# 算術符号の簡易化・高速化について
## ～STT-coder の紹介～

小野 文孝† 上野幾朗‡

† 東京工芸大学 (現在 早稲田大学・東京大学)
‡ 東京工芸大学 (現在 三菱電機(株) )

E-mail: ono@image.t-kougei.ac.jp

**あらまし** 算術符号は高い圧縮性能と様々な情報源に適用できる汎用性を有する．しかし，その演算負荷が高いため，如何に効率を保ったまま演算負荷の低減を行うかという検討が長年進められてきた．その結果，最近では，算術符号が多くの画像／映像符号化標準に採用されるようになっている．しかしながらハフマン符号と比較すると依然としてその負荷は大きく，算術符号の普及の妨げとなっていると考えられる．そこで，ハフマン符号と同様に状態遷移先と符号とを出力する状態遷移テーブルを参照して算術符号化処理を実行させることにより算術符号化の簡易化を図る STT-coder が筆者らにより提案されている．本報告では STT-coder の狙いとその性能並びに今後の発展の可能性について紹介する．

**キーワード** 算術符号，静止画符号化， ハフマン符号，ROM

# Simplified and High-Speed Arithmetic coder
## Introduction of STT-coder

Fumitaka ONO† Ikuro UENO†‡

† Tokyo Polytechnic University (now with Waseda Univ. and The Univ. of Tokyo)
‡ Tokyo Polytechnic University (now with Mitsubishi Electric Corp.)

E-mail: ono@image.t-kougei.ac.jp

**Abstract** Arithmetic coding has flexible adaptability to various information sources, and provides high coding efficiency. Its disadvantage is the complexity of operations, and the most studies to simplify and speed up arithmetic coding have been tuned for the simplification of the interval calculation for symbols such as in range coder, Q-Coder and MQ-Coder. Thanks to the effort, it is now adopted in many image and video coding standards. However, the simplification is still necessary to be prevailed more widely. Recently, STT-coder has been proposed by us, in which, arithmetic operation is realized by state transition table, just like the case of Huffman coding. In this report, we will introduce the background idea of STT-coder and its performance and future possibility to be developed.

**Keyword** arithmetic coder, still image coding, Huffman coding

## 1. まえがき

算術符号は高い圧縮性能と様々な情報源に適用できる汎用性を有する.しかし,その演算負荷が高いため，如何に効率を保ったまま演算負荷の低減を行うかという検討が長年進められてきた．その結果，最近では，算術符号が多くの画像／映像符号化標準に採用されるようになっている．しかしながらハフマン符号と比較すると依然としてその負荷は大きく，さらなる普及のためにはより簡易化する必要があると考えられる．これまでの算術符号の簡易化は主として領域計算の分野で行われてきた．初期のものとしては LR 符号 [1)]があり LPS の確率を(1/2)のべき乗に限定し，シフト演算で LPS の領域幅を求めるものである．続いて提案された Q-Coder[2)]では LPS の領域幅を有効領域幅によらず固定としていた．有効領域幅は最大で 2 倍の範囲を動くので想定確率としては 2 倍の範囲で変動が生じるが固定であっても確率を限定した LR 符号より高い性能が得られる．Q-Coder は確率が 1/2 近辺で LPS の領域幅が MPS の領域幅を上回ることがあるのでその補正を行う処理を加え QM-coder[3)]や MQ-coder[4)]として標準に採用されている．

算術符号(非ブロック符号)と通常のハフマン符号(ブロック符号）を比較するとその大きな相違はハフマン符号では符号ツリーをたどることで符号化／復号が行えるのに対し，算術符号では一般にそれが困難であることがあげられる．ハフマン符号において符号ツリーをたどるということは符号ツリーのノード（節）アドレスを遷移先とする状態遷移として符号化／復号を記述することであり，符号が確定する場合は出力符号語を記述し，遷移先はルート（根）に戻る．したがって算術符号を簡易化する上でハフマン符号と同様に状態遷移テーブルによって算術符号の動作を記述できるようにすることが簡易化につながるといえる．本報告ではこのような考え方に基づき筆者らによって提案された 2 値の算術符号である STT-coder[5),6)]について紹介するとともに今後の発展について述べる．

## 2. 状態遷移による符号化処理の記述

### 2.1. 3 ビットシステムでの例（方式１）

算術符号の利点はマルチコンテクスト情報源への対応であり，シングルコンテクスト情報源が対象であれば算術符号を採用する利点はないといってよい．そこで算術符号の適用情報源としてマルチコンテクスト情報源を想定する．また多値の算術符号も提案されているが，2 値の算術符号の方が汎用性が高いので符号化対象としては 2 値情報源を考えることとする．

