Afr. J. Comp. & ICT
Vol. 2, No. 2, 2009, pp. 13 – 20
ISSN 2006 – 1781

AN ENHANCED LZW TEXT COMPRESSION ALGORITHM

Adeniyi Moronfolu
% Department of Computer Science,
University of Ibadan,
Oyo State,
Nigeria.
Email: saintniyi@yahoo.com

&

'Dele Oluwade
Dewade Systems Consult
G. P. O. Box 4222,
Minna, Niger State,
Nigeria.
Email: deleoluwade@yahoo.com

Received 21 October, 2006

Thesis Communication

ABSTRACT

Data compression is the art or science of representing information in a compact form. This compact representation is created by identifying and using structures that exist in the data. Data can be characters in text file, numbers that are samples of speech or image waveforms, or sequences of numbers that are generated by other processes. Most text files contain a lot of redundancies such as multiple occurrences of characters, words, phrases, sentences and even spaces. These redundancies are the structures which most compression algorithms tend to exploit to bring about reduction in the amount of data. Achieving a high amount of compression depends not only on the compression algorithm used but also on the types of data that is being compressed. The Lempel-Ziv-Welch (LZW) algorithm is known to be one of the best compressors of text which achieve a high degree of compression. This is possible for text files with lots of redundancies. Thus, the greater the redundancies, the greater the compression achieved. However, the LZW algorithm can be further enhanced to achieve a higher degree of compression without compromising the speed of compression and decompression. In this paper, an improved hybrid algorithm, called RLW algorithm, is presented. The algorithm combines the run length encoding (RLE) algorithm with the LZW algorithm with a view to enhancing the

performance of the LZW algorithm without compromising the speed of execution.

Keywords: Text compression, Lempel-Ziv-Welch (LZW) algorithm, Run Length Encoding (RLE) algorithm,

I. INTRODUCTION

In the last decade, there have been a transformation in the way we communicate, and the process is still under way. This transformation includes the ever-present, ever-growing internet; the explosive development of mobile communications; and the ever-increasing importance of video communication. Data compression is one of the enabling technologies for these aspects of the multimedia revolution. It would not be practical to put images, let alone audio and video, on websites if it were for data compression algorithms. A data compression technique is said to be lossless if the decompressed (i.e restored) data file is identical to or is an exact replica of the original data [9]. It is said to be lossy if it only generates an approximation of the original file. Lossy image compression is used in digital cameras, greatly increasing their storage capacities while hardly degrading picture quality at all. Cellular phones would not be able to provide communication with increasing clarity were it not for compression. The advent of digital TV service would not be possible without compression. Making a long distance call is possible by the use of compression. Voice compression is used in internet telephony for example, while audio compression is used for CD ripping and is decoded by MP3 players [11]. Even modern, fax machine and satellite TV service are using compression. The applications are endless.

Data compression operates in general by taking "symbols" from an input "text", processing them, and writing "codes" to a compressed file. Symbols are usually bytes, but they could just as easily be pixels. To be effective, a data compression scheme needs to be able to transform the compression file back into an identical copy of the text [10]. Although, there are many types of compression techniques in existence today, the goals are all basically the same i.e. to make data as succinct as possible in the smallest possible time. This has direct application in storage conservation algorithm and in data transmission among many other applications [14]. The use of a compression algorithm is dependent on the efficiency of the algorithm, the nature of data to be compressed, as well as the elapsed time for which the compression and decompression can take place. Different compression algorithms are suited for different types of data e.g. JPEG (lossy) algorithm is appropriate for picture data compression, MPEG (lossy) algorithm for video data compression while HUFFMAN, LZW, Run Length Encoding (RLE) lossless algorithms etc are suited for text data compression. In particular, the RLE algorithm is an algorithm used to reduce the size of a repeating string (called run) of characters. This is typically achieved by encoding a run of symbols into two bytes, namely a count and a symbol. The LZW, on the other hand, is an algorithm which maps strings of text characters into numeric code. In [2, 3, 4, 5, 6, 7], a binary lossless text compression algorithm (called 'code presentation') was developed and applied to some model binary text data, notably American Standard Code for Information Interchange (ASCII), Extended Binary Coded Decimal Interchange Code (EBCDIC) and the International Telegraph

and Telephone Consultative Committee (CCITT) code.
In the present paper, an enhanced LZW compression algorithm, called RLW algorithm, is presented. This algorithm is an hybrid of the RLE and LZW algorithms which enhances the performance of the latter without compromising the speed of execution. A comparison of RLE, LZW and RLW in terms of the file sizes shows that the new algorithm (RLW) performs best.

