Comparison of Lossless Compression Schemes for WAV Audio Data 16-Bit between Huffman and Coding Arithmetic

Tonny Hidayat

Faculty of Computer Science
Universitas Amikom Yogyakarta
Yogyakarta, Indonesia
tonny@amikom.ac.id

Mohd Hafiz Zakaria, Ahmad Naim Che Pee

Faculty of Information and Communication Technology
Universiti Teknikal Malaysia Melaka
Melaka, Malaysia
hafiz@utem.edu.my, naim@utem.edu.my

***Abstract* – In this paper we give an overview and compare Huffman and Arithmetic algorithm using PCM audio data 16 bits stereo. The motivation for the study is to increase the transmission speed in the information exchange which is greatly dependent upon the size of the information. Compression has become one of the solutions to the above problem. The quest for efficient audio data compression rate is still ongoing and good quality compression particularly for music files are greatly sought after by the industry as well as users. Currently music files are stored in various formats and the paper will focus on a lossless technique. The technique is applied to raw WAV files with a comparison of two algorithms, namely Huffman and Arithmetic, and to compare the parameters of file size, ratio, speed, the factor of compression and waveform. The results are then compared to previous existing research and then present the difference of the test result on other data such as text and images.**

Keywords - component; Audio; Huffman; Arithmetic Coding; Lossless Compression; WAV

I. Introduction

The speed of information transmission that combines text, sound, and images has become a major part of the exchange of information. The speed of delivery was commonly dependent upon the size of the information. One of the solutions to the above problems is run compression before the data, text, sound, and images are transmitted, and then being decompress by the receiver to obtain the original data (decompress).

Data compression is an essential issue. This compression technique is crucial because of the size of the data the more extended the increasingly large and complex was added, for it needs to be supported by the development of the technology of continuous bandwidth (speed for downloading data from the internet) are balanced. It is human nature that they prefer the best quality of data with minimal size (quantity). Considering the issues above, then the solution is the maximization of existing compression, i.e. reducing the slots used by the compressed data.

Given the scenario, there is still a need to compare and determine which algorithm should be used in performing data compression specifically for audio files that are formatted in mp3, and wav as a very sensitive will lose data [1]. The size of WAV files are large, then the required compression to reduce the size of the data. WAV format is usually used as the master data or original audio data compression, which allows it be used without removing the original data is a Lossless compression [2].

Technology has grown rapidly and people quest for quality entertainment has increased [2][1]. Several developments are also observed in the audio aspect. Good audio quality causes the listeners also to enjoy it. The audio content that a person usually listens is usually in the form of music. This sound quality is very attractive to people both as consumers and industry players. Therefore, the format of sounds in a file is very important.

This audio has various formats when stored in a file. In this file storage, audio also requires compression to minimize the storage capacity and ease of access in real time. Files with small capacity are easily accessible and transmitted [3]. The compression technique consists of lossless and lossy. Lossy compression represents a reduction of some frequencies from the original sound and still leaves the frequency of sounds that humans can hear, while lossless compression attempts to shrink the actual file size without reducing the content by combining linear approximations and entropy coding[4]. Through compression, data is minimized to maximize the channels allocated for data. It is used to increase data transmission capacity.

This lossless approach is applied to raw WAV files with a comparison of two algorithms, namely Huffman and Arithmetic [3]. Both of these methods are popular in compression as well as having their advantages and disadvantages. The Huffman algorithm can minimize data for a particular symbol larger than its entropy. This algorithm uses symbols that often appear to represent in a single symbol. Furthermore, the Arithmetic Algorithm depends on the probability of occurrence of symbols. It follows a universal source code procedure that depends on the source statistic model.

II. Literature Review

Compression is a change in the size of the data into a size smaller than before [5].This is useful for storing smaller and easier data in the process of sending data. This compression process is in contrast to the decompression process. Decompression process is a process that attempts to restore data or files that have been compressed into the original form. When restoring the data to its original form, it has the configuration of the refilled contents of the data or files. This often happens the data section after experiencing the decompression process.

