AVL 木を利用した適応的数値データ圧縮法とその改良

横尾英俊† 杉浦由紀江†☆

計算機内の数値データは，実数値とみなされる場合でも離散データの一種である．しかし，語長の短い整数型のデータでも16ビット以上あるのが普通であり，アルファベット・サイズの非常に大きな離散データと考えることができる．そのような大きなアルファベット上のデータを効率良く符号化するには，従来の一般的なデータ圧縮法では不十分で特別のアルゴリズムの開発が必要となる．本論文は，このような数値データの適応的圧縮の問題をデータ構造の視点から考察するものである．特に，平衡木の代表例であるAVL木の利用について議論を行い，AVL木のこのような問題向けの修正とその結果を利用した数値データの新しい圧縮法を提案する．提案する符号化法は，符号化したデータからもとのデータをひずみなく復元できるもので，対象データの統計的性質についての予備知識を仮定しないユニバーサルな符号化法になっている．こうような数値データ圧縮法は，計算機内の数値データファイルの圧縮のためばかりでなく，多くの応用可能性を有している．本論文では，各方法の性能の比較のために行った階調画像符号化への適用結果について報告する．

Application of AVL Trees to Adaptive Compression of Numerical Data

Hidetoshi Yokoo† and Yukie Sugiura†☆

Computer data of fixed-precision and floating-point numbers are *discrete* even though they may represent *real* numbers. However, they are quite different from purely discrete data in that the size of an alphabet defining such numerical data is very large and they can be thought of as realization of a random variable with smooth distribution. This paper discusses the adaptive compression of computer files of numerical data whose statistical properties are not given in advance. Specifically, two new algorithms utilizing AVL trees are proposed and the improvement of AVL trees for that purpose is considered. Comparisons them with existing algorithms are made mainly from the viewpoint of data structures. This paper includes also an application of these algorithms to the noiseless compression of gray-scale images in order to show wider applicability of the adaptive compression of numerical data.

1. はじめに

計算機内の数値データは，実数値とみなされる場合でも離散データの一種である．しかし，語長の短い整数型のデータでも16ビット以上あるのが普通であり，アルファベット・サイズの非常に大きな離散データと考えることができる．本論文では，このような数値データから成るファイルを効率良く符号化するための適応的データ圧縮法について検討する．

従来の適応的データ圧縮法には，動的 Huffman 符号[4]のように記号の発生頻度の偏りを利用するものとUNIX の compress コマンドとして実現されているLZW 法[7]のように記号列の反復を利用するものとがある．しかし，数値データは，そのアルファベットの大きさのため，ひとつのファイルに含まれるデータの重複が相対的に低頻度であるのが普通である．そのため，これらの従来の一般的なデータ圧縮法では十分な圧縮効果が得られず[2]，特別のアルゴリズムの開発が必要になる．最も単純な方法は，数値を上位ビットと下位ビットに分割して，上位ビットだけを上述の一般的な方法の対象とする方法である．この方法では，上位ビットと下位ビットとが恣意的に分割されることになり，分割点の妥当性を理論的に説明することが困難なことが多い．これに対し，本論文で対象とするのは，このような分割をデータに応じて適応的に行う，より理論的な手法である．これまで，このような適応的な方法として提案された手法は決して多くなく，Yokoo[9]や Itoh ら[2]の方法が存在する程度である．前

† 群馬大学工学部情報工学科
Department of Computer Science, Faculty of Engineering, Gunma University
☆ 現在 沖電気工業(株)
Presently with Oki Electric Industry Co., Ltd.

者の Yokoo[9]の符号化法は，数値の自然データとしての特性に着目してアルファベットの大きさの問題に対処しようとするものであるが，漸近的な性能が必ずしも満足できるものではない．この点を改良した Itoh らの方法[2]は，漸近的に最良なユニバーサル符号化法になっているばかりでなく，問題の本質をより明確にする優れたアルゴリズムである．しかし，語長に相当する深さのヒープを必要とするので，たとえば，64 ビットのデータを対象とすることは実際には困難である．

本論文で新たに提案する方法は，これら二つの方法—これ以降，“従来の”といえば，この2方法だけを指すものとする—の欠点を相補うものである．特に，データ構造として AVL 木[1]を採用している点に特徴がある．Itoh らの方法と異なり，必要な記憶容量がデータの語長よりむしろデータ数で決まるものである．提案する圧縮法は，符号化したデータからもとのデータをひずみなく復元できる，対象データの統計的性質についての予備知識を仮定しないユニバーサルな符号化法になっている．データ圧縮法に関連した AVL 木の利用可能性については既に Williams[8]などによって指摘されている．しかし，そこでは AVL 木の探索木としての利用，すなわち適応化のための過去のデータ記憶用の補助手段としての利用可能性が指摘されているだけである．結果として，このようなデータ構造の選択はアルゴリズムの効率を左右することはあっても，圧縮力に直接的な寄与はしない．一方，本論文で取り上げる従来の2方法を含む各方法は，アルゴリズムの基礎に共通性があるにもかかわらず，採用しているデータ構造の差異によって圧縮力に違いを生じている．本論文では，各方法の性能の比較を，階調画像の可逆符号化への応用という，それ自身でも興味深い手法を新たに実現して行う．このような応用例は，数値データの適応的圧縮法が単に計算機内の数値データファイルのためばかりでなく，多くの応用可能性を持つものであることを示している．

