コンピュータビジョン 90－3
（1994．9．20）

適応的データ圧縮の最近の技法

横尾 英俊
群馬大学工学部情報工学科

データ圧縮とは，各データに含まれる冗長性を除去することによって情報通信や記憶装置使用の効率を向上させる技術である．本稿では，近年急速に開発の進んだデータ圧縮技術の中でも特に基礎的と考えられる離散データの歪のない圧縮について解説する．データの構造についての何らかの学習と符号化とを1走査の中で並行に行う適応的なデータ圧縮法の基本的考え方について述べ，それを実現する符号化法の例として算術符号とZiv-Lempel符号を紹介する．また，Ziv-Lempel符号に内在するデータの学習機能を検討することでデータ圧縮のためのモデリングの基本的問題が明らかになる．

Recent Trends in Adaptive Data Compression

Hidetoshi Yokoo

Department of Computer Science
Gunma University, Kiryu, Gunma 376, Japan

The aim of data compression is to reduce redundancy in stored or transmitted data in order to increase effective data density. This paper surveys several adaptive data compression techniques, which can adapt to unexpected data by a single sequential pass. Basic ideas of arithmetic and Ziv-Lempel codes are given. Arithmetic coding encourages a separation between the model for representing data and the encoding of the data with respect to that model. A pair of fundamental problems on the construction of models are revealed by investigating the underlying model construction mechanism of Ziv-Lempel coding.

1. はじめに

大規模な情報の通信や格納の効率化に関わる諸問題の最も直接的な解決法は，それらの基盤となるハードウェア技術を発展させることである．しかし，そのようなハードウェア技術の著しい進歩から見れば逆説的に，情報通信や記憶装置使用の効率をソフトウェアによって相対的に向上させようとする技術，すなわちデータ圧縮技術がますます重要視されるようになってきている．その背景には，大規模なデータの処理が可能となればなるほど更に大規模なデータの処理の要求が生ずること，また一方では，そのような要求に対処するためのデータ圧縮の基本概念の整備が急速に進んだことがあげられる．本稿では，そのような基本概念の中でも特に基礎的と考えられる離散データの歪のない圧縮 (lossless compression) の最近の技法と課題について解説する[1]．

離散データの歪のない圧縮（今後は単に「データ圧縮」と呼ぶ）の最も基本的な手法は，出現頻度の高い記号に短いビットを割り当て，頻度の低い記号により長いビットを割り当てるものである．本稿全体を通じて，データを構成する記号の集合（これを「アルファベット」という）を

$$A = \{a_0, a_1, \ldots, a_{M-1}\}$$

で表す．アルファベット A の各記号の頻度が

$$p_i = \Pr[a_i], \quad i = 0, \ldots, M-1$$

としてあらかじめ与えられているなら，それらは平均

$$H = -\sum_{i=0}^{M-1} p_i \log_2 p_i \qquad (1)$$

ビット未満では符号化できないこと，および，限界 H にいくらでも近いデータ圧縮が可能であることが知られている[2]．実際のデータがこのような定式化に常に合致するとすれば，データ圧縮研究に残された本質的課題は何もないことになる．しかし，このような前提は，次のような点から現実的ではない．

1. A 上の確率分布 p_i が既知であるとは限らない．
2. p_i が一定値とはならず，時間や前後の記号に依存するのが普通である．
3. 既存の方法が計算量の上でも十分効率的であるとは限らない．

このほかにも，アルファベットの大きさ M をあらかじめ固定できないような現実の状況などもあろう．データ圧縮法の最近の研究は，対象とするデータの種類に応じた様々なものがあっても，それらの多くは，これらの問題をそれぞれ個々に，あるいはいくつかを同時に解決しようとする点で共通している．仮に問題1が解決できているとするなら，そこで得られた確率分布に対する圧縮法だけを議論すればよいことになる．そのような圧縮法を**静的** (static) 符号化という．一方，実際の確率分布を知るためにデータを圧縮以前に走査し，その結果と静的符号化とを組み合わせる場合もあろう．これは，**準静的** (semi-static) 符号化などと呼ばれる．これに対し本稿では，主として**適応的** (adaptive) な手法について議論する．これは，データの構造についての何らかの学習と符号化とを1走査の中で並行に行うものである [8]．より理論的な議論では，これらの概念に明確な定義を与えることが必要になる場合もある．しかし，本稿では**適応的データ圧縮法**をこの程度の意味で用いる．また，本稿ではほとんど言及しない，静的な方法と適応的な方法との関係の一般的な議論は [3], [7] などで行われている．

2. 適応的データ圧縮法の潮流

最近のデータ圧縮法の多くは，実用化されているものも研究レベルのものも，何らかの意味で算術符号と Ziv-Lempel 符号と呼ばれる手法に関連している．両者は1970年代後半にほぼ同時に出現したインパクトの大きいものだっただけに，それ以前のデータ圧縮法は様々なヒューリスティクスの組み合わせにすぎず，確たる理論に基づくものはなかったとする見方もある [2]．しかし，両者とも手法的には決して斬新というわけではなく，算術符号は Shannon の符号化法に深く関係し，Ziv-Lempel 符号も直観的にはごく自明な手法を採用したものである．むしろ，この両者は，理論的にも実際にも“安心して”使える道具であることの根拠を明確にすることによって，その後のデータ圧縮研究を加速したという意義が大きい．そのような点を具体的に述べるために，データ圧縮の基礎について多少の準備を行う．

