# Minimizing Error and VLSI Complexity in the Multiplication Free Approximation of Arithmetic Coding

Gennady Feygin P. Glenn Gulak Paul Chow
Department of Electrical and Computer Engineering
University of Toronto, Ontario, Canada

### Abstract

Two new algorithms for performing arithmetic coding without employing multiplication are presented. The first algorithm, suitable for an alphabet of arbitrary size, reduces the worst case normalized excess length to under 0.8% versus 1.911% for the previously known best method of Chevion et al. The second algorithm, suitable only for alphabets of less than twelve symbols allows even greater reduction in the excess code length. For the important case of the binary alphabet the worst case excess code length is reduced to less than 0.1% versus 1.1% for the method of Chevion et al. The implementation requirements of the proposed new algorithms are discussed and shown to be similar to those of the algorithm proposed by Chevion et al.

## 1 Introduction

Arithmetic Coding [1, 2] is a data compression technique that represents the source data as a fraction that assumes a value between zero and one. The encoding algorithm is based upon recursive subdivision of an interval and retaining one of the created subintervals as a new interval for the next step of the recursion.

An encoding process begins by initializing an interval $A = [0, 1.0)$ and codeword $C = 0$. The interval $A$ is subdivided in proportion to the (known or estimated) symbol probabilities. Then a sub-interval corresponding to a current symbol being encoded is selected, and the values of $A$ and $C$ are updated to contain the size of the selected sub-interval and the position of the lower boundary of the selected sub-interval respectively. For an alphabet of size $m$, with symbol probabilities $p_i, i = 1, \dots, m$ the encoding process for the $k$-th symbol of the alphabet when encoding the $(n+1)$-th symbol in the input stream requires the following computations: $A_{n+1} = A_n p_k$, $C_{n+1} = C_n + A_n \sum_{i=1}^{k-1} p_i$. At the completion of the encoding process the value of the code-string $C$ is transmitted to the decoder. At the decoder, the iterative process of constructing the code-string $C$ is reversed.

Arithmetic coding was made practical by Rissanen and Langdon [1, 3] who developed a method for dealing with the infinite precision that results from iterative multiplications. They proposed that the value of the interval $A$ must be kept in an interval $[0.5, 1)$ following every step of the iteration. Whenever the value of $A$ drops below 0.5, both $A$ and $C$ are "re-normalized" by one or more left shifts until the

0-8186-3392-1/93 $3.00 © 1993 IEEE

value of $A$ returns to its proper interval. Further development of practical arithmetic coders [4, 5, 6] involve replacing the computationally expensive multiplication required in every cycle of the iterative update of the $A$ and the $C$ registers with approximations.

In the Q-coder [7] multiplication is avoided by using the approximation that $A = 1$ when doing multiplication, while the contents of the $A$ register are allowed to be in the range $[0.75, 1.5)$. The proposed method works only for binary alphabets. The worst case excess code length (defined below) is approximately 3.5% [3].

An alternative method that is suitable for an arbitrary-size alphabet was introduced by Rissanen and Mohuiddin [4]. The multiplication is made unnecessary by always setting the value of $A$ to either 0.5 or 1.0. Although simple and elegant, the method is not particularly efficient, with the authors reporting an excess code length of 12% [6].

The work of Chevion et al. [6] is an extension of the method of Rissanen and Mohuiddin, with the value of $A$ being approximated by a truncated value. The truncation method chosen guarantees that an approximation will require at most one addition and one shift operation. The excess code length is shown to be less than 2% for an arbitrary alphabet and less than 1.1% for the binary alphabet.