なお，本論文での対数の底はすべて2である．

2. 数値データの適応的圧縮

2.1 基本的考え方

数値データを発生する未知の情報源を仮定する．ただし，数値の表現形式は既知の有限語長のある形式に固定されているとする．したがって，取り得る数値の上限・下限も既知であると考える．本論文では，これらをそれぞれ $+\infty$，$-\infty$ で表す☆．数値データの適応的圧縮とは，このような情報源から発生する離散データの系列を1走査で符号化し，より効率的で可逆な表現へと変換するものである．

このような数値データは，純粋の離散データとみなして符号化しても，アルファベットの大きさなどの問題のため十分な効率を得ることができない．一方，離散化されたデータとは言っても，全く離散的な分布に従うのではなく，離散化とは独立した，ある適当な性質を満たす分布関数の実現値となっているのが普通である．そのような分布関数があらかじめ分かっている場合には，その情報を利用することも可能であるが，ここでは，それを仮定しない．したがって，分布関数の推定に相当する操作と符号化とを並行に実行しながら，推定精度を徐々に高めることが必要になる．この考えを陽的に実現する手段として算術符号の利用がまず考えられるが，本論文で取り上げる手法は分布関数の推定機能を内在しているため，算術符号の考えを必ずしも必要としない符号化法となっている．

一般のデータ圧縮の基本的な考え方としては，データの頻度の偏りを利用して可変長符号に変換する場合（FV：Fix-to-Variable 符号）と，逆に偏りを一様にならす変換を導入してその結果を等長符号化する場合（VF 符号と呼ばれるものなど）とがある．数値データの圧縮では，まず，後者の考え方を利用する．これを理解するには，任意の分布関数に従う疑似乱数の発生法である逆関数法とのアナロジーがよい．逆関数法は，一様乱数 U を生成して，分布関数 F の逆関数値 $F^{-1}(U)$ を求めるというものである．**図1**に示すように，この逆関数法とは正反対の関係を利用することで分布関数の推定を行うことができる．分布関数の推定結果を区分的に線形な連続関数に限定すると，その導関数（密度関数）は区分的に一定で不連続な関数となる．従来の数値データ圧縮法は，このような考え方によって，**図2**に示すような，各領域を等面積に設定した密度関数の推定を過去のデータから行う．続いて，次に符号化すべきデータがどの部分区間に含まれているかを等長符号化し，さらに，その部分区間での対象データの位置を等長符号化する．部分区間での対象データの位置の符号化に必要な語長は各部分区間ごと

☆ 本論文で新たに提案する方法は，従来の方法と異なり，上下限が単調に変化する場合や上下限が未知の場合に拡張することができる．この点は本論文の方法の重要な特長の一つであるが，そのような場合については別途発表する．

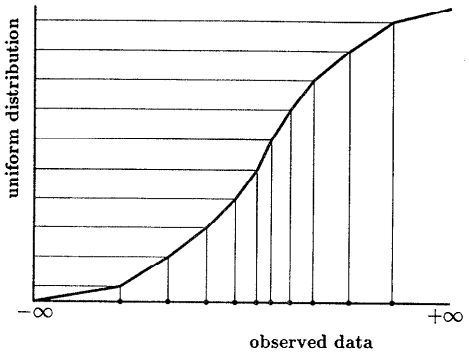


図 1 既に観測済みの数値データを横軸にプロットし，それらが一様に分布するように（縦軸）変換すると，その変換は数値データの分布関数に対する一つの推定と考えることができる．＋∞，－∞は，それぞれ，取り得る数値の上限と下限とを表す．
Fig. 1 Estimation of distribution function.

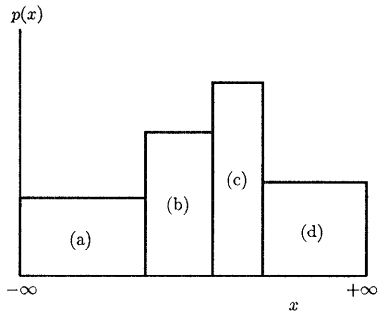


図 2 密度関数 $p(x)$ の推定．各部分区間の面積：$S_a=S_b=S_c=S_d$.
Fig. 2 Estimation of density function.