II. RLE AND LZW ALGORITHMS

Run-length Encoding (RLE) is a technique used to reduce the size of a repeating string of characters. This repeating string is called a run. Typically, RLE encodes a run of symbols into two bytes; a count and a symbol. RLE can compress any type of data regardless of its information content, but the content of data to be compressed affects the compression ratio. RLE cannot achieve high compression ratios compared to other popular compression methods, but it is easy to implement and is quick to execute. Run-length encoding is supported by most bitmap file formats such as TIFF, BMP and PCX.

Consider a character run of 15 'A' characters which normally would require 15 bytes to store: AAAAAAAAAAAAAAA becomes 15A. With RLE, this would only require two bytes to store; the count (15) is stored as the first-byte and the symbol (A) as the second byte. Consider another example, with 16 character string : 000ppppppXXXXaaa. This string of characters can be compressed to form 3(0)6(p)4(X)3(a). Hence, the 16 byte string would only require 8 bytes of data to represent the string. In this case, RLE yields a compression ratio of 2:1 [12]

For example, consider a string of text S = abaaabbbaaabaaaabbb. Running it through a RLE assuming run length of one is not allowed. The string will be compressed to ab3a3b3ab3a3b. There is thus a compression from 18 bytes 13 bytes.

Unlike the RLE, LZW (Lempel-Ziv-Welch) method maps strings of text characters into numeric code. To begin with, all characters that may occur in the string are assigned a code. For example, suppose that the string S = abaaabbbaaabaaaabbb used above in the RLE example is to be compressed via LZW. Firstly, all occurring characters in string are assigned a code. In this case the occurring characters are "a" and "b" and are assigned 0 and 1 respectively. The mapping between character and their codes are stored in the dictionary. Each dictionary entry has two fields: key and code. The characters are stored in the key field while the numbers are stored in the code field. The LZW compressor repeatedly finds the longest prefix, p, of the unencoded part of S that is in the dictionary and outputs its code. If there is a next character c in S, then pc (pc is the prefix string p followed by the character c) is assigned the next code and inserted into the dictionary. This strategy is called the LZW rule [8].

Let S = pc be an uncompressed string where p is the longest prefix of the uncompressed string found in dictionary and c is the suffix of the uncompressed string. To compress the string using LZW algorithm, the following steps are followed:

> *Step 1*: Replace the p with q code where the dictionary key for q code must already be defined in the dictionary.
> *Step 2*: Add new code inside the dictionary with key text (q) fc(p), where fc(p) represents the first character of the uncompressed string [8, 13].

To decompress using the algorithm, each code is fed as input one at a time and is replaced by the texts it represents. The code-to- text mapping can be reconstructed in the following ways: First, a predefined dictionary of all single occurring characters is setup. As before, the dictionary entries are code-text pairs. This time, however, the dictionary is searched for an entry with a given code (rather than with a given text). The first code in the compressed file corresponds to a single character and so may be replaced by the corresponding character. For all other codes p in the compressed file, two cases can be considered namely, the case in which code p is in the dictionary, and the case in which the code p is not in the dictionary. To decompress the compressed string the qp code is considered.

Case 1: Code p is found in dictionary. Replace p with the text (p) and add new code into dictionary based on key text(q)fc(text(q)), where p represents the current code, fc the first character and q the previous code

Case 2: Code p is not found in dictionary. Replaced p with text(q)fc(text(q)) and add new code into dictionary based on key text(q)fc(text(q))[8, 13].

III. COMBINED ALGORITHM (RLE+LZW)

Though, LZW is a good compressor of text, yet its performance can still be enhanced by passing the string to be compressed through a RLE Compressor first before passing it through a LZW compressor. This results in a better compression gain than using LZW compressor alone. The RLE+LZW Compression algorithm consists of two steps:

*Step 1***:** Compress the source file using RLE first.

*Step 2***:** Compress the output of the RLE compressor using LZW compressor.

Similarly, the RLE+LZW decompression algorithm consists of two steps:

*Step 1***:** Decompress the source file using LZW first.

*Step 2***:**Decompress the output of the LZW decompressor using the RLE decompressor.

As an example, the string S = abaaabbbaaabaaabbb which was earlier considered will be further used to demonstrate the little, yet significant, compression gain of the RLE + LZW combined compressor. To start with, assume the string is first passed through the RLE compressor, the output of the RLE compressor will be 'ab3a3b3ab3a3b'. If this output is now passed through the LZW compressor, the following stages will occur until the whole string has been compressed.
First, the predefined dictionary is setup with all the occurrence of characters in the string to be compressed. Table 3.1 below shows the initial configuration of the dictionary before compression begins and its various stages. To begin compression, set up the initial predefined dictionary as done in stage (1) of Table 3.1 below.