まず，前節での議論を踏襲して，アルファベット A の各記号 a_i を

$$B = \{0, 1\}$$

上の記号列に対応させてデータ圧縮を実現する場合を考える．このときの B 上の記号列を**符号語**，A の各記号に対応するすべての符号語の集合を**符号**と呼ぶ．B を2元に固定する必要はないが，本論文では簡単のために2元符号だけを考える．この場合，a_i を表す符号語の長さを $L(i)$ とすると，その単位はビット (bit) となる．また，これ以降，log の底を2とする．このような前提でのデータ圧縮は，$L(i)$ をいかに小さくするかという問題に帰着できる．歪のない圧縮では，符号化された記号列からもとのデータが一意的に復元できなければならない．これを一意復号可能性と呼び，そのための $L(i)$ の満たすべき条件として，クラフトの不等式 (Kraft's inequality)

$$\sum_{i=0}^{M-1} 2^{-L(i)} \leq 1$$

が知られている．クラフトの不等式を制約条件として平均符号長の最小値を求めたものが前節のエント

[1] より一般のデータ圧縮技術の中での「離散データの歪のない圧縮」の位置づけについては [5] などで知ることができる．

[2] H はエントロピー (entropy) と呼ばれる．データ圧縮法の圧縮能力は，A の各記号に対する平均の符号長で評価することが多い．平均符号長をエントロピーの近くまで短くするためのデータ圧縮法をエントロピー符号化という．エントロピー符号化の代表例としてはハフマン符号 (Huffman code) がよく知られている．エントロピーやハフマン符号などのデータ圧縮の情報理論的基礎については情報理論の教科書を参照されたい．

ロピー H である．前節の前提が現実的でない可能性として，前節では1から3を指摘した．しかし，より基本的な問題点として，“確率”という言語を用いないデータ圧縮法の存在を指摘することも考えられる．実際，4節で述べるZiv-Lempel符号はデータの確率的構造を全く前提としていない．しかし，そのような場合であっても，確率的構造を暗に前提としていることを指摘することができる．このことをみるために，歪のない圧縮の可能な任意のデータ圧縮法を C とし，A 上の記号列 s に対する C の符号長を $L_C(s)$ で表す．データ圧縮法 C が歪なく圧縮するものである限り，$L_C(s)$ に対しても次のようにクラフトの不等式が成立する．

$$\sum_{s\in A^l} 2^{-L_C(s)} \leq 1.$$

ここで，A^l は A の元を l 個接続した記号列すべての集合である．この式は

$$p(s) = 2^{-L_C(s)} \tag{2}$$

とおくと，

$$\sum_{s\in A^l} p(s) \leq 1$$

と表記できる．また，式(2)は

$$L_C(s) = -\log p(s)$$

と等価である．式(1)からもわかるように，s が“確率”$p(s)$ を有するなら，その最適符号長は $-\log p(s)$ である．一方，式(2)の $p(s)$ はその和が1を越えないことから確率としての性質を満たすものになっている[3]．すなわち，データ圧縮法 C はデータ s の確率を式(2)で与えられる値とみて符号化していることになる．このような考察から，確率モデルに基づいてデータ圧縮を行うことと，任意のデータ圧縮法にその確率的構造を与えることには差がないことがわかる．また，A 上の記号列 s に記号 a_i を接続した記号列 sa_i に要求されるクラフトの不等式

$$\sum_{s\in A^l}\sum_{a_i\in A} 2^{-L_C(sa_i)} \leq 1$$

は，次のような意味での単調性

$$L_C(s) < L_C(sa_i) \quad \text{for any } s \in A^l \text{ and } a_i \in A$$

を持つ適応的符号化法を考える限り，

$$p(a_i|s) = 2^{L_c(s)-L_c(sa_i)}$$

によって決まる“条件つき確率”の想定を意味する．つまり，適応的データ圧縮法によってデータを1記号ごと，あるいは，ある長さのブロックごとにデータ圧縮を行うことと，1記号ごとの確率を決めるモデル[4]を与えることは本質的に等価であることになる．ここで，“等価である”とは同一の符号長を達成するデータ圧縮が可能であるということにほかならない．しかし，各記号ごとの確率の列が与えられても，それに基づくデータ圧縮を実際にどう実現するかは，このような議論からだけでは生まれてこない．これに対する答を与えるのが算術符号である．

算術符号 (arithmetic codes) は，ハフマン符号などと異なり，ある特定の符号化法に対する呼び名ではなく，次節で述べるような性質を持つ符号化法の総称である．1976年にPascoとRissanenがそれぞれ独立に提案したものが最初の算術符号とされている．多くの符号が記号列（符号語）の接続を基礎としているのに対し，算術符号は加算や乗算などの算術演算を基本操作としているため，このような呼び名がある．算術符号は，1記号あるいは1ビットごとにそれに対応する確率の値が生成できるようなモデルがあれば，そのモデルで原理的に可能な最適のデータ圧縮に任意に近い精度の圧縮を行うことができる．しかも，そのようなモデルが，たとえば，同一の記号 a_i の出現確率が時刻やそれ以前の記号に応じて変化するものであっても，算術符号はその変化に追従することができる．そのような意味で算術符号は万能であり，算術符号を利用してデータ圧縮を行う限り，データ圧縮は，データの各記号の確率を生成するモデルの構成に帰着される．しかし，万能であるだけに，また，算術演算を基礎としているだけに，高速性を要求される場合には不向きな場合が多い．あるいは，もっと“手軽”な方法が求められる場合も当然考えられる．

