International Journal of Application or Innovation in Engineering & Management (IJAIEM)
Web Site: www.ijaiem.org Email: editor@ijaiem.org
Volume 3, Issue 6, June 2014 ISSN 2319 - 4847

Evaluation of Dominant Text Data Compression Techniques

[1]Somefun, Olawale M. [2]Adebayo, Adewale O.

[1,2]School of Computing and Engineering Sciences, Babcock University,
Ilisan-Remo, Ogun, Nigeria

Abstract

Compression refers to reducing the quantity of data, bits used to represent, store and/or transmit file content, without excessively reducing the quality of the original data. Data Compression technique can be Lossless which enables exact reproduction of the original data on decomposition or Lossy which does not. Text data compression techniques are normally lossless. There are many text data compression algorithms, and it is essential that best ones be identified and evaluated towards optimizing compression purposes. This research focused on proffering key compression algorithms, and evaluating them towards ensuring selection of better algorithms over poorer ones. Dominant text data compression algorithms employing Statistical and Dictionary based compression techniques were determined through qualitative analysis of extant literature for convergence and expositions using inductive approach. The proffered algorithms were implemented in Java, and were evaluated along compression ratio, compression and decompression time using text files obtained from Canterbury corpus. Huffman and Arithmetic coding techniques were proffered for statistical, and Lempel-Ziv-Welch (LZW) for dictionary-based technique. LZW was indicated the best with the highest compression ratio of 2.36314, followed by Arithmetic with compression ratio of 1.70534, and Huffman with compression ratio of 1.62877. It was noted that the performance of the data compression algorithms on compression time and decompression time depend on the characteristics of the files, the different symbols and symbol frequencies. Usage of LZW, in data storage and transmission, would go a long way in optimizing compression purposes.

Keywords: Data Compression, Lossy Compression, Lossless Compression, Statistical Data Compression, Dictionary-Based Data Compression.

1. INTRODUCTION

Information is part of life and it comes in various forms. The increase in information requires a large amount of storage and bandwidth. In order for this to happen efficiently, it necessitates the need to use fewer bits to represent the information in order to save storage and bandwidth. Compression refers to reducing the quantity of data used to represent a file, image or video content without excessively reducing the quality of the original data. It also reduces the number of bits required to store and/or transmit digital media. To compress something means that you have a piece of data and you decrease its size [1]. There are numerous data compression techniques, and it is essential that best ones be identified and evaluated towards optimizing compression purposes. This has to been done to avoid the setback of installing poorer algorithms over better ones, which will be space and time costly and detrimental to computing. It is therefore necessary to proffer key compression algorithms and evaluate them towards ensuring selection of better algorithms over poorer ones. The objectives of this paper are, therefore, to proffer key data compression techniques algorithm and evaluate the techniques based on Compression ratio, Compression time and Decompression time.

Dominant text data compression algorithms employing Statistical and Dictionary based compression techniques were determined through qualitative analysis of extant literature for convergence and expositions using inductive approach. The proffered algorithms were implemented in Java, and were evaluated along compression ratio ($\frac{number\ of\ bytes\ before\ compression}{number\ of\ bytes\ after\ compression}$), compression time (the time taken to reduce or compress file or document) and decompression time (time taken to reproduce or decompress a file or document), using text files obtained from Canterbury corpus (alice29.txt (English text), asyoulik.txt (Shakespeare), lcet10.txt (Technical writing), plrabn2.txt (Poetry), bible.txt (The King James' version of the bible), and world192.txt (The CIA world fact book)). Table 1 presents relevant configuration of the platform of execution of the compression algorithms Java programs.

Table 1: Configuration of the Platform for Execution of the Algorithms Java Programs

Laptop HP 650	
Ram	DDR3 - 4GB
Processor Type	Intel (R) Pentium (R)
Number of Cores of Processor	2
Clock Speed of Processor	2.40 GHz
Cache of Processor	8 MB
Operating System	Windows 8 - 64bit, Ubuntu 12.10
Application package	Netbeans

Compression in computing and other related fields is important due to the fact that it helps to keep data in a little space as possible in a file, thereby making more storage capacity available. Data compression is also increasingly crucial to efficient and secure transfer/communication of data in medical, commercial and government-related computing. Compression as well helps system designers exploit limited communication channel bandwidth (e.g. in video transmissions and wireless computing) and storage resources (e.g. remote sensing medical imaging). Proffering and evaluation of key text algorithm is a step towards ensuring selection of better algorithms over poorer ones.