## 2 Analysis of the Multiplication Free Approximation of Chevion et al.

In the method of Chevion et al. [6] the value of $A$ is restricted to the range $[0.5, 1)$. $L$ is defined to be one greater than the number of zeroes between the first two ones on the left (MSB) side of the binary representation of the fraction $A$. The number of the most significant bits of $A$ being considered in computing the approximation of $A$ is denoted by $N$, where $N$ is less than[1] the total number of bits in $A$. When there is only a single one in the $N$ most significant bits ($L \geq N$), $L$ is set equal to $N$. For each $i$, $1 \leq i \leq m$, one half of the probability of the corresponding $i$-th symbol is denoted $p'_i = p_i/2$ and one half of the cummulative probability of all symbols from 0 to $i - 1$ is denoted by $S'_i = \Sigma_{j=0}^{i-1} p'_j$. Chevion et al. suggest that the true value of $A$ be approximated by $\hat{A} = 2^{-1} + 2^{L}$ for $L < N$ and by $\hat{A} = 2^{-1}$ for $L = N$; this approximation is shown in Figure 1(a). The corresponding encoding algorithm is shown in Figure 2 and the decoding algorithm in Figure 3.

By considering the mutually exclusive events $L = l$, $l = 1, 2, \ldots, N$, and performing a summation over the range of $l$, Chevion et al. [6] computed the following upper bound for $E(D)$, the *expected value of the excess code length $D$* that is introduced by their approximation technique:

[1] The number $N$ must be less than or equal to the total number of bits in the $A$ register minus the number of bits used to represent the symbol probabilities plus one. Otherwise, each iteration would increase the number of bits required to represent $A$.

$$E(D) \leq \sum_{l=1}^{N-1}(1+2^{-l})G_m(\frac{1}{2^l+1}) + G_m(\frac{1}{2^{N-1}}) \tag{1}$$

$$G_m(Y) = \frac{1}{ln2}[(1+Y)ln(1+Y) - \frac{1}{m^2}(1+mY)ln(1+mY) - (1-\frac{1}{m})Y] \tag{2}$$

In Figure 6 we illustrate the cumulative contributions to $E(D)$ of the terms in the summation of Equation 2 for $l$ going from $N = 12$ to 1. It is readily apparent that the contribution of the term $l = 1$ dominates, accounting for well over 50% of the total value of the upper bound of $E(D)$. This result is hardly surprising, since, as is apparent from Figure 1(a), $l = 1$ is the most common case (corresponding to one half of all possible values of $0.5 \leq A < 1$). Furthermore, the average error made in replacing $A$ by its approximation $\hat{A}$ is larger for the values of $A$ which correspond to $l = 1$ than the error made for the values of $A$ that correspond to other values of $l$.

The absolute value of the error is an exponentially decreasing function of $l$. It is apparent that any significant reduction in the excess code length will be dependent on improving the accuracy of our approximation for $l = 1$.

In what follows, we present a method for achieving that goal.

## 3 Enhanced Multiplication Free Approximation to Arithmetic Coding

Let $L$ be one plus the number of zeroes between the first two ones on the left of $A$, as in [6]. In the case where $L = 1$, let $\Gamma$ be one less than the number of ones on the left of $A$.

We will use a new approximation to the value of $A$, namely $\hat{A}_1$ defined as follows: For $L \geq 2$ use $\hat{A}_1 = \hat{A} = 2^{-1} + 2^{-(L+1)}$ as used in [6], and for $L = 1$, use $\hat{A}_1 = 2^0 - 2^{-(\Gamma+1)}$, as shown in Figure 1(b). We can now follow the methodology suggested in [6] to compute the expected value of the excess code length $E(D_1)$ corresponding to the new method for approximating $A$ by $\hat{A}_1$.

The actual mathematical derivation of the value of $E(D_1)$ is quite lengthy and is omitted, except for the final result:

$$\begin{aligned} E(D_1) \leq\ & (\sum_{l=2}^{N-1}(1+2^{-l})G_m(\frac{1}{2^l+1}) + G_m(\frac{1}{2^{N-1}})) + \\ & \sum_{\gamma=1}^{N-2} 2(1-2^{-(\gamma+1)})G_m(\frac{1}{2\times(2^{\gamma+1}-1)}) + 2(1-2^{-N})G_m(\frac{1}{(2^N-1)}) \end{aligned} \tag{3}$$

