号化の初期段階など辞書に単語がまだ十分登録されていない場合，辞書内に一致系列が見つかりにくくなり，全体として圧縮率が悪くなる場合がある．

辞書が，すでに符号化の終了した部分系列 $x(1, s-1)$ をそのまま蓄えている単純な Buffer の場合に対して，(f 1)と(f 2)の符号化方式の例を示す．

(**例 1**) $x(s, \cdots)$ に最も長く一致する部分系列を Buffer 内で探した結果，ある m, $1 \leq m \leq s-1$, に対して $x(s, t) = x(m, m+t-s)$ であるとき，x_m と x_s の**インターバル長** $s-m$ と，**一致長** $l_{st} = t-s+1$ を用いて $\langle s-m, l_{st} \rangle$ と符号化する[20)etc.]．たとえば，$x(1, 12) = abaabaaabbab$, $x(13, \cdots) = aabbaab\cdots$ のとき，$x(13, 17) = aabba$ を $\langle 6, 5 \rangle$ と符号化する．

(**例 2**) $x(1, s-1)$ がある規則で，$x(1, s-1) = x(1, t_1)x(t_1+1, t_2)\cdots x(t_{u-1}+1, t_u)$ と分解されているとする．このとき，ある $j(\leq u)$, に対して，$x(s, t) = x(t_{j-1}+1, t_j)$ のとき，$x(s, t)$ を $\langle j \rangle$ と符号化する．

たとえば，上記の例で $x(1, 12) = a, b, aa, ba, aab, bab$ と分解されているとき，$x(13, 15) = aab$ を $\langle 5 \rangle$

☆ 各 n を葉に対応させ，n の 2 値符号語を根から葉までの枝に対応させたもの．

と符号化する．

$x(1, s-1)$ を分解する方法はいろいろ考えることができるが，例2で示した $x(1, 12)$ の分解は次の性質を満たす分解である．

$$x(t_{i-1}+1, t_i)=x(t_{j-1}+1, t_j)x_{t_i}, \quad \text{for some } j, 0\leq j\leq i-1 \qquad (1)$$

$$x(t_{i-1}+1, t_i)\neq x(t_{j-1}+1, t_j), \quad \text{for any } j, 0\leq j\leq i-1 \qquad (2)$$

ただし，$x(t_{-1}+1, t_0)=\lambda$（空系列）である．この性質を満たす分解法は，Lempel-Ziv[19]により提案され，**増分分解 (Incremental Parsing)** とよばれている．

5. 辞書の構成法

ユニバーサル符号の性能(a 1)～(a 4)は，用いる辞書に大きく依存するが，具体的には次のような点に注意しなければならない．

(**g 1**) 辞書のデータ構造

(**g 2**) 辞書の初期値，辞書への単語（すなわちデータの部分系列）の登録方法，単語の削除方法

1. で述べたように(a 1)～(a 4)には trade-off の関係が存在するため，唯一の最適な方式が存在するわけではない．また，(g 1)，(g 2)は **4.** で述べた(e 1)，(e 2)，(f 1)，(f 2)と互いに関連しており，それぞれを別々に取り扱えるものではないが，以下では，(g 1)，(g 2)に関してそれぞれ一般的な考え方をまとめておく．

辞書のデータ構造

辞書のデータ構造としては，Unsorted List, Sorted List, Link Buffer, Binary Tree, Trie, Hash table, etc. や，それらを組み合わせたものが考えられる．どのデータ構造を用いるかは，(a 1)～(a 4)のどの性能に重きを置くかにより異なる．符号化では，符号化したい部分系列と一致する単語を辞書内から高速に探索する必要があるため，Tree や Tric 構造がとられることが多い．たとえば，Bell[25]の LZSS 符号では二分探索木が用いられており，Fiala-Greene 符号[31]では，Patricia tree が用いられている．

復号の辞書は符号化の辞書と同じデータ構造を用いるものもあるが，符号によっては，まったく異なっている場合もある．(例1)の符号化では，$x(s, \cdots)$ と一致するものを辞書内から高速に探す必要があるため Tree 構造を用いる必要があるが，復号は Tree を用いなくても $x(1, s-1)$ を蓄える Buffer があれば，$\langle s-m, l_{st}\rangle$ の情報から $x(s, t)=x(m, m+l_{st}-1)$ を容易に復号できる．