2.Literature Review

2.1. Information Theory

The basis for data compression is the mathematical value of information. Information contained in a symbol x is given by $l(X) = \log_2 \frac{1}{p(x)}$. This value also describes the number of bits necessary to encode the symbol. This definition reinforces the notion of information. First, the more probable the occurrences of a symbol, the fewer bits are used to represent it. Conversely, the least frequent symbols provide more information by their occurrence. Secondly, if there are n equally probable messages, $\log_2 n$ bits will be required to encode each message. This is the information value of each message $l= \log 2\frac{1}{p} = \log_2 n$. Finally, information of two independent messages should be additive.

Consider two independent messages A and B. Here we have,

$I(AB) = \log_2 \frac{1}{p(A)p(B)} = \log_2 \frac{1}{p(A)} + \log_2 \frac{1}{p(B)} = I(A) + I(B)$ [2].

2.1.1 Entropy

Entropy can also be defined as the measure of the average information [2]. According to Shannon, entropy of a discrete source for a finite alphabet X is given by

$$H(x) = \sum_{x \in X} p(x) \log_2 \frac{1}{p(x)}$$

Properties of Entropy

Theorem: $0 \leq H(X) \leq \log_2 n$ where $X = \{X_1, X_2, ..., X_n\}$

Proof:

If p(X) =1 for some x, then H (X) = 0.

$$H(X) = \sum_{x \in X} p(x) \log_2 \frac{1}{p(x)} \leq \log_2 \sum_{x \in X} p(x) \frac{1}{p(x)} = \log_2 n$$

If p(X) = 1/n for all X, we have

$$H(X) = -\sum_{x \in X} \frac{1}{n} \log_2 \frac{1}{n} = -\frac{1}{n} \sum_{x \in X} log_2 1 - \sum_{x \in X} log_2 n = \sum_{x \in X} log_2 n = \log_2 n$$ [3].

2.2. Types of Compression Data

Data to be compressed can be divided into symbolic and diffused data. Symbolic data is data that can be discerned by human eye. These are combinations of symbols, characters or marks. Examples of this type of data are text and numeric data. Unlike the symbolic data, diffuse data cannot be discerned by the human eye. The meaning of the data is stored in its structure and cannot be easily extracted. Examples of this type of data are speech, image and video data [3].

2.3. Broad Data Compression Categories

Data compression techniques are categorized according to loss of data into two groups, namely lossless data compression techniques and lossy data compression techniques. Lossless algorithms reconstruct the original message exactly from the compressed message, and lossy algorithms only reconstruct an approximation of the original message. Lossless algorithms are typically used for text and lossy for images and sound where a little bit of loss in resolution pitch or other is often undetectable, or at least accepted. Lossy is used in an abstract sense, however, and does not mean random lost pixels, but instead means loss of a quantity such as frequency component, or perhaps loss of noise [4]. Lossy compression algorithms are JPEG (Joint Photographic Expert Group), MPEG (Moving Picture Experts Group), MP (Media Player) and Fractal while Lossy coding techniques include DCT (Discrete Cosine Transform), DFT (Discrete Fourier Transform), DWT (Discrete Wavelet Transform).

Lossless compression techniques are used to compress, of necessity, medical images, text and images preserved for legal reasons, computer executable files, among others. Lossy compression techniques reconstruct the original message with loss of some information. It is not possible to reconstruct the original message using the decoding process, and is called irreversible compression. The decompression process produces an approximate reconstruction, which may be desirable where data of some ranges that could not be recognized by the human brain can be neglected. Such techniques could be used for multimedia images, video and audio to achieve more compact data compression [5].