そのような要求に対し，特にテキスト圧縮で強力な性能を示すのが **Ziv-Lempel 符号**である．この符号は，1977年頃から発表されたZivとLempelの2名による一連の研究に基礎を置くものである．その後，数々の改良や変形が発表され，今や算術符号と同様，特定の符号化法というより多数の変形算法を含むデータ圧縮法のひとつのクラスに対する名前となっている．また，このクラスに含まれる特定の方法に言及する場合，二人の頭文字をアルファベット順にした“LZ”を冠して呼ぶことが多い[2]．Ziv-Lempel符号は，直観的には，記号列の反復によるデータの冗長性を除去することで圧縮を実現するものになっている．少なくとも算法の上では，上でふれたように，データの確率的構造を利用することは全くない．つまり確率的構造の未知のデータに対しても使用することが可能であり，データのそのような予備知識を前提としないという意味でユニバーサル符号 (universal codes) と呼ばれる．ZivとLempelの最大の功績のひとつは，そのような意味で確率的構造とは一見無縁な方法を独自に定式化し，その性能の記述に成功したという点にある．しかし，その後は実用化に向けた算法の改良が相次ぎ，しかも，極めて強力な方法の発表が続いている．その結果，現在ファイル圧縮のために実用化されている方法のほとんどは，このZiv-Lempel符号に基づくものとなっている．

[3] 確率和（クラフトの不等式では“クラフト和”ということがある）が必ずしも1に一致しない(これを“劣確率分布”という)のは，冗長性が残っている場合である．データ圧縮の立場から言えば，圧縮法 C にむだがあるためであり，確率の言葉で言えば，“余分”な要素に確率を割り当てているためである．

[4] このモデルを“symbolwise model”という．もっとミクロには，各ビットごとに確率を与えるモデルであってもよい．これを特に“bitwise model”と呼ぶことがある．

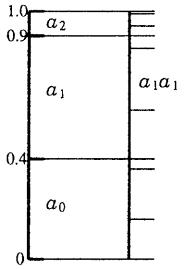


図 1: 算術符号のための区間分割の例.

表 1: 算術符号のための区間分割の例.

現区間	部分区間への分割 a_0	a_1	a_2	入力記号
$[0.0000, 1.0000)$	$[0.0000, 0.4000)$	$[0.4000, 0.9000)$	$[0.9000, 1.0000)$	a_1
$[0.4000, 0.9000)$	$[0.4000, 0.6000)$	$[0.6000, 0.8500)$	$[0.8500, 0.9000)$	a_1
$[0.6000, 0.8500)$	$[0.6000, 0.7000)$	$[0.7000, 0.8250)$	$[0.8250, 0.8500)$	a_1
$[0.7000, 0.8250)$	$[0.7000, 0.7500)$	$[0.7500, 0.8125)$	$[0.8125, 0.8250)$	a_2
$[0.8125, 0.8250)$				

以上がデータ圧縮研究の主たる流れである．ここで，現在を含む今後の基礎的研究の潜在的可能性を整理すると，次のように分類できる．まず第 1 は，算術符号でも Ziv-Lempel 符号でもない基本的に新しい方法を作ることである．もちろんある程度の性能が要求されるので，決して容易な課題とは言えない．第 2 は，それぞれのクラスの中での問題の解決や改良である．たとえば，“よい算術符号” を設計することである．第 3 は，算術符号で必要とするようなモデルについての研究，つまりモデル器の設計である．これは応用に依存する可能性の最も強い分野であり，また，データ圧縮という目的にとどまらない意義を有するものである．以下では，まず，上の第 2 の方向に注意しながら個々の手法の簡単な解説を行ったあと，Ziv-Lempel 符号でも確率的構造とは実は無縁ではないということを材料にして，モデリングに関係した基本的問題について述べる．

3. 算術符号

算術符号の原理そのものは極めて単純であるが，それを細部まで理解して完全にインプリメントすることは必ずしも容易ではない．

算術符号は，入力データ系列 $x_1x_2\ldots$, に対応する確率の値の列 $\Pr[x_1], \Pr[x_2], \ldots$, を数直線上の区間に対応させるものである [2]．このとき，$\Pr[x_1]$ などの値は予め与えられた一定値であってもよいし，また，データの過去の部分から推定した (したがって時間とともに変化する) ものであってもよい．確率を区間へ対応させること自体は極めて単純である．たとえば，$M = 3$ の場合のアルファベットとその上の確率分布の対が時間等に依存しない一定値として

$$(p_0,\ p_1,\ p_2) = (0.4,\ 0.5,\ 0.1)$$

と与えられたとする．これに基づき，図 1 の左端に示すように，区間 $[0, 1)$ をこれらの確率の比で分割する．その結果の部分区間をさらに同様の割合で分割する．この場合，データが a_1a_1 であれば，下から 5 番目の区間 $[0.6, 0.85)$ がそれに対応する．これを一般的に述べるために，確率の累積値を表す次の二つの量を使うことにする．

$$R_i = \sum_{j=0}^{i} p_j, \quad Q_i = \sum_{j=0}^{i-1} p_j, \quad i = 0, \ldots, M-1.$$

ここで，p_j は符号化しようとする時刻における記号 a_j の出現確率である．区間の初期値を $[L, H) := [0, 1)$ とし，データから 1 記号 a_i を入力するたびに区間の端点を次のように更新する．

$$\begin{aligned} H &:= L + R_i(H - L), \\ L &:= L + Q_i(H - L). \end{aligned}$$

これを反復して最後の入力記号に対する区間が決まったなら，その区間に含まれる数 (の 2 進表示の小数点以下の部分) を一つ，他の区間と区別可能な最小の長さで出力する．上記の例を入力 $a_1a_1a_1a_2$ に対して続けたものを表 1 に示す ([6] の例による)．この場合の最終的な部分区間 $[0.8125,\ 0.825)$ を 2 進数で表示すると，$[0.1101,\ 0.11010101\ldots)$ となる．この区間から，符号語としては，たとえば，**1101000** を選ぶことができる．