辞書の初期値

辞書の初期値に関しては，

(**h 1**) 空（または，ほぼ空の状態☆）の辞書

(**h 2**) あらかじめ用意した比較的大きな辞書

から始める二通りが考えられる．現在提案されているユニバーサル符号はユニバーサル性を高めるために，(h 1)の方式が取られているものがほとんどである．今後 CD-ROM などを利用した標準の辞書が各計算機で利用できるような環境が整った場合，後者の方式も検討する必要があろう．

単語(部分系列)の登録方法

(Step-A3) で述べたように，部分系列 $x(s, t)$ の符号化が終了すると，$x(s, t)$ に基づいて辞書に新たな単語を登録する必要がある．単語の登録方法は，辞書のデータ構造や辞書の参照方法に大きく依存しているが，ほぼ次のように分類できる．

(**i 1**) $x(s, t)$ に対して，$x(i, i+l), s\leq i\leq t-1, 0\leq l$（または，ある与えられた L に対して，$0\leq l\leq L$）[17),31)etc.] をすべて登録する[18)etc.]．

(**i 2**) $x(s, t)$（あるいは $x(s, t)x_{t+1}$）のみを単語として登録する[19),21)etc.]．

(**i 3**) (i 1)と(i 2)の中間の方式．いろいろな登録法が可能であるが，たとえば，次のような登録法が知られている．

(**i 3-1**) $x(s, t)$ に対して，$x(s, s+l), 0\leq l,$ を単語として登録する[31)etc.]．

(**i 3-2**) すでに $x(t_{j-1}, t_j), 1\leq j\leq u,$ が登録されているとき，各 $x(s+l, \cdots), 0\leq l,$ ごとに，式(1)，(2)を満たすものを登録する[37)]．

(i 1)，(i 3-1)は(f 1)に対応した方式であり，(i 2)，(i 3-2)は(f 2)に対応した方式である．また，(例1)，(例2)はそれぞれ(i 1)，(i 2)に対応している．

(f 2)，(i 2)を用いた方式では，登録された各単語がほぼ等確率で参照されると見なせる場合が多いので，(c 1)の2値符号化法を利用するものが多い[19),21)etc.]．一方，(i 1)あるいは(i 3-1)に(f 1)を用いた方式では，辞書内の場所と長さを用いて符号化することになるが，一致する単語の

☆ たとえば，長さ1のすべての単語を登録した状態．

長さは短いほうが長い場合より生起しやすいので，(c2)の2値符号化法が用いられる[17),25)etc.]．場所を表す数字は(i1)では(c2)が用いられるが，(i3)のように登録する部分系列を制限した場合は，Tree 構造を用いて(c1)の2値符号化法に適した辞書にすることもできる[31),37)]．

単語の削除方法

通常符号化が進むにつれて，単語が次々に登録されていくため，辞書（メモリ容量）がいっぱいになったときの処理を考える必要がある．一般に，辞書がいっぱいになったときの処理法としては次のような方法がある[29)]．

(j1) 辞書を Reset する．

(j2) 辞書を固定する（単語の登録を中止する）．

(j3) Swap 用の辞書を用意する．これは，辞書がいっぱいになった時点（あるいは辞書の容量がある基準値を超えた時点）で新しい辞書を作り始め，その辞書がある程度充実した時点で辞書を Swap する方式である．

(j4) 辞書の一部を削除しながら新しい単語を登録する．この方式はどの単語を削除するかによって，さらに次のように分類できる．

(j4-1) 最も以前に登録された単語を消去する[25)etc.]．

(j4-2) 最も以前に参照された (LRU, Least Recently Used) 単語を消去する[26)etc.]．

(j4-3) 最も参照された回数の少ない (LFU, Least Frequently Used) 単語を消去する[28)etc.]．

これらの方式にはそれぞれ次のような欠点がある．(j1)の方式は，Reset 直後の圧縮率が悪くなる．(j2)では，辞書の固定後は適応性がなくなる．(j3)では辞書を二重に用意するため，メモリの半分（あるいは何割か）を有効に利用できない．(j4)は適応性を保ったまま，メモリを有効に利用できるので，最もすぐれた方式ではあるが，辞書が Window-Buffer からできている簡単な場合などを除いて，一般に処理が複雑になり，符号化復号化に時間がかかる．