Table 1 shows the expected upper bounds for various alphabet sizes $m$ evaluated numerically. For all values of $m$ considered, the upper bound of the excess code

| $m$ | 2 | 12 | 22 | 32 | 52 | 72 | 256 |
|---|---|---|---|---|---|---|---|
| $E(D)/\log(m)\%$ | 1.101 | 1.747 | 1.866 | 1.903 | 1.911 | 1.896 | 1.735 |
| $E(D_1)/\log(m)\%$ | 0.321 | 0.613 | 0.703 | 0.745 | 0.784 | 0.798 | 0.787 |

Table 1: Upper bound of the of the excess code length (normalized to the source entropy) for various alphabet sizes that result from the method of [6] $(E(D)/\log(m))$ and from the proposed new method $(E(D_1)/\log(m))$. Probabilities of all symbols are $(p_1, p_2, \cdots p_m = \frac{1}{m})$ for a given alphabet size $(m)$.

| $p$ | 0.025 | 0.075 | 0.125 | 0.225 | 0.325 | 0.425 | 0.475 | 0.500 |
|---|---|---|---|---|---|---|---|---|
| $E(D)/H\%$ | 0.187 | 0.257 | 0.318 | 0.449 | 0.616 | 0.851 | 1.008 | 1.101 |
| $E(D_1)/H\%$ | 0.052 | 0.072 | 0.089 | 0.127 | 0.176 | 0.245 | 0.293 | 0.321 |
| $E(D_2)/H\%$ | 0.015 | 0.020 | 0.025 | 0.036 | 0.051 | 0.072 | 0.087 | 0.096 |

Table 2: Expected value of the excess code length using the method of [6] $(E(D)/H)$ and two proposed new methods $(E(D_1)/H$, $(E(D_2)/H))$ for a coder with binary alphabet and unequal symbol probabilities for various values of the probability of the least probable symbol $(p)$.

length has been reduced by more than 40%. Table 2 contains the results for the binary alphabet with various probabilities of the least probable symbol $p$. Here the reduction is approximately 60%.

# 4 A Further Enhancement for the Small Alphabets

In our discussions thus far we have relied on an approximation of $A$ by $\hat{A}$ that is derived by truncating [2] $A$. It is natural to ask whether it is possible to decrease the excess code length through a better approximation.

Consider a new approximation of $A$ by $\hat{A}_2$ that employs an alternative procedure illustrated in Figure 1(c). A sub-interval between $2^{-1} + 2^{-l}$ and $2^{-1} + 2^{-(l+1)}$ is subdivided into two equal halves; any value of $A$ is estimated to the closest possible value of $\hat{A}_2$.

We once again omit the lengthy mathematical derivation of $E(D_2)$ for the sake of conciseness and give only the final result:

$$E(D_2) \;\leq\; \sum_{l=2}^{N-1}(1+2^{-l})\left\{G_m(\frac{1}{2\times(2^l+1)}) - G_m(-\frac{1}{4\times(2^l+1)})\right\} -$$

[2] The reader must keep in mind that the truncation employed here is not the standard truncation to the $n$-th significant bit, but rather a truncation of all bits following the first bit set to a one in the $A$ register (not counting the MSB).

$$\frac{3}{2}G_m(-\frac{1}{12}) + G_m(\frac{3}{2^{N+1}}) +$$
$$\sum_{\gamma=2}^{N-2} 2 \times (1 - 2^{-(\gamma+1)}) \times \left\{ G_m(\frac{1}{4 \times (2^{(\gamma+1)} - 1)}) - G_m(-\frac{1}{2 \times (2^{(\gamma+1)} - 1)}) \right\} +$$
$$\frac{3}{2}G_m(\frac{1}{12}) + \left\{ G_m(\frac{1}{2^N - 1}) - G_m(-\frac{1}{2 \times (2^N - 1)}) \right\} \tag{4}$$

The expected value of the excess code length has been computed numerically and summarized in Table 2 for the binary case $(E(D_2)/H)$. The excess code length is reduced by a factor of ten or more for all values of the probability of the least probable symbol $p$.

Unfortunately, the technique employing rounding cannot be used for large values of $m$ due to the following:

In [6] the approximation $q_i = \frac{\hat{A}}{A}p_i \ \forall i, 1 \leq i < m$ is used. Since $q_i \leq p_i \ \forall i, 1 \leq i < m$, $q_m = 1 - \frac{\hat{A}}{A}(1 - p_m) > 0$.

