Australian Journal of Basic and Applied Sciences, 4(12): 6564-6569, 2010
ISSN 1991-8178

An Efficient Text Compression Technique Based on Using Bitwise Lempel-Ziv Algorithm

[1]Ahmed S. Musa, [2]Ayman Al-Dmour and [3]Mansour I. Irshid

[1]Department of Computer Engineering Al-Huseein Bin Talal University Ma'an – Jordan
[2]Department of Information Technology Al-Huseein Bin Talal University Ma'an – Jordan
[3]Department of Electrical Engineering Jordan University of Science and Technology Irbid – Jordan

Abstract: This paper presents an efficient data compression technique based on using Lempel-Ziv coding algorithms such as the LZ-78 algorithm. The conventional LZ-78 algorithm was applied directly to a non-binary information source (i.e., original source) with a large number of alphabets (such as 256 characters in English text). However, this modified technique applies the LZ-78 algorithm to an nth–order extension equivalent binary source which includes 2^n symbols. The equivalent binary source is generated by applying an efficient source encoding scheme on the original one. In this encoding scheme, each alphabet in the original source is given a weighted fixed-length code (e.g. eight bits/character). The weighted codes are chosen in such a way that the entropy of the generated binary source is made as close as possible to that of the original one. The reduction in number of symbols in the extended binary source leads to a better compression algorithm performance which can be measured based on the algorithm implementation complexity, memory usage, compression-decompression speed, and compression ratio. Analysis and simulation results obtained based on using the modified compression technique show that the bitwise LZ-78 encoder of the fourth-order extended binary source, which includes 16 symbols, achieves compression efficiency close to that of the conventional LZ-78 encoder, which includes 256 symbols.

Keywords: Lossless Data Compression, Lempel-Ziv 78 algorithm, Text Compression, Source mapping, ASCII code, EBCDIC code.

INTRODUCTION

Recently, many applications such as teleconferencing, environmental remote sensing, word processing programs, etc., have demanded large capacity storage devices along with high bandwidth transmission channels. To meet these massive demands and to enhance higher data store and transfer capabilities, the service providers, organizations, publishing companies, continuously upgrade their equipment and install new transmission channels. (Held and Marshall, 1991). As a matter of fact, the deployment of additional infrastructure (installation, operation, and maintenance) requires additional capacity that is more costly. Therefore, in most instances, service providers rely on new technologies which increase the amount of data that the current infrastructure can handle.

Data compression is a promising technology that is used to enhance the existing infrastructure and to enable the aforementioned applications. Thus, a potential cost saving can be achieved by utilizing a high performance data compression technique. Moreover, several benefits can be achieved from compressing digital data:

1. The effective speed of transmission will increased. The benefit can be realized when downloading a large file via internet connection, where approximately a saving of one-quarter of time is possible when the file is in its compressed format.
2. The security against illicit monitoring can be achieved since the original data, usually represented in American Standard Code for Information Interchange (or ASCII), is converted into a different code (Held and Marshall, 1991).

Data compression encompasses a wide variety of techniques and algorithms. These algorithms are applied to a variety of problems that might be experienced when extremely large amounts of data are stored or

Corresponding Author: Ahmed S. Musa, Department of Computer Engineering Al-Huseein Bin Talal University Ma'an – Jordan
E-mail:shorman@ahu.edu.jo

transmitted (Weiss and Shremp, 1993). A question arises at this juncture: What is the most appropriate compression technique that can be used to achieve high compression efficiency? As described in (Bell *et al.,* 1990), compression ratio, which is given by the average number of bits needed to represent one character from the original source (number of bits per character or bits/character), is not the only metric that can be used to select the compression algorithm. Other metrics such as the processing time (speed) and the memory size, along with the implementation complexity, play a major role in improving the compression algorithm efficiency.

The aforementioned metrics (required memory, speed, compression ratio, etc.) are dramatically increased when the number of alphabets or symbols in the original information source (256 symbols for English text, Sound, and image) is large (Held and Marshall, 1991; Elabdalla and Irshid, 2001; Bell *et al.,* 1990; langdon and Rissanen, 1981). Therefore, to reduce the metric size, the number of symbols in the source must be reduced. For instance, an information source with a binary alphabet such as a black-white image has a simple source encoding technique that simplifies their hardware implementation (langdon and Rissanen, 1981; 1982).

In this research, an efficient text compression technique based on using Ziv and Lempel compression algorithms is proposed. In this devised technique, each character in the input text file is given a new weighted fixed-length binary code similar to the ASCII, Extension Binary Coded Decimal Interchange Code (EBCDIC), etc. This fixed-length binary code is chosen in such a way that makes the first-order entropy of the resulting binary source, H(B), multiplied by the code length, N, as close as possible to that of the input text file, H(S). Afterwards any compression technique such as LZ-78 algorithm will be utilized to manipulate the resulting binary file on a bitwise basis in lieu of the bytewise basis (or character basis).