に異なるので，結果として，データ圧縮が実現できることになる．従来の方法は，このときの部分区間の境界を決定するための制約によって特徴づけることができる．たとえば，数値データの範囲を区間 $[0,1)$ に限定した場合，Yokoo の方法では，2進数の有限小数で表すことのできる数だけが境界の候補となるのに対し，Itohらの方法では，過去に観察された数値データがそのような候補となる．

以上のように，基礎にあるのは VF 符号の考え方であるが，結果として得られる符号は FV 符号である．

2.2 データ構造の選択

図1からも想像できるように，以上の考え方を利用するには，過去のデータに対するソーティングが必要になる．値の比較による交換を基本操作とするソーティングでは，データ数 n に対し $O(n\log n)$ の時間計算量が必要である．しかし，オンライン・データ圧縮を目的とする場合には，計算量が $O(n)$ を越えることは望ましくない．Itoh らの方法では，線形時間でソート可能なビンソートあるいは基数ソートの考え方を応用してこの問題を解決している．したがって，すべての可能な数値のためにあらかじめビン—Itoh らはヒープ構造を有する配列によって実現している—を用意しなければならず，数値表現の語長が長くなるほど実用性が低下することになる．一方，Yokoo の方法では，過去のデータ記憶に必要な精度を2分探索風に適応的に増すようにし，出現データに対する完全なソーティングを避けた結果となっている．密度関数の部分区間の境界が有限の2進小数—その有効桁数は1ビットから始まり適応的にしだいに長くなる—しか選ばれないのもそのためである．入力データのすべての桁の情報によって動的化する Itoh らの方法との漸近的性能の差は，このような利用している数値の精度の差に起因していると考えられる．

ここで，新しい方法の提案を行う．しばらくは，議論を簡単にするために，出現するデータに同一数の重複が全くない場合を考える．一例として，**図3**に示す2分木の各節点に記入した7個のデータがこれまでに観察されているとする．図3は，そのようなデータを平衡な2分探索木になるように配置したものである．各節点 x に続く2本の枝のうち左の枝を“x 未満”，右の枝を“x 以上”と解釈すると，一定の深さ d ($d=1,2,\cdots$) までの節点によって，2^d 個の部分区間への分割を表現することができる☆．また，左の枝に0，右の枝に1という記号を対応させると，各部分区間は

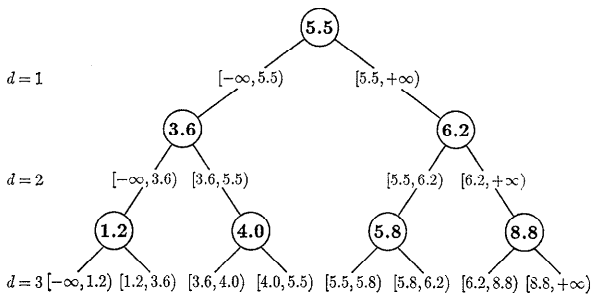


図 3 観測済みのデータによる部分区間の設定（完全平衡2分木の場合）．
Fig. 3 Subintervals corresponding to the completely balanced binary tree of already observed data.

☆ 数値の取り得る範囲に －∞ が含まれ ＋∞ が含まれないのは，単なる形式上の都合にすぎない．

長さ d ビットの符号語で表現できる．なお，本論文では，このように木構造に部分区間の分割を対応させる場合に，根を1と定義した“深さ (depth)”という表現を用いる．これに対し，木に含まれる最長のパスの長さをその木の“高さ (height)”と呼ぶ．根だけから成る木の高さは0である．

さて，既に n 個のデータが符号化済みで，第 $n+1$ 番目のデータ ($n=0,1,\cdots$) が符号化の対象になっているとする．これを，n に依存して決まる深さ $d=d(n)$ を選んで，まず 2^d 個の部分区間のうち対象データを含む範囲を長さ d ビットで符号化し，その区間の中でそのデータを特定するのに必要な情報をそれに続けることで符号化する．この方法は，部分区間の個数が 2^d という形式に限定されるという点で，Itoh らが Modified algorithm [文献2), p. 2502] と呼んでいるものの一般的な場合に相当する．

次に，第 $n+1$ 番目のデータの登録による木構造の更新が必要になる．しかし，その結果が再び平衡な2分探索木となるようにすることは，平衡性の定義によっては不可能か，あるいは多くの計算量を要することになる．そこで，各データの登録後の木構造としてAVL木[1]を採用する．AVL木は，任意の節点において，その左部分木と右部分木の高さの差が1以下であるような2分探索木である．AVL木への新たなデータの挿入は，まず通常の2分探索木の場合と同様に行う．その結果がAVL木の条件を満たさなければ，回転と呼ばれる操作によってAVL木に変換する．AVL木で重要なことは，この変換を含む種々の操作が通常の2分探索木操作に必要な計算量をオーダの意味で悪化させることなく実行できることである．なお，データに同一数の重複がないと仮定している限り，データの挿入は必ず挿入として実現される．

