Journal of Computer and Communications, 2018, 6, 301-315
http://www.scirp.org/journal/jcc
ISSN Online: 2327-5227
ISSN Print: 2327-5219

A Complexity Analysis and Entropy for Different Data Compression Algorithms on Text Files

Mohammad Hjouj Btoush, Ziad E. Dawahdeh

Department of Computer Science, Al-Balqa Applied University, Mutah University, Karak, Jordan
Email: hujooj@bau.edu.jo, m_ziad_d@yahoo.com

How to cite this paper: Btoush, M.H. and Dawahdeh, Z.E. (2018) A Complexity Analysis and Entropy for Different Data Compression Algorithms on Text Files. *Journal of Computer and Communications*, **6**, 301-315.
https://doi.org/10.4236/jcc.2018.61029

Received: November 21, 2017
Accepted: January 9, 2018
Published: January 12, 2018

Copyright © 2018 by authors and Scientific Research Publishing Inc.
This work is licensed under the Creative Commons Attribution International License (CC BY 4.0).
http://creativecommons.org/licenses/by/4.0/
Open Access

Abstract

In this paper, we analyze the complexity and entropy of different methods of data compression algorithms: LZW, Huffman, Fixed-length code (FLC), and Huffman after using Fixed-length code (HFLC). We test those algorithms on different files of different sizes and then conclude that: LZW is the best one in all compression scales that we tested especially on the large files, then Huffman, HFLC, and FLC, respectively. Data compression still is an important topic for research these days, and has many applications and uses needed. Therefore, we suggest continuing searching in this field and trying to combine two techniques in order to reach a best one, or use another source mapping (Hamming) like embedding a linear array into a Hypercube with other good techniques like Huffman and trying to reach good results.

Keywords

Text Files, Data Compression, Huffman Coding, LZW, Hamming, Entropy, Complexity

1. Introduction

Data compression has important applications in the areas of data transmission and data storage despite of the large capacity storage devices that are available these days. Hence, we need an efficient way to store and transmit different types of data such as text, image, audio, and video to reduce execution time and memory size [1].

In 1977, Abraham Lempel and Jakob Ziv created the first of what we now call the LZ family of substitution compressors [2]. Lempel-Ziv-Welch (LZW) is a universal lossless data compression algorithm created by Abraham Lempel, Jacob Ziv, and Terry Welch. LZW is a general compression algorithm capable of working on almost any type of data [3].

The general principle of data compression algorithms on text files is to transform a string of characters into a new string, which contains the same information, but with new length as small as possible. The efficient data compression algorithm is chosen according to some scales like compression size, compression ratio, processing time or speed, and entropy [4].

Data compression is very important in business and data processing, which reduces data volume and cost of saving it [5].

Data compression is necessary in many fields in application data processing and also it is very important in distributed systems and data transfer. Data compression is a part of information theory helping in reducing data redundancy over network [6]. So it is very important to study different data compression algorithms used on text files to find what the efficient algorithm is that reduces data volume with saving quality of data.

1.1. Definition: Compression Size

Is the size of the new file in bits after compression is complete?

1.2. Definition

Compression ratios a percentage that results from dividing the compression size in bits by the original file size in bits and then multiplying the result by 100%.

1.3. Definition: Processing Time or Speed

Is the time in millisecond that we need for each symbol or character in the original file for compression? It results from dividing the time in millisecond that is needed for compressing the whole file by the number of symbols in the original file and scales as millisecond/symbol.

1.4. Definition: Entropy

Is the number that results from dividing the compression size in bits by the number of symbols in the original file and scales as bits/symbol?

1.5. Definition: Symbol Probability

A probability for each symbol in the original file is calculated by dividing the frequency of this symbol in the original file by the number of the whole symbols in this file.

1.6. Definition: Hamming Weight

Is the number of ones in the N-bits (fixed-length) code word [7]?

In Section 2, four different data compression techniques (LZW, Huffman, Fixed-length code (FLC), and Huffman after using Fixed-length code (HFLC)) are reviewed and explained. In Section 3, these techniques are tested on different text files with different sizes and the results are tabulated and analyzed. Finally, Section 4 presents the conclusions and future work.

2. Data Compression Techniques

In this section, we will give a short review and explanation with an example for each one of the four techniques that we check in this paper. We use, as an example, the following string of characters as input string
S = "/WED/WE/WEE/WEB/WET" in all techniques and see the compress file that results [8] [9]. Note that the results on this example do not represent standard results and not scale the efficient of those techniques but only as an example because the size of the string (file) is very small.

2.1. LZW

In 1977, Abraham Lempel and Jakob Ziv created the first of what we now call the LZ family of substitutional compressors. In 1984, Terry Welch modified the LZ78 compressor for implementation in high-performance disk controllers. The result was LZW algorithm that is commonly found today [2].

LZW is a general compression algorithm capable of working on almost any type of data [3]. LZW compression creates a table of strings commonly occurring in the data being compressed, and replaces the actual data with references into the table. The table is formed during compression at the same time at which the data is encoded and during decompression at the same time as the data is decoded [10].

The algorithm is surprisingly simple. LZW compression replaces strings of characters with single codes. It does not do any analysis of the incoming text. Instead, it just adds every new string of characters it sees to a table of strings. Compression occurs when a single code is output instead of a string of characters. It starts with a "dictionary" of all the single character with indexes 0.255. It then starts to expand the dictionary as information gets sent through. Pretty soon, redundant strings will be coded as a single bit, and compression has occurred [11]. This means codes 0 - 255 refer to individual bytes, while codes 256 - 4095 refer to substrings [11]. By applying LZW algorithm on the example S, we get the following see **Table 1**.