Source Entropy and Mapping Model:

In any existing language, each character has a frequency of occurrence different from any other character. In this article, attention is centered on the English language and the probability distribution of its own characters. Based on the probability distribution and statistics of the language characters, a new source mapping technique is proposed to map the input text file into a binary file.

In the proposed mapping technique, each character in the source file (e.g., typical English text file) is given a unique fixed-length binary codeword. This codeword is like ASCII or EBCDIC one which is an eight-bit length code. The codeword is assigned based on the application of the following rule: The fixed-length binary codewords with the largest Hamming weights (i.e., the largest number of "ones") are assigned to the characters with the largest frequency of occurrence. In general, codewords with a Hamming weight of (N-m) are assigned to the characters:

$$\binom{N}{m} = \frac{N!}{(N-m)!m!} \tag{1}$$

Where N is the length of the assigned binary code and m is ranged from 0 to N.

According to the aforementioned rule, the all "ones" in a eight-bit binary codeword, whose Hamming weight equals to eight, is assigned to the character that has a large probability of occurrence, p_0, (e.g., space character, denoted by `•`, in English text file). In the same way, the next most probable characters are given codewords having the maximum Hamming weight (e.g., codewords with a Hamming weight of seven are assigned to the next eight characters that possess a high frequency of occurrence). This assignment is repeated until the character with the smallest frequency of occurrence is assigned a codeword with the lowest Hamming weight (i.e., zero).

The assignment process results in a binary information source having two symbols (0 and 1). The probability of occurrence of each zero and one symbol reflects the degree of correlation between consecutive characters in the original source. On the contrary, if the assignment process had been done in an arbitrary way such as ASCII or EBCDIC the resulting binary file will not reflect the statistics of the original source. Therefore, this proposed technique is the key contribution of this work.

Table 1 lists the statistics of some individual characters available in English text files and the code assigned to each character based on previously defined rule along with its corresponding ASCII code given in (Mano, 1984). The statistics of these characters are gleaned from a large body of written English texts known as the Brown Corpus (Bell *et al.,* 1990). Based on the probability of each character given in Table 1, the entropy of the original source, H(s), is found to be approximately 4.47 bits/character (Elabdalla and Irshid, 2001; Abdel-Rahman *et al.,* 2000; 2001). On the other hand, if the entropy is measured based on the resulting binary file, it is found that the probabilities of zero and one are 0.147 and 0.853, respectively. Thus, the entropy of the first-order binary source, H(B), is found to be 0.6 bits/symbol. The entropy of the nth-order

extended binary source is n times that of the first-order binary source, nH(B). For instance, the entropy of eighth-order extended binary source, which has the same number of symbols as that of the original source or text file, is 4.8 bits/character.

Now, if the same text file is mapped using ASCII code, it is found that the probabilities of zero and one are 0.44 and 0.56, respectively. Therefore, the entropy of the first-order binary source and its corresponding eighth-order extended binary source are 0.99 bits/symbol and 7.9 bits/character, respectively. From these figures it is easy to observe that the difference in entropy between the original source and the nth-order extended binary source {H(s) – nH(B)} is less when the proposed mapping technique is used.

Bitwise Lz-78 Compression Algorithm:

LZ-78 algorithm, which is a fixed-length coding technique, is one of the most popular dictionary compression algorithms. The compression, achieved based on this algorithm, is accomplished by mainly parsing the incoming data stream into shortest groups of consecutive characters that are not encountered previously. Afterwards, these groups of characters are replaced by shorter representations. These representations are mainly a binary representation of indices to an earlier occurrence strings stored in a buffer or a dictionary (Weiss and Shremp, 1993; Haykin, 2001; Ziv and Lempel, 1977; 1978).

LZ-78 algorithm lends itself to highly efficient implementation while possessing a significant performance in compression ratio (Bell *et al.,* 1990). In this study, LZ-78 algorithm is used to manipulate the resulting binary source on a bitwise basis in lieu of applying the same algorithm directly to the original source file (i.e., on bytewise basis). LZ-78 algorithm builds a string-based dictionary without limitation on how far back in the dictionary the match can be achieved with the previous strings (Bell *et al.,* 1990; Nelson, 1991).

Having no explicit limit on the dictionary size enlarges the probability of finding a match with the previous strings. Furthermore, unlimited dictionary size allows the decoder (or the decompressor) to implicitly decode (or expand) the compressed data without receiving any information about the dictionary size and format from the compressor. In addition, the unlimited size of the dictionary will meet the important theoretical property of LZ-78 algorithm which is stated that the compression is optimal when the text file is large (Held and Marshall, 1991; Bell *et al.,* 1990; Nelson, 1991; Ziv and Lempel, 1978). In practice, text files can be large to some extent because of the limited memory size, storage device size, etc. Therefore, a further study can be performed to scrutinize the effect of dictionary size on the different compression metrics include the compression-decompression speed, memory requirement, implementation complexity along with compression ratio.