ここで，第 $n+1$ 番目のデータの符号化に議論を戻して，部分区間の個数について検討を加える．図3からも分かるように，完全に平衡な2分木（以下では，完全平衡2分木，completely balanced binary tree と呼ぶ）の場合に選択できる木の深さは $0\leq d\leq\lfloor\log(n+1)\rfloor$ である．なお，$d=0$ は全区間を全く分割せずに1区間として扱うことを意味する．一方，Itoh らは，部分区間の個数として $\lceil\sqrt{n}\rceil$ を推奨している．これは，$d=0.5\log n$ に相当する．一般の AVL 木で $d=0.5\log n$ とすることが可能かどうかをみるために，選択可能な最大深さ d を固定したときに節点数の最大値を与える AVL 木を $G(d)$ で表す．$G(1)$ は，

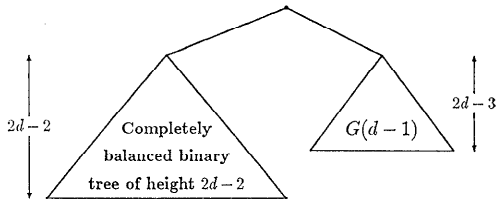


図 4 $G(d)$, $d\geq 2$：選択可能な最大深さが d で節点数最大の AVL 木.
Fig. 4 $G(d)$: AVL tree with the maximum feasible depth d that attains the maximum number of nodes.

根およびそれに接続した1個の葉から成る節点数2の木である．一般に，$G(d)$ は**図4**に示すような再帰的な構造を有し，その節点数を n_d で表すと，

$$n_1=2 \qquad (1)$$
$$n_d=n_{d-1}+2^{2d-1},\ d\geq 2$$

を満たす．図4に示すように，選択可能な最大深さを1ずつ増すと，$G(d)$ の高さは2ずつ増加する．式(1)より

$$n_d=\frac{2}{3}(2^{2d}-1)$$

を得る．これを d について解き，n_d を単に n と表記すると，

$$d=0.5\log\left(n+\frac{2}{3}\right)+0.5\log\frac{3}{2}$$

となる．この式は，観測済みのデータが n 個あるとき，選択可能な最大深さ d が最も浅くなる場合の n と d との関係を表している．すなわち，AVL木を使った場合でも区間の個数の上では Itoh らの方法の採用が可能なことを示している．実際，後述する実験では，$d(n)=\lfloor 0.5\log(n+1)\rfloor$ を使用している．

2.3 AVL木の修正と符号化法

これまでは出現するデータに同一数の重複はないと仮定してきたが，重複があっても符号化そのものは全く問題なく実行することができる．同一数の重複が問題となるのは AVL 木にデータを登録するときである．同一データを AVL 木の別々の節点に割り当てることを許すと，いわゆる退化が生じて AVL 木としての条件を満たさなくなることがある．したがって，同一データが生起した場合には AVL 木への登録をすることができず，そのようなデータの寄与を部分区間の設定に反映できない．この問題を解決する方法として，2分探索木の各節点にデータの重複度を記録する方法が考えられるので，まず，そのような可能性を Nievergelt ら[5]の重さ平衡木（BB木という）を利用

して検討してみた．

AVL 木においても，BB 木においても，平衡度の判定は部分木単位に行われ，根により近い節点の情報は参照されることなく局所的に再平衡化が行われる．ところが，そのために回転を行うと，節点間の子孫関係に逆転が生じるので，節点の重複度まで考慮した再平衡化を行おうとしても，回転の結果が再び平衡度の条件を満たすということは保証されなくなる．また，平衡度の劣化に関する性能が平均的には BB 木より AVL 木のほうが優れていることが知られている[5]．このようなことから，節点の重複度を記録した重さ平衡木を数値データ圧縮法に組み込もうとしても，同一数の重複の問題を簡単には解決できないことになる．そこで，本論文では，このような重さ平衡木を採用する代わりに，AVL 木自身を次のように修正する方法を提案する．

まず，以上述べてきた通常の AVL 木に含まれる節点を実節点と呼ぶことにする．これに対し，新たに“疑似節点”という概念を導入する．疑似節点は，数値データを格納しないという点を除けば，実節点と同じ構造を有し，同じような処理を受ける節点である．以下，このような疑似節点を導入した AVL 木を修正 AVL 木と呼ぶ．修正 AVL 木の節点数は挿入したデータ数に一致し，そのうち，重複したデータが疑似節点として登録される．以下，その操作について述べる．

次のデータ x を木に挿入する場合を考える．この場合，各節点に格納済みのデータと x とを比較しながら x を木の根から葉に向かって移動することになる．この間の比較で，木の節点上のデータと x とが等しい場合にはそのことを示すフラグ（*flag* で表す）を立て，x が疑似節点か葉☆に到達するまで x の移動を続けることにする．この結果，x の最終位置について次の4可能性が生じる．