6. 圧縮率の改良

各種のユニバーサル符号に関して，圧縮率を改善する数多くのアイデアが出されているが，本稿では多くのユニバーサル符号に共通に適用できる圧縮率の改善方法についていくつか紹介する．

4. (例1)の符号化法では，データの部分系列 $x(s,t)$ が，$x(s,t)=x(m,m+t-s), m<s$, を満たすとき，$x(s,t)$ をそのインターバル長 $s-m$ と最大一致長 $l_{st}=t-s+1$ を用いて，$\langle s-m, l_{st}\rangle$ と符号化する*．通常，これらの整数値は小さいほうが生起しやすいので，(c2)に対応した2値符号化法を用いて2値系列に符号化される．したがって，このような場合は，インターバル長と一致長をそれぞれ $s-m$, l_{st} より小さい正整数で表現できれば，より効率よく圧縮できることになる．以下では，インターバル長と一致長のより小さい正整数を用いた表現方法について紹介する．

一致長の短縮化表現

インターバル長 $s-m$ が分かっているとき，一致長 l_{st} をより小さい数字で表現する方法を考える．$x(m,m+t-s)$ は，$x(s,t)$ に最も近い最大一致系列なので，$x(i,i+l_{st}-1)\neq x(s,t), m<i<s$, であるが，$x(i,i+l-1)=x(s,s+l-1), m<i<s, 1\leq l<l_{st}$, を満たす i と l が存在する．このような l の中で，最も長いものを l_M とすると，$\langle s-m, l_{st}\rangle$ の代わりに，$\langle s-m, l_{st}-l_M\rangle$ と表現しても，$x(s,t)=x(m,m+t-s)$ を復元できる[35)]．

たとえば，$x(1,10)=cabcdaabca$, $x(11,\cdots)=abcdc\cdots$ の場合，$x(11,14)=abcd$ が $x(2,5)$ に等しいので，$x(11,14)$ をインターバル長と一致長を用いて符号化すると $\langle 9,4\rangle$ となるが，$x(11,13)=x(7,9)=abc$ より，$l_{st}-l_M=4-3=1$ であることを利用すると $\langle 9,1\rangle$ と符号化できる．このとき，$\langle 9,1\rangle$ より実際の一致長は次のように求まる．$\langle 9,1\rangle$ と $s=11$ から $x(s,t)$ が $m=2$ から始まる系列と一致することが分かる．次に x_2 から始まる系列と最も長く一致する系列 $x(i,i+l-1)$ を $3\leq i\leq s-1$ の範囲で探す．この例の場合は，$x(2,4)=x(7,9)=abc$ が最大一致系列となり，$l_M=3$ であることが分かる．したがって，実際の一致長が $l_{st}=(l_{st}-l_M)+l_M=1+3=4$ であることが分かる．

この符号化法は，Fiala-Greene 符号[31)]のC方式で取り入れられている．C方式では，辞書内の単語に対する一致長を Patricia Tree 上で最後に枝分かれした節点からの長さを用いて表現しているが，これは $l_{st}-l_M$ を表しているのにほかならない．

---

* $x(s,\cdots)$ に最も長く一致する部分系列が複数個存在する場合は，インターバル長の小さいものを用いるものとする．

インターバル長の短縮化表現 1

次に一致長 l_{st} が分かっているときに，インターバル長 $s-m$ を小さい正整数で表現する方法について考える．

$x(s,t)$ に一致するものを探すときに，m が $m<s-l_{st}+1$ の範囲の $x(m,m+l_{st}-1)$ のみに制限をする（つまり，$x(s,t)$ と重ならない範囲だけに制限する）と，インターバル長を用いた表現 $\langle s-m-l_{st}+1, l_{st}\rangle$*の代わりに，$x(m,m+l_{st}-1)$ が $x(s,t)$ から，長さ l_{st} の系列として何種類前の系列であるかを示す Recency-Rank（RR と略す）値を用いて，$\langle RR, l_{st}\rangle$ と符号化できる[27]．