Table 3.1: Various Stages of LZW Compression on the RLE Compressor Output

Stage 1

Key	a	b	3
Code	0	1	2

Original string: ab3a3b3ab3a3b
Output code = null

Initial configuration (Predefined dictionary)

Stage 2

Key	a	b	3	ab
Code	0	1	2	4

Original string: ab3a3b3ab3a3b
Output code = 0
a has been compressed, ab added to dictionary

Stage 3

Key	a	b	3	ab	b3
Code	0	1	2	4	5

Original string: ab3a3b3ab3a3b
Output code = 01
ab has been compressed, b3 added to dictionary

Stage 4

Key	a	b	3	ab	b3	3a
Code	0	1	2	4	5	6

Original string: ab3a3b3ab3a3b
Output code = 012
ab3 has been compressed, 3a added to dictionary.

Stage 5

Key	a	b	3	ab	b3	3a	a3
Code	0	1	2	4	5	6	7

Original string: ab3a3b3ab3a3b
Output code = 0120
ab3a has been compressed, a3 added to dictionary

Stage 6

Key	a	b	3	ab	b3	3a	a3	3b
Code	0	1	2	4	5	6	7	8

Original string: ab3a3b3ab3a3b
Output code = 01202
ab3a3 has been compressed, 3b added to dictionary

Stage 7

Key	a	b	3	ab	b3	3a	a3	3b	b3a
Code	0	1	2	4	5	6	7	8	9

Original string: ab3a3b3ab3a3b
Output code = 012025
ab3a3b3 has been compressed, b3a added to dictionary

Stage 8

Key	a	b	3	ab	b3	3a	a3	3b	b3a	ab3
Code	0	1	2	4	5	6	7	8	9	X

Original string: ab3a3b3ab3a3b
Output code = 0120254
ab3a3b3ab has been compressed, ab3 added to dictionary

Stage 9

Key	a	b	3	ab	b3	3a	a3	3b	b3a	ab3	3a3
Code	0	1	2	4	5	6	7	8	9	X	Y

Original string: ab3a3b3ab3a3b
Output code = 01202546
ab3a3b3ab3a has been compressed, 3a3 added to dictionary

Stage 10

Key	a	b	3	ab	b3	3a	a3	3b	b3a	ab3	3a3
Code	0	1	2	4	5	6	7	8	9	X	Y

(Final Dictionary)

Original string: ab3a3b3ab3a3b
Output code = 012025468
ab3a3b3ab3a3b has been compressed, 3a3 added to dictionary

Stage 11
Original string: ab3a3b3ab3a3b
Output code = 012025468
ab3a3b3ab3a3b has been compressed, nothing added to dictionary

The final compressed string of the RLE output through the LZW compressor is 012025468, which occupies 9 bytes.

Initially, there was a 18 byte string passed through the RLE compressor. However, the string was reduced to 13 bytes by the RLE compressor. The output of the RLE compressor is now passed through the LZW compressor and there is a further reduction from its 13 bytes to 9 bytes. Thus, the combined RLE and LZW reduces the original string from 18 bytes to 9 bytes – a 50% reduction. When the same sample string was manually run through the LZW compressor, it compresses it to 13 bytes. Thus, the combined RLE+LZW algorithm outperforms LZW compressor.

IV. PERFORMANCE EVALUATION

In order to verify and justify the manual walkthrough of the RLE + LZW combined algorithm done in Table 3.1 above, some sets of files were run through the implemented version of RLE, LZW and RLW (combined algorithm) to compare their result and performance. The implementation was carried out using C language due to the language's support for manipulating strings, among other features. Table 4.1 below shows the comparative result of the compression. While LZW and RLW compete favourably, it was observed that when it comes to text files, RLW compresses a little better than LZW. Though the compression gain is very little, yet it is significant. It is also noteworthy that in some types of files that are mostly non text-based, LZW seems to equally have a little performance gain than RLW. There are however some types of text-based files where LZW still has a performance gain over RLW. Such files are either small in size with little redundancies, or bigger in size and yet with little redundancies. In Table 4.1 below, the files D1, D2, D3, D4 and D8 are text based files while the remaining files (i.e. D5, D6 and D7), though containing text, are not text-based files.

Table 4.1: Comparison of Algorithms Performance for Different File Sizes and Format.

Serial No	Name	Original File Sizes (KB)	RLE Size (KB)	LZW Size (KB)	RLW Size (KB)
1	D1	19.5	10.6	2.27	2.03
2	D2	147	116	49.2	49
3	D3	73	50.2	25.2	24.9
4	D4	32.5	14.6	8.5	8.34
5	D5	227	213	102	103
6	D6	444	191	64.7	64.9
7	D7	22.7	22.5	10.1	10.2
8	D8	41.5	25.2	12.8	12.6

In file D4, for instance, whereas the original file size is 32.5KB, use of RLE compression algorithm reduces the size to 14.6KB, while LZW, on the other hand, reduces it to 8.5KB. By using the combined algorithm (RLW), the size is reduced to 8.34KB which gives the best compression result.

