A Fast-Adaptive Huffman Coding Algorithm

Wei-Wei Lu and M. P. Gough

***Abstract*— The Huffman code in practice suffers from two problems: the prior knowledge of the probability distribution of the data source to be encoded is necessary, and the encoded data propagate errors. The first problem can be solved by an adaptive coding, while the second problem can be partly solved by segmenting data into segments. But the adaptive Huffman code performs badly when segmenting data into relatively small segments because of its relatively slow adaptability. This paper offers a fast-adaptive coding algorithm which tracks the local data statistics more quickly, thus yielding better compression efficiency.**

I. Introduction

SUPPOSE a data compression scheme is to be chosen to encode a data source driven by a known probability distribution, and the channel is noiseless; the Huffman code [1] may be one of the best choices. Indeed, the Huffman code has been proven capable of encoding a data source with the bit rate very close to its entropy, provided its statistics are known in advance. The excellent performance is due to the following encoding strategy: the symbols with rare occurrence are represented by the longer codewords, and those with frequent occurrence are represented by the shorter codewords. This results in a minimum bit rate with which the data source is encoded. However, in practice, the Huffman code suffers mainly from two problems.

One problem is that the statistical model of the data source must be known before the design of the Huffman code. The better the model and the data source agree, the better the Huffman code performs. A bad model can lead to a low compression efficiency, sometimes even worse than no compression at all. This is a very real problem because in many applications, the prior knowledge of the statistics of a data source is impossible, or the statistics are changeable with time. However, there is an adaptive coding, known as the adaptive Huffman code [2], which deals with this problem. In the adaptive coding a fixed model is no longer used. Instead, a counter is set up for each symbol of the source alphabet, and the count is initially set to 1 or to an expected occurrence frequency. Each time a symbol occurs, its count increases by 1, and the code tree is then updated to fit the accumulative counts which approximately represent the local statistics of the data source. Every time a certain amount of data has been encoded, equivalent to a time constant, each count is multiplied by some fixed factor, which was suggested equal to 0.5 in [2]. Since both sender and receiver update their own code tree based on the same previous data sequence, the codes used by both sides always agree.

The other problem is that the compressed data propagate errors. Suppose an error bit occurs during the transmission of the compressed data; this error bit will cause the receiver to mismatch a codeword. The mismatched codewords are likely to have different lengths and cause the desynchronization of the bit streams on both sides of the channel—this will be maintained until resynchronization occurs. Segmentation of the data into segments consisting of a fixed number of data is an obvious solution whereby the error propagation is limited within the range of a segment. A related discussion can be found in [3].

But a new problem of low compression efficiency immediately arises when the adaptive coding is applied to segmented data. In fact, even with the adaptive coding, there is still a low compression efficiency at the beginning of the data because not enough statistical information has been gained to establish a good model. However, the problem becomes more serious when segmenting data into segments. Since every segment cannot use the previous segment's model, which could have been corrupted by channel noise, initialization is basically necessary for every segment, resulting in a large percent of data encoded with lower compression efficiency.

This paper presents a new algorithm tracking the local statistics more quickly than the above adaptive coding. The new algorithm employs two trees: the front tree and the back tree. The front tree is an adaptive Huffman code, except the number of leaves (codewords) is a variable, while the back one is an symmetric or almost symmetric tree. Each symbol is encoded only in one tree and moves dynamically between two trees, according to the occurrence frequency. The symbols with higher frequency have more chances to be encoded in the front tree, and those with lower frequency often stay in the back tree. The faster adaptability is made possible by the relatively small size of the front tree, and so the data are more efficiently encoded.

The structure of the paper is as follows. We state the algorithm in the next section. The experimental results are shown in Section III. A summary is given in the last section.

II. Description of the Algorithm

The algorithm sets up two trees, the front tree and the back tree, to encode symbols. Each symbol is encoded only in one tree and moves dynamically between two trees. The front tree is an adaptive Huffman code, while the back one is a binary code. Initially, all symbols are put in the back tree, and the front tree has only a special codeword W. After a symbol occurs and is encoded in the back tree, it is then

Paper approved by the Editor for Image Communications Systems of the IEEE Communications Society.