2.4. Lossless Data Compression Techniques

There are three main categories of lossless data compression techniques: those imploying statistical models, those that require the use of a dictionary, and those that use both statistical and dictionary-based techniques. Dictionary-based compression schemes tend to be used more for archiving applications (sometimes in conjunction with other methods), while real-time situations typically require statistical compression schemes. This is because dictionary-based algorithms tend to have slow compression speeds and fast decompression speeds while statistical algorithms tend to be equally fast during compression and decompression. Statistical compression schemes determine the output based on the probability of occurrence of the input symbols and are typically used in real-time applications. The algorithms tend to be symmetric (the decoder mirrors the steps of the encoder); therefore, compression and decompression usually require the same amount of time to complete. Dictionary compression schemes do not use a predictive statistical model to determine the probability of occurrence of a particular symbol, but they store strings of previously input symbols in a dictionary. Dictionary-based compression techniques are typically used in archiving applications such as compress and gzip because the decoding process tends to be faster than encoding. Hybrid compression methods share characteristics with both statistical and dictionary-based compression techniques. These algorithms usually involve a dictionary scheme in a situation where simplifying assumptions can be made about the input data [6].

Lossless data compression algorithm includes**:** Limpel Ziv family (Dictionary-based encoding), Run-length Encoding compression (Statistical coding), Huffman Encoding (Statistical coding), Arithmetic Encoding (Statistical coding), Bit-mask coding (Dictionary-based). Figure 1 depicts data compression techniques.

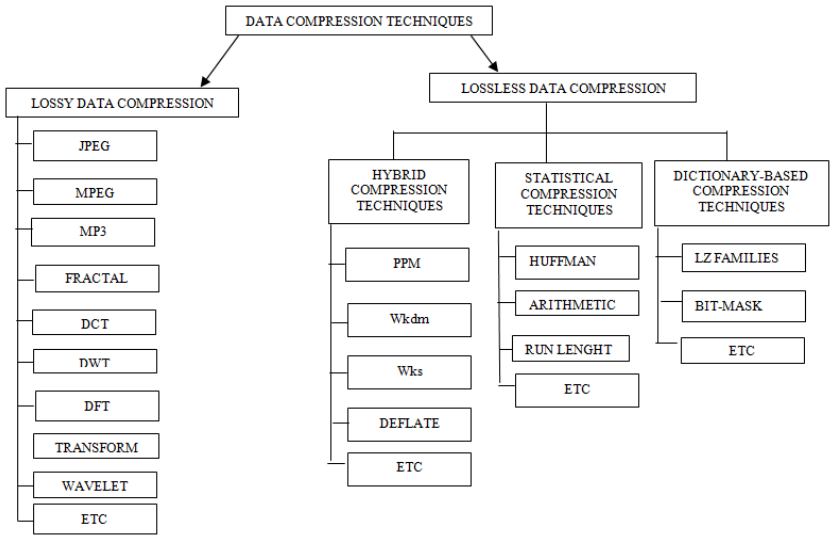

Figure 1 - Data compression techniques

2.5. Dictionary-Based Data Compression Techniques

Dictionary Compression selects entire phrases of common characters and replaces them with a single codeword. The codeword is used as an index into the dictionary entry which contains the original characters.

2.5.1 LZW

The LZW algorithm uses dictionary while decoding and encoding but the time taken for creating the dictionary is large. LZW compression uses a code table common choice to provide 4096 entries in the table. In this case, the LZW encoded data consists of 12 bit codes, each referring to one of the entries in the code table. Decompression is achieved by taking each code from compressed file, and translating it through the code table to find what character or characters it represents. Codes 0-255 in the code table are always assigned to represent single byte from the input file. When the LZW program starts to encode a file, the code table contains only the first 256 entries, with the remainder of the table being blank. This means that the first code in the compressed file is of 12 bit codes, each referring to one of the entries in the code table. Decompression is achieved by taking each code from the compressed file, and translating it through the code table to find what character or characters it represents. As the encoding continues, the LZW algorithm identifies repeated sequences in the data, and adds them to the code table. Compression starts the second time a sequence is encountered. The key point is that a sequence from the input file is not added to the code table until it has already been placed in the compressed file as individual characters (codes 0 to 255). This is important because it allows the decompression program

to reconstruct the code table directly from the compressed data, without having to transmit the code table separately [5]. Table 2 presents summary of principal LZ variations.

Table 2: Summary of Principal LZ Variations [5].