さて，いまの例の最終的な区間幅 0.0125 は入力記号の確率の積 $0.5^3 \times 0.1$ に当然一致する．この確率の積を p と書くと，最終的な部分区間では，$H = L + p$ が成立している．このとき，区間 $[L, H)$ を特定するのに必要な小数点以下の (2 進での) 桁数は $-\log p$ 程度である．これは前節で述べた，確率 p の事象に対する最適な符号長に一致する．これが，算術符号でデータ圧縮が可能であることの直観的理由である．しかし，上の手順を実際の符号化法として使うにはいくつかの問題がある．まず，区間幅の計算を任意の長さの入力系列に対して継続することは，任意に高い精度の計算の必要性を意味し，実際には不可能である．さらに，最終的に得られた区間を符号語に変換するため，全ての入力系列の入力を終了するまでは圧縮結果を全く出力することができない．そこで，これらの問題に対処するため，次のような手続きを考えてみる．この手続きは，1 記号 a_i を入力したなら上のようにして新しい区間を計算し，その結果得られた区間に対して適用するものである．

1. 得られた部分区間 (現区間) が $[0,\ 1/2)$, $[1/4,\ 3/4)$, $[1/2,\ 1)$ のいずれかに含まれるならば以下に進み，そうでなければこの手続きを終了する．
2. 現区間が $[0, 1/2)$ の部分区間であるなら，0 を出力し，それに続けて 1 を f ビット出力する．さらに，区間幅を次のように 2 倍に拡大したあと上の 1 に戻る．

$$H := 2H, \quad L := 2L.$$

表 2: 算術符号による符号化の例.

現区間	出力など	部分区間 a_0	部分区間 a_1	部分区間 a_2	入力記号
$[0.00, 1.00)$	分割	$[0.00, 0.40)$	$[0.40, 0.90)$	$[0.90, 1.00)$	a_1
$[0.40, 0.90)$	分割	$[0.40, 0.60)$	$[0.60, 0.85)$	$[0.85, 0.90)$	a_1
$[0.60, 0.85)$	**1** を出力; 区間幅を拡大				
$[0.20, 0.70)$	分割	$[0.20, 0.40)$	$[0.40, 0.65)$	$[0.65, 0.70)$	a_1
$[0.40, 0.65)$	*follow*++; 区間幅を拡大				
$[0.30, 0.80)$	分割	$[0.30, 0.50)$	$[0.50, 0.75)$	$[0.75, 0.80)$	a_2
$[0.75, 0.80)$	**1** を出力; **0** を $f=1$ ビット出力; 区間幅を拡大				
$[0.50, 0.60)$	**1** を出力; 区間幅を拡大				
$[0.00, 0.20)$	**0** を出力; 区間幅を拡大				
$[0.00, 0.40)$	**0** を出力; 区間幅を拡大				
$[0.00, 0.80)$	**0** を出力				

3. 現区間が $[1/2, 1)$ の部分区間であるなら，1 を出力し，それに続けて 0 を f ビット出力する．さらに，区間幅を次のように 2 倍に拡大したあと上の 1 に戻る．

   $$H := 2H - 1, \quad L := 2L - 1.$$

4. 現区間が $[1/4, 3/4)$ の部分区間であるなら，変数 *follow* の値を 1 だけ増す．さらに，区間幅を次のように 2 倍に拡大したあと上の 1 に戻る．

   $$H := 2H - 1/2, \quad L := 2L - 1/2.$$

上の手続きの 2, 3 に含まれる f の値は，それ以前の入力記号に対してこの手続きを適用したときに設定された変数 *follow* の値である．なお，この手続きを開始するときの *follow* の値は 0 に設定しておく．上の手続きは，これらの変数の扱いを別にすれば非常に単純である．すなわち，初めに述べた算術符号化法では区間幅が狭くなる一方であったのを，区間が $[0, 1/2)$ または $[1/2, 1)$ に含まれることが判明するたびに 1 ビットを出力し，区間幅を 2 倍にするようにしたものである．したがって，問題は，これらの両区間に含まれることが判明していないときの処理にある．この処理を担うのが *follow* などの変数である．その働きを具体的に示すために，先の例がいまの方法でどのように符号化されるかを表 2 に示す．表 2 についての詳しい説明は省略するが，表 1 の場合と同じ出力結果が得られるのに対し，区間を表す桁数がそれほど増加しない様子が観察できる．しかし，これでも実用的な符号化法として十分なわけではない．桁数が増加しないとはいえ，その限界が明らかではないので，有限精度の計算で確実な符号化を保証するためには更に工夫が必要となる．また，区間幅の計算に必要となる乗算も，場合によっては，より高速な演算で代用しなければならないこともある．その結果，そのような問題を解決する算術符号の変種がいくつも提案され，その応用例も着実に増え続けている．なお，以上の考えに基づく比較的利用しやすい算術符号のプログラムが [11], [2] にある．

4. Ziv-Lempel 符号

Ziv-Lempel 符号については既に優れた解説があるので [2],[12]，ここでは，次節の議論に必要な最低限の算法と “よい Ziv-Lempel 符号” を設計するということの一般論だけ述べる．

ここで述べるのは，Ziv-Lempel 符号の中でも LZ78[16] と呼ばれるもので，UNIX の標準的な圧縮ツールである compress の基礎となっている算法である．まず，データ系列が十分長いものとして，第 n 番目の記号までの系列 $x_1x_2...\ldots x_n$, $x_i \in A$ を $k+1$ 個の部分列 $X_1X_2\ldots X_{k+1}$ に次のように分解する (これ以降，同一の記号を異なる意味で用いることが多いので注意されたい)．