Manuscript received May 21, 1991; revised August 30, 1991.

The authors are with the School of Engineering and Applied Sciences, University of Sussex, Fulmer, Brighton BN1 92T, England

IEEE Log Number 9209483.

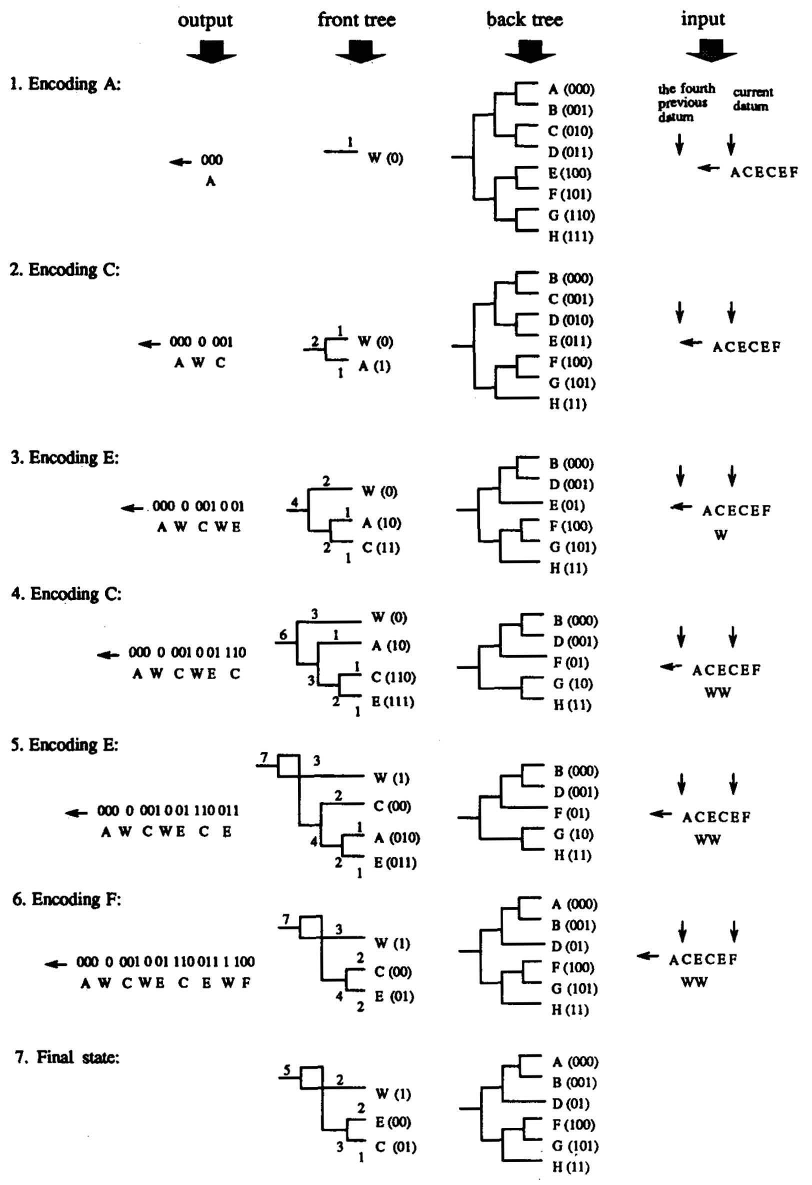

Fig. 1. An example of the fast-adaptive Huffman encoding.

taken to the front tree. The occurrence frequency of every symbol is counted, and the front tree is updated to fit the occurrence frequencies after each datum is encoded. After a certain number N (time constant) of data has been encoded, each time a datum is encoded, the Nth previous datum has to be removed from the front tree, i.e., the corresponding symbol's frequency number is decreased by 1. A symbol will be taken back to the back tree if its frequency number decreases to zero. A special codeword W is always in the front tree; it is used to identify the back codewords by preceding them during transmission.

