L. R. Bahl
H. Kobayashi

Image Data Compression by Predictive Coding II: Encoding Algorithms

Abstract: This paper deals with predictive coding techniques for efficient transmission or storage of two-level (black and white) digital images. Part I discussed algorithms for prediction. Part II deals with coding techniques for encoding the prediction error pattern. First, we survey some schemes for encoding if the error pattern is assumed to be memoryless. Then a method is developed for encoding certain run-length distributions. Finally, some experimental results for sample documents are presented.

Introduction

In Part I [1] we discussed several prediction algorithms for transforming a two-dimensional binary image into a prediction error pattern. Because this transformation is deterministic and invertible, it is possible to reconstruct the original image from the error pattern. The purpose of this transformation is to convert the two-dimensional redundancy of the image into a form that is suitable for easy encoding. It is generally much simpler to encode the error pattern than the original image. The error pattern can be approximated as the output of a binary memoryless source which produces 0's with probability p (the probability of correct prediction) and 1's with probability p_e (the probability of prediction error) $= 1 - p$. Many schemes for encoding the output of a memoryless source are known in the literature. The first half of this paper, which is a discussion and comparison of three such schemes, is intended to be tutorial in nature. The second half of the paper is the development of an encoding scheme that is useful when the memoryless source assumption is not justified. Finally, we present some experimental results related to the compression of the three sample documents discussed in Part I.

Encoding the error pattern by memoryless encoding schemes

The error pattern is assumed to be the output of a memoryless source producing 0's and 1's with probabilities p and p_e, respectively. The entropy $H(p)$ of this source is known to be

$$H(p) = -p \log_2 p - (1-p) \log_2 (1-p), \tag{1}$$

which is equal to $H(p_e)$.

It follows from Shannon's Source Coding Theorem [2] that the maximum possible compression G_{max} that can be achieved by encoding this source is

$$G_{max} = \frac{1}{H(p)}. \tag{2}$$

There are three basic properties of a code that are of interest to us: 1) efficiency, 2) stability, and 3) ease of implementation. By efficiency we mean the closeness with which the code comes to achieving G_{max}, the upper limit on compression for a particular value of p. By stability we mean the code's efficiency over a range of prediction error probabilities. This is important in an image compression system where many different types of images having varying prediction error rates may be processed. By ease of implementation we mean the simplicity of the hardware or software needed to do the encoding and decoding.

To encode the error sequence we first decompose it into blocks that can be of either a fixed or a variable length. Each block will be assigned a codeword and these too can be of either fixed or variable length. Clearly, assigning fixed-length codewords to fixed-length error blocks cannot provide any compression. We will consider one example of each of the three other possibilities: 1) fixed to variable encoding, 2) variable to fixed encoding, and 3) variable to variable encoding. A comparative evaluation follows the description of the three encoding methods.

- *Huffman encoding—fixed to variable*

The error sequence is decomposed into fixed-length

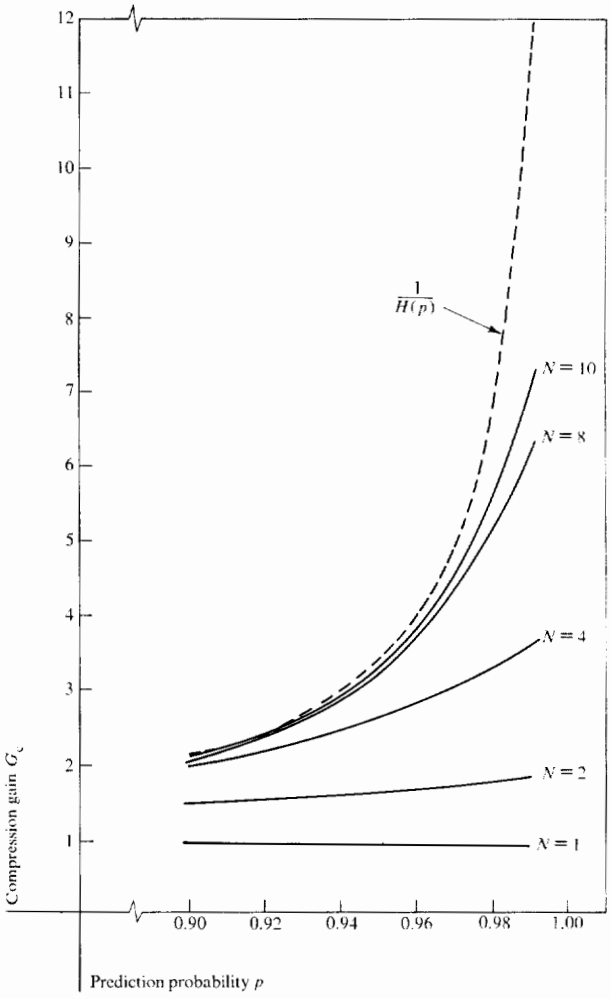

Figure 1 Compression gain of Huffman codes.