STT-coder ではまず符号化のためのレジスタ長（数直線アドレスの精度）が 3 ビットの例をとりあげ状態遷移による符号化処理の考え方について考察している．

3 ビットシステムでは，ハフマン符号であれば符号語の種類としては最大で図 1 のように 1 ビット符号（2 通り），2 ビット符号（4 通り），3 ビット符号（8 通り）の 14 通りがあり，算術符号の処理過程では途中状態としてそれらを複数含む連続領域が考えられる．つまり各状態を遷移し有効領域がこの 14 通りのどこかに達した時に符号を発生すると考えることでハフマン符号と全く同様に算術符号の動作を状態遷移により記述できると考えられる．

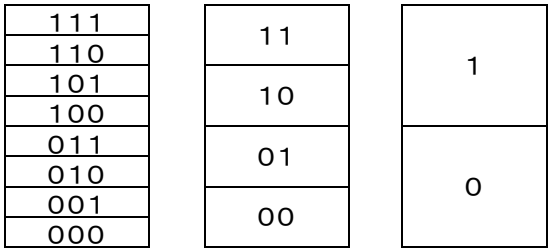


図 1　3 ビットシステムでとりうる符号

通常の算術符号においては 3 ビットシステムなら有効領域幅が 4 以下になれば再正規化を行い有効領域幅を 5 から 8 の間の値に回復する．また符号出力は有効領域の下限アドレスと上限アドレスを比較して確定した部分が取り出せることになる．最終的にはフラッシュ処理を行い領域を特定できる最小限のアドレスを出力する.このため有効領域の下限を常に記憶しておき，座標確定部を符号として出力する作業が必要になるが，この処理は状態遷移による記述では効率よく表現できないといえる．

一方状態遷移テーブルにより記述する算術符号ではこの 14 通りの領域のどれかに達するまでは再正規化は行わず，再正規化では常にフラッシュ処理を行うものとして考えることができる．通常の算術符号ではフラッシュ処理を行う際に有効領域を符号にするうえで無駄な領域が存在してしまうが，これは最後のシンボルである以上避けられないことである．しかし状態遷移型算術符号ではフラッシュを行うのは最後のシンボルとは限らないので算術符号として領域の無駄がないようにできる．たとえば図 2 の a)に示す 100 から 110 の領域は大きさが(1/2)のべき乗ではないので，通常の算術符号でフラッシュを行う場合はこの領域に対応した符号として 10（100 と 101 の共通部分）とせざるを得ず 110 の部分に無駄が生じる．また b)の 001 と 010 の領域であれば大きさは(1/2)のべき乗であるが対応符号は 001 か 010 のいずれかとせざるを得ず，領域がこの半分の場合と同じになってしまい，やはり無駄が生じる．

状態遷移型算術符号では次の情報源シンボルが存在するので次のシンボルの符号化において a)は c)に b)は d)に分割ができるため領域の無駄が生じない．STT-coder ではこのように後続のシンボルを組み合わせた情報源の拡大に相当する処理を取り入れており，これを「拡大フラッシュ」処理と称している．ただ分割がシンボル生起確率によらず固定となり情報源のシンボル出現確率の相違（コンテクスト）に基づく領域分割制御が行えないことになる．逆に通常の算術符号であれば a)の場合も b)の場合も最終シンボルでない限り再正規化を行うので有効領域を回復することが可能になり，効率低下を防げるがそのためには有効領域の下端アドレスを記憶しておく必要があり，状態遷移テーブル参照型算術符号とはならないといえる．