Method	Author	Description
LZ77	Ziv and Lempel	Send pairs with pointer and character. Pointer is fix-size and indicates a substring in the previous N characters.
LZR	Rodeh et al.	Send pairs with pointer and character. Pointer is variable-size and indicates a substring anywhere in the previous characters.
LZSS	Bell	A flag bit distinguishes sending of pointer or character. Pointer is fix-size and indicates a substring in the previous N characters.
LZB	Bell	Same as LZSS, except that pointer is variable-size
LZH	Bell	Same as LZSS, except that Huffman coding is used for pointers on a second pass.
LZ78	Ziv and Lempel	Send pairs of pointer and character. Pointer indicates a previously parsed substring which is stored in a dictionary
LZW	Welch	Include all alphabets in dictionary initially. Therefore only outputs fix-size pointers.
LZC	Thoman et al.	As suggested by Welch, a variable-size pointer scheme is implemented.
LZT	Tischer	Same as LZW, except that the parsed strings in dictionary are stored as a Least Recently Used list.
LZJ	Jakobsson	Same as LZW, except that pointers can reach anywhere in the previous characters.
LZFG	Fiala and Greece	By breaking up strings in the sliding window, pointers are formed from a tree data structure.

2.5.2 Dictionary-Based Code Compression Techniques Using Bit-mask technique

Bit-mask improves the dictionary based compression technique by considering mismatches. The basic idea is to determine the instruction sequences that are different in few bit positions (hamming distance) and store that information in the compressed program and update the dictionary (if necessary). The compression ratio will depend on how many bit changes are considered during compression [7].

2.6. Statistical Compression

Statistical Compression uses the frequency of single characters to choose the size of the codewords that will replace them. Frequent characters are encoded using shorter codewords so that the overall length of the compressed text is minimized.

2.6.1 Huffman Compression

The Huffman compression algorithm is named after its inventor, David Huffman. Huffman compression is a lossless compression algorithm that is ideal for compressing text or program files. Huffman Coding [8] is perhaps the most common and widely-used statistical compression technique. During the encoding process, this method builds a list of all the input symbols, sorting them based on their probabilities. The algorithm then constructs a tree, with a symbol at every leaf, and traverses the tree to determine the codes for each symbol. Commonly occurring symbols have shorter codes. Decoding is simply the reverse: the code is used to transverse the tree until the symbol is determined. The Huffman coding ensures that the longest codes get assigned to the least frequent brightness and vice versa. Huffman coding converts the pixel brightness values in the original image to a new variable-length codes, based on their frequency of occurrence in the image [9]. Figure 2 shows Huffman Compression flow graph.

Figure 2 - Huffman Compression Flow Graph [10].

2.6.2 Arithmetic Coding

The method of arithmetic coding was suggested by Elms and presented by Abramson in his test on information theory [11]. In arithmetic coding a source ensemble is presented by an interval between 0 and 1 on the real number line.

Practical implementations of Arithmetic Coding are very similar to Huffman coding, although it surpasses the Huffman technique in its compression ability. The Huffman method assigns an integral number of bits to each symbol, while arithmetic coding assigns one long code to the entire input string. Arithmetic coding has the potential to compress data to its theoretical limit. Arithmetic coding combines a statistical model with an encoding step, which consists of a few arithmetic operations. The most basic statistical model would have a linear time complexity of N[log(n)+a] +Sn where N is the total number of input symbols, n is the current number of unique symbols, a is the arithmetic to be performed, and S is the time required, if necessary, to maintain internal data structure [12]. The Arithmetic flow graph is shown in Figure 3.

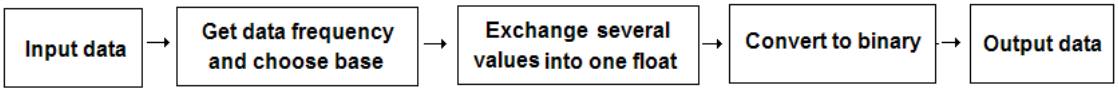

Figure 3 - Arithmetic Compression Flow Graph [10].