When rounding is introduced, $q_i$ is no longer guaranteed to be less than or equal to $p_i$ and, as a result, $q_m$ may be less than or equal to zero. This leads to an invalid encoding with no code space reserved for the $m$-th symbol. The following equation must be satisfied to ensure that the code space is reserved for the most probable $m$-th symbol:

$$1 - \frac{\hat{A}_2}{A}(1 - p_m) > 0$$
$$\frac{\hat{A}_2}{A}(1 - p_m) < 1$$
$$\max\left\{\frac{\hat{A}_2}{A}\right\} \max\{(1 - p_m)\} < 1 \tag{5}$$

Recall that $p_m$ is the probability of the most probable symbol ($p_m \geq p_i \ \forall i, 1 \leq i < m$). Thus $\max\{(1 - p_m)\}$ occurs when $\min\{(p_m)\} = \frac{1}{m}$, or $\max\{(1 - p_m)\} = \frac{m-1}{m}$. Thus

$$\max\left\{\frac{\hat{A}_2}{A}\right\} < \frac{m}{m-1} \tag{6}$$

The maximum value of $\left\{\frac{\hat{A}_2}{A}\right\}$ occurs when $\Gamma = 1$, with $A = .1011$ and $\hat{A}_2 = 0.1100$.

$$\max\left\{\frac{\hat{A}_2}{A}\right\} = \frac{2^{-1} + 2^{-2}}{2^{-1} + 3 \times 2^{-4}} \max\left\{\frac{\hat{A}_2}{A}\right\} = \frac{12}{11} \tag{7}$$

Substituting back into Equation 6, we obtain the desired result: $m < 12$.

Thus, although rounding can be employed advantageously for small alphabets (up to $m = 11$), and in particular for the important case of binary alphabets, rounding cannot be employed in encoders with larger alphabets.

## 5 Implementation

We now proceed to evaluate some trade-offs and design choices that must be made if our proposed algorithm were to be implemented in VLSI. The block diagram of the encoder based on the algorithm proposed by Chevion et al. [6] is shown in Figure 4. The cycle of operations begins by the Thermometer Encoder[3] unit evaluating the value $L$. The resultant value of $L$ is used to control the setting of the Barrel Shifter in both the $A$-register data-path and the $C$-register data-path. In the meantime, values of $p'_k$ and $S'_k$ are made available from the look-up table (LUT). Finally, the updated values of $A$ and $C$ are computed in their respective Adder units and written back into the $A$- and the $C$-registers. The block diagram of the encoder based on the modified algorithms proposed in this paper is shown in Figure 5. Some additional circuitry is required, including a new 0/1 bit shifter unit, changing the Adder to be an Adder/Subtracter and modifying the Thermometer Encoder unit to search for either $L$ or $\Gamma$ as required. Fortunately, out of all the additional circuitry required, only the Adder/Subtracter unit is in the critical path of the computation. Furthermore, the delay of the Adder/Subtracter unit may be reduced by performing the one's complement and selection of the complemented or non-complemented values of $p'_k$ and $S'_k$ inside the lookup table. It is possible to design the encoder/decoder that implements our proposed method without impacting the length of the critical path.

On the other hand, for a given acceptable excess code length, the design based on our modified algorithm can be implemented with a lower value of $N$, reducing the length of the carry chains in the Adder units, the amount of multiplexing to be performed in the Barrel Shifter units and the number of bits of $A$ that must be examined in the Thermometer Encoder unit. This results in a reduction of the overall area required and the delay on the critical path. The dependence of the excess code length on the register length $N$ is illustrated in Figures 7a) and 7b). The comparison of the results shows that our proposed new methods outperform the method of Chevion et al. We must also point out that the Excess Code Length remains essentially unchanged for values of $N \geq 5$ for a given method, slightly worse but quite acceptable for $N = 4$ and unacceptably large for $N = 2$ or $N = 3$. Thus $N = 4$ or $N = 5$ are the best choices for practical design.