1. $X_1 = x_1$ を第 1 番目の部分列とする．
2. 第 j 番目の部分列 X_j は最後の X_{k+1} を除き，それ以前のどの部分列とも異なるようにする．つまり，$X_j \notin \{X_1, \ldots, X_{j-1}\}$, $j = 2, \ldots, k$.
3. X_j はその最後の 1 記号を除くと，空系列かあるいは過去のある部分列に一致する．すなわち，

   $$X_j = X_q a, \ a \in A$$

   となる整数 q が存在する．ただし，$q \in \{0, 1, \ldots, j-1\}$, X_0 は空系列 λ である．

$A = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\}$ に対する分解の例をあげる．

$$\mathtt{a} \cdot \mathtt{b} \cdot \mathtt{ac} \cdot \mathtt{bd} \cdot \mathtt{ab} \cdot \mathtt{bdc} \cdot \mathtt{a}$$

この例では，$k = 6$ である．このような系列の分解を増分分解 (incremental parsing) と呼ぶ．最初の部分列の候補は A の要素であり，アルファベットの大きさ M だけの可能性がある．次に，この例で $X_1 = \mathtt{a}$ が確定したあとの第 2 番目の部分列の候補は，a を除く A の要素か，a に A の要素を接続したもののいずれかである．つまり，$M - 1 + M$ 通りの可能性がある．一般に，第 j 番目の部分列 X_j の候補は，$j = k+1$ の場合を除き $(M-1)j+1$ 個存在する．それらの候補のうち，実際に出現したものを特定するのに必要な記述の長さは

$$\log((M-1)j + 1) \tag{3}$$

ビットである．

LZ78 の圧縮力を説明するこれ以降の議論は本稿の範囲を越えるので，原論文 [16] あるいは [4] などを参照されたい．それらで明らかにされていることは，LZ78 の理論的な性能，特に漸近的最良性と呼ばれる非常に強力な結果である．しかし，そのような性質にもかかわらず，有限長のデータに対する実用的な性能は極めて不十分で，多くの改良を生む原因となっている．改良を一般的に考えるとき，LZ78 を特別な場合として含む，次のような Ziv-Lempel 符号のより広い記述が指針となる．系列の分解による各部分列 X_j は，より一般的には，同一データの過去のある部分列 Y_q と誤差 $\varepsilon(X_j, Y_q)$ とを使って

$$X_j = \langle Y_q, \varepsilon(X_j, Y_q) \rangle$$

と記述することができる．ここで，LZ78 の手順 3 の X_q に換えて Y_q と表記したのは，過去の部分列 Y_q を分解の結果得られた部分列以外からも選びうることを示す．この部分列 Y_q を長くするだけなら，問題は単なる系列の照合問題となる．それに加えて，X_j の記述長

$$||\langle Y_q, \varepsilon(X_j, Y_q)\rangle|| \leq ||Y_q|| + ||\varepsilon(X_j, Y_q)|| \qquad (4)$$

を短くしなければならない点が一層重要である．ここで，不等式 (4) は，Y_q の記述長 $||Y_q||$ と誤差の記述長 $||\varepsilon(X_j, Y_q)||$ との和よりも $||\langle Y_q, \varepsilon(X_j, Y_q)\rangle||$ が長くなることはないという一般的事実を示したにすぎない．したがって，“よい Ziv-Lempel 符号” の設計は局所的には[5]

$$\frac{||\langle Y_q,\ \varepsilon(X_j, Y_q)\rangle||}{|X_j|}$$

をできるだけ小さくするような

1. X_j, Y_q の決定法，
2. $\langle Y_q, \varepsilon(X_j, Y_q)\rangle$ の実際の記述法

の設計に帰着できる．しかも，実際のデータ圧縮法としては，これらを高速に実現できるものでなければならない．たとえば，項目 2 の実際の符号化法を効率的なものにしようとすると，実現が現実的に難しくなることが多い．本来の Ziv-Lempel 符号をはじめ多くの変形算法でも，この実際の符号語の記述法に明らかな冗長性を残しているものが少なくない．高速性と冗長性とのトレードオフを考慮しながら，上記の 1 と 2 とに具体的な方法を与えることが今後も必要とされる課題である．なお，Ziv-Lempel 符号の実用算法として流通しているものの中で最も強力と思われるものに *gzip* がある．この方法は，本稿の分類では，適応的符号化というより準静的な側面の強い方法である [10].

5. モデリング

1 節で簡単にふれたように，確率分布を知るためにデータを圧縮以前に走査し，その結果を利用してデータを再び走査して圧縮する手法を準静的符号化と呼ぶ．アルファベット A の各記号の頻度 p_i, $i = 0, \ldots, M-1$ だけを推定して圧縮に利用する場合，準静的符号化では p_i の推定値 $\widehat{p_i}$ として最尤推定量を使うべきことが簡単に示せる．すなわち，n 記号から成るデータに含まれる記号 a_i の出現回数 c_i を測定したなら，

$$\widehat{p_i} = \frac{c_i}{n}, \quad i = 0, \ldots, M-1$$

として符号化すればよい[6]．このとき，準静的符号化では，$\widehat{p_i}$ の値，あるいはそれに基づいて構成した符号そのものも符号化結果に含めなければならない．しかし，本稿ではこの点は問題にしない．より問題となるのは，適応的符号化では最尤推定量を使うことが必ずしも適当ではないという点である．たとえば，最初の 1 記号を符号化したあと 2 番目の記号の符号化に最尤推定量を使うということは，それが最初の記号に 100% 一致すると期待することを意味し，明らかに適切ではない[7]．そこで，最尤推定量に代えて，データの n 記号までの符号化が既に済んでいる場合に次の $n+1$ 番目の符号長の期待値を最小にする推定量を使うことにする．そのために，まず，過去の n 記号に含まれる記号 a_i の出現回数をそれぞれ c_i とする．すなわち，$n = c_0 + \cdots + c_{M-1}$ である．ここで，

$$\begin{aligned} \boldsymbol{c} &= (c_0, \ldots, c_{M-1}), \\ \boldsymbol{p} &= (p_0, \ldots, p_{M-1}) \end{aligned}$$

