Journal of Theoretical and Applied Information Technology
30th June 2023. Vol.101. No 12

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

COMPARISON ANALYSIS OF BOLDI-VIGNA ζ2 ALGORITHM AND END-TAGGED DENSE CODE ALGORITHM ON AUDIO FILE COMPRESSION

[1]HANDRIZAL, [2]PAUZI IBRAHIM NAINGGOLAN, [3]AFIF REFANO MUFID
[1,2,3]Department of Computer Science, Faculty of Computer Science and Information Technology, Universitas Sumatera Utara, Jl. Universitas No. 9-A, Medan 20155, Indonesia

E-Mail: handrizal@usu.ac.id

ABSTRACT

In this modern era, everything is done online including data-sending activities. However, the problem faced today is the large size of the data, which causes the length of the sending process and the use of large storage capacity. Therefore, to overcome this problem, data processing techniques are needed, one of which is compression techniques. Data compression aims to reduce the size of the data, reduce the bandwidth required when transmitting data, and reduce the need for data storage. Wav files are widely used in game creation, which is commonly used for sound effects and music, wav files tend to have a large size. So compression techniques are needed to reduce the size of data or files. The Boldi-Vigna algorithm takes advantage of locality and similarity in graph data. This algorithm mostly supports random access, except for the implementation of the required reference compression. In the End-Tagged Dense Code (ETDC) algorithm, instead of using a flag bit to signify the beginning of a codeword, the flag bit is used to signify the end of a codeword. This study tested a comparison of data compression between the Boldi-Vigna ζ2 algorithm and the End-Tagged Dense Code algorithm using audio files in 8-bit format with an extension of *.wav. Based on testing in this study, it was found that the Boldi-Vigna ζ2 algorithm is better in the audio file compression process with an average Ratio of Compression of 73.81%, an average Compression Ratio of 1.58, and an average Space Savings of 26.18%. While the End-Tagged Dense Code algorithm has a better compression time with an average compression time of 1.05 seconds. It can be concluded that the Boldi-Vigna ζ2 algorithm is better at compressing audio files, while the End-Tagged Dense Code algorithm is faster at performing the compression process. The compression results obtained are influenced by the number of the same digital values contained in the wav audio file that needs to be compressed. Both algorithms can restore the whole audio file like the original audio file through the decompression process.

Keywords: *Compression, Audio, Boldi-Vigna ζ2, End-Tagged Dense Code*

1. INTRODUCTION

In this modern era, everything is done online including data-sending activities. This certainly has a positive impact on human life. But the problem faced today is the large size of the data, thus causing the length of the delivery process and u and storage capacity. Therefore, to overcome this problem, data processing techniques are needed, one of which is compression techniques. Data compression aims to reduce the size of the data, reduce the bandwidth required when transmitting data, and reduce the need for data storage.

Audio is a sound or voice produced when molecules in the air are transformed by a motion caused by an object that produces a vibration. This amount of vibration is referred to as the frequency of the vibration. One such forward and backward movement is called a cycle. Then the unit for frequency is cycle per second or cps. This unit is commonly known as Hertz (Hz). Sounds or sounds that can be heard by the human ear range from 20 Hz to 20,000 Hz, also known as audio sonic [1].

Digital audio is a digital version of analog audio, which is processed by converting the amplitude of an analog wave into time intervals (samples) to produce digital audio. In digital audio, analog audio is converted into digital audio so that it can be processed by a digital system or computer using a tool called an Analog to Digital Converter (ADC), then digital audio is converted back into analog audio so that it can be heard by humans using a tool called a Digital to Analog Converter (DAC) [2].

Wav is a standard audio file developed by Microsoft and IBM with a file extension (*.wav). Wav allows sound to be recorded in various

qualities, such as 8-bit or 16-bit with a sample rate of 11025 Hz, 22050 Hz, or 44100 Hz. For good quality, namely: 44100 Hz, 16-bit will take up about 150 Kb of capacity every second. Wav files are widely used in the creation of games, which are commonly used for sound effects and music. Wav files tend to have a large size, it is because the Wav file format is not compressed so it has the advantage of faster loading times [3].