The compression ratio = 144/152 × 100% = 94.73% from the original size, it means that it saves 5.27% in space or storage of the new file. And entropy = 144/19 = 7.578 bits/symbol instead of 8 bits/symbol in ASCII (where 19 is the number of symbols in the file or string).

The string table fills up rapidly, since a new string is added to the table each time a code is output. In this highly redundant input, 5 code substitutions were output, along with 7 characters. If we were using 9 bit codes for output, the 19 character input string would be reduced to a 13.5 byte output string. Of course, this example was carefully chosen to demonstrate code substitution. In real world examples, compression usually doesn't begin until a sizable table has been built, usually after at least one hundred or so bytes have been read in.

Table 1. The compression process of LZW (S = /WED/WE/WEE/WEB/WET).

Character	Code output	New code	New string
/W	/	256	/W
E	W	257	WE
D	E	258	ED
/	D	259	D/
WE	256	260	/WE
/	E	261	E/
WEE	260	262	/WEE
/W	261	263	E/W
EB	257	264	WEB
/	B	265	B/
WET	260	266	/WET
EOF	T	2	/W
Total = 152 byte	Compressed size= 12 string *12 byte =		

2.2. Huffman Algorithm

Huffman algorithm is the oldest and most widespread technique for data compression. It was developed by David A. Huffman in 1952 and used in compression of many type of data such as text, image, audio, and video. It is based on building a full binary tree for the different symbols that are in the original file after calculating the probability for each symbol and put them in descending order. After that, we derive the code words for each symbol from the binary tree, giving short code words for symbols with large probabilities and longer code words for symbols with small probabilities [1]. By applying Huffman algorithm on the example above, we get the descending probabilities shown in Table 2.

Moreover, the binary tree was built as in Figure 1.

Then, we get the code word for each symbol from the binary tree as in Table 3.

The compressed file for this string (file) will be:
011000111011000001100000001100011000110001101 = 43 bits, instead of 19 8 = 152 bits in ASCII.

2.3. Fixed-Length Code (FLC)

Most of compression text methods are done into an arbitrary fixed-length binary code 8-bit ASCII code, which is called a byte wise basis (character wise basis). Limited research has been done on a bitwise basis instead of the conventional byte wise basis [9]. The new technique: Fixed-length code (FLC) deals with more effective approach for English text source encoding, it is based on transforming the characters in the source text to be compressed onto a new weighted

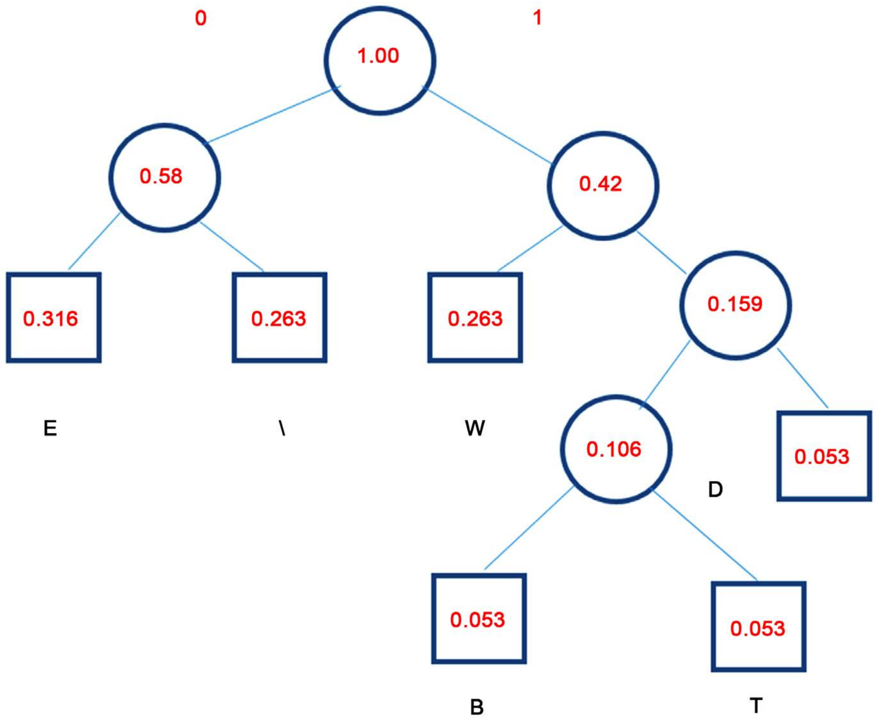

Figure 1. Binary tree for S.

Table 2. Descending probabilities for symbols in S**.**

Symbol	Probability
E	6/19 = 0.316
/	5/19 = 0.263
W	**1.1 5/19 = 0.263**
D	1/19 = 0.053
B	1/19 = 0.053
T	1/19 = 0.053

Table 3. Code words for each symbol in S.

Symbol	Probability	Codewords
E	0.316	00
/	0.263	01
W	0.263	10
D	0.053	111
B	0.053	1100
T	0.053	1101

fixed-length binary code by using a bitwise basis (its length depends on the number of different symbols in the source text) rather than a byte wise basis (8-bits ASCII) [6] [10]. Now, we apply this new mapping technique on the example S .First, we calculate the probability for each symbol in the source text and put them in descending order. The length of the new N-bit code word is calculated from m = 2 N, where m is the number of the different symbols in the source text file and N is the number of bits (fixed-length) that we need for each character in this text (file) instead of 8-bits. Here m = 6 symbols, so we need 3-bits (N = 3) for each symbol. The symbol with large probability (E) take a code word with large Hamming weight (N), the next $(^{N}{}_{r})$ symbols take a code word with (Nr) Hamming weight, and so on (where r = 1, 2, ⋯, N) [6] [10]. So, we get the results In **Table 4**.