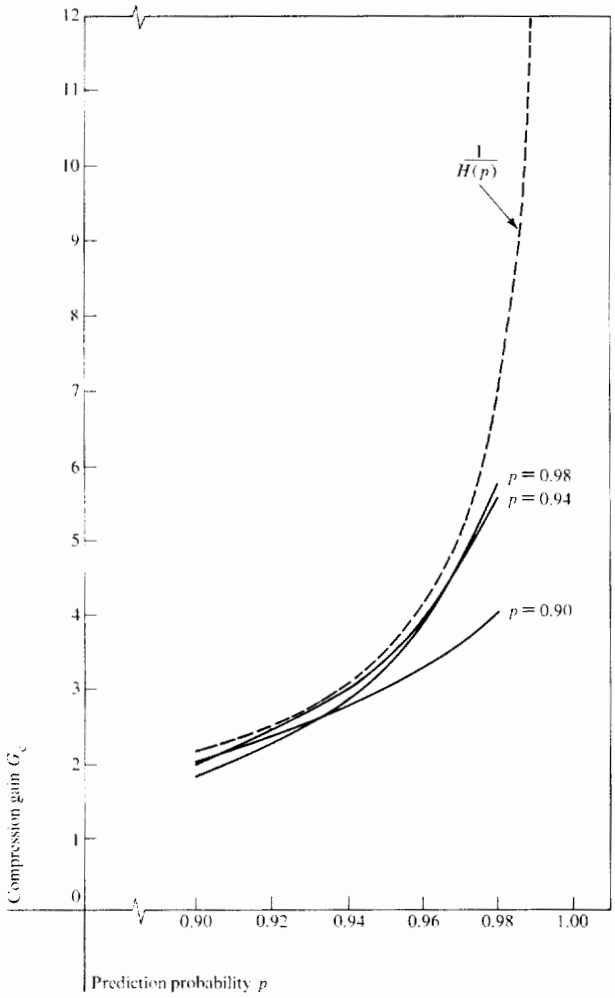

Figure 2 Compression gain of three specific Huffman codes with $N = 10$.

blocks and then a Huffman code [3] is used for encoding the blocks. Huffman coding is the most efficient fixed-to-variable length encoding method. If the fixed-block size is N, the Huffman code will have 2^N codewords, one for each of the 2^N binary patterns. From our assumption that the error pattern be regarded as essentially random, it follows that the probability of a particular pattern is $p^k p_e^{N-k}$, where k is the number of 0's in the pattern. These probabilities can be used to construct a Huffman code. We do not discuss the process of constructing the code since that is well known. For Huffman codes it is not possible to derive a closed form for the compression gain. However, it is known that the average number of bits, c, needed to encode each error digit is bounded by

$$H(p) \leq c < H(p) + N^{-1}. \tag{3}$$

Thus, the compression gain G_c is bounded by

$$\frac{1}{H(p)} \geq G_c > \frac{1}{H(p) + N^{-1}}. \tag{4}$$

Figure 1 shows the relationship between the actual compression gain G_c and the prediction probability p for $N = 1, 2, 4, 8,$ and 10. The compression gain was obtained by actually constructing the Huffman codes for various values of N and p and then determining their performance. Figure 1 also contains a plot of $G_{\max} = 1/H(p)$, which is the upper bound on compression. It should be noted that each curve in Fig. 1 does not necessarily represent a single code, but a sequence of codes, each optimal for a particular value of p.

The stability of a fixed Huffman code can be seen in Fig. 2, where we have plotted the performance of three codes of block length $N = 10$, which are optimal at $p = 0.90, 0.94$ and 0.98, respectively.