The Boldi-Vigna (ζ2) algorithm was introduced by Paolo Boldi and Sebastiano Vigna as a family of variable-length codes which is a great option for compression. This algorithm takes advantage of locality and similarity in graph data. Boldi-Vigna mostly supports random access, except for the implementation of required reference compression [4]. Boldi-Vigna code zeta starts with a positive k integer, a factor in shrinking the code.

The End-Tagged Dense Code (ETDC) Algorithm is inspired by the Huffman Code Tagged Algorithm and is achieved through very simple changes. Compared to using a flag bit to signify the beginning of a codeword, in the End-Tagged Dense Code (ETDC) Algorithm, the flag bit is used to signify the end of a codeword. That is, the bit flag is 0 for the first bit of each codeword byte except for the last bit, which has 1 on its first bit. The bit flag is sufficient to ensure that the code is a code prefix, apart from the other 7 bits [5].

Data compression is the art or science of representing information in a concise form [6]. Data compression is used to reduce the number of bits generated from each symbol that appears. With this data compression, it is expected to save storage space by reducing data size. File delivery can be done quickly if the file is small, but not all data sent by people has a small capacity so large capacity memory is needed to store all the data needed, and, unstable internet connection speeds take a relatively long time in the process of sending data. Therefore, in large-scale data transmission, a compression process is also required [7].

Data compression can be divided into 2 categories, lossless compression, where Y is equal to X, and lossy compression, where Y is different from X but provides much higher compression than lossless compression [6].

On lossless compression, the compressed data will not lose the information in it, and the data that has been compressed can be decompressed again. Lossless compression uses simple bit operations to reduce the number of bits required to store a single sample thereby reducing the size of the entire audio file [8]. Lossless compression is usually used in data that does not tolerate any discrepancy between the original data and the data after being compressed [6]. Therefore, the data that has been generated after the lossless compression process is maintained authenticity and quality.

In lossy compression, data that has been compressed loses some information, so it cannot be decompressed again, due to some estimation or approximation methods used in algorithms [9]. The compression ratio obtained in lossy compression is much higher than that of lossless compression because some information is omitted in lossy compression [6]. Thus making lossy compression more suitable for use in data that does not require data authenticity after the compression process is carried out.

Based on the above background, which algorithm has the best and most efficient compression results. Therefore, in this research, the author applies a lossless compression method, so that the compressed audio file does not lose data and still maintains its quality. This research uses Boldi-Vigna (ζ2) algorithm and End-Tagged Dense Code (ETDC) Algorithm in compressing wav audio files, to compare the performance between these algorithms.

2. FORMULATION OF THE PROBLEM

In this research, the formulation of the problem discussed is that the large size of the audio file causes the length of the transmission process, as well as the use of a large storage capacity, so a data compression process is needed using the Boldi-Vigna ζ2 Algorithm and the End-Tagged Dense Code Algorithm, to find an algorithm that has the best and efficient compression results.

3. SCOPE AND LIMITATION

The scope and limitations of this research are as follows:

- Comparing Boldi-Vigna ζ2 algorithm and End-Tagged Dense Code algorithm.
- The compressed file type is a *.wav audio file in 8-bit format.
- The audio file used is a maximum of 10 MB in size.
- The parameters used to measure data compression performance are calculating the value of Rc (Ratio of compression), Cr (Compression ratio), Ss (Space savings), and compression time (Running Time).

Journal of Theoretical and Applied Information Technology
30th June 2023. Vol.101. No 12

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

- The programming language used is Kotlin which is created using the Android Studio IDE (Integrated Development Environment).

4. THEORETICAL FRAMEWORK

4.1. Boldi-Vigna ζ2 Algorithm

The Boldi-Vigna Zeta code's algorithm begins with a positive integer k which is the shrinkage factor of the code. The set of all positive integers is partitioned into intervals $[2^0, 2^k - 1]$, $[2^k, 2^{2k} - 1]$, $[2^{2k}, 2^{3k} - 1]$, with a general form $[2^{hk}, 2^{(h+1)k} - 1]$. The length of each interval is $2^{(h+1)k} - 2^{hk}$.