## 6 Summary and Future Work

We have demonstrated two new methods for performing arithmetic coding without employing multiplication. The first method, suitable for an alphabet of arbitrary size, reduces the worst case normalized excess length to under 0.8% versus 1.911% for the previously known best method of Chevion et al. [6]. The second method, suitable only for alphabets of less than twelve symbols allows even greater reduction in the excess code length. For the important case of the binary alphabet the worst case excess code length is reduced to less than 0.1% versus 1.1% for the method of Chevion et

[3] The name thermometer encoder is used due to the similarity of the required logic to the Thermometer Encoders employed in Flash A/D converters

al. [6]. Furthermore, the VLSI implementation of the proposed new methods can be accomplished without adding significantly to the circuitry required to implement the encoder of Chevion et al. and without impacting the length of the critical path.

Although the two methods proposed in this paper are sufficiently good to be used in all but the most demanding applications, it would be of interest to derive some theoretical bounds on the dependence of the excess code length on the complexity of the operations required.

## Acknowledgements

This work was supported by funding from the Natural Sciences and Engineering Research Council of Canada Operating Grants.

## References

[1] J. J. Rissanen and G. G. Langdon. Universal Modelling and Coding. *IEEE Transactions on Information Theory*, 27(12):12–23, 1981.

[2] N. Abramson. *Information Theory and Coding*, pages 61–62. McGraw-Hill, New York, NY, 1963. Refers to unpublished work of Elias.

[3] G. G. Langdon and J. J. Rissanen. Compression of Black-White Images with Arithmetic Coding. *IEEE Transactions on Communications*, 29(6):858–867, 1981.

[4] J. J. Rissanen and K. Mohuiddin. U.S. Patent 4,652,856, IBM, 1987.

[5] G. G. Langdon and J. J. Rissanen. A Simple General Binary Source Code. *IEEE Transactions on Information Theory*, 28(5):800–803, 1982.

[6] D. Chevion, E. D. Karnin, and E. Wallach. High Efficiency, Multiplication Free Approximation of Arithmetic Coding. In *Proceedings of Data Compression Conference*, pages 43–52, Snowbird, Utah, March 1991.

[7] W. B. Pennebaker et al. An overview of the basic principles of the Q-Coder adaptive binary arithmetic coder. *IBM Journal of Research and Development*, 32(6):717–726, November 1988.

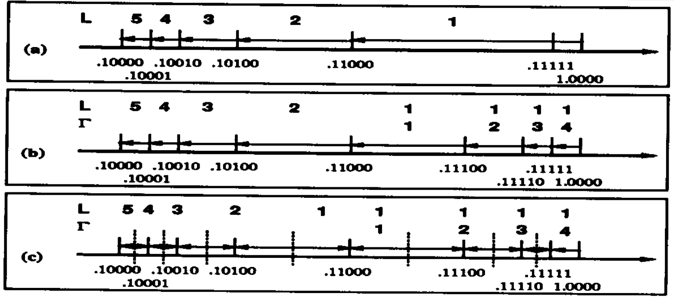

Figure 1: (a) Interval sub-division and approximation in the method of [6] , and two new methods: (b) subdivision into more intervals, (c) subdivision into more intervals with rounding to nearest permissible value.

```
$C \leftarrow 0.00\ldots0$
$A \leftarrow 0.11\ldots1$
loop
  compute $L$ as $\mathcal{F}(A)$
  input_symbol $k$
  if $1 \le k < m$
    $C \leftarrow \begin{cases} C + S'_k(1+2^{-L}) & \text{if } 1 \le L < N \\ C + S'_k & \text{if } L = N \end{cases}$
    $A \leftarrow \begin{cases} p'_k(1+2^{-L}) & \text{if } 1 \le L < N \\ p'_k & \text{if } L = N \end{cases}$
  else if $k = m$
    $C \leftarrow \begin{cases} C + S'_m(1+2^{-L}) & \text{if } 1 \le L < N \\ C + S'_m & \text{if } L = N \end{cases}$
    $A \leftarrow \begin{cases} A - S'_m(1+2^{-L}) & \text{if } 1 \le L < N \\ A - S'_m & \text{if } L = N \end{cases}$
  end if
  while $A < 0.100\ldots0$
    $A \leftarrow 2A$
    $C \leftarrow 2C$
    output_bit CarryOutFromC
  end while
end loop
```