（1） **not** *flag* かつ現在位置が実節点の葉である．

（2） **not** *flag* かつ現在位置が疑似節点である．

（3） *flag* かつ現在位置が実節点の葉である．

（4） *flag* かつ現在位置が疑似節点である．

新しい節点の挿入は，各場合に応じて行う．

場合（1）のときは，通常の AVL 木の場合と同様に，x を表す実節点を生成してそれを現在位置の子節点として挿入する．（2）の場合には，現在位置の疑似節点を実節点に変えてデータ x を登録し，（4）の場合に述べる疑似節点の挿入を行う．（3）の場合には，現在位置が真の葉ならば x の値に応じた枝を下に伸ばし，その先に疑似節点を生成する．現在位置が真の葉でなければ，枝を可能な側に伸ばして疑似節点を生成・付与する．（4）の場合は，新たに疑似節点を1個生成することにし，それを現在位置を根とする部分木に，その平衡性ができるだけくずれないような葉として挿入する．

以上のようにして，実節点あるいは疑似節点の挿入を行い，その結果が AVL 木の条件を満たさなくなれば木の再平衡化を行う．このための回転の操作は，通常の AVL 木の場合と同一である．すなわち，回転は平衡度だけを基準に行うので，実節点と疑似節点とを区別して扱う必要はない．ただし，疑似節点が実節点よりも上にあるパスができないようにする．これを実現する例を，いわゆる一重回転の場合で示したのが**図5**である．これは，二重回転の場合も同様に行うことができる．

以上で明らかなように，修正 AVL 木とは，同一数のデータがあるときに，それを疑似節点として“ほぼ”対応する位置に格納したものであると言うことが

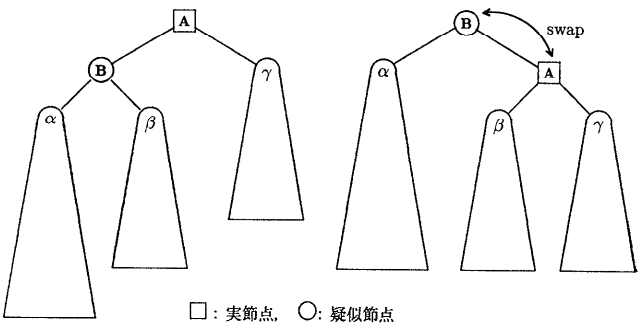


図 5　修正 AVL 木での一重回転の例．回転後の木（右図）で実節点Aと疑似節点Bとを交換した後，必要ならばさらに同様の操作を木のパスに沿って続け，疑似節点が実節点よりも上にあるパスができないようにする．すなわち，疑似節点は，たとえば，BがAより左にあるべきかどうかといった個性が問われない節点である．

Fig. 5　Single rotation in modified AVL tree.

☆ ここでは，x がそれ以上先に進めなくなった場合を便宜的に葉と呼んでいる．したがって，真の葉のほか，子節点を1個だけ有する節点において，子節点の接続されていないほうに x を移動させなければならないときも，この場合に含まれる．特に真の葉だけに言及する必要がある場合には，そのように呼ぶ．

できる．このような修正 AVL 木を使った符号化法をアルゴリズムとしてまとめると，その概略は次のようになる．

```
begin
1   n:=0; d:=0; N:=3; D:=3;
2   while データがある限り do begin
3     x:=n+1 番目のデータ;
4     α:=−∞; β:=+∞;
5     カレント・ノード:=修正 AVL 木の根;
6     flag:=false;
7     for i:=1 to d do begin
8       if カレント・ノードが疑似節点 then
            for ループを終了し17行目へ;
9       if x<カレント・ノードの値 then begin
10          "0" を出力; β:=カレント・ノードの値;
11          カレント・ノード:=左の子節点 end
12      else begin
13        if x=カレント・ノードの値 then
              flag:=true;
14  "1" を出力; α:=カレント・ノードの値;
15  カレント・ノード:=右の子節点 end
16  end; {for-end}
17  code(x−α, β−α) を出力;
18  葉に到達するまで flag をセットしながらカレント・ノードを進めxを修正 AVL 木に挿入;
19  if 平衡でない then 再平衡化;
20  n:=n+1;
21  if n≥N then begin
22  d:=d+1;
23  D:=4×D;
24  N:=N+D end
25  end {while-end}
end.
```

7行目から16行目までの **for** ループで部分区間が指定される．部分区間が決まると，その中でのデータの符号化が17行目の code(k, K) によって行われる．これを，k と K および符号化の対象が整数で $0 \leq k < K$ である場合で述べる．すなわち，$2^c \leq K < 2^{c+1}$ なる c と $j = 2^{c+1} - K$ とに対して，$k < j$ なら k の c 桁の2進数展開，$k \geq j$ なら $j+k$ の $c+1$ 桁の2進数展開を符号語とするものである．これは，K 個の対象をほぼ均等な長さで，できるだけ効率的に2元符号化する符号である[2]．復号も同様のアルゴリズムで実現可能で，これらの7行目から17行目を対応する復号法で置き換えればよい．18行目では，データ x を木に登録して，登録の結果不平衡が生じれば19行目で修正AVL木としての再平衡化を行う．次の21行目から24行目までは，部分区間を決める深さを $d(n) = \lfloor 0.5 \log(n+1) \rfloor$ に設定するものである．これ以降，上記の符号化法を“修正 AVL 木法”と呼ぶ．