2.6.3 Run Length Encoding (RLE)

RLE is one of the simplest forms of compression, which works well on data with little noise. It is often used both in lossless and in lossy compression algorithms. It is very time efficient because of its simple nature but its major weakness is that its compression ratio is highly dependent on the input data. RLE compresses data by iterating over all the elements and expressing the data as a collection of tuples of counters and values that express how many occurrences of the value are present in one sequence. For example, if we were to compress the string “AAABBAAACCCCBBB” we would get the compressed string “3A2B3A4C3B”. Here one can see that the string of 15 bytes can be expressed as a string of 10 bytes. This shows that RLE is a comparison of n-1 element in an array of size n, which explains its efficient execution time. However, RLE has a weakness of its reliability on the nature of the input data. In some cases, it can even give a larger compressed string than the original. For example if the input string is “ABDBAC”, the compressed string would be “1A1B1D1B1A1C”, which is twice as large as the original. This is of course an extreme case, but it shows the unpredictable nature of the algorithm, especially for noisy data. RLE is in the same way very efficient in an opposite scenario. When it comes to implementing RLE, it is clear that it has a sequential nature. But it can be easily parallelized with a little overhead in edge cases. The parallelization can be done such that each thread starts at a different position in the input data and start comparing values. The overhead is to compare the values where one threads ends and another begins to see if they are similar, and to merge the values because it will be decompressed to the original [13]. To decompress RLE encoded data, one simply reads the tuples one by one and produces the value in the tuple the amount of times of the counter. For example, a tuple of “3A is then decompressed into “AAA” actual three A characters [10]. RLE is only efficient with files that contain lots of repetitive data.

2.7. Algorithms of the selected compression techniques

2.7.1 Huffman Compression Technique Algorithm

The Huffman Compression technique is as follows:

For Encoding

1. Read Text File.
2. Get the probabilities of every character in the file.
3. Sort the characters in descending order according to their probabilities
4. Generate the binary tree from left to right until you have just one symbol left.
5. Read the tree from right to left assigning different bits to different branches.
6. Store the final Huffman dictionary.
7. Encode every character in the file by referring to the dictionary.
8. Transmit the encoded code along with the dictionary.

For Decoding

1. Read the encoded code bitwise
2. Look for the code in the dictionary
3. If there is a match gets, its corresponding symbol else read the next bit and repeat Steps 3 & 4
4. Write the decoded text in a file [16].

2.7.2 Arithmetic Compression Technique

The Arithmetic Compression technique algorithm is as follows:

1. Calculate the probability of each symbol.
2. Calculate its cumulative probability Pc.
3. Begin with a current interval [L,H) initialized to [0,1)

4. For each event in the file, perform two steps.
   a. Subdivide the current interval into subintervals, one for each possible event. The size of a event's subinterval is proportional to the estimated probability that the event will be the next in the file, according to the model of the input.
   b. Select the subinterval corresponding to the event that actually occurs next, making it the new current interval.
   Lower interval L = L + Pc * (H-L) *L;
   Higher interval H = L Pc * (H-L) * H;
5. Assign a unique identical tag for the message.

For Decoding

The tag value is taken; the new tag value is calculated using formula:

Tag = (Tag – L(s)) / Pc(s)

The corresponding lower interval and cumulative probability of symbol s is taken to decode the tag value. The process continues till the end of the text file. As the length of the source sequence increases, the length of the subinterval specified by the sequence decreases, and more bits are required to precisely identify the subinterval.

2.7.3 Lempel-Ziv-Welch Compression Algorithm

The Lempel-Ziv-Welch Compression technique algorithm is as follows:

Compression

```
1   Initialize table with single character strings
2   P = first input character
3   WHILE not end of input stream
4       C = next input character
5       IF P + C is in the string table
6         P = P + C
7       ELSE
8         output the code for P
9       add P + C to the string table
10        P = C
11      END WHILE
12  output code for P
```

Decompression

```
1   Initialize table with single character strings
2   OLD = first input code
3   output translation of OLD
4   WHILE not end of input stream
5     NEW = next input code
6     IF NEW is not in the string table
7           S = translation of OLD
8           S = S + C
9     ELSE
10          S = translation of NEW
11    output S
12    C = first character of S
13    OLD + C to the string table
14    OLD = NEW
15  END WHILE [15].
```

3. Evaluation Results

3.1. Data Presentation

Attention was focused to compare the performance of Huffman, Arithmetic and LZW compression techniques, as dominant text data compression techniques. Research works done to evaluate the efficiency of any compression algorithm are carried out with regards to three important parameters, namely compression ratio, compression time and decompression time. Table 3 presents the evaluation results. Figures 4, 5 and 6 depict compression ratio, compression time, and decompression time, respectively.