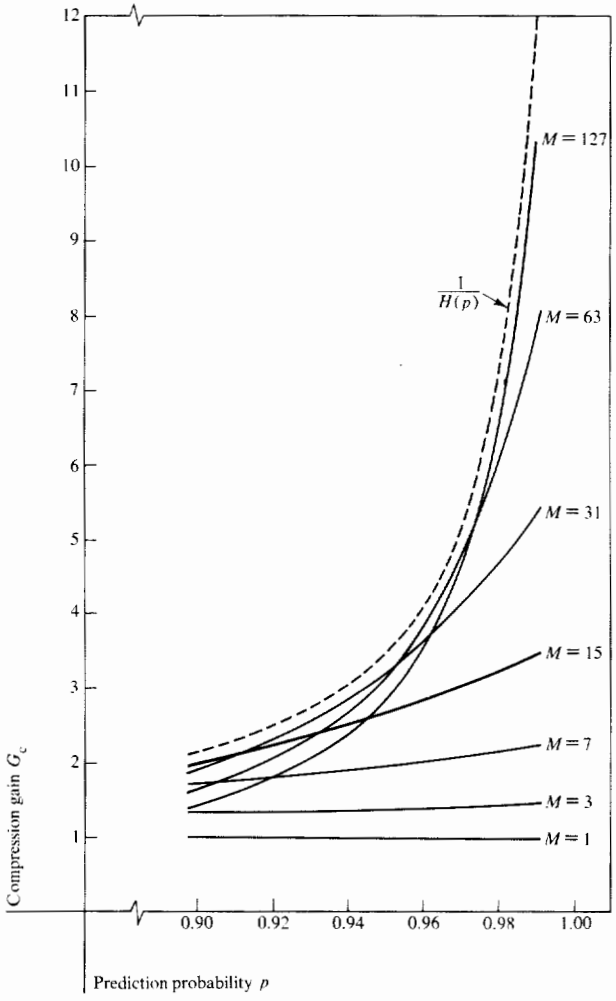

Figure 3 Compression gain of conventional run-length encoding.

Table 1 Example of a conventional run-length code with $M = 7$.

Sequence	*Run-length*	*Codeword*
1	0	000
01	1	001
001	2	010
0001	3	011
00001	4	100
000001	5	101
0000001	6	110
0000000	7	111

The implementation of Huffman encoding and decoding must be done by some form of table lookup. A code of block length N requires a table of size 2^N. For $N = 10$, the table size is 1024, which is quite large. In particular, the decoding of Huffman codes requires a considerable amount of computational effort.

• *Conventional run-length encoding—variable to fixed* There is a kind of run-length encoding in which the image is first decomposed into runs of successive 0's and runs of successive 1's. Then the lengths of these runs are encoded and transmitted. This type of run-length encoding is usually applicable to images directly rather than to prediction error patterns. Capon [4] and Huang [5] discuss this type of coding in great detail. Another type of run-length encoding, which is more applicable to the encoding of prediction error patterns, is one in which we encode and transmit only runs of 0's that are terminated by a single 1. It is this second type of run-length encoding that we consider in this paper.

An example of conventional run-length encoding is shown in Table 1. This kind of encoding has been discussed by Elias [6], Wholey [7], and Arps [8], among others. The encoding process is in two steps. First, the error sequence is decomposed into run-lengths. Columns 1 and 2 of Table 1 show the correspondence between the error sequence and the run-lengths. An upper bound M ($M = 7$ in Table 1) is placed on the longest run-length, so if the number of consecutive 0's, L, is M or greater, then each group of M 0's is considered to be a run of length M and the remaining bits are treated as a separate run-length; M is chosen in such a way that $M = 2^N - 1$ for some positive integer N.

The second step is to assign codewords to each run-length. In conventional run-length encoding each run-length L is assigned a codeword, which is the N-bit binary representation of L. Columns 2 and 3 of Table 1 show an example of this codeword assignment.

It is a simple matter to derive an expression for G_c in this case. The probabilities of the run-length are given by

$$p(L) = \Pr\{\text{run-length} = L\} = \begin{cases} p^L \cdot p_e & \text{for } 0 \le L \le M-1; \\ p^M & \text{for } L = M. \end{cases} \quad (5)$$

The average number of bits in each block, λ, is then

$$\lambda = Mp(M) + \sum_{L=0}^{M-1} (L+1)p(L) = \frac{1-p^M}{1-p}. \quad (6)$$

Because all codewords have length $N = \log_2(M+1)$, the compression gain is

$$G_c = \frac{\lambda}{N} = \frac{1-p^M}{(1-p)\log_2(M+1)}. \quad (7)$$

Figure 3 shows the relationship between p and G_c for different values of M. Since each curve in this figure represents a single code, both the efficiency and the stability of this method are evident from these curves. The implementation of this encoding method is quite simple. All that is needed is an N-bit binary counter which has as its input the error sequence. Its operation is as follows:

1) For each input of 0, the counter counts up by 1.
2) When the counter reaches its maximum value $2^N - 1$, it outputs a sequence of N 1's and is then reset to 0.
3) When a run-length terminates, i.e., when the input is a 1, the N-bit counter outputs its contents and is then reset to 0.

The operation of the decoder is similarly quite simple.