たとえば，$x(1,11)=cabcdaaaaaa$, $x(12,\cdots)=abcdc\cdots$ の場合，$x(12,15)=x(2,5)=abcd$, $m=2$, $s=12$, $l_{st}=4$ より，インターバル長 $s-m-l_{st}+1$ を用いた符号化では，$x(12,15)$ は $\langle 7,4\rangle$ と符号化される．これに対して，$x(2,5)$ と $x(12,15)$ の間には，$bcda$, $cdaa$, $daaa$, $aaaa$ の 4 種類の長さ $l_{st}=4$ の部分系列が存在し，$x(2,5)$ は $x(12,15)$ に対して 5 種類前（$RR=5$）の系列になるので，RR 値を用いた表現では $\langle 5,4\rangle$ となる．

RR 値はインターバル値以下になるので，RR 値を用いればインターバル値より小さい正整数で表現できる．

l_{st} が固定長 L の場合，RR 値はワープロなどで用いられている move-to-front 方式（出現した単語 $\boldsymbol{x}^L$ を辞書の一番前に並び換える方式）を用いて求めることができる[24)etc.]．つまり，すべての種類の単語を一列に並べた逐次リストを用意し，単語 $\boldsymbol{x}^L$ が出現するごとに，その単語 $\boldsymbol{x}^L$ を move-to-front 方式で逐次リストの先頭に移動する．このとき，単語 $\boldsymbol{x}^L$ の RR 値は，その逐次リストにおける単語 $\boldsymbol{x}^L$ の順位で求まる．

逐次リストを用いる方法としては，move-to-front 方式以外に，出現した単語 $\boldsymbol{x}^L$ を逐次リスト上で k 個前に移動する move-ahead-k 方式や，その k が $k=1$ の場合に相当する transpose 方式などを用いることも可能である[28),32)]．

辞書として木構造を用いている場合，部分系列の長さ l_{st} が決まるとその長さの節点だけを考えればよく，対象となる節点数を少なくできる．その結果，対象となる節点の区別を小さい整数値で行え，圧縮率がよくなる．この効果は，本質的に上記の RR 値符号化と同じである．また，文献 38) の符号化法も RR 値法と本質的に等価である．

インターバル長の短縮化表現 2

$x(1,s-1)$ を利用して，$x(s,t)$ を符号化するとき，（例 1）では x_s から x_t までの系列の繋がりやすさを考慮した符号化になっているが，$x(1,s-1)$ と $x(s,t)$ にまたがる系列の繋がりやすさをまったく考慮していない．これを考慮に入れた符号化法として，次のような方式を考えることができる[36)etc.]．

$x(s,t)$ を符号化するとき，$x_{s-1}x(s,t)$ と一致する系列 $x_{m-1}x(m,m+l_{st}-1)$ を探す．$x_{s-1}=a$ のとき，そのインターバルを文字 a に続くところだけで数えあげる．たとえば $x(1,11)=cabcdaabcac$, $x(12,\cdots)=abcdc\cdots$ の場合，（例 1）の符号化法では通常のインターバル値は 10 となり，$\langle 10,4\rangle$ と符号化される．これに対して，$x_{11}=c$ で条件を付けて探すと，c は x_9, x_4, x_1 で現れており，3 個前の $c=x_1$ に繋がる $x_1x(2,5)$ が $x_{11}x(12,15)$ と一致している．したがって，$x_{11}=c$ で条件付けて数えたインターバル値 3 を用いて，$\langle 3,4\rangle$ と符号化できる．

上の説明では，x_{s-1} の一文字で条件付けることを考えたが，一般に $x(s-j,s-1)$ の j 文字で条件付けることも可能である．条件の長さ j を長くすると，その条件が出現する個数が少なくなるため，インターバルをより小さい数字で表現できる．しかし，$x(s-j,s-1)x(s,t)$ に一致する $x(m-j,m-1)x(m,m+l_{st}-1)$ を探すので，条件の長さ j が長くなると一致長 l_{st} の長いものが存在する確率が小さくなり，かえって圧縮率が悪くなる．

上記の短縮化表現を実際の符号化法に組み込むには，辞書を複数個用意し，直前部分系列 $x(s-j,s-1)$ に基づいて，使う辞書を決めて符号化すればよい[36),39)]．