とおく．過去の n 記号に対する実際の記号の出現回数 $\boldsymbol{c}$ が得られている場合に次の $n+1$ 番目の記号を推定値 $(\widehat{p_i})$ によって符号化した場合の符号長の期待値は

$$\mathrm{E}[L(n, \boldsymbol{c})] = -\int \sum_{i=0}^{M-1} p(\boldsymbol{p}|\boldsymbol{c}) p_i \log \widehat{p_i} d\boldsymbol{p} \qquad (5)$$

と表される．ここで，$p(\boldsymbol{p}|\boldsymbol{c})$ は $\boldsymbol{c}$ に対する $\boldsymbol{p}$ の条件つき確率であり，積分は M 次元空間中の

$$p_0 + \cdots + p_{M-1} = 1, \quad 0 \leq p_i \leq 1$$

を満たす範囲で行う．ベイズの定理に基づく計算を行うと，式 (5) は

$$-\frac{\displaystyle\sum_{i=0}^{M-1} \int p_0^{c_0} \cdots p_{M-1}^{c_{M-1}} p(\boldsymbol{p}) p_i \log \widehat{p_i} d\boldsymbol{p}}{\displaystyle\int p_0^{c_0} \cdots p_{M-1}^{c_{M-1}} p(\boldsymbol{p}) d\boldsymbol{p}} \qquad (6)$$

となる．続いて，$\boldsymbol{p}$ の事前分布 $p(\boldsymbol{p})$ に一様分布を仮定し，平均符号長 $\mathrm{E}[L(n, \boldsymbol{c})]$ を最小にする推定値を求めると

$$\widehat{p_i} = \frac{c_i + 1}{n + M} \qquad (7)$$

が得られる．これをラプラスの推定量と呼ぶ．特に，$M = 2$ の場合

$$\widehat{p_i} = \frac{c_i + 1}{n + 2}, \quad i = 0, 1 \qquad (8)$$

だけをラプラスの推定量と呼ぶ場合も多く，式 (8) は種々の適応的データ圧縮法で利用されている．

ラプラスの推定量は 1 節で指摘した問題点の 1 に対する一つの解答例になっている．そのような推定量の使用を前提として，次に第 2 の問題点を検討して

[5] X_j のもともとの長さを $|X_j|$ で示す．このような局所的な圧縮比 (compression ratio) を最適にする貪欲算法がデータ系列全体に対しても最適であるという保証はない．しかし，そのような大域的に最適な方法を適応的な方法で実現するのは困難であり，また実際的には，局所的に高性能な方法は大域的にも性能が良いのが普通である．

[6] この点で準静的符号化は静的符号化に類似している．前者を準適応的と呼ばない理由のひとつがここにある．

[7] 逆に言えば，それ以外の記号の出現確率の推定値を 0 とすることになる．確率 0 の記号に対する最適な符号長は $-\log 0 = +\infty$ であり，意味のある符号化ができない．一般に，過去に実際に観察されたことのない事象，すなわち零頻度の事象にどの程度の推定確率を割り当てるかという問題を零頻度問題 (zero-frequency problem[9]) という．

みる．特に，各記号の出現確率がその直前のデータに依存して変化するような場合（この代表例としてマルコフ情報源からの出力がある）を考える．この場合，次の記号の出現確率の推定のためには，記号 a_i の単純な出現回数だけでなく，直前の記号（列）で条件づけた回数を測定することが必要となる．記号列 s に続いて記号 a が出現した回数を $c(a|s)$ で表す．ここで一般論の前に，前節で例をあげた Ziv-Lempel 符号が最初に予告したように確率的構造の推定とは無縁でないことをまず示す．前節の増分分解を使って分割が済んだばかりの長さ l の部分列を

$$x_1 x_2 \cdots x_l, \quad x_i \in A$$

で表す．このとき，記号 a の出現回数 $c(a|\lambda)$ の測定のために x_1 だけを利用することにする．すなわち，$c(x_1|\lambda)$ を 1 だけ増すことにする[8]．同様に，x_1 の直後の記号の出現回数の測定に x_2 を利用し，$c(x_2|x_1)$ の値を 1 増す．一般に，$c(x_i|x_1 \cdots x_{i-1})$ に 1 を加えるものとする $(i = 1, 2, \ldots, l)$．すると，$j-1$ 個の部分列の処理が済んだ時点で

$$\sum_{a_i \in A} c(a_i|\lambda) = j - 1, \tag{9}$$

および，$c(a|s) > 0$ を満たす A 上の任意の記号 a と記号列 s に対し

$$c(a|s) = 1 + \sum_{a_i \in A} c(a_i|sa) \tag{10}$$

が成立する．これらの量を使って，次の第 j 番目の記号列に対し，以下の操作を $c(a|s) = 0$ となるまで繰り返す．ただし，記号列の初期値を $s = \lambda$ とする．

