Active and Passive Elec. Comp., 2001, Vol. 24, pp. 211–221
Reprints available directly from the publisher
Photocopying permitted by license only

A SIMPLE BINARY RUN-LENGTH COMPRESSION TECHNIQUE FOR NON-BINARY SOURCES BASED ON SOURCE MAPPING

ABDEL-RAHMAN JARADAT[a,*] and MANSOUR I. IRSHID[b]

[a]*Computer Engineering Department, University of Sharjah, Sharjah – United Arab Emirates;*
[b]*Electrical Engineering department, Jordan University of Science & Technology, Irbid – Jordan*

(Received 15 July 2001; In final form 22 August 2001)

In this paper, we propose a very simple and efficient binary run-length compression technique for non-binary sources. The technique is based on mapping the non-binary information source into an equivalent binary source using a new fixed-length code instead of the ASCII code. The codes are chosen such that the probability of one of the two binary symbols; say zero, at the output of the mapper is made as small as possible. Moreover, the "all ones" code is excluded from the code assignments table to ensure the presence of at least one "zero" in each of the output codewords. Compression is achieved by encoding the number of "ones" between two consecutive "zeros" using either a fixed-length code or a variable-length code. When applying this simple encoding technique to English text files, we achieve a compression of 5.44bits/character and 4.6bits/character for the fixed-length code and the variable-length (Huffman) code, respectively.

1 INTRODUCTION

The main objective of researchers working in the field of data compression is to design simple and efficient data compression algorithms. The implementation complexity, the large execution time, and the

*Corresponding author. E-mail: jaradatsharjah.ac.ae
On leave from Jordan University of Science & Technology, Irbid, Jordan
E-mail: mabbadijust.edu.jo

large memory size found in many popular source encoders are due to the large number of symbols in the alphabet of the original information sources (256 symbols for text, sound, and image) [1,2]. On the contrary, information sources with small number of alphabet such as black-and-white images has very simple source encoding techniques and they are very suitable for hardware implementation. For such small alphabet sources, run-length encoders are found to be efficient and very simple to be implemented in hardware [4].

In run-length encoders, the sequences of consecutive identical symbols are replaced by the length of the run where a special symbol is reserved as a flag indicating that the next symbol contains the length of the run. Run-length encoding can be applied to any information source provided that it contains large number of sequences of consecutive identical symbols in order to achieve good data compression. The technique is very useful and very easy to implement in the case of binary sources such as black-and-white pictures where long sequences of black and white pixels can be found [3,5]. To exploit the simplicity of bit-wise run-length encoders, non-binary sources can be converted to binary sources and then the resulting binary source is applied to the bit-wise run-length encoder. For text sources, it is found that this approach is not useful if the conversion process is done arbitrarily such as in ASCII text [1]. Methods have been proposed to reassign the character codes so that this approach would be effective. The idea of reassigning character codes so that they are suitable for run-length encoding was proposed by Lynch [1]. A large number of consecutive sequences of "ones" and "zeros" can be obtained by assigning the "all ones" and the "all zeros" codes to the two most common characters of the English text and other codes are assigned in a similar way [6].

In this paper, we presents a simple method of assigning the codes to the symbols of the non-binary information source. The codes are chosen such that the probability of occurrence of symbol "zero" is made as small as possible and this will results in a large consecutive sequences of symbol "one" and a rare sequences of symbol "zero". Applying the binary run-length encoder to the resulting binary source is found to be very effective both in implementation simplicity and in compression efficiency. In Section 2, the proposed source mapping process is discussed. In Section 3, a fixed-length binary run-length encoder applied to the resulting binary source is studied. In Section 4,

a variable-length binary run-length encoder is presented. Conclusions are given in Section 5.