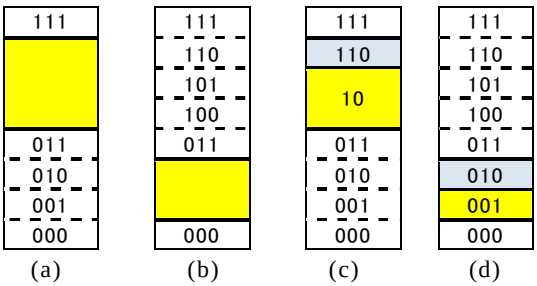


図 2　有効領域と符号

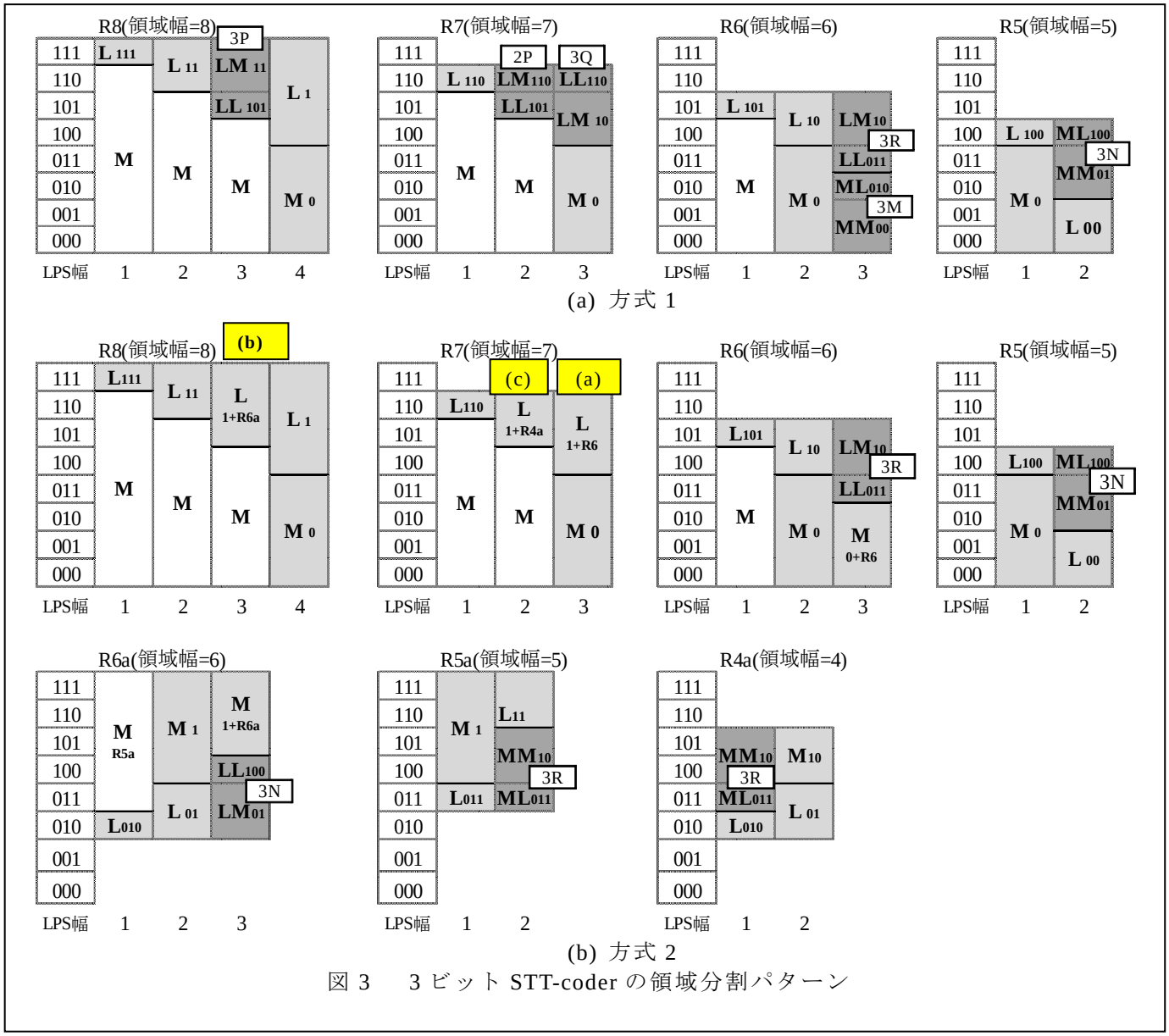


図 3　3 ビット STT-coder の領域分割パターン

さて, 2 値の算術符号では MPS／LPS の概念が一般的に用いられている．このため STT-coder でもまずは MPS／LPS での符号化を前提とする．そこで数直線の有効領域と，情報源のコンテクストから定まる LPS 幅とを組み合わせて記載したのが図 3 の方式 1 である．

ここで MPS と LPS の領域配置であるが LPS の上位配置／下位配置は事前に個別の規則をとりきめておけば受信側でも正しく判断できるので自由度がある．そこで図 3 にあるように，基本的には LPS を上位配置としているが，有効領域幅が 5 で LPS 幅が 2 の場合のみ LPS を下位配置としている．これはもし上位配置で LPS が出現すると有効領域幅が 2 で次の符号化では MPS／LPS とも領域幅が 1 で固定となり効率の低下をもたらすためである．一方 MPS が発生すれば有効領域幅が 3 でこれは続く符号化では MPS 幅が 2，LPS 幅が 1 に固定的に分けることになるが，この場合は MPS が上位配置でも下位配置でも同じであり効率に影響がない．