Table 3: Evaluation Results

File Name (.txt)	Encoding Time(m sec)	Decoding Time (m sec)	Original size (Byte)	Compressed (Byte)	Compression Ratio
A. Huffman					
alice29	200.0	100.0	7,233,536	86,016	84.0952

Asyoulik	100.0	100.0	126,976	77,824	1.6316
Bible	200.0	400.0	4,050,944	2,220,032	1.8247
lcet10	100.0	0.0	430,080	253,952	1.6935
World192	100.0	200.0	2,473,984	1,966,080	1.2583
plrabn12	100.0	0.0	483,328	278,528	1.7352
B. Arithmetic Coding					
alice29	100.0	100.0	7,233,536	86,016	84.0952
Asyoulik	100.0	100.0	126,976	77,824	1.6316
Bible	200.0	200.0	4,050,944	2,199,552	1.8417
lcet10	100.0	100.0	430,080	253,952	1.6935
World192	100.0	200.0	2,473,984	1,548,288	1.5979
plrabn12	100.0	100.0	483,328	274,432	1.7620
C. Lempel-Ziv-Welch					
alice29	100.0	100.0	7,233,536	69,632	103.8824
Asyoulik	0.0	0.0	126,976	61,440	2.0667
Bible	300.0	500.0	4,050,944	1,781,760	2.2736
lcet10	100.0	100.0	430,080	180,224	2.3864
World192	300.0	300.0	2,473,984	864,256	2.8626
plrabn12	100.0	100.0	483,328	217,088	2.2264

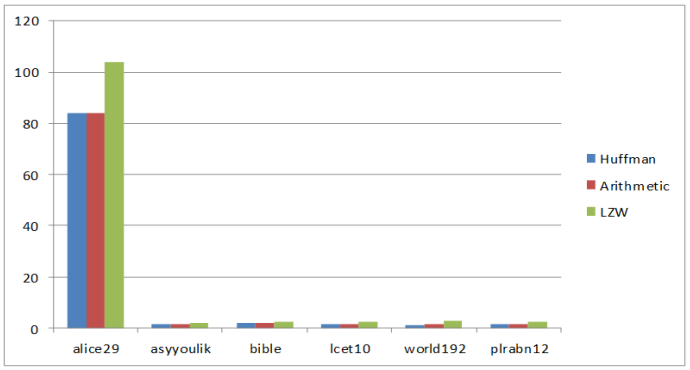

Figure 4 - Compression Ratio

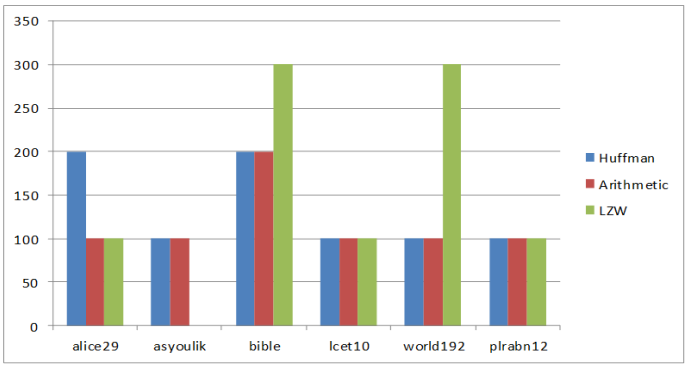

Figure 5 - Compression time

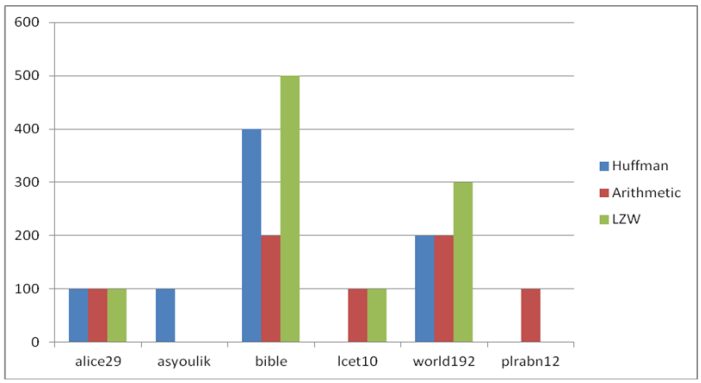

Figure 6 - Decompression Time

3.2. Discussion

Table 4.1 shows the utilized filenames, compression and decompression times in milliseconds for each of those files, the original file sizes in bytes and the compressed file sizes for each file as performed by each of the three techniques, Huffman, Arithmetic and LZW. LZW, with compression ratio of 2.36314, is indicated as the best one in terms of Compression ratio, followed by Arithmetic (compression ratio of 1.70534) which shows little difference compared with Huffman (with compression ratio 1.628766). It is observed that the time for compression and decompression depends highly on the type and the content of the files. It is also of note that the performance of the data compression techniques depends on the characteristics of the files, the different symbols contained in it, and symbol frequencies.