The encoding of the Boldi-Vigna ζ2 algorithm is as follows:

1. The set of positive integers is partitioned into intervals $[2^{hk}, 2^{(h+1)k} - 1]$.
2. Input value of n.
3. The value of $k = 2$, because it uses the Boldi-Vigna ζ2 algorithm.
4. Find n in the interval, starting from $h = 0$.
5. See the unary code for h+1, the unary code used is reverse type.
6. Calculate the minimum binary code, $x = n - 2^{hk}$.
7. Calculate the length of the interval, $z = 2^{(h+1)k} - 2^{hk}$.
8. Calculate $s = \lfloor \log_2 z \rfloor$.
9. If $x < 2^s - z$, then x is encoded as binary code, in $s - 1$ bit. If $x \geq 2^s - z$, then $(x - z + 2^s)$ is encoded as *binary code* in s bits.
10. Boldi-Vigna Code = Unary Code + Binary Code

Boldi-Vigna Code can be seen in Table 1.

Table 1. Boldi-Vigna ζ2 Code

n	ζ2
1	10
2	110
3	111
4	01000
5	01001
6	01010
7	01011
8	011000
9	011001
10	011010
11	011011
12	011100
13	011101
14	011110
15	011111
16	00100000

4.2. End-Tagged Dense Code Algorithm

The End-Tagged Dense Code algorithm is marked with the symbol b bit as the compiler of the codeword, starting with bit 0 for each first bit of a codeword byte, except for the last byte starting with 1 on the first bit.

For values $n = 0$ to $n = 2^{b-1} - 1$, formed by b bits of the codeword, consisting of 1 and followed by a combination of b-1 bits, the value starts from 2^{b-1} to $2^b - 1$. Then for the value of $n = 2^{b-1}$ to $n = (2^{b-1})^2 + (2^{b-1}) - 1$, formed using $2b$ bits of a codeword, the value of which is between [0 and $2^{b-1} - 1$] and the second [2^{b-1} and $2^b - 1$]. Next, the value for $n = (2^{b-1})^2 + (2^{b-1})$ to $n = (2^{b-1})^3 + (2^{b-1})^2 + (2^{b-1}) - 1$ formed using $3b$ bits of a codeword, the value of which is located between [0 and $2^{b-1} - 1$], the second [0 and $2^{b-1} - 1$], and the third [2^{b-1} and $2^b - 1$], and so on.

The encoding of the End-Tagged Dense Code algorithm is as follows:

1. Character sets are sorted by frequency of occurrence.
2. Input value of n.
3. The *value of b* used is 3.
4. The smallest value of the last set of codes, $s = 2^{b-1}$
5. x = *binary code* $((n \bmod s) + s)$
6. $y = n \text{ div } s$, if $y \leq 0$, then the *codeword* of the *End-Tagged Dense Code* algorithm stops there, but if $y > 0$, then the calculation $n_2 = y - 1$ is carried out.

End-Tagged Dense Code can be seen in Table 2.

Table 2. End-Tagged Dense Code

n	ETDC (b =3)	Bit
0	**1**00	3
1	**1**01	3
2	**1**10	3
3	**1**11	3
4	**0 00 1**00	6
5	**0**00 **1**01	6
6	**0**00 **1**10	6
7	**0**00 **1**11	6
8	**0**01 **1**00	6
9	**0 01 1**01	6
10	**0**01 **1**10	6

n	ETDC (*b* =3)	Bit
11	**0**01 **1**11	6
12	**0**10 **1**00	6
13	**0**10 **1**01	6
14	**0 1**0 **1**10	6
...	...	...
19	**0**11 **1 1**1	6
20	**0 0**0 **0 0**0 **1** 00	9
21	**0 0**0 **0**00 **1**01	9
...	...	...
83	**0 1**1 **0**11 **1**11	9
84	**0 0**0 **0 0**0 **0 0**0 **1** 00	12
...	...	...
339	**0 1**1 **0 1**1 **0**11 **1** 11	12
...	...	...

4.3. General Architecture

The general architecture is a system design scheme that describes the overall flow. General architecture can also be a guideline for the creation of system modeling. The general architecture is depicted in Figure 1.