Data compression can run in two ways (Lossy and Lossless) [6]–[8]. Symbols to data and encoding symbols that often appear with a small number of bits. Each algorithm on data compression focus on some data transformation. This further minimizes random or reduces the number of symbols and takes advantage of the symbols that often appear and encode some symbols that have smaller bits.

Furthermore, the usual sound heard every day is an analog waveform. These waves are tempered by the air pressure around us, which we can hear with the help of the eardrum. As with pictures, digital audio can be broken down into numbers that can then be processed [9]. Audio data digitization is done by measuring the voltage at multiple points in the vulnerable time and interpreting each measurement in the form of numbers and then writing the numbers in a file. This process is called sampling. The advantage of digital audio is the perfect sound production quality. The perfect quality of reproduction is its ability to reproduce audio signals repeatedly without diminishing sound quality [1]. However, the original sound file has a large size. Therefore, it is necessary to do good compression in order to shrink the file size and maintain its quality.

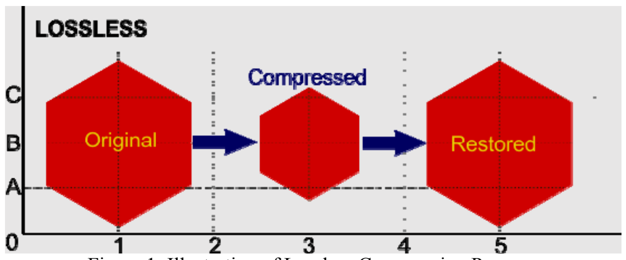

Figure 1: Illustration of Lossless Compression Process.

The data compression approach has two types, that are lossless and lossy compression. Lossless data compression is a class of data compression algorithms that allow the original data to be rearranged from compression data, whereas lossy compression is a method for compressing data and decompressing it. Lossless compression is used to compress data to be accepted in original condition [8]. Whereas lossy compression produces smaller compression files compared to existing lossless methods. Lossless compression is used if the accuracy of sanitary data is important, while lossy compression usually removes parts of the data that are not very useful and not perceived by us so still think that the data can still be used even in compression. The illustration of the former compression approach is shown in Figure 1.

III. Problem Definition and Algorithm Development

In this study, the lossless approach is used to compress WAV files. This approach includes several algorithms such as Huffman and Arithmetic. The Huffman algorithm begins by listing all the alphabetical symbols in the order of their probabilities [10].This then builds a tree, with symbols on each leaf, from bottom to top. This is done by the steps, where at each step the two symbols with the smallest probability are selected, added to the top of the partial tree, removed from the list, and replaced with auxiliary symbols representing the two original symbols. When the list is reduced to just one auxiliary symbol (representing the entire alphabet), the tree is finished. The tree is then passed to determine the symbol code.

A. Huffman Algorithm

Huffman algorithm is a type of optimal prefix code commonly used for data compression without loss. The process of finding or using the code was continued by Huffman coding[11]–[13]. An algorithm is developed by David A. Huffman when he became a D.Sc. student at MIT and published his paper in 1952 entitled "Method for Minimum Redundancy Code Construction”.

The output of the Huffman algorithm can be seen as a variable-length code table for encoding source symbols (such as characters in a file). This algorithm produces this table from the estimated probability or frequency of occurrence (weight) for each possible source symbol value [12], [13]. As with other entropy coding methods, the more common symbols are generally represented using fewer bits than the less common symbols. The Huffman method can be implemented efficiently, finding the code in linear time with the number of input weights if this weight is sorted. Although there are encoding methods partially, Huffman encoding is not always optimal if it is compared by all compression method.

The Huffman algorithm flow begins by assigning some code from each piece of data to become multiple symbols [17]–[19]. Then, the next step is to write the first symbol. This symbol is added to the code for the next symbol. It will check whether there is still the last symbol. If the last symbol does not exist, then the process of recurrence by adding more code to the next symbol. Last symbol checking is done again. If this last symbol does not exist, then the process stops.