非圧縮モードの導入

部分系列 $x(s,t)$ が 2 値系列に圧縮符号化されたときの符号語を $B(x(s,t))$ で表すとき，

$$|B(x(s,t))|<|x(s,t)| \qquad (3)$$

を満たせば，$x(s,t)$ が圧縮されたことになる．ただし，$|\cdot|$ はビット長を示す．

しかし，場合によっては符号化を行うと式（3）

---

* $s-m$ から $l_{st}-1$ を引いているのは，$m<s-l_{st}+1$ の範囲に制限しているため．

が満たされず，かえって長くなる場合が起こりえる．特に，符号化の初期などで辞書が十分成長していない場合などによく生じる．

このような膨張を防ぐために，非圧縮モードを導入し，次のような符号化出力 $\tilde{B}(x(s,t))$ を出す[25),31)etc.]．

$$\tilde{B}(x(s,t))=\begin{cases}0B(x(s,t)), & \text{if (3) holds.}\\ 1x(s,t), & \text{otherwise}\end{cases} \quad (4)$$

非圧縮モードを導入すると $B(x(s,t))$ だけを用いた場合に比べて，圧縮モードであるか非圧縮モードであるかを示すフラグに1ビット必要な分だけ効率が悪くなる☆．しかし，上記のような膨張を防ぐことができ，全データ $x(1,\cdots)$ に対する圧縮率は，非圧縮モードを用いない場合より非圧縮モードを導入したほうがよくなる場合が多い．

また，辞書の更新に(j4)のような単語の削除を許す場合は，辞書内に一致する系列がまったく存在しなくなる可能性がある．そのような場合には，非圧縮モードは不可欠となる．

先読みによる改良

2. の (Step-A1)-(Step-A3) のように，符号化はデータを部分系列に分解することと，それを符号化することを繰り返して行われる．長い部分系列を一度に符号化するほうが通常効率がよいので，できるだけ長く一致するものを辞書内から見つけて，その長さで分解する．しかし，最も長く一致するものを用いないほうが，次の分解のときに非常に長い一致が得られて，トータルすると効率がよくなる場合がある．このようにデータを先読みして，最適に分解すれば効率はよくなるが[20)]最適な分解を求めるためには，実際に幾通りにも分解符号化してみる必要があり，符号化の速度はかなり遅くなる．実用的には，1文字ずらして符号化したものと，ずらさずに符号化したものとを比べて，性能のよいほうを用いる方法が取られている[5)]．

7. おわりに

本稿では，各種のユニバーサルデータ圧縮符号に共通に存在する概念を明らかにし，高性能な符号を作るために必要な基本的な手法をいくつかの例を用いて紹介した．紙面の都合上，各ユニバーサル符号の個々のアルゴリズムには触れなかったが，興味のある人は，たとえば文献1)の Ch. 8 などを参照して欲しい．また，ユニバーサルデータ圧縮関連の詳しい文献リストは，文献4)，6)などにあげられている．

本稿では取り上げなかったが，重要なテーマとして，非常に大きなアルファベットサイズのデータに対する圧縮法や，データ圧縮のハード化の問題などがある．前者は，16ビットや32ビット程度の桁数で表した実数値データなどを符号化するとき，アルファベットサイズが非常に大きいため，新たな部分系列が過去の部分系列と完全に一致することが少なく，通常のユニバーサル符号のアルゴリズムでは効率よく圧縮できない．しかし，実数値の上位桁だけを取り出せば，同じ値が繰り返し出現する．このような特徴を利用すれば，アルファベットサイズが非常に大きなデータに対して効率よく圧縮する符号を作ることができる[34)etc.]．後者に関しては，多数のマイクロプロセッサで並列処理できるようなアルゴリズムを考案したり[23)etc.]，汎用のマイクロプロセッサの処理に適したアルゴリズムを考える必要がある[21)]．

本稿で示した(a)～(j)の各項目を組み合わせれば，多種多様のユニバーサル符号が構成できる．しかし，商業用のユニバーサルデータ圧縮符号を作る場合はともかく，ユニバーサル符号を学問的に研究する場合は，圧縮率の細かい数字を競うよりも，研究対象の符号がどのような点にどのような新しいアイデアが含まれているかを明らかにすることのほうが意味がある．そのような意味付けを行うときに，本稿で示したような分類が役に立つものと思われる．