Figure 2: Pseudo-code for the encoder algorithm of Chevion et al.

```
for $i$ from 1 to $SizeOfC$ by 1
  $C \leftarrow 2C$ + input_bit
end for
$A \leftarrow 0.11\ldots1$
loop
  compute $L$ as $\mathcal{F}(A)$
  find largest $k$ satisfying
    $C \ge \begin{cases} S'_k(1+2^{-L}) & \text{if } 1 \le L < N \\ S'_k & \text{if } L = N \end{cases}$
  if $1 \le k < m$
    $C \leftarrow \begin{cases} C - S'_k(1+2^{-L}) & \text{if } 1 \le L < N \\ C - S'_k & \text{if } L = N \end{cases}$
    $A \leftarrow \begin{cases} p'_k(1+2^{-L}) & \text{if } 1 \le L < N \\ p'_k & \text{if } L = N \end{cases}$
  else if $k = m$
    $C \leftarrow \begin{cases} C - S'_m(1+2^{-L}) & \text{if } 1 \le L < N \\ C - S'_m & \text{if } L = N \end{cases}$
    $A \leftarrow \begin{cases} A - S'_m(1+2^{-L}) & \text{if } 1 \le L < N \\ A - S'_m & \text{if } L = N \end{cases}$
  end if
  output_symbol $k$
  while $A < 0.100\ldots0$
    $A \leftarrow 2A$
    $C \leftarrow 2C$ + input_bit
  end while
end loop
```

Figure 3: Pseudo-code for the decoder algorithm of Chevion et al.

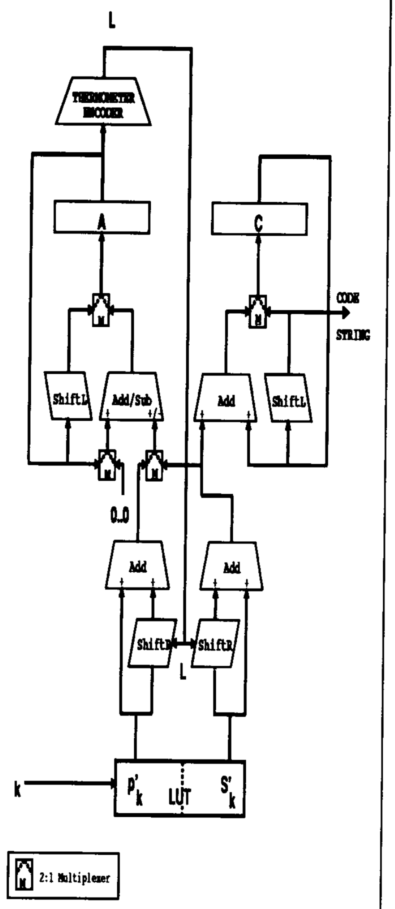

Figure 4: The architecture of the encoder based on the algorithm of Chevion et al.

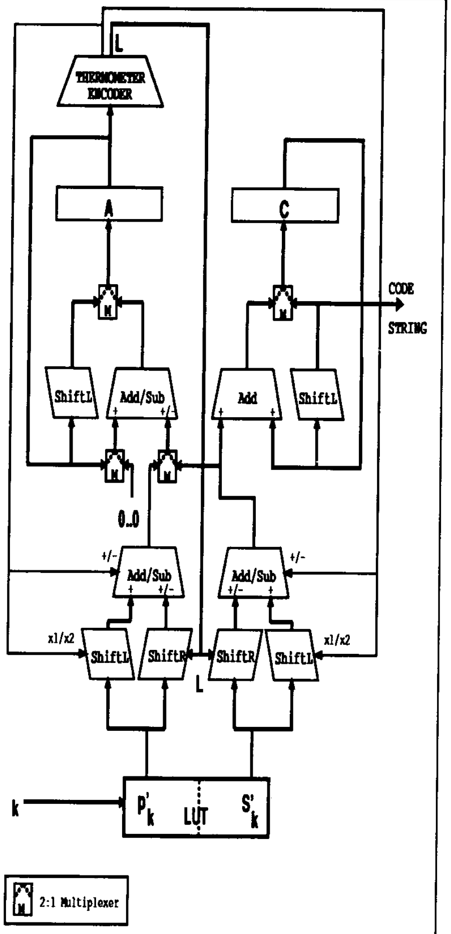

Figure 5: The architecture of the encoder based on the modified algorithms.