これに対し，通常の AVL 木を使用し，同一データの反復があった場合にはそれを符号化はするが，AVL 木への登録をしない符号化法を“AVL 木法”と呼ぶ．

3. 性能評価

3.1 画像符号化への応用

提案した2方法—AVL 木法，修正 AVL 法—の圧縮力を評価し従来の方法と比較するには，採用している木の平衡度の劣化の解析が必要である．AVL 木の平均高さについては，完全平衡2分木と比較してほとんど劣化がないことが知られているので，データに同一数の重複がない場合の圧縮力の期待値は Itoh らの方法と大差ないと考えられる．しかし，AVL 木のこのような性質は半ば経験的にいくつか知られているにすぎず，同一数の重複がある場合や修正 AVL 木への理論の修正は容易ではない．そこで，厳密な解析に代えて乱数データによるシミュレーションのほか，画像データを対象とする圧縮実験によって各方法の性能を評価した．

Itoh らの方法と Yokoo の方法の漸近的性能の差は，1データ当たり 0.3〜0.4 ビット程度である．このことを示す Itoh らの実験[2]と同一規模の乱数データ（16ビット整数）によって実験したところ，修正 AVL 木法は Itoh らの方法と同等かわずかに劣る程度で Yokoo の方法より劣化することはなく，AVL 木法でも多くのデータで Yokoo の方法よりは効率的であるという結果が得られた．このような傾向は，乱数データだけでなく，どのようなデータに対しても同様に観察されたので，ここでは，乱数データによる実験結果の詳細を省き，現実的な例として興味深い画像符号化の例を報告する．この例は，数値データの適応的圧縮法が計算結果としての純然たる数値データのためばかりでなく，いろいろな応用可能性を持つものであることを示すものである．

議論を具体的にするために1画素256階調の正方画像の可逆符号化を考える．画像の2次元的な冗長性を除去する基礎的な手法として，予測子（predictor）の使用がよく知られている．この方法では，仮に高性能

の予測子を選んだ場合でも，(i) 2次元的な冗長性が完全に除去できるわけではない；(ii) 予測誤差の統計的性質が画像ごとに異なる；(iii) 予測誤差データを符号化する具体的な方法に何を選ぶか，といった問題の解決が必要である．このような問題を解決するため，Todd ら[6]は，近年さかんな情報源のモデリングと符号化とを分離する考え方に算術符号を組み合わせた方法を提案している．しかし，一般にモデリングと算術符号とを組み合わせる方法はコストの点で難点があり，十分に普及するまでに至っていない．本論文では，前半で議論した数値データの圧縮法を利用した，より単純で効率的なアプローチを提案する．

簡単な例として，**図6**に示す，近傍2画素から次の画素値を推定する予測子を考える．推定値 $P=(L+U)/2$ と実際の画素値 X との差 $E=P-X$ を予測誤差と呼ぶ．画素値の範囲を $0\leq X\leq 255$ とすると，予測誤差の範囲は

$$-255\leq E\leq 255$$

となる．予測子が高性能であれば，E は 0 を中心に対称に分布することが期待できる．その場合，Eの値を，その絶対値と符号ビットとをこの順序で接続した9ビット・データとして表示すると，MSB (Most significant bit) から LSB (Least significant bit) に向かって各ビットのランダム性が増すことになる．とは言っても，そのような特徴は一般的傾向にすぎず，具体的な性質は個々の画像ごとに異なる．ここで，上述の問題(i)に対処するために，このようにして表示した予測誤差データを隣接する数画素間でインタリーブを行って1個のデータに合成する．これを隣接する4画素で行う例を**図7**に示す．このようにして各画像から得られるたとえば36ビット・データの系列を数値データの系列とみなして前半で述べた圧縮法を適用することにする．

この符号化法では，インタリーブの結果，画素間

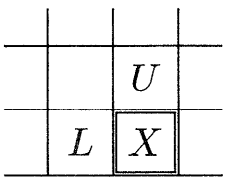


図 6 画素値Xを $P=(L+U)/2$ として推定するための近傍画素．画素Xが画像の境界にある場合には適当に扱う．この予測子は，1回の加算とシフトだけで実現することができる．
Fig. 6 Linear predictor $P=(L+U)/2$ for pixel X.

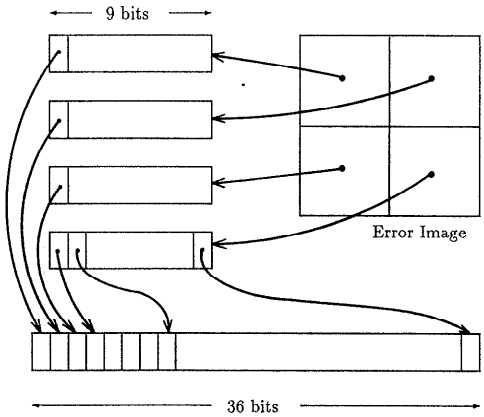


図 7 予測誤差データのインタリーブ．9ビット表示した各画素の予測誤差データを36ビットのデータに合成する．
Fig. 7 Numerical data of 36-bit long interleaved from 4 pixels on error image.