The compress file by this technique is: 1101011110111110101111110101111111110101111100110101111010 = 57 bits. The compression ratio = 57/152 × 100% = 37.5% from the original size, it saves 62.5% in the space or storage. The entropy for this technique in this file is 3 bits/symbol instead of 8 bits/symbol in ASCII.

2.4. Huffman after Using Fixed-Length Code (HFLC)

This technique is a complement to the previous approach. Here, we use Huffman Algorithm on the new fixed-length code (FLC) that we obtained before. First, we calculate the probability of the symbols one and zero from the compressed file that results from the previous technique, then calculate the new probability for each fixed-length code by using the following equation:

$$\text{New probability} = q^{u}\left(1-q\right)^{N-u}$$

where u is the number of one's in the given fixed-length code, $(N - u)$ is the number of zero's in this code, q is the probability of the symbol one, and $(1 - q)$ is the probability of zero [1]. After that, we apply Huffman algorithm on the new probability that we get after sorting them in descending order and building the full binary tree as we done in Section 2.2. By applying this technique on the results that we get from S, the probability for symbol one = 42/57 = 0.737 and the probability for the symbol zero = 1 − 0.737 = 0.263. The new binary tree is represented by **Figure 2**.

The new probability and Huffman code words are illustrated in **Table 5**.

The compressed file that results from this technique is: 000001101000000011000000111000000110110000000110111 = 47 bits. The compression ratio = 47/152 × 100% = 30.92% from the original size, so it saves 69.08% of the storage.

Table 4. Codeword for each symbol in FLC.

Symbol	Probability	Codeword (FLC)
E	0.316	111
/	0.263	110
W	0.263	101
D	0.053	011
B	0.053	100

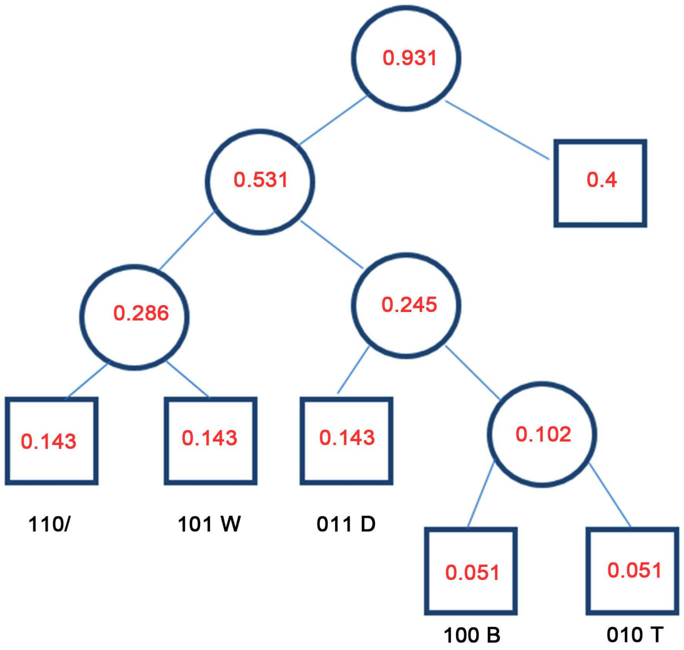

Figure 2. New binary tree.

Table 5. New probability and Huffman code words.

Symbol	Fixed-length code	New probability	Huffman code words
E	111	0.400	1
/	110	0.143	000
W	101	**0.143**	001
D	011	0.143	010
B	100	0.051	0110
T	010	0.051	0111

The entropy for this technique on this file is = 47/19 = 2.474 bits/symbol. Thus, from the results that we got from applying the four techniques (LZW, Huffman, FLC, and HFLC) on the given example, we note that the compression ratios for them are: 94.73%, 28.28%, 37.5% and 30.92%, respectively. So, the best one on this example was Huffman, HFLC, FLC, and LZW respectively. We also note that the entropys for these techniques on this example were: 7.578, 2.263, 3.0, and 2.474 bits/symbol, respectively. It is clear that, the best one on this example was Huffman, then HFLC, FLC, and LZW. But we must note that, these results are not standard but only as an example on each one, because LZW gives best results on the big files but its results are worst on the small files see Table A1 (Appendix A), Figure 3 and Figure 4.

Table A2 (Appendix B) shows the files and the compression ratio for each file in each technique, we note that it is high (worst) in LZW for the small files and low (best) in Huffman, but on the big size the best technique is LZW, then Huffman, HFLC, and FLC, respectively.