3 ビット算術符号の有効領域幅は通常の算術符号であれば最大領域の(1/8)の領域を単位として 5~8 であるが，方式 1 の状態遷移型では 4 以下でも再正規化を行わないことがありうるので 2~8 となる．そこで MPS 確率が 0.5 から 0.99 の場合について有効領域幅が 2~7 の場合における符号化効率と，有効領域幅が 8 の場合の符号化効率との差をとり，効率低下の指標として示したのが図 4 である．状態遷移型算術符号ではこのように有効領域の小さな状態を如何に避けるかが課題であることがわかる．

## 2.2.　3 ビットシステムでの例(方式 2)

さて, 方式 1 では算術符号の動作をブロック符号と全く同様の条件で考えたわけであるが算術符号の考え方をとり入れることでより効率を高めることを考える．

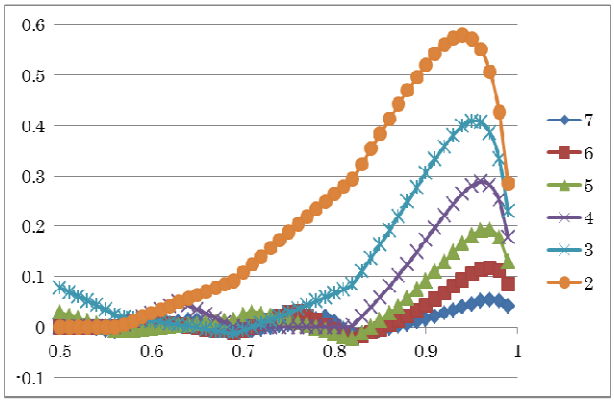


図 4 有効領域幅が 2 から 7 の場合の符号化効率低下（横軸は MPS 確率；縦軸は有効領域幅が 8 の場合に対する効率低下量）

通常の算術符号化であれば領域が 4 以下になればその下位座標に関わらず有効領域を 2 倍（2 のべき乗倍）する再正規化を行う．このため下位座標を記憶する処理が必要になる．そこでそれぞれの符号化過程をより詳細にみていくと(a)[有効領域幅が 7 で LPS 領域が 3 の場合に LPS が出現したとき]については符号の最上位は 1 と確定するので 1 を符号として出力したうえで精度をあげる（この領域を大きさ 6 に拡大する）ことが可能である．つまりフラッシュではない再正規化を限定的に許容するわけである．このとき次の遷移先（有効効領域 6）はたとえば有効領域幅が 8，LPS 幅が 2 で MPS が出現した場合の遷移先と全く同じでありもともと存在している．つまりこの場合は算術符号の再正規化にあたる処理をしても下位アドレスの記憶は不要でありまた全体の状態数も増えない．よって方式 1 ではこの幅 3 の領域を次の符号化で MPS 幅が 2，LPS 幅が 1 と分割してその後領域幅 8 に移るようにしていたのに対し，方式 2 では符号 1 を出力して有効領域幅 6 に移るようにすることにする．遷移テーブルの大きさとしては方式 1 であれば符号が出力されるときは必ず初期状態（有効領域幅 8)に戻るのに対しこの処理を許容すると符号が出力されても初期状態以外に遷移する場合があり，このため遷移時での符号出力の有無を示すための情報として遷移テーブルの出力ビットが 1 ビット増すことになる．

さて，最上位ビットが確定した時には再正規化を許容するとした場合に同様の例を個別に調べると(b)[有効領域幅が 8 で LPS 領域が 3 の場合に LPS が出現したとき], (c) [有効領域幅が 7 で LPS 領域が 2 の場合に LPS が出現したとき]も考察対象となる．しかし，両者については(a)とは異なり，遷移先は新たな状態であり状態数が増加してしまう．

(b)の時は符号 1 を出力すれば有効領域幅が 6（符号 000，001 が領域外）となるが，これはもともと存在する有効領域幅 6（符号 110,111 が領域外）とは異なる．またこの後の符号化処理では LPS 幅として 1,2,3 の 3 通りのいずれかが考えられ，たとえば LPS 幅が 1 の場合で，MPS が発生すると LPS 配置が上位でも下位でもさらに新たな状態が発生する（図 3 では下位配置を選択）．LPS 幅が 2 の場合には LPS 下位配置がよくこのあとには新たな状態は生じない．また LPS 幅が 3 の場合に MPS が出現すると有効領域幅が 6（符号 000，001 が領域外）という状態に戻る．