で符号化および復号化の時間遅れの差が生じる．しかし，前半で述べたどの数値データ圧縮法を組み合わせた場合でも，画像全体に対し，ラスタ・スキャン1走査で適応的に符号化を行うことができる．また，ひずみのない復号が一意に可能なことも，以上述べた符号化の過程を逆にたどることで理解できる．ただし，インタリーブを行う画素数を増やすと，数値データとしての語長が長くなって Itoh らの方法の適用は困難となる．

3.2 結果と考察

上述の画像符号化法に数値データ圧縮法として前半で述べた各方法をそれぞれ組み合わせた圧縮法をプログラムし，SIDBA 標準画像を含む自然画像の符号化実験を行った．その結果の主なものを**表1**に示す．画像はすべて 256 階調で画素数 65536 である．インタリーブを4画素単位に行った場合の数値データとしての個数は 16384 となる．そのうち，相異なるデータの個数が AVL 木法での節点数となる．なお，Itoh らの方法は 36 ビット・データに対しては適用できず，横2画素をインタリーブした 18 ビット・データに適用した結果だけである．ユニバーサル符号を画像符号化に応用する試みは既に数多く報告されているが，テキスト・ファイル圧縮で高性能を示す手法であっても，画像に対する性能は満足できる水準には達していないようである．一例として，比較のために，Welch[7]の LZW 法を応用した伊藤ら[3]の結果を表に含めた．

この伊藤らの方法に対し，本論文の画像符号化法は，どの数値データ圧縮法を組み合わせた場合でも

表 1 画像符号化の圧縮率の比較．節点数と時間の欄を除いて，結果はすべて圧縮率（出力符号長／入力データ長）である．時間の単位は秒である．“—”は，データが不明であることを表す．
Table 1 Compression ratios for natural images using various compression methods.

インタリーブ		2画素（18 bits/数値データ）				4画素（36 bits/数値データ）				伊藤-朴-今井[3]
組み合わせた数値データ圧縮法		修正 AVL 木法		Itoh-Goto[2]		AVL 木法		修正 AVL 木法	Yokoo[9]	
		圧縮率	時間［s］	圧縮率	時間［s］	節点数	圧縮率			
画像	aerial	0.757	19.8	0.751	17.5	16087	0.758	0.758	0.769	0.841
	couple	0.661	19.2	0.626	16.8	11725	0.595	0.588	0.574	0.674
	girl	0.663	19.5	0.630	17.0	14335	0.623	0.623	0.641	0.732
	gm	0.659	19.4	0.655	17.1	15915	0.665	0.664	0.709	—
	lady	0.649	19.0	0.607	16.7	10023	0.580	0.562	0.578	—
	moon	0.652	19.4	0.649	17.0	15863	0.658	0.658	0.709	0.741
単純平均		0.674	19.4	0.653	17.0	13991.3	0.647	0.642	0.663	—

10% 前後から 15% 程度の改良となっている．これは，1画素当たり1ビットに近い量であり，著しい改善が得られていることを意味する．採用した数値データ圧縮法の中では，4画素をインタリーブしたデータに修正 AVL 木法を適用した場合が平均的に良好な結果を示している．AVL 木法を採用した方法は，節点数が十分多い場合には修正 AVL 木法程度の性能が得られるが，節点数が少なくなると劣化が見られる．これは，AVL 木法が同一データの重複を考慮できないことによる．また同じように，Itoh らの方法でも，同一データの重複が多い場合には性能の劣化が見られる．しかし，2画素だけのインタリーブにもかかわらず，そのほかの画像のほとんどで最良の性能を示している．Yokoo の方法は，同一データの重複があっても，他の方法ほど性能が劣化しない点が特徴的である．いずれにしても，このような相違は各数値データ圧縮法の特性から直ちに説明できるものである．実験は，このほか，他の予測子やインタリーブ画素数等の様々な組み合わせについても行ったが，そのような組み合わせの最適な場合を見つけることが本論文の目的ではないので詳細は省略する．