☆ 非圧縮モード時の $x(s,t)$ の長さを可変にする場合は，$x(s,t)$ の長さを示すためのビットも必要になる．

参考文献

1) Bell, T., Cleary, J. G. and Witten, I. H.: *Text Compression*, Prentice Hall, Inc. (1989).
2) Williams, R. N.: *Adaptive Data Compression*, Kluwer Academic Publishers (1991).
3) Bell, T., Witten, I. H. and Cleary, J. G.: Modeling for Text Compression, *ACM Computing Surveys*, Vol. 21, No. 4, pp. 557-591 (Dec. 1989).
4) 山本博資：ユニバーサルデータ圧縮アルゴリズムについて，信学会技報，Vol. IT 90, No. 97, pp. 7-14 (Jan. 1991).
5) 奥村，吉崎：圧縮アルゴリズム入門，*C Magazine*, Vol. 3, No. 1, pp. 44-68 (Jan. 1991).
6) 山本博資：ユニバーサルデータ圧縮アルゴリズム

とその改良方法について，ミニワークショップ「情報理論における最近のアルゴリズムとその応用」予稿集，情報理論とその応用学会，高知（1992年12月11日，12日）．

7) Witten, I. H., Neal, R. M. and Cleary, J. G.: Arithmetic Coding for Data Compression, *Communications of the ACM*, Vol. 30, No. 6, pp. 520-540 (June 1987).

8) Rissanen, J.: A Universal Data Compression Scheme, *IEEE Trans. Inform. Thoery*, Vol. IT-29, No. 5, pp. 656-664 (Sep. 1983).

9) Rissanen, J.: Universal Coding, Information, Prediction, and Estimation, *IEEE Trans. Inform. Theory*, Vol. IT-30, No. 4, pp. 629-636 (July 1984).

10) Gallager, R. G.: Variations on a Theme by Huffman, *IEEE Trans. Inform. Theory*, Vol. IT-24, No. 6, pp. 668-675 (Nov. 1978).

11) Knuth, D. E.: Dynamic Huffman Coding, *J. Algorithms*, Vol. 6, pp. 163-180 (1985).

12) Vitter, J. S.: Design and Analysis of Dynamic Huffman Coding, *J. ACM*, Vol. 34, No. 4, pp. 825-845 (Oct. 1987).

13) Elias, P.: Universal Codeword Sets and Representations of the Integers, *IEEE Trans. Inform. Theory*, Vol. IT-21, No. 2, pp. 194-203 (Mar. 1975).

14) Amemiya, T. and Yamamoto, H.: A New Class of the Universal Representation for the Positive Integers, *IEICE Trans.* Vol. E 76-A, No. 3, pp. 447-452 (Mar. 1993).

15) Capocelli, R. M.: Comments and Additions to 'Robust Transmission of Unbounded Strings Using Fibonacci Representations', *IEEE Trans. Inform. Theory*, Vol. 35, No. 1, pp. 191-193 (Jan. 1989).

16) Yamamoto, H. and Ochi, H.: A New Asymptotically Optimal Code of the Positive Integers, *IEEE Trans. Inform. Theory*, Vol. 37, No. 5, pp. 1420-1429 (Sep. 1991).

17) Ziv, J. and Lempel, A.: A Universal Algorithm for Sequential Data Compression, *IEEE Trans. Inform. Theory*, Vol. IT-23, No. 3, pp. 337-343 (May 1977).

18) Ziv, J.: Coding Theorems for Individual Sequences, *IEEE Trans. Inform. Theory*, Vol. IT-24, No. 4, pp. 405-412 (July 1978).

19) Ziv, J. and Lempel, A.: Compression of Individual Sequences via Variable-Rate Coding, *IEEE Trans. Inform. Theory*, Vol. IT-24, No. 5, pp. 530-536 (Sep. 1978).

20) Storer, J. A. and Szymanski, T. G.: Data Compression via Textual Substitution, *J. ACM*, Vol. 29, No. 4, pp. 928-951 (Oct. 1982).

21) Welch, T. A.: A Technique for High-Performance Data Compression, *IEEE Computer*, Vol. 17, No. 6, pp. 8-19 (June 1984).

22) 横尾英俊：ユニバーサル情報源符号化のための修正 Ziv-Lempel 符号，信学会論文誌，Vol. J 68-A, No. 7, pp. 644-671 (July 1985).