• *Golomb's run-length encoding—variable to variable*
This method is due to Golomb [9]. We give a simple interpretation of this encoding method. The first step is to obtain the run-lengths, as we did in conventional run-length encoding, except that no upper bound on the run-length is assumed. The codewords for the run-length can be constructed in the following simple manner. First, find an integer m such that

$$p^m \approx 0.5. \tag{8}$$

Once m is found, we partition the run-lengths into groups of size m: the set of run-lengths $\{0, 1, 2, \cdots, m-1\}$ forms group A_1; the set $\{m, m+1, \cdots, 2m-1\}$, group A_2; etc. In general, the set of run-lengths $\{(k-1)m, (k-1)m+1, \cdots, km-1\}$ comprises group A_k. To each group is assigned a group prefix, which for group A_k is $k-1$ 1's followed by a 0 and which we denote by $1^{(k-1)}0$. If m is chosen such that $m = 2^N$, each group contains 2^N members and an N-bit sequence (called the tail) uniquely identifies each member within the group. The simplest way of generating this tail for a run-length L is to construct the N-bit binary representation of $L - (k-1)m$, i.e., the binary representation of L modulo m. The codeword for a run-length L which belongs to group A_k consists of its group prefix, $1^{(k-1)}0$, followed by the N-bit tail, L modulo m. The encoding process can be understood from Table 2, which shows the Golomb code for $m = 4$.

It is not necessary to choose m to be a power of 2. In this paper we consider only codes with $m = 2^N$ and $p^m = \frac{1}{2}$ because they are simpler to implement. Codes with $m \neq 2^N$ are similar to the codes discussed here, with some minor modifications which are discussed in Golomb [9] and our earlier report [10].

We now obtain an expression for the compression achieved by this method for the case when $m = 2^N$. For an independent binary source, the run-length distribution is geometric in nature, i.e.,

$$\Pr\{\text{run-length} = L\} = g(L) = p^L \cdot p_e \text{ for } L \geq 0. \tag{9}$$

Equation (9) is similar to (5) except that there is no limit on the maximum run-length.

The average number of digits per block, λ, is the average run-length plus one, i.e.,

$$\lambda = 1 + \sum_{L=0}^{\infty} Lp^L \cdot p_e = \frac{1}{1-p} = \frac{1}{p_e}. \tag{10}$$

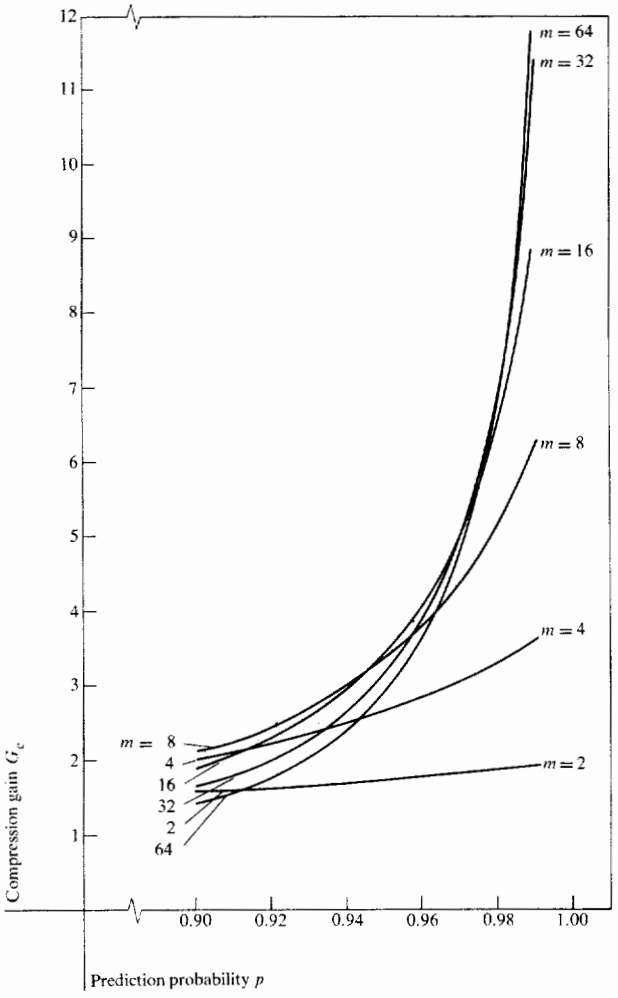

Figure 4 Compression gain of Golomb run-length encoding.

Table 2 An example of Golomb's run-length encoding for $m = 4$.