1. 次の記号を a とする．
2. 記号 a を推定確率
$$\frac{c(a|s) + (M-1)^{-1}}{\sum_{a_i \in A} c(a_i|s) + M(M-1)^{-1}} \tag{11}$$
で符号化する．
3. $c(a|s) = 0$ であれば反復を終了．
4. 記号列 sa をあらたに s とする．

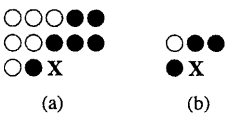
ここで，符号化中の第 j 番目の部分列を $x_1 x_2 \cdots x_l$ とすると，それに対してこのモデルが想定している符号長は

$$\begin{aligned} &-\log \frac{c(x_1|\lambda) + (M-1)^{-1}}{\sum_{a_i \in A} c(a_i|\lambda) + M(M-1)^{-1}} \\ &-\log \frac{c(x_2|x_1) + (M-1)^{-1}}{\sum_{a_i \in A} c(a_i|x_1) + M(M-1)^{-1}} - \cdots \\ &-\log \frac{(M-1)^{-1}}{\sum_{a_i \in A} c(a_i|x_1 \cdots x_{l-1}) + M(M-1)^{-1}} \\ = \ &\log((M-1)j + 1) \end{aligned}$$

となる．なお，式 (9) および (10) の関係を用いた．この長さは，増分分解による部分列の符号長 (3) に完全に一致する．すなわち，増分分解による符号化法はいま述べた 1 記号単位の符号化モデルと等価である．このモデルは，Ziv-Lempel 符号がデータの確率的な構造を推定するものであることを示すばかりでなく，次のような問題を提起するものとして重要である．

1. Ziv-Lempel 符号では，次の 1 記号を式 (11) に示す推定量で符号化している．式 (11) とラプラスの推定量 (7) とを比較すると，両者は
$$\hat{p}_i = \frac{c_i + \rho}{n + M\rho} \tag{12}$$
と一般化できることが予想できる．式 (11) は上式で $\rho = (M-1)^{-1}$ としたもの，ラプラスの推定量は $\rho = 1$ としたものになっている．特に，$M = 2$ の場合に両者は一致し，$\rho \to 0$ の極限で式 (12) は最尤推定量となる．ラプラスの推定量の導出では，式 (6) で $p(\boldsymbol{p})$ に一様分布を仮定した．しかし，多くの現実のデータの圧縮を試みると，この仮定が必ずしも現実には妥当でないことが観察できる．式 (12) の推定量で言えば，経験的には ρ を 1 より小さな値に取る方が適切な場合が多い．
2. 増分分解による部分列 $x_1 x_2 \cdots x_l$ に含まれる第 i 番目の記号 x_i は，$s = x_1 \cdots x_{i-1}$ によって条件づけた推定確率 (11) で符号化されている．記号 x_i の出現確率を条件づける直前の記号列を文脈と呼ぶと，Ziv-Lempel 符号が採用している $s = x_1 \cdots x_{i-1}$ という文脈が "最適" であるという保証はない．

上記の問題点 2 を別の視点から見るために，白黒画像の歪のない圧縮を考えてみる．白黒画像の各画素をラスタ走査によって適応的にデータ圧縮し，図 2 に示す画素 X の符号化が問題になっているとする．仮に，図 2(a) に示すものと同一の画素値の周辺画素が過去に 10 回出現し，そのいずれの場合も対応する X の位置の画素値が黒であったとする．また，(a) の一部に当たる (b) に示す画素だけを近傍画素として着目すると，過去に同一の画素値の組み合わせの出現が 100 回あり，そのうち X に相当する画素値が黒となった場合が 64 回，白となった場合が 36 回あったとしよう．このとき，いま符号化しようとしている画素 X のための参照画素として，(a)，(b)，あるいはそれ以外の周辺画素の組み合わせのうち，いずれを文脈として採用するのが適切かという問題が生じる [14]．もし，(a) を採用すれば黒を非常に高い確率で



(a) (b)

図 2: 画素値 X の符号化のための文脈（参照すべき近傍画素）の例．

[8] $c(a|\lambda)$ の測定に x_1 だけでなくその他の記号も利用するようにし，それに応じて，ほかの条件つき出現回数の測定頻度も増すようにすると，Ziv-Lempel 符号の変形算法 [15] が得られる．

期待することになってXの符号長が短くなると予想できるが，過去に10回しか出現していない(a)による推定は，過去に100回観察できた(b)ほどの推定精度はないはずである．つまり，文脈を大きく(あるいは「長く」)すればするほど次の事象は予測しやすくなる（エントロピーが減少する）が，その分，そのような推定の精度は劣化することになる．エントロピー（の推定値）と推定精度とのトレードオフ[9]を考慮した文脈選択の問題がこうして生じることになる．