23) Gonzalez Smith, M. E. and Storer, J. A.: Parallel Algorithms for Data Compression, *J. ACM*, Vol. 32, No. 2, pp. 344-373 (Apr. 1985).

24) Bentley, J. L., Sleator, D. D., Tarjan, R. E. and Wei, V. K.: A Locally Adaptive Compression Scheme, *Commun. ACM*, Vol. 29, No. 4, pp. 320-330 (Apr. 1986).

25) Bell, T. C.: Better OPM/L Text Compression, *IEEE Trans. Communication*, Vol. COM-34, No. 12, pp. 1176-1182 (Dec. 1986).

26) Tischer, P.: A Modified Lempel-Ziv-Welch Data Compression Scheme, *Australian Computer Science Communications*, Vol. 9, No. 1, pp. 262-272 (1987).

27) Elias, P.: Interval and Recency Rank Source Coding: Two On-line Adaptive Variable-Length Schemes, *IEEE Trans. Inform. Theory*, Vol. 33, No. 1, pp. 3-10 (Jan. 1987).

28) Horspool, R. N.: A Locally Adaptive Data Compression Scheme, *Commun. of the ACM*, Vol. 30, No. 9, p. 792 (Sep. 1987).

29) Storer, J. A.: Parallel Algorithms for On-Line Data Compression, *IEEE ICC '88*, pp. 385-389 (1988).

30) Willems, F. M. J.: Universal Data Compression and Repetition Times, *IEEE Trans. Inform. Theory*, Vol. 35, No. 1, pp. 54-58 (Jan. 1989).

31) Fiala, E. R. and Greene, D. H.: Data Compression with Finite Windows, *Comm. of the ACM*, Vol. 32, No. 4, pp. 490-505 (Apr. 1989).

32) 朴，高島，今井：Self-Organizing Rule に基づく適応的データ圧縮法，信学会論文誌，Vol. J 72-A, No. 8, pp. 1353-1359 (Aug. 1989).

33) Wyner, A. and Ziv, J.: Some Asymptotic Properties of the Entropy of a Stationary Ergodic Data Source with Applications to Data Compression, *IEEE Trans. on Inform. Theory*, Vol. 35, No. 6, pp. 1250-1257 (Nov. 1989).

34) Yokoo, H.: A Lossless Coding Algorithm for the Compression of Numerical Data, *The Trans. of The IEICE*, Vol. E 73, No. 5, pp. 638-643 (May 1990).

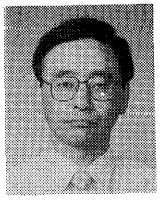
35) Bender, P. E. and Wolf, J. K.: New Asymptotic Bounds and Improvements on the Lempel-Ziv Data Compression Algorithm, *IEEE Trans. on Inform. Theory*, Vol. 37, No. 3, pp. 721-729 (May 1991).

36) Plotnik, E., Weiberger, M. J. and Ziv, J.: Upper Bounds on the Probability of Sequences Emitted by Finite-State Sources and on the Redundancy of the Lempel-Ziv Algorithm, *IEEE Trans. on Inform. Theory*, Vol. 38, No. 1, pp. 66-72 (Jan. 1992).

37) Yokoo, H.: Improved Variations Relating the Ziv-Lempel and Weltch-Type Algorithms for Sequential Data Compression, *IEEE Trans. on*

Inform. Theory, Vol. 38, No. 1, pp. 73-81 (Jan. 1992).
38) 伊瀬，田中：Ziv-Lempel 符号の性能向上に関する一考察，信学会論文誌，Vol. J 76-A, No. 3, pp. 473-479 (Mar. 1993).
39) 大峰，山本：複数の辞書によるユニバーサルデータ圧縮の改良法について，信学会論文誌（掲載予定）.

(平成5年11月1日受付)



山本 博資

1952 年生．1975 年静岡大学工学部電気工学科卒業．1980 年東京大学大学院工学系研究科電気工学専門課程博士課程修了．工学博士．同年徳島大学工学部助手．同講師，助教授を経て，1987 年電気通信大学電気通信学部助教授，1993 年東京大学工学部助教授，現在に至る．情報理論（Shannon 理論，多端子情報理論，データ圧縮アルゴリズムなど），通信理論，暗号理論などの研究に従事．IEEE，電子情報通信学会，情報理論とその応用学会，応用数理学会各会員．