Group	*Run-length*	*Group prefix*	*Tail*	*Codeword*
A_1	0	0	00	000
	1		01	001
	2		10	010
	3		11	011
A_2	4	10	00	1000
	5		01	1001
	6		10	1010
	7		11	1011
A_3	8	110	00	11000
	9		01	11001
	10		10	11010
	11		11	11011
	⋮			

The combined probability of run-lengths belonging to group A_k is

$$P(A_k) = \sum_{L=(k-1)m}^{km-1} p^L \cdot p_e = p^{km} = 2^{-k}. \quad (11)$$

Because each run-length in group A_k has a codeword of length $N + k$, the average codeword length is

$$c = \sum_{k=1}^{\infty} (N + k) \cdot 2^{-k} = N + 2. \quad (12)$$

Thus

$$G_c = \frac{N + 2}{1 - p} = \frac{2 + \log_2 m}{1 - p}, \quad (13)$$

where $m = -1/\log_2 p$. (14)

Even when $m \neq 2^N$ the above formula (13) is a close approximation to the compression gain [10].

Figure 4 shows the performance of Golomb's codes for $m = 2, 4, 8, 16, 32$, and 64. As in the case of conventional run-length encoding, each curve represents a single code. The curve for the upper bound $1/H(p)$ could not be distinguished from the envelope of the curves plotted in that figure.

Encoding can be done by an N-bit binary counter as in the case of conventional run-length encoding. The operation of the counter is as follows:

1) For each input of 0, the counter counts up by 1.
2) When the counter overflows (i.e., when it reaches $m = 2^N$), the encoder outputs a 1 and resets the counter to 0.
3) When a run-length terminates, i.e., when the input is a 1, the encoder outputs a 0 followed by the contents of the N-bit counter. The counter is then reset to 0.

Decoding can similarly be performed in a simple way. Circuits for encoding and decoding Golomb codes are discussed in detail in our report [10]. Golomb coding is closely related to Shannon-Fano [2] encoding. This is also discussed in detail in our earlier report [10].

• *Comparison of the three encoding methods*

In our experiments with compressing real images, we encountered prediction probabilities in the range of roughly $p = 0.93$ to $p = 0.99$, which corresponds to prediction error rates between about seven and one percent, respectively. We are therefore interested in comparing the performance of the codes over this region.

A comparison of Figs. 1, 2, 3, and 4 shows the superiority of Golomb codes over both Huffman coding of fixed lengths and conventional run-length encoding. The main deficiency of Huffman codes is the poor performance at higher values of p. For example, with $p = 0.98$ and $N = 10$, Huffman codes achieve only 81 percent of the maximum possible memoryless compression. With conventional run-length codes, the efficiency is about 92 percent and for Golomb codes it is above 99 percent. In terms of stability, Golomb codes are superior, although conventional run-length codes are almost as good. Huffman codes are comparatively poor. In terms of implementation, encoders and decoders for both Golomb and conventional run-length codes require very little logic, whereas Huffman codes require the storage of a table of considerable size.

Encoding run-lengths with non-geometric distributions

In the preceding section we assumed the error sequence to be the output of a binary memoryless source. In this section we consider the encoding of error patterns which do not meet this idealized situation. We limit ourselves to run-length encoding because of its superior performance in the memoryless case. Some schemes for encoding nongeometric run-length distributions have been considered by Huang [5].

We have previously shown that for a geometric run-length distribution, the average error sequence block length is

$$\lambda = \frac{1}{p_e}, \quad (15)$$

where p_e is the prediction error probability. This relation must, however, hold for any run-length distribution. Consider any binary source which produces 1's with probability p_e. Then in a sequence of length N, the number of 1's is $N \cdot p_e$ as $N \to \infty$. Because the number of run-lengths is equal to the number of 1's, the average block length λ is given by

$$\lambda = \frac{N}{N \cdot p_e} = \frac{1}{p_e}. \quad (16)$$

For a memoryless source, the run-length distribution is given by

$$g_L = p^L \cdot p_e \quad \text{for } L = 0, 1, 2, \cdots. \quad (17)$$

Replacing p_e by $\frac{1}{\lambda}$ and p by $1 - \frac{1}{\lambda}$ we have

$$g_L = \left(\frac{\lambda - 1}{\lambda}\right)^L \cdot \frac{1}{\lambda} \quad \text{for } L = 0, 1, 2, \cdots. \quad (18)$$

The run-length entropy H_G, i.e., the average number of bits needed to encode a run-length, is

$$H_G = -\sum_{L=0}^{\infty} g_L \log_2 g_L. \quad (19)$$

Substituting for g_L from (18) we have

$$H_G = \lambda \log_2 \lambda - (\lambda - 1) \log_2 (\lambda - 1). \quad (20)$$

It is easily verified that H_G is the same as $\lambda H(p)$.