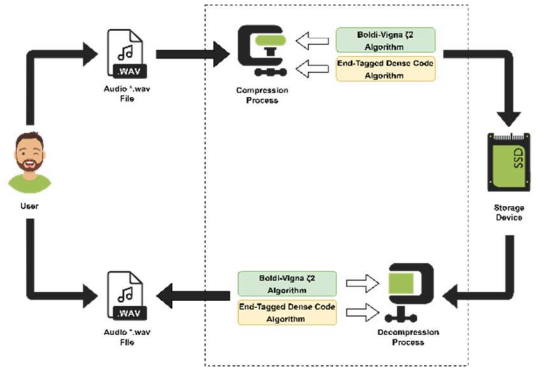

Figure 1. General Architecture

The explanation of the general architecture above is as follows:

1. The user inputs an audio file in wav format into the system.
2. Then the file is compressed using the algorithm the user chooses, which is Boldi-Vigna ζ2 or End-Tagged Dense Code.
3. After that, the system will output a file that has been compressed and saved to the storage device.
4. The compressed file will be decompressed using the algorithm chosen by the user, namely Boldi-Vigna ζ2 or End-Tagged Dense Code.
5. After the decompression process, the system will output in the form of an original audio file in wav format.

4.4. Research Design

Research design is divided into functional requirements and non-functional requirements.

4.4.1. Functional Requirements

Functional requirements are requirements that must exist for the system to run according to its purpose. The functional requirements for the system created are as follows:

1. The system should be able to read the bit values present in audio files with the *.wav extension.
2. The system should be able to perform compression and decompression on audio files using the Boldi-Vigna ζ2 algorithm.
3. The system must be able to compress and decompress audio files using the End-Tagged Dense Code algorithm.
4. The system must be able to display the results of the parameters used to measure the performance of the data compression used, namely Rc (Ratio of compression), Cr (Compression ratio), Ss (Space savings), and compression time (Running Time).

4.4.2. Non-Functional Requirements

Non-functional needs are additional needs that support the capacity of the running system. Non-functional needs in the system created are as follows:

1. Simple
   The system must have a user-friendly appearance, so users can easily use the system.
2. Ease
   The system can be used offline so that it has high mobility and can be used anywhere.
3. Performance
   The system must be able to perform tasks relatively in compressing and decompressing audio files.
4. Economical
   The system can work well without having to spend money.

5. Information
   The system can display information on files.

4.5. Compression Performance Measuring Parameters

Data compression performance can be calculated based on the following parameters:

1. Ratio of Compression (Rc)

   The ratio of compression (Rc) is the percentage of data size after the compression process is carried out with the size of the data before the compression process is carried out.

$$Rc = \frac{\text{Data Size After Compression}}{\text{Data Size Before Compression}} \times 100\%$$

2. Compression Ratio (Cr)

   Compression ratio (Cr) is a comparison of the size of the data before the compression process with the data after the compression process.

$$Cr = \frac{\text{Data Size Before Compression}}{\text{Data Size After Compression}}$$

3. Space Savings (Ss)

   Space Savings (Ss) is a percentage of storage space savings.

$$Ss = \left(1 - \frac{\text{Data Size After Compression}}{\text{Data Size Before Compression}}\right) \times 100\ \%$$

5. RECENT RESEARCH

Research that is relevant to the research that the author will conduct is as follows:

1. Boldi & Vigna [2005] entitled "Codes for the World Wide Web" explains that the Boldi-Vigna code ζ is very suitable for storing Web-Graphs. In the test results, codes ζ2 and ζ3 almost always obtained the best results compared to codes γ (ζ1), ζ4, δ, and nibble. But in tests where there was no internalization code δ performed slightly better [10].
2. Brisaboa et al [2007] entitled "Lightweight natural language text compression" explains that End-Tagged Dense Code obtained a compression ratio 8% higher than Tagged Huffman Code, and 2.5% higher than Plain Huffman Code, and obtained an encoding time of 40% lower than Plain Huffman Code, and obtained the same search time as Tagged Huffman Code [11].
3. Buehrer & Chellapilla [2008] entitled "A Scalable Pattern Mining Approach to Web Graph Compression with Communities" explains that Boldi-Vigna code ζ achieves high edge compression and scales well for large graphs. When compared to Huffman code, Boldi-Vigna code ζ gains no more than a 5% loss in the number of bits per edge, and the typical loss is between 2-3% [12].
4. Claude & Navarro [2010] entitled "Fast and Compact Web Graph Representations" explained that Web-Graph (Boldi-Vigna code ζ) obtained compression ratio results in 1.5-2 times faster to navigate compressed graphs [13].
5. Valencia et al [2010] entitled "Translation Table Compression under End-Tagged Dense Code" explains that in phrase-based translation tables, End-Tagged Dense Code reduces space savings between 30-60% [14].
6. Mancebo et al [2019] entitled "Saving Energy in Text Search Using Compression" explains that End-Tagged Dense Code is a good compression alternative, based on its compression ratio which is around 30-35%, as well as its fast compression and decompression time [5].
7. Lin et al [2019] entitled "Text Compression for Myanmar Information Retrieval" explained that the End-Tagged Dense Code method obtained an average compression ratio on Myanmar language text of 49%. In End-Tagged Dense Code, the higher the word frequency, the higher the compression ratio obtained, so word segmentation and construction of dictionary files are very important [15].

6. PROBLEM ANALYSIS

Problem analysis is the process of identifying problems, the causes of problems, and the consequences of these problems, so that the system can run by the objectives. In this research, the problem discussed is that the large size of the audio file causes the length of the transmission process, and the use of large storage capacity, so a data compression process is needed. In this research, the author compares two algorithms to find out which

algorithm has the best and most efficient data compression results for compressing audio files with *.wav extension, using the Boldi-Vigna ζ2 algorithm and the End-Tagged Dense Code algorithm, with parameters Rc (Ratio of compression), Cr (Compression ratio), Ss (Space savings), and compression time (Running Time).

7. IMPLEMENTATION AND TESTING

The system is created using Kotlin programming language using the Android Studio IDE (Integrated Development Environment). In this study, the file used was Audio files in 8-bit format with a size of 96 bytes and an extension of *.wav. Comparison based on comparison parameters, namely Ratio of Compression, Compression Ratio, and Space Savings to know the effectiveness and efficiency of the two algorithms in compressing the wav audio file. The digital audio value (hex value) can be seen in Figure 2.

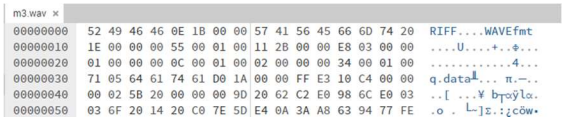

Figure 2. Digital Audio Value

7.1. Compression Using Boldi-Vigna ζ2 Algorithm

The compression process using the Boldi-Vigna ζ2 algorithm is as follows:

1. Digital values contained in the sample audio file with 96 bytes size in Figure 4 are:

   52 49 46 46 0E 1B 00 00 57 41 56 45 66 6D 74 20 1E 00 00 00 55 00 01 00 11 2B 00 00 E8 03 00 00 01 00 00 00 0C 00 01 00 02 00 00 00 34 00 01 00 71 05 64 61 74 61 D0 1A 00 00 FF E3 10 C4 00 00 00 02 5B 20 00 00 00 9D 20 62 C2 E0 98 6C E0 03 03 6F 20 14 20 C0 7E 5D E4 0A 3A A8 63 94 77 FE
2. The hexadecimal digital audio values are sorted first by frequency, then the compression process using the Boldi-Vigna ζ2 algorithm can be seen in Table 3.