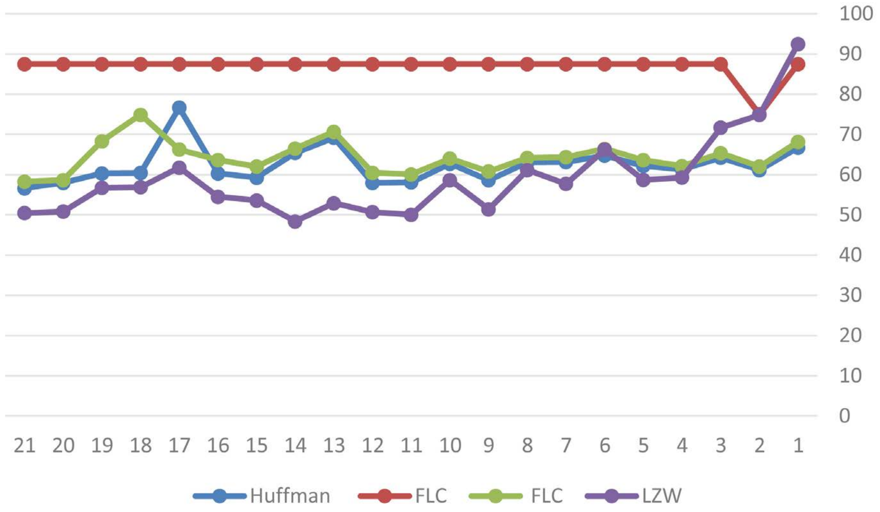

Figure 3. Compression size.

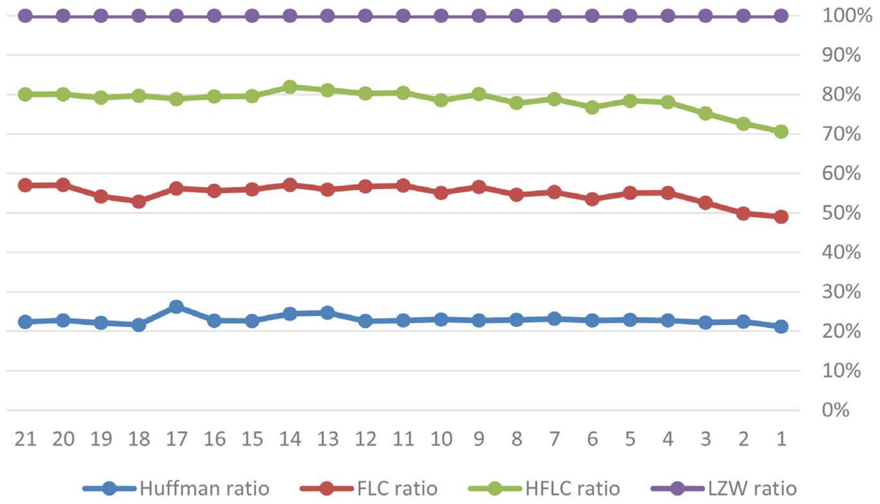

Figure 4. Compression ratio.

3. Analysis and Results

In this Section, tests are made on the four types of techniques on different text files (21 files) from different sizes. Some of these files are taken from the Calgary Corpus; which is a set of traditionally files used to test data compression programs [5]. The results are tabulated and analyzed in order to reach to the best technique, advantage and disadvantage for each one, and when each one is best to use. Source code is written for each technique; in C++ for Huffman, FLC, and Huffman after using FLC, and in Java for LZW. The execution for these programs is done on Pentium 4 with 2.4 G, Ram 248 M, and full cache. The following results are obtained: **Table A1** shows tested files names, original size in bytes and bits, and the new size for each file after compression in the four techniques that we tested. It is clear that Huffman Algorithm is the best one on the small files, then HFLC, FLC, and LZW, but when the size of the files increase LZW will be the best one, then Huffman, HFLC, and FLC, respectively.

Table A2 (Appendix B) shows the files and the compression ratio for each file in each technique, we note that it is high (worst) in LZW for the small files and low (best) in Huffman, but on the big size the best technique is LZW, then Huffman, HFLC, and FLC, respectively.

Table A3 (Appendix C) and Figure 5 represents the compression time in milliseconds that is needed for each character in the source files to complete compression in the four techniques. The best (smallest) time is in LZW, the time in Huffman and FLC is nearly the same, and in HFLC is the worst (long) time because as we saw in the example in Section 2.4, we need more calculations before building the binary tree and obtain the code words.

Now, we illustrate all the results that we obtained from all tested files by taking the average for each of: the original size, the compression size, the compression ratio, the compression time, and the entropy in the four techniques. It is clear from Table A4 (Appendix D) and Table A5 (Appendix E), Figures 6-9 that the average of: compression size, compression ratio, compression time, and entropy is the best in LZW, then in Huffman, HFLC, and FLC, respectively.

From all of the above, LZW is the best technique in all of the compression scales that we tested especially on the files of big sizes, then Huffman, HFLC, and FLC, respectively. But we must note that, the performance of the data compression depends on: the characteristics of the files, the different symbols contained in it, and symbols frequencies. We also note that FLC is a good technique and give a good result when the file contained little different characters or symbols,