Now consider another general, non-memoryless source that also has prediction error rate p_e and hence average run-length λ. Let the run-length distribution be any arbitrary distribution, denoted by

$$p_L = \Pr\{\text{run-length} = L\}, \quad L = 0, 1, 2, \cdots, \tag{21}$$

with average block length

$$\lambda = \sum_{L=0}^{\infty} (L+1)p_L. \tag{22}$$

The run-length entropy of this source, denoted by H_L, is given by

$$H_L = -\sum_{L=0}^{\infty} p_L \log_2 p_L. \tag{23}$$

Theorem Among all sources having the same average run-length, the source with geometric run-length distribution has maximal run-length entropy.

Proof We need to show that $H_L \leq H_G$. Consider the quantity

$$\begin{aligned} -\sum_{L=0}^{\infty} p_L \log_2 g_L &= \sum_{L=0}^{\infty} p_L[(L+1)\log_2 \lambda - L \log_2 (\lambda - 1)] \\ &= \lambda \log_2 \lambda - (\lambda - 1)\log_2 (\lambda - 1) \\ &= H_G. \end{aligned} \tag{24}$$

Then

$$\begin{aligned} H_L - H_G &= \sum_{L=0}^{\infty} p_L \log_2 \left(\frac{g_L}{p_L}\right) \\ &= \log_2 e \sum_{L=0}^{\infty} p_L \ln\left(\frac{g_L}{p_L}\right). \end{aligned} \tag{25}$$

Using the well-known inequality $\ln(x) \leq x - 1$ for $x \geq 0$, we have

$$H_L - H_G \leq \log_2 e \sum_{L=0}^{\infty} p_L\left(\frac{g_L}{p_L} - 1\right) = 0. \quad \text{Q.E.D.} \tag{26}$$

The maximum compression that can be achieved by any method which encodes the run-lengths of the source specified by (21) is

$$G_{\max} = \frac{\lambda}{H_L}. \tag{27}$$

Since $H_L \leq H_G$, we can see that if the distribution is non-geometric, it is possible to achieve greater compression for the same prediction error rate.

One obvious way of obtaining a compression close to $G_{\max}$ would be to construct a Huffman code for the run-lengths. Such Huffman codes are unattractive since they require table-lookup encoding and decoding and, in particular, the decoding requires considerable computation. We now develop a simple encoding scheme that we have found to be fairly efficient for the kind of non-geometric run-length distributions encountered in practice. The encoding is most readily understood as a generalization of Golomb's run-length encoding method and we refer to it as multi-mode Golomb encoding.

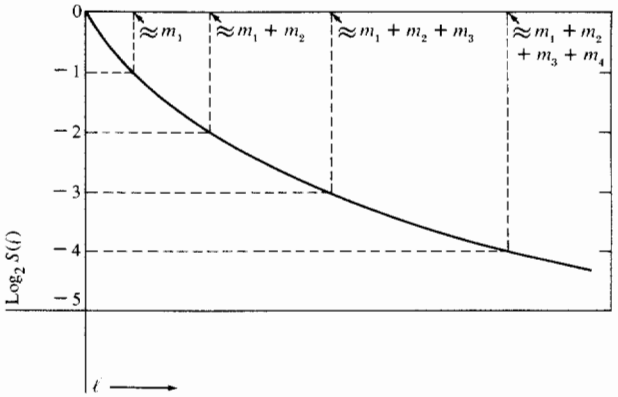

Figure 5 Selection of parameters $m_1, m_2, \cdots$ from $\log_2 S(\ell)$ curve.

For the arbitrary run-length distribution of (21), let us define a survivor function

$$S(\ell) = \Pr\{\text{run-length } L \geq \ell\} = 1 - \sum_{L=0}^{\ell-1} p_L,$$
$$\ell = 0, 1, 2, \cdots. \tag{28}$$

Now choose a sequence of integers $m_1 = 2^{N_1}$, $m_2 = 2^{N_2}$, $\cdots$, $m_k = 2^{N_k}, \cdots$ such that

$$\begin{aligned} &S(m_1) \approx 1/2, \\ &S(m_1 + m_2) \approx 1/4, \\ &\vdots \\ &S(m_1 + m_2 + \cdots m_k) \approx 1/2^k, \\ &\vdots \end{aligned} \tag{29}$$

Run-lengths L in the range $m_1 + m_2 + \cdots + m_{k-1} \leq L < m_1 + m_2 + \cdots + m_k$ are said to belong to group A_k and, as in the case of Golomb encoding, are encoded by a prefix $1^{(k-1)}0$, followed by an N_k-bit representation $L - (m_1 + m_2 + \cdots + m_k)$. A useful guide in constructing such a code is to plot $\log_2 S(\ell)$ against ℓ as in Fig. 5. The way to choose $m_1, m_2, \cdots, m_k, \cdots$ is self-explanatory from this curve. If the run-length distribution is geometric, then the curve of Fig. 5 becomes a straight line and we have $m_1 = m_2 = \cdots = m_k = \cdots = m$, where $p^m \approx \frac{1}{2}$, which leads to a Golomb code.