The assigning algorithm of encoding to each symbol is begun using determining probability of each symbol. The arrangement of node probability depends on command. Next, combine the 2 least probabilities into a single node. There is a check on whether only 2 left nodes are in a row. If this condition is met, then the process is stopped.

However, if this condition is not met then there is a loop going to the stage of preparation of the probability of the dependent node with the command[18], [19].

After passing the examination of this condition, the stage continues to the process of filling the number 0 at the vertex of the node and 1 at the base of the node. Furthermore, it leads to a separate node in the node below the line in question. There is a check on the condition whether there is any probability of a symbol. If this does not exist, then there is a loop toward the filling stage of value 0 at the vertex of the node and 1 below the node. If the examination shows that there is a probability of the symbols appearing then reading a bit from the end line to the original line streams to form the word code.

Examples of compression technique using Huffman on text files are shown below. Suppose a text file whose contents are represented by "AAAABBBCCCCCD". This file has a size of 13 bytes and one-character equal to 1 byte. Based on the descriptions above, compression can be done as follows:

a. Noting the existing character and number of each character. A = 4, B = 3, C = 5, D = 1.
b. Sorts the characters of which there are at least to most: D, B, A, C.
c. Create a binary tree based on a sequence of characters that has a frequency of smallest to largest.
d. Replace the existing data with the code bits based on binary tree that is created. Replacement of characters into binary code, as seen from the top node or root node is called: A = 01, B = 001, C = 1, D = 000.
e. Next, based on the binary code of each of these characters, all characters in the file can be changed to: 01010101001001001111110001111111 because the numbers 0 and number 1 represent 1 bit, so the above data bits consist of 32 bits or 4 bytes (1 byte = 8 bits).
f. The way to store data type of character is sorting it from the data which have highest frequency to lowest so it becomes: [C,A,B,D] such as data symbol above. After it has been compression, data had size about 1+4+4=9 byte. This amount consists of 1 byte character code which has lowest frequency, 4 character type = 4 byte and 4 byte of whole character code.

B. Arithmetic Coding

Arithmetic coding has important history because at the algorithm has successfully replaced the Huffman coding for 25 years. Arithmetic coding is part of entropy encoding which converts data into a other form with more often uses a little bit and seldom uses more bits characters [18]. This coding technique separates the input message into symbols and swaps each symbol with a floating-point. Arithmetic coding encodes the entire message into a fractional number n where (0.0 = n <1.0) [20], [21].

Arithmetic coding algorithm replace a sequence of inputted symbols with floating point. As the encoded messages are longer and complex, Some bits are needed for this requirement. A sequence of symbols into a unique number with interval (0.1) and notation (0.1) has mean that all real numbers from 0 to 1, including 0 but excluding 1 or it can be written with 0 = x < 1. This number can be uniquely decoded so it generates a sequence of symbols which are used to generate these numbers [22], [23].

As a note of this table, each character has mentioned range, except the number which has high frequency. So, symbol "3e" has range from 0.8 to 0.9999. Next, the encoding is explained according to below:[5].

```
1. Set low = 0.0 (initial conditions)
2. Set high = 1.0 (initial conditions)
3. While (input symbol is still there) do
4. Take the symbol input.
5. CR = high – low.
6. High = low + CR*high_range(symbol)
7. Low = low + CR*low_range(symbol)
8. End while
9. Print low
```

Algorithm above, such as low, high and CR state lower and upper limit then interval range of each symbol (character). At beginning, low and high is initialized using 0 and 1 then next process of both is updated using formulation:

1. High = low + CR*high_range(symbol)
2. Low = low + CR*low_range(symbol)

Generally, the steps to compressing audio file using Arithmetic Coding as follow:

1. Open the audio file to read the header and audio sample.
2. Read audio files to get the data sample.
3. Take the value of the audio sample from 1 to n.
4. Arrange audio samples on the list chain with special characters' list delimiter among data sample audio (#).