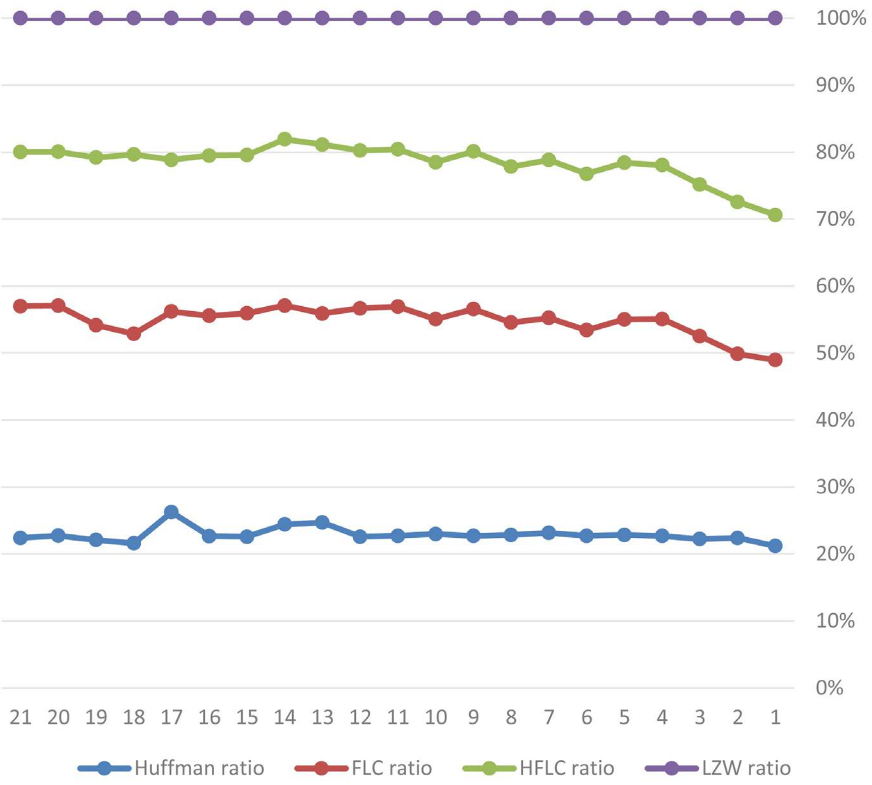

Figure 5. Compression time.

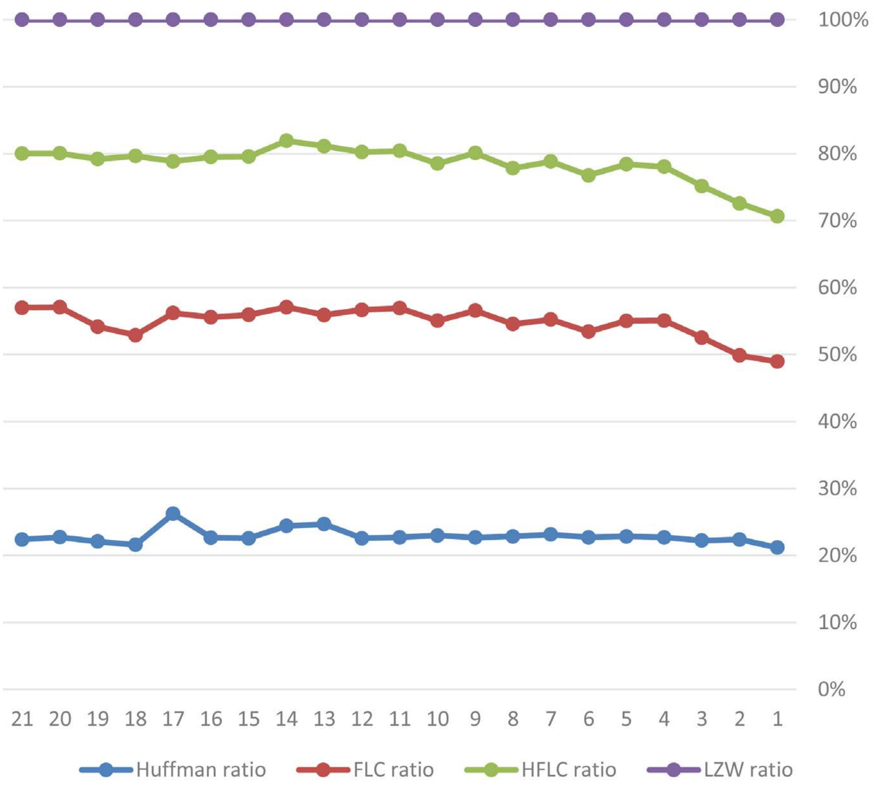

Figure 6. Entropy.

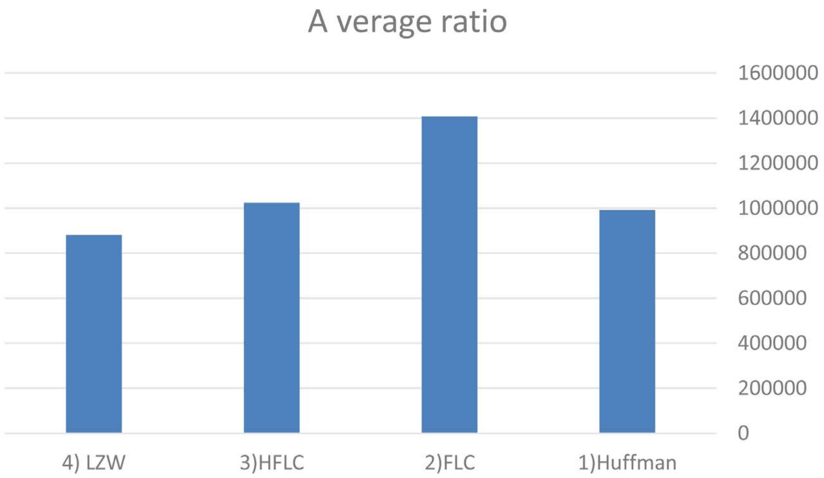

Figure 7. Average ratio.

for example less than 16 different symbols. The advantage of this technique is in the memory space because it deals with 4 bits instead of 8 bits in ASCII for each character. But in fact, we need data compression for large files that contains different characters in order to reduce its size, so this technique will not give a good results on it because if the number of different symbols in the file is more than 64, we will need 7 or 8 bits for each character which is nearly the same as in ASCII, and in this case we need more time for the new calculations that we need in this technique.