2 SOURCE MAPPING

The first stage of encoding in this proposed source encoding technique is to map the original non-binary information source onto an equivalent binary source using an N-bits fixed-length binary encoder. The mapping process is done such that it satisfies two conditions: first; the codes are chosen such that the probability of one of the two binary symbols at the output of the mapper; say zero, is made as small as possible, secondly; the "all ones" codeword is excluded from the code assignments table to ensure the presence of at least one "zero" in each codeword and limits the run length of symbol "one" to $(2N-2)$ where N is the code length of the mapper. The first condition can be satisfied if the codes with the largest Hamming weights (Hamming weight of z is defined to be the number of non-zero components (ones) of z [7]) are assigned to the symbols with largest probability of occurrence in a descending order. For a mapper with an N-bit code length, codes having a Hamming weight of $(N-1)$ are assigned to the first N symbols, codes having a Hamming weight of $(N-2)$ are assigned to the next $N(N-1)/2$ symbols, or, in general, codes having a Hamming weight of $(N-m)$ are assigned to the next $N!/m!(N-m)!$ symbols. Such a mapping process results in a binary source having a small probability for symbol "zero" and, of course, a large probability for symbol "one" provided that the original source has a non-uniform probability distribution as it is the case in many information sources. Interestingly, it is found that the entropy of the resulting binary source multiplied by N is found to be very close to the entropy of the original information source. On the contrary, if the mapping process is done in an arbitrary way such as in the case of ASCII code, the resulting probabilities of zero and one are almost equal and the entropy of the resulting binary source is far from that of the original source.

Let the original information source S has an alphabet of $M < 2^N$ symbols with probabilities $\{p_0, p_1, p_2, \ldots\}$ arranged in descending order of their magnitude, and let the equivalent binary source B has a

probability of q for symbol "zero" and a probability of $(1-q)$ for symbol "one". The entropy of the original source is given by:

$$H(S) = -\sum_{i=0}^{M-1} p_i \log_2(p_i) \tag{1}$$

When the mapping process is done according to the above mentioned mapping rule, the probability of symbol "zero" of the resulting binary source can be determined as follows: multiply the number of zeros in each codeword by the probability of its corresponding symbol, then take the sum over all symbols and divide by N which is the length of the codeword. Since the number of zeros can take one of N values ranging from 1 to N, then the codewords can be grouped into N groups with the ith group composed of $\binom{N}{i}$ codewords each with i zeros. Therefore, the probability of symbol "zero" is found by taking the summation of the probabilities of the symbols in each group multiplied by i/N and then taking the summation of the results over the N groups as it is shown by the following relation:

$$q = \sum_{i=1}^{N} \frac{i}{N} \sum_{k=c(i-1)+1}^{c(i)} p_{k-2} \tag{2}$$

where $c(i) = \sum_{j=0}^{i} \binom{N}{j}$ and $\binom{N}{j} = N!/j!(N-j)!$

Accordingly, the entropy of the binary source is given by:

$$H(B) = -q\log_2(q) - (1-q)\log_2(1-q) \tag{3}$$

An ideal source mapper is one that results in a binary source whose entropy is equal to the entropy of the original information source divided by N:

$$H(B) = \frac{H(S)}{N} \tag{4}$$

This ideal mapping can be achieved under special conditions, the straight forward case is when the original information source generates symbols having equal probabilities. For other non-uniform informa-

tion sources the entropy of the resulting binary source is higher than the entropy of the original source divided by N:

$$H(B) > \frac{H(S)}{N} \tag{5}$$

The difference between the entropy of the Nth order extended binary source and the entropy of the original source $\{NH(B) - H(S)\}$ is near minimum when the codewords are assigned according to the above mentioned mapping rule. The minimum entropy for the resulting binary source can be achieved if the "all ones" codeword is not excluded from the code table. This is because the entropy of the binary source $H(B)$ decreases as q decreases ($q < 0.5$), and the smallest value of q is achieved only if the mapping process is done according to the above mentioned mapping rule. A source mapper for the English text (less than 256 symbols) is designed according to the proposed mapping rule. Table I shows the probabilities and the codes which are assigned to the different characters of the English text (not all of them are shown in the table). The probabilities for the characters shown in Table I were obtained by the authors via computing the occurrences of the characters in the text file 'WWEND10.TXT (a novel written by William Morris entitled: The Well at the Worlds End) of size 1.186 MBytes taken from the Gutenberg Project. The calculated probabilities are consistent with those found in [1]. The same text file 'WWEND10.TXT was used as the source of the test data for checking

TABLE I The Probabilities and the New Assigned Codes for the English Text Alphabet.