An example is given in Fig. 1. Assume that the sample space consists of eight symbols $A-H$, the time constant is 4, and the data sequence is $ACECEF$.

III. The Experimental Performance

The performance of the fast-adaptive Huffman code has been tested on three types of data, compared with the adaptive coding [2]. The first set of experimental data is a part of a scientific paper. It consists of 1024 characters, including English letters, the Arabic numerals, space character, and ⟨line feed⟩, represented as ASCII text, requiring 7 b/character. The second set of experimental data is space-acquired science data. It consisted of 1024 samples which are positive integers, requiring 8 b/sample. The third set of experimental data is 1024 random integer numbers with Normal distribution. The data range from 0 to 255, requiring 8 b/number. In our experiment, all symbols of the source alphabet are initially assumed to be equally likely. The first experiment was done to test the effect of different time constants on the compression efficiency. The first and second sets of data were encoded by the codes with different time constants, such as 64, 128, 256, 512, and 1024. The result is shown in Fig. 2. It is shown that for the first set of data, the fast-adaptive and the adaptive Huffman codes yield the best compression efficiency with time constants equal to 512 and 1024, respectively, and for the second set of data, the best compression efficiency is yielded at time constants equal to 128 and 1024, respectively. In both cases, the optimal time constant of the fast-adaptive Huffman code is smaller than that of the adaptive coding. However, in order to unify parameters for comparison, in the following experiments we set the time constant of both codes to 1024. The second experiment was done to test the performance of the fast-adaptive Huffman code in both compression efficiency and adaptability. The first and second sets of data were segmented and encoded with different segmentations, such as 32 data/segment, 64 data/segment, 96 data/segment, and so on, culminating with 1024 data/segment. The codes were initialized for each segment. The results are shown in Figs. 3 and 4. Both Figs. 3 and 4 show that either in the case of a short segment or a long segment, the fast-adaptive Huffman code yields a lower bit rate than the adaptive Huffman code. The better performance on short segments means that the fast-adaptive coding adapts the local data statistics more quickly than the adaptive coding, while the better performance on long segments means that the fast-adaptive coding compresses data more efficiently than the adaptive one. Quantitatively, the fast-adaptive Huffman code is at least 2.5 times faster to adapt than the adaptive coding (32 characters versus 80 characters at the same bit rate) for English text encoding, and at least 11 times faster to adapt than the adaptive coding (32 samples versus 352 samples at the same bit rate) for the space-acquired data encoding. Also, the fast-adaptive Huffman code reduces the bit rate by between 3.6% (the bit rate is reduced from 4.81 to 4.63) and 11.2% (reduced from 6.25 to 5.55) for English text encoding, and by between 5.9% (reduced from 5.54 to 5.20) and 22.0% (reduced from 7.49 to 5.83) for space-acquired data encoding. The third experiment was done to examine the relationship between the compression efficiency and the dispersion of the data encoded. The third set of data with a different standard deviation was encoded. The result is shown in Fig. 5. It is shown that the compression performance of the fast-adaptive coding is affected by the dispersion of the data. The less the data disperse, the higher the compression efficiency is (i.e., the bit rate is closer to the entropy of the data).

IV. Summary

A fast-adaptive Huffman code algorithm is presented. Our experimental results indicate that:

1) the fast-adaptive Huffman code requires a smaller time constant to yield the best performance compared to that required by adaptive coding;
2) for the English text encoding, the new algorithm is 2.5 times faster to adapt and yields from 3.6% to 11.2% fewer bit rates than adaptive coding;

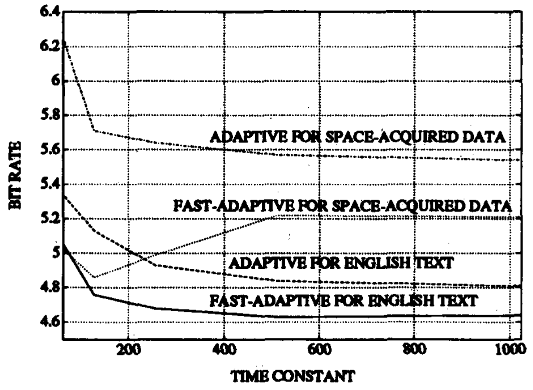

Fig. 2. The bit rate as a function of the time constant for the encoding of the English text and the space-acquired data.