To illustrate the proposed idea and how LZ-78 algorithm manipulates the resulting binary source, Table 2 and 3 show the LZ-78 binary encoded blocks and subsequences of the first-order and second-order extended binary source of the following input binary data sequence: 10111101111111111011111111100001. The two tables list also the initialization subsequences that are pre-filled in the dictionary or code book prior the beginning of the data compression process.

In Tables 2 and 3, the input binary sequence is divided into nine, and seven subsequences based on the first- and second-order extended binary source, respectively. Each subsequence has two parts; the first part is a sequence that is not encountered previously and the second one is an explicit subsequence, called innovative phrase, which is marked by a bold font in the aforementioned tables. The first column in Tables 2 and 3 indicates the numerical positions or indices of the individual subsequences in the dictionary. The subsequences shown in Column 2 are constructed based on the LZ-78 encoding technique described in (Haykin, 2001) and (Ziv and Lempel, 1978).

To illustrate this encoding technique, the light will be shed at Table 3, where the second-order extended binary source is used. For instance, in this table the first subsequence of the input data stream, 1011, is made up of the concatenation of the third entry in the dictionary, 10, with 11. Thus, it is represented by the number 34, as shown in Column 3. The second subsequence of the data stream, 1101, consists of the fourth entry in the dictionary, 11, followed by the second entry, **01**, in the dictionary. Henceforth, it is represented by the number 42. The same process is repeated until the input binary data stream has been completely parsed.

The complete set of numerical representations for the various subsequences generated based on utilizing the first-order and second-order extended binary sources are listed in the third column in Tables 2 and 3. The fourth column in Tables 2 and 3 consists of a uniform block of bits that is the binary encoded representations of the different subsequences of the input data stream. For example, in Table 3 the last two bits, 11, in the block, 01111, represent the innovative symbol for the first subsequence, 1011, the remaining bits, 011, is the equivalent binary representation of the index or pointer to the root subsequence that matches the subsequence under consideration except for the innovation symbol that is located in position three in the dictionary.

Table 1: Probabilities of some English characters along with its corresponding ASCII code and new assigned code.

Alphabet	Probability of occurrence	ASCII Code	Assigned Code
Space `•`	0.147	00100000	11111111
e	0.098	11100101	11111110
t	0.070	11110100	11111101
h	0.062	01101000	11111011
o	0.059	11101111	11110111
a	0.058	01100001	11101111
…..	…..	…..	…..
;	0.00070	00111011	01011110
Carriage Return `CR`	0.00069	00001101	00111110
…..	…..	…..	…..

Table 2: Encoding process using LZ-78 algorithm based on the first-order binary source.

Numerical Position	Subsequences	Numerical Representation	Binary Encoded Block
1	0		
2	1		
3	10	21	0100
4	11	22	0101
5	110	41	1000
6	111	42	1001
7	1111	62	1101
8	1101	52	1011
9	11111	72	1111
10	11110	71	1110
11	00	11	0010
12	01	12	0011

Table 3: Encoding process using LZ-78 algorithm based on the second-order extended binary source.

Numerical Position	Subsequences	Numerical Representation	Binary Encoded Block
1	00		
2	01		
3	10		
4	11		
5	1011	34	01111
6	1101	42	10001
7	1111	44	10011
8	111101	72	11101
9	111111	74	11111
10	1110	43	10010
11	0001	12	00101

Text Compression Performance Using Bitwise Lz-78:

To illustrate the advantages of the proposed compression technique, different English text files with different sizes have been compressed based on the proposed technique. Moreover, the proposed technique can also be applied to any text files written in any language such as Arabic (Ghwanmeh *et al.,* 2006). The advantages achieved by any compression algorithm can be evaluated based on the following metrics: memory utilization, compress-decompress speed, implementation complexity, and compression ratio. Nowadays, most of the proposed compression algorithms emphasize on the compression ratio metric measured in bits/character because of the cheapest price of memory and the availability of high speed processor at an affordable price. Therefore, in this paper a light will be shed on the compression ratio metric.

To evaluate the performance of the proposed compression technique, LZ-78 algorithm has been applied to nth-order extended binary source resulting from transforming different English text files with different natures using the proposed mapping rule and the ASCII code. As an example, a text file, which is a story written by Mark Twain called “Life on the Mississippi”, is encoded using our proposed rule and ASCII code. The results obtained based on manipulating the resulting binary source at different extension orders (e.g., n= 1, 2, 4 and 8) using LZ-78 algorithm are shown in Figure 1.