Character	*Probability*	*Assigned code*
Space	*0.174*	*11101111*
e	0.098	11110111
t	0.070	11111101
a	0.062	11011111
o	0.059	11111011
i	0.055	10111111
n	0.055	01111111
s	0.050	11111110
r	0.047	01111110
h	0.042	01111101

the proposed technique in this paper. The entropy of the original source is calculated using Eq. (1) and it is found to be 4.47 bits/character. The probabilities of symbol "one" and symbol "zero" of the resulting binary source are found to be 0.83 and 0.17 respectively, and the entropy of the 1st-order binary source is found to be 0.66 bits/symbol. The entropy of the 8th-order extended binary source is 5.28 bits/symbol compared to 4.47 bits/character for the original text source. If an ASCII code is used as a source mapper, the probability of symbol "one" and symbol "zero" of the resulting binary source are found to be 0.44 and 0.56, respectively. The entropy of the resulting 1st-order binary source is 0.99 bits/symbol and the entropy of the corresponding 8th-order extended binary source is found to be 7.9 bits/symbol which is very far from that of the original text source.

3 FIXED-LENGTH BINARY RUN-LENGTH SOURCE ENCODER

The second stage of encoding is to apply one of the well-known encoding techniques to the binary source resulting from mapping process discussed in Section 2 such as Huffman, Lempel-Ziv, run-length, etc. The proposed binary run-length encoder is extremely simple, and no flag is needed to indicate the length of the run as in conventional run-length encoder. The encoder is a simple counter, which counts the number of "ones" between two "zeros" (NOBTZ), and then encode the resulting decimal number using an m-bit fixed-length binary code. Since we exclude the all "ones" code from the assignment code table, then the number of "ones" between two consecutive "zeros" will take one of the following values; $0, 1, 2, \ldots, (2N - 2)$, and therefore, the length m of the required fixed-length binary code is the ceiling of $\log_2(2N - 1)$. The number of time m-bit blocks generated at the output of the encoder (counter) is equal to the number of "zeros" in the data at the input of the encoder, and therefore, the size of the output compressed data in bits is equal to qmN K bits where K is the number of the original symbols in the file, while the size of the input data is N K bits. Therefore, the data is compressed to (qm) times its original size. It is obvious that decreasing the probability of symbol "zero" at the output of the mapped source will decrease the size of the compressed

file. The probability of "zero" depends on both the probability distribution of the symbols of the original source and on the mapping strategy as it is discussed in Section 2.

Taking the English text as an example, the source mapper is an 8-bit ($N = 8$) fixed-length binary encoder and the mapping table is that of Table I. In this case, there are maximum $2N - 2 = 14$ ones between two zeros, and with the zero run makes 15 different run-lengths. Therefore, we need a 4-bit ($m = 4$) fixed-length encoder (counter) to encode these "ones". Since the probability of "zero" for the English text is found in section 2 to be 0.17, then the resulting file has a size of $0.17 \times 4 = 0.68$ times the size of the original file, or in other words, the average number of bits needed to encode each of the English characters is 5.44 bits/character compared to 8 bits/character for the uncompressed file.