Reading of the results from the audio data sample obtained starts from sample 1 with a #1 until sample 12 with a #12 on the last block of the audio data sample. Assume value chunk of audio sample value 1 (#1) above to be encoded as: 10-1,-1, 0, 7, 3d, 0,-1, 0, 7, 7,-1,-1, 3d and 0.

From the data that will be encoded on top, then it will ~~be~~ create a table of probabilities such as table I.

TABLE I. PROBABILITY TABLE

No	Value	Frequency	Probability
1	-1	5	5/15=0,33
2	10	1	1/15=0,06
3	07	3	3/15=0,20
4	3d	2	2/15=0,13
5	0	4	4/15=0,26

Then, it will obtain range table that shown in Table II.

TABLE II. TABLE RANGE OF PROBABILITY

No	Value	Frequency	Probability	Range
1	-1	5	5/15=0,33	0,0 ≤ -1 < 0,33
2	10	1	1/15=0,06	0,33 ≤ 10 < 0,39
3	07	3	3/15=0,20	0,39 ≤ 07 < 0,59
4	3d	2	2/15=0,13	0,59 ≤ 3d <0,72
5	0	4	4/15=0,26	0,72 ≤ 0 < 0,98

Description:

1. 0,0 = -1 < 0,33: The value "-1" has a range from 0.0 to 0.33.
2. 0,33 = 10 < 0,39: The value of "10" has a range from 0.39 to 0.33.
3. 0,39 = 07 < 0,5: The value "7" has a range from 0.39 to 0.59.
4. 0,59= 3d < 0.72: The value of "3d" has a range from 0.72 to 0.59.
5. 0,72 = 0 < 0,98: The value of "0" has a range from 0.72 to 0.98.

From data table, the result of encoding can be seen in the table III.

TABLE III. RESULTS OF AUDIO ENCODING SAMPLE

No	Value	Low	High	CR
Start		0	1	1
1	-1	0	0,33	1
2	10	0,1089	0,1287	0,33
3	07	0,116622	0,120582	0,0198
4	3d	0,1189584	0,1194732	0,00396
5	0	0,119329056	0,119462904	0,0005148

On this proses, low value for last data is low = 0.119329056. It will be used for replace encode audio sample namely -1, 10, -1, 0, 07, 3d, 0, -1, 0, 07, 07, -1, -1, 3d and 0. Furthermore, sample 1 is #1 -1 10 -1 0 07 3d 0 -1 0 07 07 -1 -1 3d 0 and they change become #1 0,119329056. Next, It will replace for sample 2 using low value and becomes 0, xxxxxxxxx etc.

To perform the decompression of audio files, then do decoding process, as follows:

1. Take the encoded symbol (ES).
2. Repeat.
3. Search range of symbols enclosing encoded symbol (ES)
4. Print symbol.
5. CodeRange ← high_range – low_range
6. ES = ES – low_range.
7. ES = ES/CodeRange.
8. Until the symbol is exhausted.

In this matter, empty symbol can be signed (symbol) using special symbol where In this research uses #. Encoded message can be decoded according to below.

ES = 0,119329056

Compare these values with the following symbol range in the following Probability Range Table IV.

TABLE IV. RANGE OF PROBABILITY

No	Value	Frequency	Probability	Range
1	-1	5	5/15=0,33	0,0 ≤ -1 < 0,33
2	10	1	1/15=0,06	0,33 ≤ 10 < 0,39
3	07	3	3/15=0,20	0,39 ≤ 07 < 0,59
4	3d	2	2/15=0,13	0,59 ≤ 3d <0,72
5	0	4	4/15=0,26	0,72 ≤ 0 < 0,98

The calculation is stopped because the retrieved value ES = 0. The results of the above calculation can be seen in table V.