(c)の場合は符号 1 を出力し有効領域幅が 4(000,001, 110，111 が領域外）という新たな状態が発生する．この次の符号化では LPS 幅が 1 か 2 かを選択できるのでこの再正規化の効果があるといえる．またこのあとに新たな状態は発生しない．

以上のように上位ビットが確定した時に再正規化を許容するという条件を採用すると状態遷移でも符号出力があるかどうかを示す情報が 1 ビット増えるが有効領域幅の期待値が大きくなり効率向上につながる．また(a)のように状態数の増加を伴わないものに限定するか(b),(c)のように状態数の増加を伴うものも許容するかという判断が必要となる．3 ビットシステムでは状態数の増加も少ないので方式 2 ではいずれをも許容したがよりビット数の大きいシステムではより系統的な検討が望まれるからである．

方式 1 と方式 2 の符号化性能は図 5 のようになり，方式 2 では 3 ビットの通常算術符号と比べた性能低下は平均として 1%程度にとどまっている．

## 3. STT-coder の一般的設計

3 ビットシステムでの個々の例の考察に基づき，以下ではより一般的な設計法の考察を行う．

### 3.1 コンテクスト（モデル情報源）の設計

3 ビットシステムではすべての有効領域に対し，とりうるすべての LPS 幅を考えていた．しかし一般的な符号化システムを考える上ではより汎用的な考えが望ましい．このため各種の算術符号との比較も考慮し，ターゲットとなる情報源のシンボル出現確率に対し効率 100%の符号方式が用意されたときに，符号化すべき情報源のシンボル確率がターゲットからずれているとしても，その符号化効率が満たすべき最低値を設定し，その基準が満たされない範囲については新たにターゲットとすべき情報源のシンボル確率を設定するようにする．このようにして MPS 確率が 0.5 近辺から増大する方向にターゲットとなる情報源を順次設定していく．このようにして個別の符号とは独立なモデル情報源群を考えることができる．これは算術符号ではターゲットとなる情報源に基づき符号化方法（有効領域に対する各シンボル領域幅の設計）が決定されるという考え方ができることによっている．

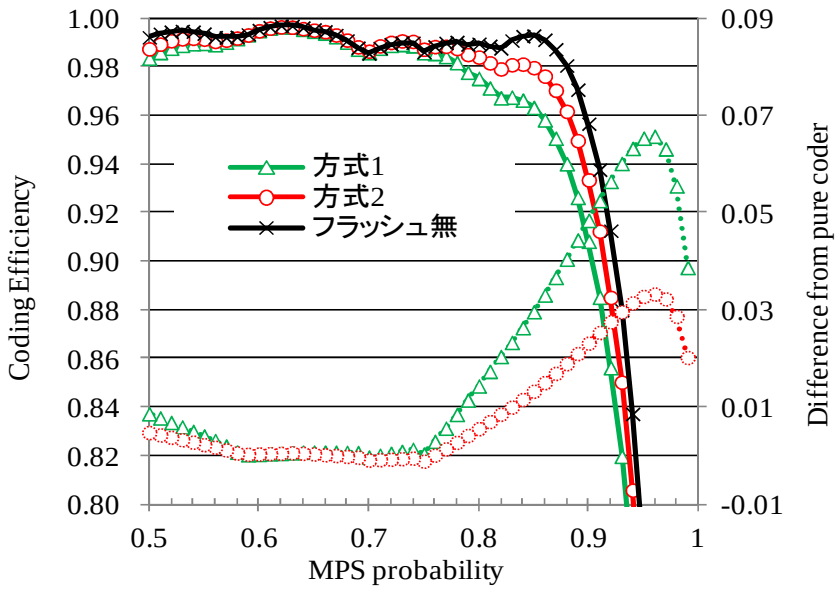


図 5　符号化効率（3 ビット STT-coder）

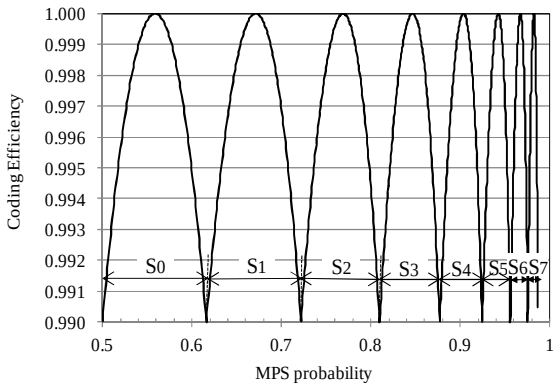


図 6　情報源モデルと理想符号での符号化効率

表 1　8 情報源モデルの中心 MPS 確率

| 情報源モデル | 中心 MPS 確率 |
|---|---|
| $S_0$ | 0.559 |
| $S_1$ | 0.671 |
| $S_2$ | 0.769 |
| $S_3$ | 0.847 |
| $S_4$ | 0.904 |
| $S_5$ | 0.942 |
| $S_6$ | 0.967 |
| $S_7$ | 0.982 |