The proposed fixed length binary run-length encoder can be easily implemented either in software or hardware. A block diagram for the hardware implementation is shown in Figure 1. Each of the original characters coming in a serial form is binary encoded into an N-bit fixed length code generated by a ROM memory which contains the assigned codes such as those shown in Table I for the English text. A parallel-to-serial converter is used to convert the data generated from the ROM into serial form. The resulting data is applied to an m-bit binary counter, which counts the number of "ones" between two consecutive "zeros". The counter is designed such that its content are read into a latch at the leading edge of the coming "zero" pulses and reset at their trailing edges to be ready for the next round of counting. Another parallel-to-serial converter is needed to convert the parallel data from the counter into serial form. If the symbol rate of the original signal is K symbol/second, then the bit rate at the output of the first parallel-to-serial converter is NK bits/second and at the output of the second parallel-to-serial converter is $qmNK$ bits/second. As an

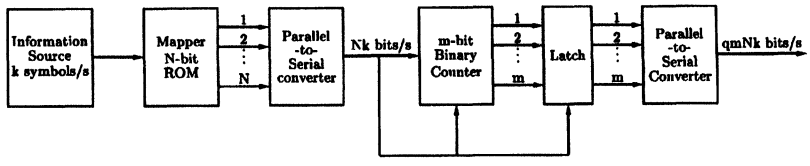

FIGURE 2 Fixed-length binary run-length decoder.

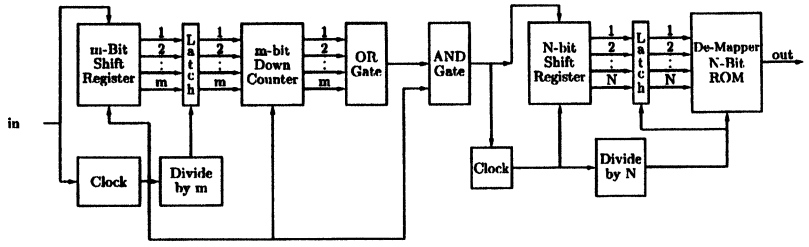

FIGURE 1 Fixed-length binary run-length encoder.

example, if an English text is applied to this encoder with a symbol rate of 1000 character/second, then the bit rate at the output of the first parallel-to-serial converter is 8000 bits/second and at the output of the second parallel-to-serial converter is 5440 bits/second.

Figure 2 shows the decoder for this proposed source encoding scheme. It consists of an m-bit shift register, which receives the compressed data as m-bits blocks. The content of the register is transferred to a buffer at the end of each block. The buffer content is used as a preset value for an m-bits countdown counter which counts at a rate equal to that of the incoming data rate. A simple logic circuit produces an output of "one" at each clock pulse as long as the content of the counter is nonzero, and once its content is zero, an output of one "zero" is produced by the logic circuit. The serial output of the logic circuit is divided into 8-bits blocks, and a ROM is used as demapper, which maps the 8-bit code into the original signal code (usually ASCII code).

4 VARIABLE-LENGTH BINARY (HUFFMAN) RUN-LENGTH SOURCE ENCODER

When studying the statistics of the binary data at the output of the counter, It is found that the probability distribution of the number of "ones" between two "zeros" (NOBTZ) is non-uniform. This suggests further compression of the original data by encoding the $(2N-1)$ "ones" by variable-length codes such as Huffman codes instead of the fixed-length codes discussed in Section 3. It is also found that shuffling the assigned codes within the groups with the same Hamming weight