IV. DISCUSSION AND CONCLUSION

From the analysis carried out in previous sections, it can be observed that the combined RLE and LZW compressor will bring about a little but significant compression than RLE or LZW alone for most text files. Since both RLE and LZW algorithms are exploiting common redundancy in text-based file (i.e. repetitiveness or multiple occurrences of phrases), combining this positive feature of the two algorithms into a single algorithm can bring about an increased performance than the individual algorithm. This increased compression gain is more noticeable in files with multiple occurrences of phrases and spaces as characterized in most day-to-day text based files. Thus, the greater the redundancies, the greater the degree of compression achieved. However, there are cases when the LZW outperforms the new algorithm; this is not always a common encounter and is observed with either smaller files with little repetitiveness or larger files with little or no repetitiveness. On the hand, since the individual algorithms are fast in execution speed, it can be argued that the speed of the combined algorithm will also be fast. It is important that the order of compression be strictly followed i.e. RLE first, then followed by LZW. This is due to the fact that LZW, from experimental observation, has been found to be an optimal compressor such that no other compression algorithm can compress its output further. The study therefore demonstrates that text-based files with lots of redundancies can be compressed to a higher degree of compression by first compressing it through the RLE algorithm before compressing it through the LZW algorithm.

POSTSCRIPT

This paper is part of the M.Sc (Computer Science) project [1] of the first author written under the supervision of the second author.

REFERENCES

[1] Adeniji Olusoji Moronfolu, *Development of a Plain Text Algorithm for Data Compression and Decompression*, M.Sc project, Department of Computer Science, University of Ibadan, Nigeria, 2006.

[2] 'Dele Oluwade, "Applications of 2-Code Error Detection Techniques", *Proceedings of the 14th*

*National Conference of COAN*_(Nigeria Computer Society), Vol.9, pp.245-251, 1998
[3] 'Dele Oluwade, 'A Novel Groupoid Code', *Journal of Science Research* (Journal of the Faculty of Science, University of Ibadan), vol.6, no.1, pp.1-3, 2000.
[4] 'Dele Oluwade, 'Algebraic Characteristics of the CCITT#2 Communication Code', *International Journal of Applied Mathematics & Statistics* (IJAMAS), vol.2, No.M04, pp.50-59, 2004; http://www.ceser.res.in/ijamas.html
[5] 'Dele Oluwade, *Design and Analysis of Computer – Coded Character Sets*, PhD thesis, Department of Computer Science, University of Ibadan, Nigeria, 2004.
[6] 'Dele Oluwade, 'A Binary Text Compression Algorithm Based on Partitioning', *Advances in Computer Science and Engineering*, vol.3, no.2, pp.165–174,2009; http://pphmj.com/journals/articles/523.htm
[7] 'Dele Oluwade, 'Application of a Data Compression Technique to the American Standard Code for Information Interchange (ASCII)', *International Journal of Information Science and Computer Mathematics*, vol.1, no.1, pp.1-7,2010 ; http://pphmj.com/journals/ijiscm.htm
[8] Sartaj Sahni, *Data Structure, Algorithms and Applications in Java*, McGraw Hill Publishing Company, pp. 437 – 450, 1999.
[9] C.E. Shannon and W. Weaver, *The Mathematical Theory of Communication*, University of Illinois Press, 1949
[10] http://dogma.net/markn/articles/arith/part2.htm <accessed in 2005>.
[11] http://www.data-compression.com <accessed in 2005>.
[12] http://www.eee.bham.ac.uk/WoolleySl/All7/body0.htm <accessed in 2005>.
[13] http://www.geocities.com/yccheok <accessed in 2005>.
[14] www.dspguide.com/datacomp.htm <accessed in 2005>.

Adeniyi Moronfolu is a product of the University of Ibadan, Nigeria where he received a Master of Science (M.Sc) degree in Computer Science.

'Dele Oluwade studied both Building Technology and Mathematics at the University of Ife / Obafemi Awolowo University (OAU), Ile-Ife, where he received a Bachelor of Science (B.Sc) degree in Mathematics. He also received a Postgraduate Diploma (PGD) in Computer Science from the University of Lagos, a Master of Science (M.Sc) degree in Mathematics from the O.A.U., Ile-Ife, as well as a Master of Philosophy (M.Phil) degree and a Doctor of Philosophy (Ph.D) degree in Computer Science from the University of Ibadan. He is a registered Computer Scientist and a Senior Member of the IEEE. He is a computer/IT consultant.