Figure 1 depicts also a comparison between the compression ratios achieved from applying the LZ-78 algorithm on a bitwise basis to the nth-order extension binary source results from applying the proposed mapping rule and ASCII cod. In this figure, it is easy to observe that the bitwise LZ-78 encoder of the fourth-order extended binary source (i.e., a source with 16 symbols) achieves a compression ratio close to that of the conventional LZ-78 encoder which includes 256 characters. In addition, one can observe that the compression ratio achieved at different extension orders, n, based on using the proposed mapping rule is less than that one when ASCII code is used. The compression ratios are equal when the extension order n is eight. This is

because the same character has been read from the original information source when LZ-78 manipulates the eighth-order extended binary source.

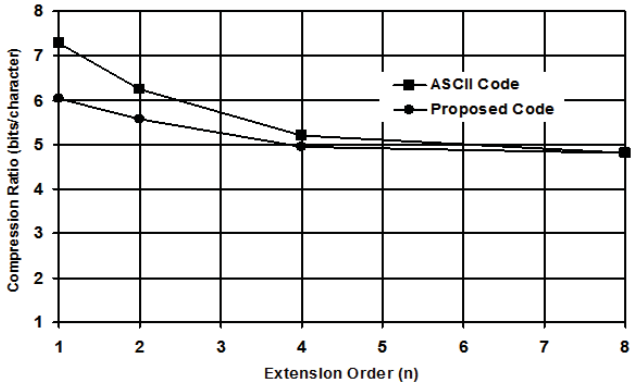

Fig. 1: Compression ratio (bits/character) versus the extension order (n) of an English text file.

Conclusions:

Transforming the non-binary information source into a binary one facilitates the software and hardware implementation of the proposed compression technique. Thus, a significant reduction in hardware and software implementations cost is acquired.

Applying a bitwise LZ-78 algorithm to nth-order extended binary source using the proposed mapping rule outperforms those conventional compression techniques in terms of compression-decompression speed and implementation complexity. Moreover, the proposed compression technique achieves a compression ratio close to that one when the order of extension made to the first-order binary source results out of the proposed mapping rule is four. To have a more reliable implementation of the proposed compression technique a limit on the dictionary or code book should be placed. To determine the optimal size of the dictionary size that leads to optimal compression metrics, another study is underway.

The proposed compression technique is not only limited to the text file but it can also be generalized to other applications such as image compression, voice compression, etc. A further study is underway to examine the performance of the proposed technique in such applications. In addition to English Text files from other languages such as Arabic are also processed to examine the proposed technique performance.

REFERENCES

Abdel-Rahman Jaradat and Mansour I. Irshid, 2000. An efficient bit-wise source encoding technique based on source mapping. Devices, Circuits and Systems, Proceedings of the 2000 Third IEEE International Caracas Conference.

Abdel-Rahman Jaradat and Mansour I. Irshid, 2001. A simple binary run-length compression technique for non-binary sources based on source mapping. Active and Passive Electronic Components, 24(4):211-221. doi:10.1155/2001/23505.

Bell, T., J. Cleary and I. Witten, 1990. Text compression. Prentice-Hall, Englewood Cliffs NJ.

Elabdalla, A. and M. Irshid, 2001. An efficient bitwise Huffman coding technique based on source mapping. Journal of Computer and Electrical Engineering, 27: 265–272.

Ghwanmeh, S., R. Al-Shalabi, and G. Kanaan, 2006. Efficient data compression scheme using dynamic Huffman code applied on Arabic language. Journal of Computer Science, pp: 885-888.

Held, G. and T. Marshall, 1991. Data compression. New York: John Wileyand Sons.

Haykin, S., 2001. Communication systems. New York: John Wiley and Sons.

Langdon, G. and J. Rissanen, 1981. Compression of black-white images with arithmetic coding. IEEE Trans. Communication, 29: 858-867.

Langdon, G. and J. Rissanen, 1982. A simple general binary source code. IEEE Trans. Information Theory, 28: 800-803.

Mano, M., 1984. Digital design. Prentice-Hall, New Jersey: Englewood Cliffs.

Nelson, M., 1991. The data compression book. Prentice-Hall.

Platos, J., V. Snasel and E. El-Qawasmeh, 2008. Compression of small text. Journal of Advanced Engineering Informatics, 22: 410 – 417.

Weiss, J. and D. Shremp, 1993. Putting data on diet. IEEE Spectrum, 30: 36-39.

Ziv, J. and A. Lempel, 1977. A universal algorithm for sequential data compression. IEEE Trans. Information Theory, 23: 337-343.

Ziv, J. and A. Lempel, 1978. Compression of individual sequences via variable-rate coding. IEEE Trans. Information Theory, 24: 530-536.