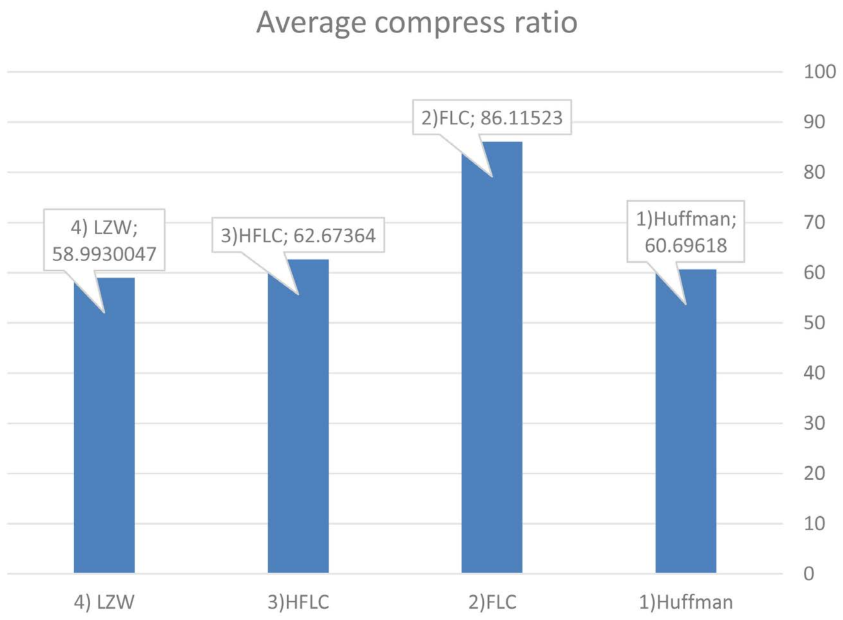

Figure 8. Average compression ratio.

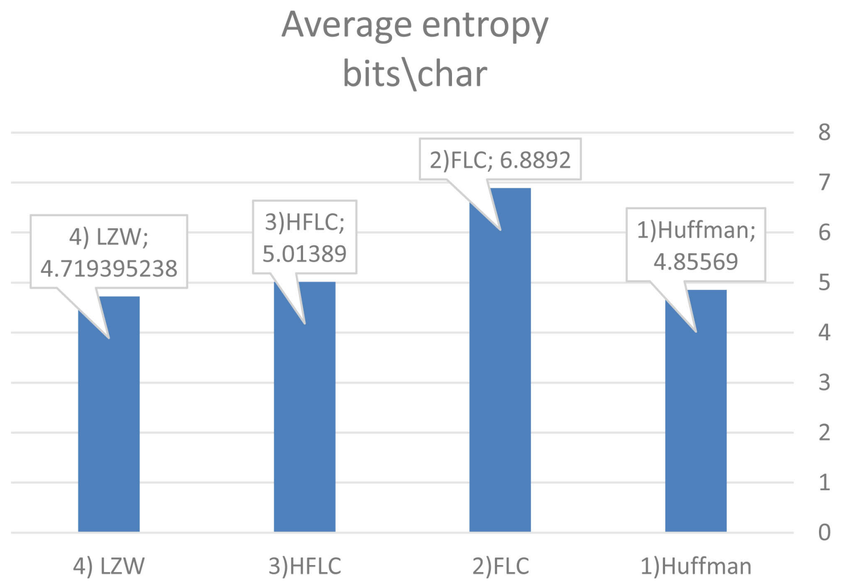

Figure 9. Average entropy.

Another advantage for LZW is that it does not need to pass the large string table to the decompression code, the table can be built exactly as it was during compression [12]. Whereas, in Huffman we must transmit the frequency table for the characters in the source file in order to enable from building the binary tree, which will increase the size of the transmitted (compress) file [13].

4. Conclusions

In this paper, we analyze the complexity and entropy of different methods of data compression algorithms: LZW, Huffman, Fixed-length code (FLC), and

Huffman after using Fixed-length code (HFLC). We test those algorithms on different files of different sizes and then conclude that: LZW is the best one in all compression scales that we tested especially on the large files, then Huffman, HFLC, and FLC, respectively.

From all of the above, LZW is the best technique in all of the compression scales that we tested especially on the files of big sizes, then Huffman, HFLC, and FLC, respectively. But we must note that, the performance of the data compression depends on: the characteristics of the files, the different symbols contained in it, and symbols frequencies.

References

[1] Elabdalla, A.R. and Irshid, M.I. (2001) An Efficient Bitwise Huffman Coding Technique *Based on Source Mapping. Computer and Electrical Engineering*, **27**, 265-272. https://doi.org/10.1016/S0045-7906(00)00022-7

[2] Cheng, C.Y. (2001) Introduction on Text Compression Using Lempel, Ziv, Welch (LZW) Method. http://www.geocities.com/yccheok/lzw/lzw.html

[3] Heggeseth, M. (2003) Compression Algorithms: Huffman and LZW, CS 372: Data Structures. http://web.mit.edu/6.02/www/s2012/handouts/3.pdf

[4] Lelewer, D.A. and Hirschberg, D.S. (1987) Data Compression. http://www.ics.uci.edu/~dan/pubs/DataCompression.html

[5] Hasan, M.R., *et al.* (2013) Comparative Data Compression Techniques and Multi-Compression Results. *IOP Conference Series: Materials Science and Engineering*, **53**, 012081. https://doi.org/10.1088/1757-899X/53/1/012081

[6] Sethi, G., *et al.* (2014) Data Compression Techniques. *International Journal of Computer Science and Information Technologies*, **5**, 5584-5586.