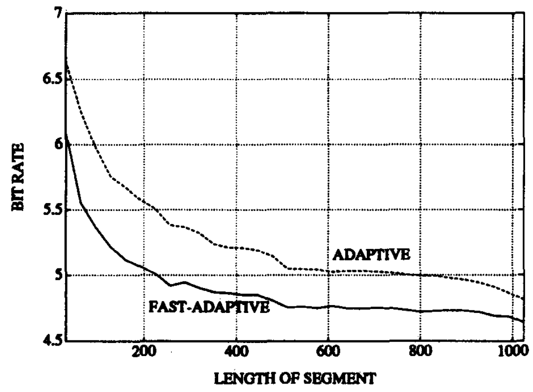

Fig. 3. The bit rate as a function of the length of segment for the encoding of the English text.

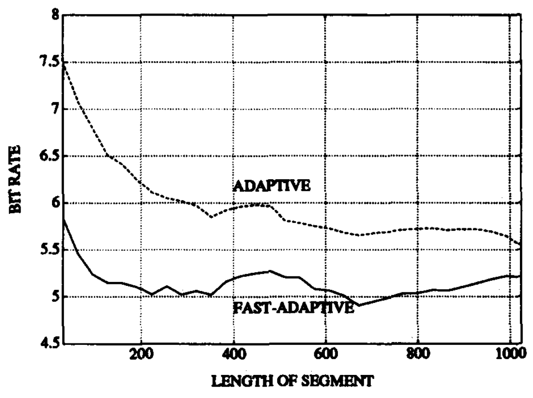

Fig. 4. The bit rate as a function of the length of segment for the encoding of the space-acquired data.

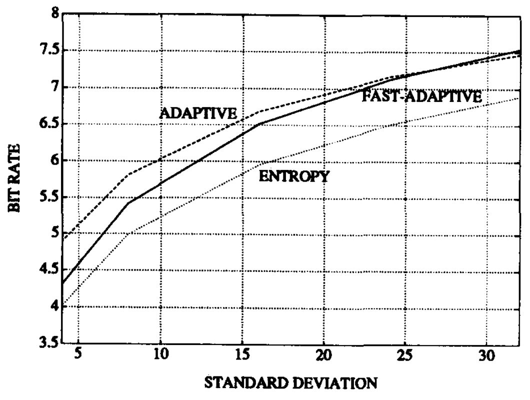

Fig. 5. The bit rate as a function of the standard deviation for the encoding of the random integer numbers with Normal distribution.

3) for the space-acquired data encoding, the new algorithm is 11 times faster to adapt and yields from 5.9% to 22.0% fewer bit rates than the adaptive one; and
4) the compression efficiency of the new algorithm is affected by the dispersion of the data to be encoded. The less the data disperse, the higher the compression efficiency is.

The fast adaptability of the new algorithm is more significant when segmenting data into segments to limit the propagation of data errors caused by channel noise.

REFERENCES

[1] D. A. Huffman, "A method for the construction of minimum-redundancy codes," *Proc. IRE,* vol. 40, pp. 1098–1101, Sept. 1952.
[2] R. G. Gallager, "Variations on a theme by Huffman," *IEEE Trans. Inform. Theory,* vol. IT-24, pp. 668–674, Nov. 1978.
[3] K. M. Rose, and A. Heiman, "Enhancement of one-dimensional variable-length DPCM images corrupted by transmission errors," *IEEE Trans. Commun.,* vol. 37, pp., 373–379, Apr. 1989.