たとえば，最低符号化効率を 0.99 に設定すると図 6 のように MPS 確率が最も小さい範囲では MPS 確率 0.559 のモデル情報源 S0 が定まる．これは MPS 確率 0.559 の情報源用の理想符号とは MPS での符号長を log（1/0.559），LPS の符号長を log（1/0.441）とする符号化であり，この符号化を MPS 確率が 0.5 の情報源に適用すると符号化効率が約 99%となることを意味している．図 6 に符号化効率のグラフを示す．また,S0 から S7 の 8 個のモデル情報源の中心となる MPS 確率は表 1 のようになる．

STT-coder として 6 ビットシステムをとりあげると S7 での LPS 幅は有効領域幅が 33 から 64 のすべてで 1 となるので 6 ビットシステムではこの 8 つの情報源を想定し，この効率が切り替わるところで符号化方式を切り替えればよいことになる．

3 ビットシステムでは各有効状態に対する LPS 幅を入力情報と考えていたが，このような考え方に立てばコンテクストと有効領域幅を入力情報として必要な LPS 幅を得て符号化を行う形の方が状態遷移テーブルサイズをより小さくできる余地がある．また動的符号化においてはコンテクストの遷移を取り扱うので，このモデル情報源の考えを動的符号化にも活用できる．

### 3.2　LPS 幅の条件とオフセットの概念の導入

上に述べたモデル情報源の考え方では有効領域幅に対する LPS 幅はモデル情報源数により制限される．また方式 2 で示したように上位ビットが確定した時に再正規化を行うという条件の採用については方式 2 の(b),(c)のように状態数の増加を伴うものも許容するかという問題が残されている．

しかし，上位ビットが確定した時に再正規化を行うかどうかを個別に規定するのは避けるべきであろうし，再正規化に伴う状態数の上限を設定しなければ方式設計が現実的にならないので上位ビットが確定した時には常に再正規化を行うという方針を採用し，状態数の制御はLPS幅の制限によってもたらされるようにすることとした．

たとえば3ビットシステムの方式2の(b)の場合には有効領域幅が6で，符号000，001が領域外となっている．このように下位の幅2の領域の符号が領域外となっている状態のことをオフセット2と呼ぶことにする．方式2の(a)で生じる有効領域6（符号110，111が領域外）はもともと存在する状態であり，オフセットゼロにあたる．

さて3ビット方式2ではオフセット2の状態からオフセット3の状態が生じている．これはLPSを下位配置としたためである．この状態からLPSを下位配置とすれば領域の上限が固定されMPSの出現により新たなオフセット値が登場する．一方LPSを上位配置とすればMPSの出現によりオフセットは固定され上限アドレスが変化することになる．もちろん方式2のように上位ビットが確定し，確定した分の符号を出力して再正規化することによりオフセットが変化する場合も生じる．

そこで，3ビット方式2で考えていた有効領域幅に対しとりうるすべてのLPS幅を考える方針は棄て，遷移状態数はオフセットの種類数により定まることからオフセットの種類数を制限し，それに基づいて逆に許容できるLPS幅を定め，その結果として状態数の上限が制限されるようにすることを考える．

たとえば3ビットシステムに戻りオフセットを1種類（オフセット=0）とすると，有効領域幅が8ではLPS幅は1,2,4の3種，有効領域幅が7ではLPS幅は1,3の2種，有効領域幅が6ではLPS幅は1,2の2種，有効領域幅が5ではLPS幅は1のみが許容される．

この時の遷移状態数は有効領域幅の種類数と等しく，8から5までの4種で済む．しかし特に有効領域幅が5のときはLPS幅が1のみなのでLPS確率が0.5に近い情報源での効率低下が大きい．そこでMPS確率が0.5から0.98の範囲について有効領域幅が5~7の場合における符号化効率と8の場合の符号化効率との差を調べ，効率低下の指標として示したのが図7である．図4と比較するとオフセットをゼロのみとすることでMPS確率が0.5付近での効率低下が大きいといえる．