[7] Jaradat, A.R. and Irshid, M.I. (2001) A Simple Binary Run-Length Compression Technique for Non-Binary Source Based on Source Mapping. *Active and Passive Electronic Components*, **24**, 211-221.

[8] Nelson, M. (1989) LZW Data Compression, Dr. Dobb's Journal. http://www.dogma.net/markn/articles/lzw/lzw.htm

[9] MIT 6.02 DRAFT Lecture Notes (2010) Source Coding: Lossless Compression. http://web.mit.edu/6.02/www/f2010/handouts/lectures/L22.pdf

[10] Ferreira, A.J., Oliveira, A.L. and Figueiredo, M.A.T. (2009) On the Suitability of Suffix Arrays for Lempel-Ziv Data Compression. In: Filipe, J., Obaidat, M.S., Eds., e-Business and Telecommunications. *ICETE* 2008, *Communications in Computer and Information Science*, Vol. 48, Springer, Berlin, Heidelberg.

[11] Arimura, M. and Yamamoto, H. (1998) Asymptotic Optimality of the Block Sorting Data Compression Algorithm. *IEICE Transactions on Fundamentals*, E81-A, 2117-2122.

[12] Salomon, D. (2008) A Concise Introduction to Data Compression. Springer-Verlag, London. http://www.springer.com/gp/book/9781848000711

[13] Nelson, M. (1989) LZW Data Compression. Data Compression, Magazine Articles. http://marknelson.us/1989/10/01/lzw-data-compression/

Appendix A

Table A1. List of files before and after compression.

File name	Original size (bytes)	Original size (bits)	Huffman size (bits)	FLC size (bits)	HFLC size (bits)	LZW size (bits)
1) test 1.txt	1024	8192	5463	7168	5585	7576
2) test 2.txt	2048	16,384	10,016	12,288	10,162	12,264
3) test 3.txt	4096	32,768	21,031	28,672	21,407	23,488
4) test 4.txt	8192	65,536	40,199	57,344	40,700	38,848
5) paper 5.txt	11954	95,632	59,445	83,678	60,819	56,136
6) test 5.txt	16384	131,072	84,815	114,688	87,073	86,800
7) test 6.txt	32768	262,144	165,508	229,376	168,711	151,224
8) paper 6.txt	38105	304,840	192,182	266,735	195,599	186,408
9) paper 3.txt	46526	372,208	218,195	325,682	226,239	191,328
10) paper 1.txt	53161	425,288	266,692	372,127	272,009	249,400
11) test 7.txt	65536	524,288	304,480	458,752	314,980	262,344
12) paper 2.txt	82199	657,592	380,918	575,393	397,497	333,312
13) trans.txt	93695	749,560	507,249	641,438	517,973	396,528
14) bib.txt	111,261	890,088	582,085	778,827	591,827	430,752
15) test 8.txt	131,072	1,048,576	621,241	917,504	650,108	561,960
16) test 9.txt	262,144	2,097,152	1,264,664	1,835,008	1,334,646	1,142,976
17) news.txt	377,109	3,016,872	2,312,572	2,639,763	1,998,063	1,862,464
18) test 10.txt	524,288	4,194,304	2,533,927	3,670,016	3,140,471	2,385,400
19) book 2.txt	610,856	4,886,848	2,946,397	3,817,240	2,980,735	2,772,232
20) book 1.txt	768,771	6,150,168	3,564,655	5,381,397	3,605,872	3,126,448
21) test 11.txt	1,048,576	8,388,608	4,748,053	7,340,032	4,887,941	4,233,840
Average	204,274.5238	1,634,196.1	991,894.62	1,407,291.8	1,024,210.33	881,510.857

Appendix B

Table A2. Compression ratio.

File name	Original size (bytes)	Original size (bits)	Huffman ratio (%)	FLC ratio (%)	HFLC ratio (%)	LZW ratio (%)
1) test 1.txt	1024	8192	66.6870	87.50	68.1762	92.4804
2) test 2.txt	2048	16384	61.1328	75.00	62.0239	74.8535
3) test 3.txt	4096	32768	64.1815	87.50	65.3289	71.6796
4) test 4.txt	8192	65536	61.3388	87.50	62.1032	59.2773
5) paper 5.txt	11,954	95632	62.1601	87.50	63.5969	58.7
6) test 5.txt	16,384	131072	64.7087	87.50	66.4314	66.2231
7) test 6.txt	32,768	262144	63.1362	87.50	64.3581	57.6873

Continued

8) paper 6.txt	38,105	304840	63.0435	87.50	64.1644	61.1494
9) paper 3.txt	46,526	372208	58.6217	87.50	60.7829	51.4035
10) paper 1.txt	53,161	425288	62.7085	87.50	63.9587	58.6426
11) test 7.txt	65,536	524288	58.0749	87.50	60.0776	50.0381
12) paper 2.txt	82,199	657592	57.9261	87.50	60.4473	50.6867
13) trans.txt	93,695	749560	69.1949	87.50	70.6578	52.9014
14) bib.txt	111,261	890,088	65.3963	87.50	66.4908	48.3943
15) test 8.txt	131,072	1,048,576	59.2461	87.50	61.9991	53.5926
16) test 9.txt	262,144	2,097,152	60.3038	87.50	63.6408	54.5013
17) news.txt	377,109	3,016,872	76.6546	87.50	66.2296	61.7349
18) test 10.txt	524,288	4,194,304	60.4135	87.50	74.8746	56.8723
19) book 2.txt	610,856	4,886,848	60.2923	87.50	68.3253	56.7284
20) book 1.txt	768,771	6,150,168	57.9603	87.50	58.6304	50.8351
21) test 11.txt	1,048,576	8,388,608	56.6012	87.50	58.2688	50.4713

Appendix C

Table A3. Compression time.