TABLE V. THE RESULTS OF DECODING THE AUDIO SAMPLES

No	ES	Value	Low	High	CR
1	0,3616032	-1	0	0,33	0,33
2	0,52672	10	0,33	0,39	0,06
3	0,6836	07	0,39	0,59	0,2
4	0,72	3d	0,59	0,72	0,13
5	0 (Finish)	0	-	-	-

Description:

1. Then the retrieved ES = 0,3616032
2. Corresponding to the value of the sample #1 *audio* "-1, 10, 1, 0, 07, 3d, 0, -1, 0, 07, 07, -1, -1, 3d and 0"

IV. Implementation and Indicator

The comparison parameters of the Huffman Algorithm and the Arithmetic consist of speed, factor, ratio, waveform and file size after compression. Speed is measured from the start of the compression stage to completion. The unit for this speed determination is second. Then, the compression factor is the size of the data that shrinks during compression. It is measured in percentage form. Likewise the compression ratio, is also in the form of a percentage which is a representation of the size comparison after compression is done. If factor and compression ratio are added then it will yield 100% value. This is also achieved by observing the file size after compression. The last parameter of the comparison result is the waveform. The results of this test show the existing wave images. The existing waveform is compared with the patterns after the compression process is complete.

There are different criteria to measure the performance of a compression algorithm. However, the main concern has always been the space efficiency and time efficiency.

Compression Ratio is a ratio between the size of the compressed file and the size of the source file. The compression factor is the inverse of the compression ratio [3][24][25][26][27].

$$Compression\ Ratio = \frac{Size\ after\ compression}{Size\ before\ compression} \times 100 \quad (1)$$

$$Compression\ Factor = \frac{Size\ before\ compression}{Size\ after\ compression} \times 100 \quad (2)$$

Speed (in compression and decompression): how fast is it? When evaluating data compression algorithms, compression and decompression speed must be taken into consideration. Compression speed is the number of uncompressed bits that can be handled in one second, and decompression speed is the number of compressed bits that can be handled per second. Compression and decompression speed can be calculated according to [28][29].

$$Compression\ Speed = \frac{Uncompressed\ bits}{Seconds\ to\ compress} \quad (3)$$

$$Decompression\ Speed = \frac{Compressed\ bits}{Seconds\ to\ compress} \quad (4)$$

To perform data compression in Matlab, we only use WAVE Sound file (*.Wav). This file was originally obtained (J.S.Bach; Partita E major, Gavotte en rondeau (excerpt) - Sirkka Väisänen, violin), from resource http://www.music.helsinki.fi/tmt/opetus/uusmedia/esim/index-e.html.

The data file is "a2002011001-e02.wav" (Original recording (PCM encoded 16 bits per sample, sampling rate 44100 Hertz, stereo, Duration 54.3 Second). Size 9.13 MB (9,580,594 bytes).

A. Matlab Script of Huffman Coding

```
compression_t = 0;
for i = 1 : size(signal,2)
   signal_pross = signal(:,i);
   [symbols,~,ic] = unique(signal_pross);
   p = tabulate(signal_pross);
   p(p(:,3) == 0,:) = [];
   pp = p(:,2);
   pp2{1,i} = pp;
   p = p(:,3);
   p = reshape(p,1,length(p));
   p2{1,i} = p;
   tic;
   dict = huffmandict(symbols,p/100);
   dict2{1,i} = dict;
   codex = dict(:,2);
   codex2 = codex(ic);
   comp = cell2mat(codex2');
   signal_comp{1,i} = comp;
   comp_t = toc + comp_t;
end
dcomp_t = 0;
tic;
for i = 1 : size(signal,2)
   dsig(:,i) = huffmandeco(signal_comp{1,i},dict2{1,i});
   dcomp_t = toc+dcomp_t;
end
```

Limitations of Huffman Coding:

1. This code gives an optimal solution if the probability of distribution in source symbols has been known.
2. The encoding of each symbol is an integer number of bits.
3. If the source statistics is changed then Huffman coding is not efficient.

Because the probability of symbol has long size, storage in a single word or basic storage is complex.