なお一般的にはオフセットの種類を制限するうえで，どのオフセット値を許容するのが効果的かを調べる必要が生じる．

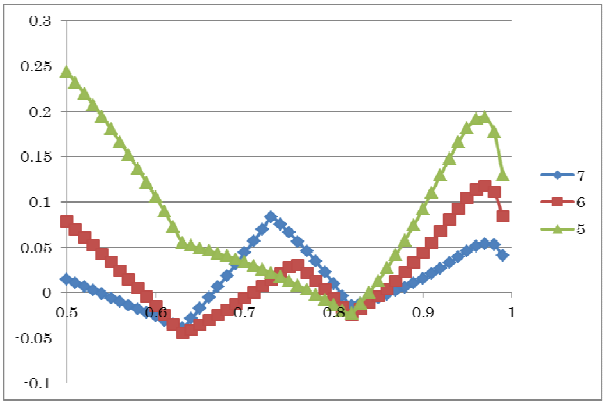


図7　オフセットをゼロに限定することでLPS幅を制限した場合の有効領域幅5〜7での符号化効率の低下（横軸はMPS確率，縦軸は領域幅が8の場合と比較した効率低下量）

### 3.3.　6ビットSTT-coderの設計

上記の考えをもとに6ビットSTT-coderを設計する．ここでシステムパラメータとして決定する必要があるのはオフセットの種類数である．遷移状態数はオフセットの種類と有効領域の上端アドレスの種類で定まり，有効領域の上端アドレスは32から63の32種類なのでオフセットの種類の32倍が状態数となる．オフセットの種類が1種類であればオフセット値はゼロのみであり，2種類であればゼロと1から31までのどれかとの組み合わせである．ゼロ以外のオフセット値としては領域の上位ビットが決まり有効領域を拡大する場合に新たなオフセット値が生じることも考え合わせると偶数に限定してよいと思われるのですべての偶数値について，MPS確率が0.5から0.98までの平均符号化効率を計算し，その値が最大となるオフセット値を求めた．その結果ゼロに追加すべきオフセット値はまず24がよいことがわかった．さらに許容するオフセット値についてはゼロと24は含めるとして他の偶数値との組み合わせをすべて調べた．その結果3番目は16であり，4番目は28となった．

さらにオフセット数を増やし8個のオフセットを調べると0,8,16,20,24,26,28,30の組み合わせとなった．これらの効率を図8に示す．

オフセット数を増やすとMPS確率が0.5付近の効率は上昇する．これは特に有効領域幅の小さい状態でLPS幅の許容値が増える効果が大きいためと思われる．しかしオフセットの値自体は大きなものが選択されるため有効領域幅の期待値は小さくなる．

この結果オフセットの種類数が4の場合に最も高い平均符号化効率が得られることがわかる．なお，オフセット値が24であればLPS幅8，オフセット値が28であればLPS幅4の場合が上位ビット確定につながる．

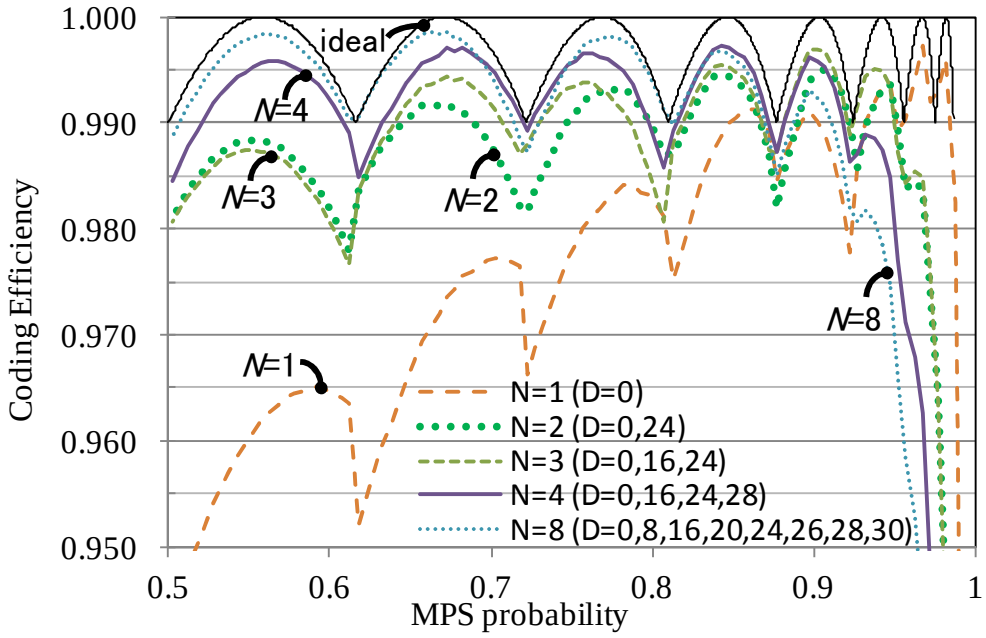


図 8　符号化効率（6 ビット STT-coder）

このため符号化効率で最適とされる範囲を超えてそれぞれ LPS 幅 8，LPS 幅 4 を選択するようにすることで平均的な符号化効率が向上することも確認されている．