File name	Original size (bytes)	Original size (bits)	Huffman time (ms/char)	FLC time (ms/char)	HFLC time (ms/char)	LZW time (ms/char)
1) test 1.txt	1024	8192	0.9765	0.9765	0.9765	0.0585
2) test 2.txt	2048	16,384	0.4882	0.4882	0.4882	0.0244
3) test 3.txt	4096	32,768	0.2441	0.2441	0.2441	0.0122
4) test 4.txt	8192	65,536	0.12207	0.12207	0.2441	0.0134
5) paper 5.txt	11,954	95,632	0.0836	0.0836	0.0836	0.0092
6) test 5.txt	16,384	131,072	0.06103	0.06103	0.12207	0.0067
7) test 6.txt	32,768	262,144	0.0305	0.0305	0.0610	0.0067
8) paper 6.txt	38,105	304,840	0.0262	0.0262	0.0524	0.0057
9) paper 3.txt	46,526	372,208	0.0214	0.0214	0.0429	0.0058
10) paper 1.txt	53,161	425,288	0.0188	0.0188	0.0376	0.0052
11) test 7.txt	65,536	524,288	0.01525	0.01525	0.0305	0.005
12) paper 2.txt	82,199	657,592	0.0121	0.0243	0.0364	0.0046
13) trans.txt	93,695	749,560	0.0218	0.0218	0.0327	0.00469
14) bib.txt	111,261	890,088	0.0179	0.0179	0.0269	0.00494
15) test 8.txt	131,072	1,048,576	0.0076	0.0076	0.0152	0.0045
16) test 9.txt	262,144	2,097,152	0.0038	0.0076	0.0114	0.0041
17) news.txt	377,109	3,016,872	0.0079	0.0079	0.0132	0.00392
18) test 10.txt	524,288	4,194,304	0.0019	0.00653	0.0076	0.00398
19) book 2.txt	610,856	4,886,848	0.0065	0.0065	0.0110	0.00386
20) book 1.txt	768,771	6,150,168	0.0230	0.0230	0.0345	0.00386
21) test 11.txt	1,048,576	8,388,608	0.0019	0.0028	0.0038	0.00387

Appendix D

Table A4. Entropy.

File name	Original size (bytes)	Original size (bits)	Huffman entropy (bits/char)	FLC entropy (bits/char)	HFLC entropy (bits/char)	LZW entropy (bits/char)
1) test 1.txt	1024	8192	5.3349	7.00	5.4541	7.3984
2) test 2.txt	2048	16,384	4.8906	6.00	4.9619	5.9882
3) test 3.txt	4096	32,768	5.1345	7.00	5.2263	5.7343
4) test 4.txt	8192	65,536	4.9071	7.00	4.9682	4.7421
5) paper 5.txt	11,954	95,632	4.9728	7.00	5.0877	4.696
6) test 5.txt	16,384	131,072	5.1766	7.00	5.3145	5.2978
7) test 6.txt	32,768	262,144	5.0509	7.00	5.1486	4.6149
8) paper 6.txt	38,105	304,840	5.0434	7.00	5.1331	4.8919
9) paper 3.txt	46,526	372,208	4.6897	7.00	4.8626	4.1122
10) paper 1.txt	53,161	425,288	5.0166	7.00	5.1167	4.6914
11) test 7.txt	65,536	524,288	4.6459	7.00	4.8062	4.003
12) paper 2.txt	82,199	657,592	4.6341	7.00	4.8357	4.0549
13) trans.txt	93,695	749,560	5.5355	7.00	5.6526	4.2321
14) bib.txt	111,261	890,088	5.2317	7.00	5.3192	3.8715
15) test 8.txt	131,072	1,048,576	4.7396	7.00	4.9599	4.2874
16) test 9.txt	262,144	2,097,152	4.8243	7.00	5.0912	4.3601
17) news.txt	377,109	3,016,872	6.1323	7.00	5.2983	4.9387
18) test 10.txt	524,288	4,194,304	4.8330	7.00	5.9899	4.5497
19) book 2.txt	610,856	4,886,848	4.8233	7.00	5.4660	4.5382
20) book 1.txt	768,771	6,150,168	4.6368	7.00	4.6904	4.0668
21) test 11.txt	1,048,576	8,388,608	4.5281	7.00	4.6615	4.0377
Sum	4,289,765	34,318,120	104.7817	146	108.0446	99.1073
avg.	204,274.52	1,634,196	4.9896048	6.952380952	5.144980952	4.719395238

Appendix E

Table A5. Average ratio.

Algorithm name	Average original size (bytes)	Average original size (bits)	Average compress size (bits)	Average compress ratio (%)	Average time (ms/char)	Average entropy (bits/char)
1) Huffman	204,274.5238	1,634,196.19	991,894.62	60.69618	0.10438	4.85569
2) FLC			1,407,291.81	86.11523	0.105408	6.8892
3) HFLC			1,024,210.33	62.67364	0.12265	5.01389
4) LZW			881,510.857	58.9930047	0.00929429	4.719395238