B. Matlab Script of Arithmetic Coding.

```
play2_state = 0;
comp_t = 0;
for j = 1 : size(signal,2)
   signal_pross = signal(:,j);
   symbol = unique(signal_pross);
   symbol2{1,j} = symbol;
   seq = zeros(1,length(signal_pross));
   for i = 1:length(signal_pross)
      seq(i) = find(symbol==signal_pross(i));
   end
   seq2{1,j} = seq;
   p = tabulate(signal_pross);
   p(p(:,2) == 0,:) = [];
   p = p(:,2);
   p = reshape(p,1,length(p));
   p2{1,j} = p;
   tic;
   signal_comp{1,j} = arithenco(seq,p);
   comp_t = toc+comp_t;
end
dcomp_t = 0;
for j = 1 : size(signal,2)
   tic;
   dseq =
arithdeco(signal_comp{1,j},p2{1,j},length(seq2{1,j}));
   symbol = symbol2{1,j};
   for i = 1:length(dseq)
      a = dseq(i);
      dsig(i,j) = symbol(a);
   end
   dcomp_t = toc+dcomp_t;
```

```
end
```

The Compression Process:

```
Input      : file audio
Output     : file Compressed
Process    : Read audio files from the initial byte to the final byte of data
Set format chunk file audio (data information audio files)
Set data chunk (the data to be compressed)
Calculate the low range and high range on CR
Set low = 0,0 (initial conditions)
Set high = 1,0 (initial conditions)
While (the input symbol is still there)
Do
Take the symbol input Take the symbol input
 CR = high – low
High = low + CR * high_range (symbol)
Low = low + CR * low_range (symbol)
End while
Print low
Store the results of the mapping on a file (Encoded-Symbol)
```

The Decompression Process

```
Input   : file compressed
Output  : file audio
Process : Read the Last coding values
Take Encoded Symbol (ES)
Do
Search range from the symbol of the enclosing ES
Print Symbol
CR = high_range – low_range
ES = ES – low_range
ES = ES / CR
Until the symbol is exhausted
Store the results of decompression on the audio file.
```

V. Result and Comparisons

The results and comparison of compression that has been done is in table VI.

TABLE VI. PERFORMANCE COMPARISON OF HUFFMAN AND ARITHMETIC ALGORITHMS

Parameter	Huffman	Arithmetic
Original Size (KB)	9356.004	9356.004
Compression Size (KB)	1684.561	3505.8469
Compression Ratio %	18.0051	37.4716
Compression Factor %	81.9949	62.5284
Compression Speed (s)	2.6666	75.4395
Decompression Speed (s)	316.8802	91.5487

Table VI shows the data with the comparison result parameters of the Huffman and Arithmetic. The comparison is done with some predefined parameters. The compression object is a sample file with WAV format. This file was originally sized 9356.004 KB. The file is then compressed with Huffman and Arithmetic and the data obtained from the comparison results is shown in Table 6. These data indicate that the Huffman has an advantage in compression size that affects the ratio and its factors as well. In addition, the Arithmetic also has an advantage in speed to restore compressed files (Decompression Speed). However, Huffman's algorithm has an advantage in its speed of compression. Compression ratio reach 18.0051% using Huffman algorithm. It is better than using arithmetic algorithm. Arithmetic algorithm has reached 37.4716%. However, arithmetic is better in decompression speed namely 91.5486 s. Furthermore, Usage Huffman reach 316.8802 s. These data are expected to be reference for next developing both algorithm.

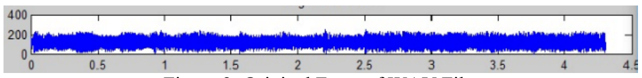
Figure 2: Original Form of WAV File

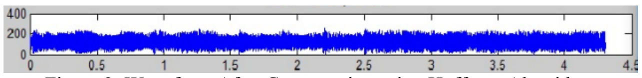
Figure 3: Waveform After Compression using Huffman Algorithm

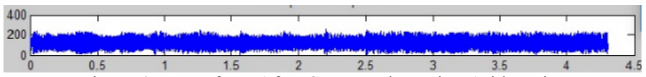
Figure 4: Waveform After Compression using Arithmetic

Furthermore, the actual size and compression result is shown in Figure 2. It is shown in waveform. After compression process is done with both of these methods, the result of waveform is shown in Figures 3 and 4. The compression results in this waveform is shown in Figures 3 and 4. These have similarity with Figure 8. it proves that the compression of both algorithms does not change the waveform. It is caused, the algorithm uses the Lossless approach so no part of the data is omitted. Compression is done by rearranging the encoded data into the new file so there is no signal change. The compressed WAV file has the same quality as before. However, the compression size is smaller than before.