文脈選択の具体的な手法については既にいくつもの方法が提案されている．しかし，それらは実用的なデータ圧縮のための発見的な方法がほとんどで(代表的なものにPPMモデル[1], [2]がある)，「文脈選択問題に対する解」と呼べるほどの一般的な方法は，筆者の知る限り，まだ存在していない．もちろん，そのような解が一般的に得られる可能性が低いことは，たとえば，近傍画素間の相関が低く，遠く離れた画素同士に強い相関のあるような画像を考えれば想像がつく．言語データの次の1文字を人間が予想する場合には，直前の数文字ばかりでなく，文法的知識をはじめとする様々な知識を使うのが普通であろう[10]．形式的には，種々の文脈に対して計数した頻度によって次の1記号の平均符号長(6)を求め，それを最小とする文脈を選択すればよさそうに思える．ラプラスの推定量の導出では，式(6)の$p(\boldsymbol{p})$に一様分布を仮定した．ところが，種々の文脈に対し式(6)を最小化することと，文脈によらずに$p(\boldsymbol{p})$に一様分布を仮定することは，実は整合性がない．したがって，$p(\boldsymbol{p})$にどのような分布を仮定すれば合理的かが明らかにされない限り，式(6)の最小化による文脈の選択はできないことになる．一方，$p(\boldsymbol{p})$に仮定すべき分布は，上述の問題点1に直接関係している．結局，上述の問題点の1と2とは相互に関連づけて解決すべきものであることがわかる．しかも両者は，単にZiv-Lempel符号のためのモデルの改良に関係するだけでなく，データ圧縮の基本的で重要な問題となっている．

6. おわりに

歪のないデータ圧縮には当然限界があるはずであり，そのような限界をある程度知ることによって，研究の正しい方向づけがなされるといえる．しかし，任意にデータを一つ与えたときに，それがどこまで圧縮可能であるかを推定することは一般には不可能である．一つの可能性としては，自然言語やプログラムなどの言語データの1文字を人間に推定させ，それによって決まる(人間による)圧縮の限界を推定値とすることが考えられる．そのような立場から，現在最も強力とされる方法(*gzip*やPPMなど)の圧縮力と古くから行われてきた英文のエントロピーの推定の結果とを比較すると，両者の間には英文字1記号あたり1ビット近い開きがある．これを大きな差だと考えれば，現在のデータ圧縮法はまだまだ未熟であり，今後に大きな発展の余地を残す分野だと言える．一方，この差を十分小さいとみなせば，主要課題を圧縮それ自身よりもほかの演算との組み合わせなどに移行すべきであると言える．たとえば，多くのデータを圧縮して格納しておく場合，必要なデータの検索をどのようにして効率化すべきかといった問題が生じる[10]．しかし，いずれの場合であっても，データ圧縮の基本である「離散データの歪のない圧縮」が重要であることに変わりはない．

[9] このようなトレードオフは，もっと一般的には，データを記述するモデルを複雑なものにすればデータを忠実に記述できるようになる一方で，モデル自身の記述長が増大するという事実として述べることができる．データを記述する複数のモデルの中で，モデル自身の記述を含むデータの記述長を最小とするモデルを採用しようとする考え方をMDL(Minimum Description Length)原理という[13]．式(4)のような定式化では，与えられたデータを近似するための記述の長さと誤差の記述長との総和を最小化するモデルといってもよい．

[10] MDL原理によるモデル選択では，選択すべきモデルの範囲をあるクラスに限定して，その中に真のモデルが含まれる場合にはそれを(漸近的に)学習する選択法を見い出すというアプローチを取るのが普通である．しかし，MDL原理を適応的な符号化法で利用する具体的方法については，未だ十分に検討されているとは言えず，今後の研究が強く期待される分野である．

参考文献

[1] T. C. Bell, J. G. Cleary, and I. H. Witten: Modeling for text compression. *ACM Computing Surveys*, 21, 4, pp.557–591 (1989).

[2] T. C. Bell, J. G. Cleary, and I. H. Witten: *Text Compression*. Prentice Hall (1990).

[3] J. G. Cleary and J. H. Witten: A comparison of enumerative and adaptive codes *IEEE Trans. Inf. Theory*, IT-30, 2, pp.306–315 (1984).

[4] T. M. Cover and J. T. Thomas: *Elements of Information Theory*. John Wiley & Sons (1991).

[5] E. A. Fox: Advances in interactive digital multimedia systems. *IEEE COMPUTER*, 24, 10, pp.9–21 (1991).

[6] P. G. Howard: The design and analysis of efficient lossless data compression systems. Ph.D. Thesis, Department of Computer Science, Brown University (1993).

[7] J. Rissanen and G. G. Langdon, Jr.: Universal modeling and coding. *IEEE Trans. Inf. Theory*, IT-27, 1, pp.12–23 (1981).

[8] R. N. Williams: *Adaptive Data Compression*, Kluwer Academic (1991).

[9] I. H. Witten and T. C. Bell: The zero-frequency problem: estimating the probabilities of novel events in adaptive text compression. *IEEE Trans. Inf. Theory*, 37, 4, pp.1085–1094 (1991).

[10] I. H. Witten, A. Moffat, and T. C. Bell: *Managing Gigabytes: Compressing and Indexing Documents and Images*. Van Nostrand Reinhold (1994).

[11] I. H. Witten, R. M. Neal, and J. G. Cleary: Arithmetic coding for data compression. *Commun. ACM*, 30, 6, pp.520–540 (1987).

[12] 山本博資: ユニバーサルデータ圧縮アルゴリズム: 原理と手法. 情報処理, 35, 7, pp.600–608 (1994).

[13] 山西健司, 韓 太舜: MDL入門: 情報理論の立場から. 人工知能学会誌, 7, 3, pp.45–52 (1992).

[14] H. Yokoo, K. Sudoh, and K. Uehara: Adaptive modeling and coding for the lossless compression of binary images. *Trans. IEICE*, E73, 10, pp.1640–1646 (1990).

[15] H. Yokoo: Improved variations relating the Ziv-Lempel and Welch-type algorithms for sequential data compression. *IEEE Trans. Inf. Theory*, 38, 1, pp.73–81 (1992).

[16] J. Ziv and A. Lempel: Compression of individual sequences via variable-rate coding. *IEEE Trans. Inf. Theory*, IT-24, 5, pp.530–536 (1978).