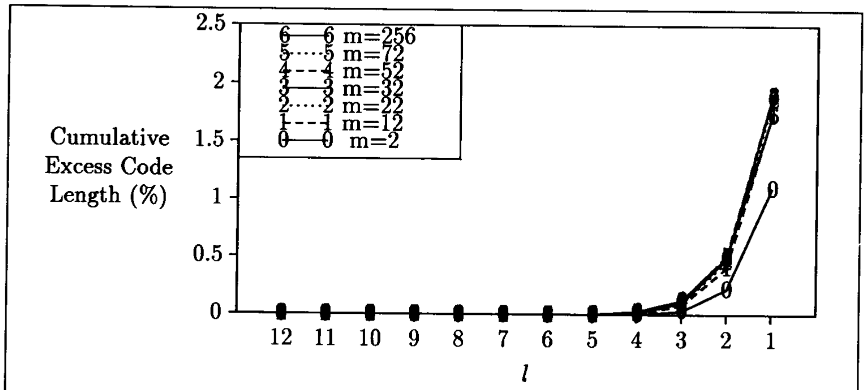

Figure 6: Cumulative contribution of subintervals of $A$ to the expected value of the excess code rate (normalized to the entropy of the source). Note that the summation proceeds in decreasing order, from $l = 12$ to 1.

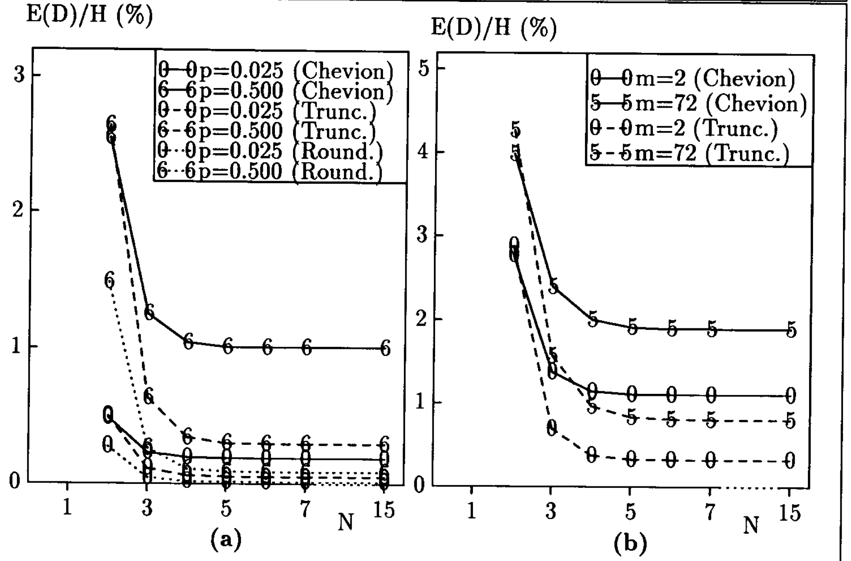

Figure 7: Excess Code Length (normalized to the entropy), as a function of the number of bits $N$, for: (a)binary alphabet ($m = 2$) with $p = 0.025$ (best case) and $p = 0.500$ (worst case), (b)non-binary alphabets with equal symbol probabilities ($m = 2$ (best case) and $m = 72$ ( worst case)) . The performance of the algorithm of Chevion et al. is shown as a solid line; the performance of our new algorithm employing truncation as a dashed line; and the performance of our new algorithm employing rounding as a dotted line (for binary alphabet only).