After it is conducted testing of both method using matlab, this gets some differences. Testing result is shown in Table 6. It can be conclude that the result of Lossless compression for wave sound using Encoded PCM 16 bit per sampling and 44100 Hertz of sampling rate as follow:

1. Huffman Compression speed is faster than Arithmetic. However, the decompression speed of Huffman is slower than the Arithmetic.
2. Arithmetic compression has compression ratio of 37.4716% and compression factor 62.5284%. Then, Huffman has 18.0051% of compression ratio and 81.9949% of compression factor. So, Huffman is better

than Arithmetic in compression ratio and compression factor.

VI. Conclusion

The result of compression in Table VI have differences result whether some studies or research result. The differences are:.

1. Compressing arithmetic algorithm is better than Huffman [30][31][32]. Difference of the results in this research shows that Huffman performance is better than Arithmetic in audio data.
2. Compression Ratio of Arithmetic algorithm is better than Huffman algorithm [33][34]. The results shows that Compression Ratio of Huffman is 18.0051% and Arithmetic is 37.4716%. It looks clearly that Huffman has better results than Arithmetic.
3. Compression and Decompression Speed of Huffman is faster than Arithmetic [34][26]. There are similarities and differences. The Compression Speed of Huffman is faster than Arithmetic. However, Decompression Speed Arithmetic is faster than Huffman.
4. Compression using audio data and performance of lossless compression have difference result when they are used on data image and text. So, both algorithm is not always optimal for particular data. It needs to be improved again in setting of the bit allocation well.

The future scope of this research is to compare all of the lossless compression algorithms with the same data so that later it can give a better picture of the results of all the lossless compression methods.

Acknowledgment

This paper is an ongoing research on deeper lossless compression technique for music files. Thank you Universitas AMIKOM Yogyakarta and Universiti Teknikal Malaysia Melaka, and everyone who provided substantial support in this study..

References

[1] L. Firmansah and E. B. Setiawan, "Lossy Audio MP3 format with Huffman Shift Coding Algorithm," vol. 3, no. c, pp. 1–5, 2016.

[2] R. Arshad, A. Saleem, and D. Khan, "Performance comparison of Huffman Coding and Double Huffman Coding," *2016 6th Int. Conf. Innov. Comput. Technol. INTECH 2016*, pp. 361–364, 2017.

[3] R. A. Bedruz and A. R. F. Quiros, *Comparison of Huffman Algorithm and Lempel-Ziv Algorithm for audio, image and text compression*, no. December. 2016.

[4] T. Hidayat, M. H. Zakaria, and A. N. C. Pee, "Lossless coding scheme for data audio 2 channel using huffman and shannon-fano," *J. Theor. Appl. Inf. Technol.*, vol. 96, no. 11, 2018.

[5] A. H. Colt Mcanlis, *Understanding Compression*. 2016.

[6] C. Science, "FINDING PATTERNS IN MUSIC DATA FILES WITH DATA An Abstract of a Thesis Computer Science Western Illinois University AMIN AKHTAR December 2014," no. December, 2014.

[7] A. Benjamin, "Music Compression Algorithms and Why You Should Care," 2010.

[8] G. Gupta and P. Thakur, "Image Compression Using Lossless Compression Techniques," no. December, pp. 3896–3900, 2014.

[9] T. Hidayat, "A Critical Assessment of Advanced Coding Standards for Lossless Audio Compression," pp. 1–10, 1948.