最後に，計算量について簡単な言及を行う．AVL 木に第 $n+1$ 番目のデータを登録して再び平衡化するのに必要な計算量は $O(\log n)$ である．したがって，n 個の数値データを符号化するのに必要な計算量は $O(n \log n)$ となり，この点だけから言えば，従来の方法より劣ることになる．しかし，より微視的に計算量を比較した場合には，この結論は必ずしも妥当なものではない．まず，符号化アルゴリズムの7行目からの **for** ループと 18 行目のデータの登録で，数値データ1個の符号化につき木の根から葉までのパスが1回走査される．従来の方法でも木のパスの走査が必要であるが，それらと比較すると，AVL 木法や修正 AVL 木法がたどる木の枝の総数は，Yokoo の方法より多くなる確率は極めて小さく，Itoh らの方法と比較しても，語長が十分長い—“語長>log(データ数)”という意味で—場合には多くなることはない．後者は，計算量 $O(n)$ のビンソート等が $O(n \log n)$ のソーティング法より必ずしも高速とは言えないという事実に類似している．したがって，より詳細な計算量の比較は，木の再平衡化のための回転の頻度やアルゴリズム実現の詳細に依存して行わなければならない．これには特別な条件の下での2数の比較のための計算量などが関係し，これ以上の議論は本論文の範囲を越えるので別途報告したい．参考までに，2画素をインタリーブした 18 ビット・データに対する秒単位の計算時間を表1に含めた．この計算時間は，画素値の画素ごとの入力から符号化が終了するまでの総時間である．Itoh らの方法に比較して，修正 AVL 木法は 10% 強の劣化ですんでいる．また，表には含めていないが，2画素をインタリーブした同じデータに AVL 木法を組み合わせた場合の圧縮率と計算時間は6画像平均でそれぞれ 0.711 と 15.1 秒となり，符号化に必要な時間が Itoh らの方法より短縮している．なお，使用計算機は Sun Microsystems 社製の SPARCstation 1，使用言語はCである．

4. おわりに

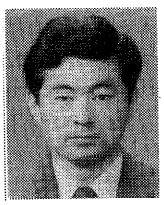
データ構造の標準的な教科書（たとえば，石畑[1]）では，アルゴリズムとデータ構造との関係をソーティング，文字列照合，探索などを例として述べるのが普通である．たとえば，ソーティングはアルゴリズムの

代表格であり，単純ながらも興味深いアルゴリズムが多数知られている．しかも，ヒープや2分探索木などの重要なデータ構造とも密接に関係している．これに対し，本論文で取り上げた数値データ圧縮の問題は，アルゴリズムがより複雑になり，応用がそれほど一般的でないという点でソーティングに相当する基礎的テーマではない．しかし，本文でみたように，アルゴリズムとデータ構造との関係という点では，これらの教科書的主題に匹敵する豊富な内容を持っている．しかも，アルゴリズムの性能として，計算量の上での性能に加え，本来の目的に対する性能が加わることになる．ソーティングでは，ソートの結果にはアルゴリズムによる差は本質的になく，性能としては，安定性を除けばその計算量だけが問題となる．一方，数値データ圧縮では，圧縮力という第一義的な性能がこのほかに存在し，そのような性能とデータ構造との間には，量的および構造的なトレードオフの関係が生じることになる．本論文で取り上げた従来の方法を含む各方法は，アルゴリズムの基礎に共通のアイデアがありながら，データ構造，性能，および計算量の3点で特徴的な対比を示すものとなっている．今後の課題が，より効率的なデータ構造でより性能の優れたアルゴリズムを見出す点にあることは言うまでもない．

参考文献

1) 石畑 清：アルゴリズムとデータ構造，岩波書店 (1989).
2) Itoh, S. and Goto, N.: An Adaptive Noiseless Coding for Sources with Big Alphabet Size, *Trans. IEICE*, Vol. E 74, No. 9, pp. 2495-2503 (1991).
3) 伊藤 豊，朴 志煥，今井秀樹：LZW 符号による画像データの圧縮に関する一考察，1990年電子情報通信学会春季全国大会，SA-6-2, pp. 437-438 (分冊1) (Mar. 1990).
4) Knuth, D. E.: Dynamic Huffman Coding, *J. Algorithms*, Vol. 6, No. 2, pp. 163-180 (1985).
5) Nievergelt, J. and Reingold, E. M.: Binary Search Trees of Bounded Balance, *SIAM J. Comput.*, Vol. 2, No. 1, pp. 33-43 (1973).
6) Todd, S., Langdon, G. G., Jr. and Rissanen, J.: Parameter Reduction and Context Selection of Gray-scale Images, *IBM J. Res. Dev.*, Vol. 29, No. 2, pp. 188-193 (1985).
7) Welch, T. A.: A Technique for High-performance Data Compression, *IEEE Comput.*, Vol. 17, No. 6, pp. 8-19 (1984).
8) Williams, R. N.: *Adaptive Data Compression*, Kluwer Academic (1991).
9) Yokoo, H.: A Lossless Coding Algorithm for the Compression of Numerical Data, *Trans. IEICE*, Vol. E 73, No. 5, pp. 638-643 (1990).

(平成4年 2 月3日受付)
(平成4年10月8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．現在同学科助教授．工学博士．データ圧縮，日本語処理などの研究に従事．電子情報通信学会，情報理論とその応用学会各会員．

杉浦由紀江

1992年群馬大学工学部情報工学科卒業．同年沖電気工業(株)入社．