### 3.4.　6 ビット STT-coder の動的符号化

STT-coder の静的符号化設計の考えについては上記で述べたが算術符号では動的符号化も重要であるので状態遷移型学習方法 [2]についても新たな提案を行っている．その目的として動的符号化特性を静的符号化特性と同様に MPS の出現確率に対しフラットにすることを掲げており，新たな考えとして遷移 LPS 比率の導入が重要であることを導いている．状態遷移型学習での状態に静的符号化での効率が 99%に近い 8 情報源モデルを採用した場合の性能の考察から従来並みの動的符号化性能を達成するには 8 情報源（8 状態）では不十分であり，その倍の 16 状態が必要であることが導かれているがそれでも従来の状態遷移型学習よりは少ない状態数で済むこと，16 状態とはいえ 8 状態のそれぞれの繰り返し（8 情報源）でよいことから静的符号化と動的符号化のシステムの共通化も容易であることが導かれている．

## 4.　今後の発展

算術符号化の簡易化の観点から筆者らが検討を進めている状態遷移テーブル参照型算術符号 STT-coder の考え方を紹介した．

一般的な STT-coder として 6 ビットシステムの設計例を紹介したが JPEG2000 や JBIG2 などの画像符号化に適用する検討はまだ十分ではない．JBIG2 などの 2 値情報源に適用するうえでは MQ-coder と同様に 12 ビットシステムが必要と思われる．その設計において 6 ビットで示されたオフセットの種類数の最適化結果やオフセット値の組み合わせ（0，1/4，3/8，7/16 という 4 種）から類推して 12 ビットのシステム設計を行うことも可能でありその場合に MQ-coder との性能比較を行うことも興味深い．

また JPEG2000 についてはこれまで JBIG2 と算術符号を共有するという前提で MQ-coder を採用しており算術符号として何ビットの精度が必要かはあまり明らかにされていない．12 ビット版に拡張した STT-coder ができた場合，その共有を図るほか JPEG2000 の算術符号化に必要なビット精度の STT-coder や JPEG2000 に専用の変形 STT-coder を検討する価値もあると思われる．

特に動的符号化については実際の情報源での検討が必要であり，それに応じてより実用的なシステムの提案が望まれる．

## 5.　むすび

JPEG2000 の仕様が固まってまもなく 15 年を迎える．その性能については今も色あせていないといえるが民生用途の応用への普及は必ずしも十分でない．そのため，算術符号化部分についてもより簡易化することが必要と思われる．本報告では筆者らがそのような用途

も含めて簡易算術符号化方式として提案しているSTT-coder の主として静的符号化部分について設計指針を紹介した．その結果下限アドレスを記憶しないという大きな簡易化の採用によっても十分実用的な性能が実現できることが示された．STT-coder の考察では簡易化と同時に算術符号全般の課題についても取り上げており，STT-coder のさらなる検討が今後の算術符号化全般への貢献となることを祈っている．

## 参考文献


1) G.G. Langdon, J. Rissanen: "Compression of Black-White Images with Arithmetic Coding", IEEE Trans., COM-29, 6, pp.858-860 (1981).
2) W.B. Pennebaker. J. L. Mitchell., G.G. Langdon, Jr., R.B. Arps: "An Overview of the Basic Principles of Q-Coder", IBM J. Res. & Develop., Vol.32, No.6 (1988).
3) ISO/IEC 11544 Information Technology -- Coded Representation of Picture and Audio Information -- Progressive Bi-level Image Compression (1993).
4) ISO/IEC 15444-1 Information Technology -- JPEG 2000 Image Coding System: Core Coding System (2000).
5) 上野 幾朗, 小野 文孝 :「状態遷移テーブル参照型算術符号 STT-coder の設計」画像電子学会誌 Vol.43, No.1, pp.62-70, 2014 年 1 月
6) 上野 幾朗, 小野 文孝 :「状態遷移テーブル参照型算術符号 STT-coder における確率推定方式と動的符号化の設計」画像電子学会誌 Vol.43, No.1, pp.71-78, 2014 年 1 月