Table 3. Calculation of Compression using Boldi-Vigna ζ2 Algorithm Process

Digital Audio	*Boldi-Vigna ζ2 Code*	Frequency	Bit × Frequency
Value (*Hex*)			
00	10	29	58
20	110	5	15
01	111	4	12
03	01000	3	15
02	01001	2	10
46	01010	2	10
61	01011	2	10
74	011000	2	12
E0	011001	2	12
05	011010	1	6
0A	011011	1	6
0C	011100	1	6
0E	011101	1	6
10	011110	1	6
11	011111	1	6
14	00100000	1	8
1A	00100001	1	8
1B	00100010	1	8
1E	00100011	1	8
2B	00100100	1	8
34	00100101	1	8
3A	00100110	1	8
41	00100111	1	8
45	00101000	1	8
49	00101001	1	8
52	00101010	1	8
55	00101011	1	8
56	00101100	1	8
57	00101101	1	8
5B	00101110	1	8
5D	00101111	1	8
62	001100000	1	9
63	001100001	1	9
64	001100010	1	9
66	001100011	1	9
6C	001100100	1	9
6D	001100101	1	9
6F	001100110	1	9
71	001100111	1	9
77	001101000	1	9
7E	001101001	1	9
94	001101010	1	9
98	001101011	1	9
9D	001101100	1	9
A8	001101101	1	9
C0	001101110	1	9
C2	001101111	1	9
C4	001110000	1	9
D0	001110001	1	9
E3	001110010	1	9
E4	001110011	1	9
E8	001110100	1	9
FE	001110101	1	9
FF	001110110	1	9
Total			525 bit

3. Because the number of bits after compression is not a multiple of eight, a calculation is made, namely, 525 mod 8 = 5 → 8 – 5 = 3, then 0 bits are added to the padding bits as much as 3 bits 000. After that, a flag bit was added which showed the number of padding bits, namely 3 (binary code = 00000011) as much as 8 bits 00000011. So that the number of string bits becomes 525 bits + 3 bits + 8 bits = 536 bits or 67 bytes.

After the calculation of compression using the Boldi-Vigna ζ2 algorithm, the data size before compression = 96 bytes, and the data size after compression = 67 bytes. Then compression performance can be calculated based on the comparison parameters:

- $Rc = \frac{\text{Data Size After Compression}}{\text{Data Size Before Compression}} \times 100\%$

  $Rc = \frac{67}{96} \times 100\%$

  $Rc = 69{,}79\%$

- $Cr = \frac{\text{Data Size Before Compression}}{\text{Data Size After Compression}}$

  $Cr = \frac{96}{67}$

  $Cr = 1{,}4328$

- $Ss = \left(1 - \frac{\text{Data Size After Compression}}{\text{Data Size Before Compression}}\right) \times 100\ \%$

  $Ss = \left(1 - \frac{67}{96}\right) \times 100\ \%$

  $Ss = 30{,}2083\%$

7.2. Compression Using End-Tagged Dense Code Algorithm

The compression process using the End-Tagged Dense algorithm is as follows:

1. Digital values contained in the sample audio file with 96 bytes size in Figure 4 are:

   52 49 46 46 0E 1B 00 00 57 41 56 45 66 6D 74 20 1E 00 00 00 55 00 01 00 11 2B 00 00 E8 03 00 00 01 00 00 00 0C 00 01 00 02 00 00 00 34 00 01 00 71 05 64 61 74 61 D0 1A 00 00 FF E3 10 C4 00 00 00 02 5B 20 00 00 00 9D 20 62 C2 E0 98 6C E0 03 03 6F 20 14 20 C0 7E 5D E4 0A 3A A8 63 94 77 FE
2. The hexadecimal digital audio values are sorted first by frequency, then the compression process using the End-Tagged Dense Code algorithm can be seen in Table 4.

Table 4. Calculation of Compression using the End-Tagged Dense Code Algorithm Process