in the mapping process discussed in Section 2 results in a different statistics for the NOBTZ. This variation in the statistics is due to the dependencies between successive letters in the text. Our objective is to look for the optimum mapping process, which gives the minimum entropy for the NOBTZ source to achieve more compression efficiency. The search for this optimum mapping process has to be done subjected to the condition that the probability of symbol "zero" is to be kept constant according to the rule discussed in Section 2. This condition will limit the shuffling process among the codes, which have the same Hamming weight only. For the N-bit mapper of Section 2, there are N groups with the ith group having Hamming weight of $(N-i)$ where $(i=1,2,\ldots,N)$, and the number of elements in each group is given by the binomial coefficient $N!/i!(N-i)!$. Since there is an extremely large number of combinations for such shuffling process, we restrict it to the first eight significant assigned codes, which have the Hamming weight of seven, and having the highest probability of occurrence. A program was written to find the statistics and the entropy of the NOBTZ using the mapping process discussed in section 2 and searching all possible combinations ($40320 = 8!$) which can be obtained from shuffling (re-assigning) the first eight codes (assigned codes with only one zero) to the characters having the eight highest probabilities. The maximum entropy resulted from the shuffling was 3.66 bits/symbol while the minimum was 3.35 bits/symbol. The combination that produced the smallest entropy was used to determine the Huffman codes from the statistics of the NOBTZ using Huffman encoding algorithm. For English text compression, Table II shows the probabilities, the fixed-length codes, and the Huffman codes for the NOBTZ having the minimum entropy of 3.35 bits/symbol. The average codeword length for NOBTZ source is calculated by multiplying the lengths of the Huffman codes by their corresponding probabilities as shown in Table II. The final average codeword length using the variable-length code is found to be 4.6 bits/character for the minimum case and 5.0 bits/character for the maximum case compared with 5.44 bits/character for the fixed-length code of Section 3. This means that the shuffling process resulted in an improvement of 0.4 bits/character and this improvement is due to the correlation between two consecutive characters. This compression efficiency is comparable to that resulting from applying Huffman compression

TABLE II Probabilities of the "NOBTZ" with the Corresponding Fixed-Length and Huffman Codes.

NOBTZ	*Probabilities*	*Fixed-length codes*	*Huffman codes*
0	0.0617	0000	0111
1	0.0607	0001	1000
2	0.0898	0010	0000
3	0.1043	0011	101
4	0.1531	0100	001
5	0.1516	0101	010
6	0.1803	0110	11
7	0.0758	0111	0001
8	0.0207	1000	10011
9	0.0306	1001	01101
10	0.0430	1010	01100
11	0.0177	1011	100100
12	0.0071	1100	1001010
13	0.0028	1101	10010110
14	0.0008	1110	10010111

technique directly to the original English text file ('WWEND10.TXT) which resulted in 4.52 bits/character. However, the proposed technique uses only 15 Huffman codes compared to 128 Huffman codes in the direct method, which results in decreasing the execution time. The hardware implementation of the variable-length Huffman encoder can be done in a similar way as in the case of fixed-length encoder. A Huffman encoder and decoder have to be added to the run-length encoder and decoder. The major problem in this implementation is that the data rate at the output of the encoder is variable which complicates the synchronization between the transmitter and the receiver.

5 CONCLUSIONS

A very simple and efficient binary run-length source encoder is proposed. Unlike conventional run-length encoder, the proposed encoder can be applied to any non-binary information sources having non-uniform probability distribution. The technique is based on transforming the original non-binary information source onto an equivalent binary source with a high probability of occurrence for symbol "one". Data compression can be achieved by counting the number of

"ones" between two "zeros" (NOBTZ) and encoding the resulting number by either fixed-length codes or variable-length codes. When applying this technique to English text files, compressed files with 5.44 bits/character and of 4.6 bits/character are achieved using fixed-length codes and variable-length (Huffman) codes, respectively. The variable-length code uses 15 Huffman codes compared to 128 codes in the conventional Huffman technique.

References

[1] Bell, T. C., Cleary, J. G. and Witten, I. H. (1990). *Text Compression*. Prentice-Hall, Englewood cliffs NJ.
[2] Held, G. and Marshall, T. R. (1991). *Data Compression*. John Wiley, New York.
[3] Jones, C. B. (1981). An efficient coding system for long source sequences. *IEEE Trans. Information Theory*, **27**, 280–291.
[4] Langdon, G. G. and Rissanen, J. J. (1981). Compression of black-white images with arithmetic coding. *IEEE Trans. Communications*, **29**, 858–867.
[5] Langdon, G. G. and Rissanen, J. J. (1982). A simple general binary source code. *IEEE Trans. Information Theory*, **28**, 800–803.
[6] Lynch, M. F. (1973). Compression of bibliographic files using an adaptation of run-length coding. *Information Storage and Retrieval*, **9**, 207–214.
[7] McEliece, Robert. (1977). *The Theory of Information and Coding*. Addison-Wesley, Reading, Massachusetts.