[10] A. D. Mutiara, R. Ramadhan, J. T. Informatika, F. Teknik, and U. H. Oleo, "CODING," vol. 2, no. 1, pp. 29–38, 2016.

[11] S. Mishra, "A Survey Paper on Different Data Compression Techniques Saumya Mishra shraddha singh," no. May, pp. 738–740, 2016.

[12] D. A. Huffmant, "Minimum-Redundancy Codes *," vol. 27, 1951.

[13] R. Fano, "2 Huffman Coding," 1952.

[14] Y. Li and Y. A. O. Liang, "Temporal Lossless and Lossy Compression in Wireless," vol. 12, no. 4, 2016.

[15] M. Arif and R. S. Anand, "Run length encoding for speech data compression," *2012 IEEE Int. Conf. Comput. Intell. Comput. Res.*, pp. 1–5, 2012.

[16] H. Beigi and J. A. Markowitz, "Standard audio format encapsulation (SAFE)," no. May 2010, pp. 235–242, 2011.

[17] G. Brzuchalski, "Huffman coding in advanced audio coding standard," vol. 8454, p. 84540U, 2012.

[18] L. Barua, P. K. Dhar, L. Alam, and I. Echizen, "Bangla Text Compression Based on Modified Lempel-Ziv-Welch Algorithm," pp. 855–859, 2017.

[19] K. Sayood, *Huffman Coding*. 2012.

[20] "13. Compression and Decompression."

[21] Vatolin D.; Ratushnyak A.; Smirnov M.; Yukin V., *Data Compression Methods*. 2003.

[22] D. R. Bull, *Lossless Compression Methods*. 2014.

[23] T. Edition, *No Title*. .

[24] P. V. Krishna, M. R. Babu, and E. Ariwa, *Global Trends in Computing and Communication Systems*, vol. 269. 2012.

[25] R. D. Masram and J. Abraham, "Efficient Selection of Compression-Encryption Algorithms for Securing Data Based on Various Parameters," pp. 1–54, 2014.

[26] S. Porwal, Y. Chaudhary, J. Joshi, and M. Jain, "Data Compression Methodologies for Lossless Data and Comparison between Algorithms," vol. 2, no. 2, pp. 142–147, 2014.

[27] Y.-C. Hu and C.-C. Chang, "A new lossless compression scheme based on Huffman coding scheme for image compression," *Signal Process. Image Commun.*, vol. 16, no. 4, pp. 367–372, 2000.

[28] A. Odat, M. Otair, and M. Al-Khalayleh, "Comparative study between LM-DH technique and Huffman coding technique," *Int. J. Appl. Eng. Res.*, vol. 10, no. 15, pp. 36004–36011, 2015.

[29] S. Kumar, S. S. Bhadauria, and R. Gupta, "A Temporal Database Compression with Differential Method," *Int. J. Comput. Appl.*, vol. 48, no. 6, pp. 65–68, 2012.

[30] S. S and R. L, "Text Compression Algorithms - a Comparative Study," *ICTACT J. Commun. Technol.*, vol. 02, no. 04, pp. 444–451, 2011.

[31] S. Shanmugasundaram, "A comparative study of text compression algorithms," *Int. J.*, vol. 1, no. December, pp. 68–76, 2011.

[32] N. Sharma, J. Kaur, N. Kaur, N. Sharma, J. Kaur, and N. Kaur, "A Review on various Lossless Text Data Compression Techniques," *An Int. J. Eng. Sci.*, vol. 2, no. (December, pp. 58–63, 2014.

[33] G. Kumar, E. Sukhreet, S. Brar, R. Kumar, and A. Kumar, "A Review : DWT-DCT Technique and Arithmetic-Huffman Coding based Image Compression," no. September, pp. 20–33, 2015.

[34] A. J. Maan, "Analysis and Comparison of Algorithms for Lossless Data Compression," *Int. J. Inf. Comput. Technol.*, vol. Vol. 3, no. No. 3, pp. 139–146, 2013.