The encoding rule for the multimode Golomb encoder is similar to that of the usual Golomb encoder except that

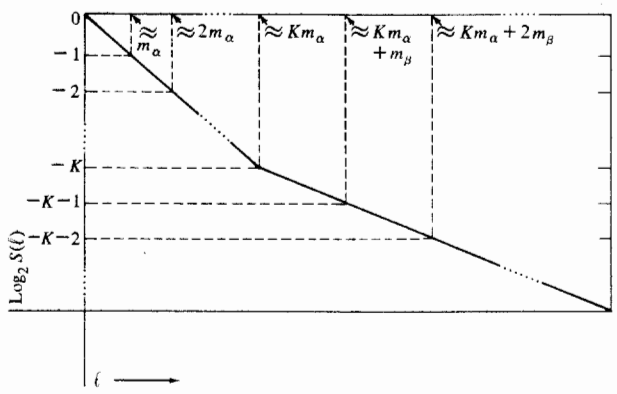

Figure 6 Selection of m_α, m_β, and K for typical $\log_2 S(\ell)$ curve.

we need to keep track of how many times the run-length counter has overflowed within a run. The modified encoding rule is

1) For each input of 0, the counter counts up by 1.
2) When the counter overflows for the kth time (i.e., when it reaches $m_k = 2^{N_k}$), a 1 is generated and the counter is reset to 0.
3) When a run terminates before the kth overflow is reached, an $(N_k + 1)$-bit codeword is generated. This codeword is a 0 followed by the N_k bits in the counter.

In practice, we have found that $\log_2 S(\ell)$ is fairly well approximated by two piecewise linear segments as shown in Fig. 6. The parameters of the multimode Golomb code are then chosen as follows:

$$m_k = \begin{cases} m_\alpha, & 1 \le k \le K; \\ m_\beta, & k > K. \end{cases} \tag{30}$$

The values of m_α, m_β are evident from Fig. 6. For brevity we denote such a code by the 3-tuple (m_α, m_β, K) which completely specifies the code.

Experimental results

Table 3 shows a comparison of the performance of Golomb and multimode Golomb codes on the three sample documents. For all three documents, the error pattern generated by the 4-pel fixed predictor was used. In the case of Golomb codes, we encoded the error pattern with codes having $m = 2, 4, 8, 16, 32, \cdots$ and chose the code that gave the highest compression. The multimode Golomb codes were designed by the method outlined in the previous section. The upper bound on compression as given by (27) is also included in the table. This result is roughly the compression that would have been obtained if we had encoded the run-lengths by Huffman codes. In the case of the journal page we find that the multimode Golomb code achieves roughly 87 percent of the maximum compression, whereas the usual Golomb code achieves 67 percent. Similar substantial improvement in compression is found for the other two documents.

Table 4 shows the actual compression obtained by a predictive coding system on the three sample documents. For each document we chose the three predictors [1] which gave the lowest prediction error rates and encoded the error pattern using a multimode Golomb code. Also included in the table is the upper bound on compression as given by (27) for each case. We find that the multimode Golomb codes achieve compressions roughly 80 to 90 percent of the upper bound.

In the case of the journal page we found the code (4, 64, 24) gave the best performance. For the jobshop charts the code (4, 128, 24) was better. However, the code (4, 64, 24) when used on the output of the 7-pel fixed predictor for the jobshop charts showed negligible deterioration from the (4, 128, 24) code. One could, therefore, use the same code over a wide range of documents with very small loss in efficiency.

Comments

In this paper we have considered the problem of compressing two-level black and white images. Our emphasis has been primarily on practical schemes that are easy to implement. There is, no doubt, room for improvement on the results presented here. We find that multimode Golomb encoding achieves roughly 80 to 90 percent of the upper bound for run-length encoding. It should be realized that the upper bound (27) is somewhat misleading because, in a single document, many of the possible run-lengths do not occur. A Huffman code constructed for the run-length distribution obtained from a single document would not assign codewords to nonoccurring run-lengths. However, in practice, it is necessary to design a code which assigns codewords to all possible run-lengths. The performance of any such code would then have to be inferior to the bound given by (27) for any particular document.