Similar tests conducted on certain algorithms including LZW, Huffman, HFLC and FLC produced LZW as leading algorithm, in line with this research. One of these tests [14] could be said to be biased because the algorithms were implemented using different programming languages (Java for LZW and C++ for Huffman, HFLC and FLC). Another [15] could also be said not to be exhaustive and, therefore not conclusive, because the data set for the test was not standard data set, which could have introduced certain errors. The results of this research prove to be more rigorous, even though the results of closely related works were corroborated.

4. Conclusion

The research proffered Huffman, Arithmetic and LZW algorithms as dominant algorithms. It also, incidentally, provided Java codes for the three algorithms. It, in addition, provided evidence to support LZW as the leading algorithm in terms of compression ratio. The use of LZW for compression where compression ratio is premium or crucial is, therefore, supported. The time for compression and decompression depends highly on the type and the content of the files, in terms of different symbols contained in it, and symbol frequencies. Further work should be done in providing intelligent text data compression.

REFERENCES

[1] M. Sharma, "Compression using Huffman coding". *IJCSNS International Journal of Computer Science and Network Security, 10*(5), pp 133-141, 2010.

[2] Khalid, Sayood (2003). Lossless Compression Handbook. San Diego: Academic Press Series Communication, Networking, and Multimedia, 2003.

[3] L. Claudio, Predictive data compression using adaptive arithmetic coding. Louisiana State University. The department of electrical engineering, 2007

[4] G.E. Blelloch, Introduction to data compression. *Computer Science Department, Carnegie Mellon University*. 2001.

[5] N. PM, R.M. Chezian, "A Survey on Lossless Dictionary Based DataCompression Algorithms". *International Journal of Science, Engineering and Technology Research, 2*(2), pp-256, 2013.

[6] M.Simpson, S. Biswas, R. Barua, "Analysis of Compression Algorithms for Program Data". *University of Maryland* 2003.

[7] S. Seok-Won, "Dictionary-Based Code compression techniques using Bit-mask for Embedded Systems". *Msc Dissertation* , University of Florida 2006.

[8] D. Huffman, "A method for the construction of Minimum Redundance codes". *In Proc. Of IRE 40(9) pp. 1098-1101*, 1952.

[9] P.Nageswara, I. Ramesh, K. Gouthami, "Data compression Techniques in Modern Communication Networks for Enhanced Bandwidth Utilization". *Georgian Electronic Scientific Journal*: Computer Science and Telecommunications, 2009.

[10] A. Aqrawi, "Effects of Compression on Data Intensive Algorithms," *Msc Dissertation* Norwegian University of Science and Technology, Department of Computer and Information Science, 2010.

[11] N. Abramson, " *Information theory and coding"* (Vol. 61). New York: McGraw-Hill, 1963

[12] I.H. Witten, R. M. Neal, J. G. Cleary, J. G. "Arithmetic coding for data compression". *Communications of the ACM, 30*(6), pp 520-540, 1987.

[13] V. B. Raju, K. J. Sankar, C. D.Naidu, C. D. "Performance Evaluation of Basic Compression Technique for Wireless Text Data". *International Journal of Advanced Trends in Computer Science and Engineering, Vol.2, No.1, Pages: 383-387,* 2009.

[14] H. Altarawneh, M. Altarawneh, "Data Compression Techniques on Text Files" A Comparison Study. *International Journal of Computer Applications, 26*, 2011.

[15] N. Jacob, P. Somvanshi, R. Torneka, "Comparative Analysis of Lossless Text Compression Techniques". *International Journal of Computer Applications, 56*, 2012.

[16] S. Shanmugasundaram, R. Lourdusamy, " A Comparative Study Of Text Compression Algorithms". *International Journal of Wisdom Based Computing, 1*(3), 68-76, 2011.