All the run-length distributions we encountered were monotonically decreasing, i.e., $P(L)$ decreases with increasing L. The independence assumption of the second section represents an attempt to model these distributions by a geometric distribution, for which we know simple and efficient encoding methods. In the third section we attempted to model the distribution as segments which are piecewise geometric. In particular, we concentrated on two-segment approximations. The advantage of this kind of approximation is that simple encoding and decoding algorithms can be constructed. Furthermore, the encoder and decoder can be varied by changing a few simple parameters, m_α, m_β and K. This would be ad-

Table 3 Comparison of Golomb and multimode Golomb encoding on error pattern generated by a 4-pel fixed predictor.

Document	Prediction error rate (percent)	Golomb code		Multimode Golomb code		Upper bound G_{max}
		Parameter m	Compression	Parameters (m_α, m_β, K)	Compression	
IBM Journal	5.85	8	2.93	(4, 64, 24)	3.81	4.36
Jobshop Chart A	2.37	16	5.75	(4, 128, 24)	6.59	8.15
Jobshop Chart B	1.35	16	8.81	(4, 128, 24)	12.33	15.15

Table 4 Experimental results for three sample documents

Document	Predictor	Prediction error rate (percent)	Multimode Golomb code		Upper bound G_{max}
			Parameters (m_α, m_β, K)	Compression	
IBM Journal	7-pel finite memory	4.79	(4, 64, 24)	4.33	4.82
	7-pel fixed	5.14	(4, 64, 24)	4.13	4.61
	4-pel finite memory	5.16	(4, 64, 24)	4.10	4.57
Jobshop Chart A	7-pel fixed	2.24	(4, 128, 24)	6.75	8.36
			(4, 64, 24)	6.73	8.36
	4-pel finite memory	2.30	(4, 128, 24)	6.68	8.25
	4-pel fixed	2.37	(4, 128, 24)	6.59	8.15
Jobshop Chart B	4-pel finite memory	1.20	(4, 128, 24)	12.92	15.71
	7-pel fixed	1.25	(4, 128, 24)	12.74	15.65
			(4, 64, 24)	12.68	15.65
	4-pel fixed	1.35	(4, 128, 24)	12.33	15.15

vantageous in a situation where one may wish to adaptively change the code depending on the data being processed.

It is evident that the efficiency of encoding could be increased by approximating $\log_2 S(\ell)$ by more than two piecewise linear segments. This, however, would increase the complexity of the encoder and decoder, which we feel is not desirable. We have also not considered the dependency between run-lengths. This refinement could be done by making a Markov model for the run-lengths and such a scheme might lead to higher compression. Some work along these lines is discussed by Arps [8]. Such schemes unfortunately lead to more complicated encoding and decoding.

Acknowledgments

We are grateful to Jacques Mommens for providing us with data on the performance of Huffman codes.

References

1. H. Kobayashi and L. R. Bahl, "Image Data Compression by Predictive Coding I: Prediction Algorithms," *IBM J. Res. Develop.* **18,** 164 (1974), preceding paper.
2. C. E. Shannon and W. Weaver, *The Mathematical Theory of Communication*, University of Illinois Press, 1949.
3. D. A. Huffman, "A Method for the Construction of Minimum Redundancy Codes," *Proc. IRE* **40,** 1098 (1952).
4. J. Capon, "A Probabilistic Model for Run-Length Coding of Pictures," *IRE Trans. Inf. Theory* **IT-5,** 157 (1959).
5. T. S. Huang, "Run-Length Encoding and its Extensions," in *Picture Bandwidth Compression*, edited by T. S. Huang and O. J. Tretiak, Gordon and Breach Science Publishers, Inc., New York, 1972, p. 231.
6. P. Elias, "Predictive Coding," *IRE Trans. Inf. Theory* **IT-1,** 16 (1955).
7. J. S. Wholey, "The Coding of Pictorial Data," *IRE Trans. Inf. Theory* **IT-7,** 99 (1961).
8. R. B. Arps, "Entropy of Printed Matter at the Threshold of Legibility for Efficient Coding in Digital Image Processing," Report No. 31, Stanford Electronics Lab., Calif., 1969.
9. S. W. Golomb, "Run-Length Encoding," *IEEE Trans. Inf. Theory* **IT-12,** 399 (1966).
10. H. Kobayashi and L. R. Bahl, "Image Compaction by Predictive Coding: Fundamentals," Research Report RC-3249, IBM Research Center, Yorktown Heights, New York 10598, February 1971.

Received September 19, 1973

The authors are located at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York 10598.