Digital Audio Value (*Hex*)	*End-Tagged Dense Code*	Frequency	Bit × Frequency
00	100	29	87
20	101	5	15
01	110	4	12
03	111	3	9
02	000 100	2	12
46	000 101	2	12
61	000 110	2	12
74	000 111	2	12
E0	001 100	2	12
05	001 101	1	6
0A	001 110	1	6
0C	001 111	1	6
0E	010 100	1	6
10	010 101	1	6
11	010 110	1	6
14	010 111	1	6
1A	011 100	1	6
1B	011 101	1	6
1E	011 110	1	6
2B	011 111	1	6
34	000 000 100	1	9
3A	000 000 101	1	9
41	000 000 110	1	9
45	000 000 111	1	9
49	000 001 100	1	9
52	000 001 101	1	9
55	000 001 110	1	9
56	000 001 111	1	9
57	000 010 100	1	9
5B	000 010 101	1	9
5D	000 010 110	1	9
62	000 010 111	1	9
63	000 011 100	1	9
64	000 011 101	1	9
66	000 011 110	1	9
6C	000 011 111	1	9
6D	001 000 100	1	9
6F	001 000 101	1	9
71	001 000 110	1	9
77	001 000 111	1	9
7E	001 001 100	1	9
94	001 001 101	1	9
98	001 001 110	1	9
9D	001 001 111	1	9
A8	001 010 100	1	9
C0	001 010 101	1	9
C2	001 010 110	1	9
C4	001 010 111	1	9

D0	001 011 100	1	9
E3	001 011 101	1	9
E4	001 011 110	1	9
E8	001 011 111	1	9
FE	010 000 100	1	9
FF	010 000 101	1	9
Total			555 bit

3. Because the number of bits after compression is not a multiple of eight, a calculation is made, namely, 555 mod 8 = 3 → 8 – 3 = 5, then 0 bits are added to the padding bits as much as 5 bits 00000. After that, a flag bit is added which shows the number of padding bits, namely 5 (binary code = 00000101) as much as 8 bits 00000101. So that the number of string bits becomes 555 bits + 5 bits + 8 bits = 568 bits or 71 bytes.

Data size before compression = 96 bytes, and the data size after compression = 71 bytes. Then compression performance can be calculated based on the comparison parameters:

- $Rc = \frac{\text{Data Size After Compression}}{\text{Data Size Before Compression}} \times 100\%$

$$Rc = \frac{71}{96} \times 100\%$$

$$Rc = 73{,}95\%$$

- $Cr = \frac{\text{Data Size Before Compression}}{\text{Data Size After Compression}}$

$$Cr = \frac{96}{71}$$

$$Cr = 1{,}3521$$

- $Ss = \left(1 - \frac{\text{Data Size After Compression}}{\text{Data Size Before Compression}}\right) \times 100\ \%$

$$Ss = \left(1 - \frac{71}{96}\right) \times 100\ \%$$

$$Ss = 26{,}0416\%$$

8. SYSTEM TEST RESULTS

Comparison testing of system compression using Boldi-Vigna ζ2 Algorithm and End-Tagged Dense Code Algorithm; was performed with 10 sample audio files in 8-bit format with the *.wav extension.

8.1. Data Size Comparison After Compression

Data size comparison after compression can be seen in Table 5.

Table 5. Data Size Comparison After Compression

File Name	File Size Before Compression (*Bytes*)	File Size After Compression (*Bytes*)	
		Boldi-Vigna ζ2	End-Tagged Dense Code
sample 1.wav	3.089.060	1.097.609	1.435.791
sample 2.wav	9.273.320	3.209.268	4.252.004
sample 3.wav	3.298.156	2.543.751	2.680.426
sample 4.wav	1.507.678	1.370.806	1.418.026
sample 5.wav	1.129.138	1.054.066	1.086.276
sample 6.wav	1.302.766	1.070.191	1.118.010
sample 7.wav	132.344	125.216	128.517
sample 8.wav	580.538	534.256	556.251
sample 9.wav	52.730	24.063	29,257
sample 10.wav	16.478	15.790	16,353

Based on Table 5. above, the results show that the file size after the compression process using the Boldi-Vigna ζ2 Algorithm is smaller compared to using the End-Tagged Dense Code Algorithm.

8.2. Performance Comparison of Algorithms

Calculation Results of Compression Performance Measurement Parameters can be seen in Table 6.

Table 6. Performance Comparison of Algorithms

File Name	*Boldi-Vigna ζ2*			*End-Tagged Dense Code*		
	Rc	Cr	Ss	Rc	Cr	Ss
sample 1.wav	35,53%	2,81	64,46%	46,47%	2,15	53,52%
sample 2.wav	34,60%	2,88	65,39%	45,85%	2,18	54,14%
sample 3.wav	77,13%	1,29	22,87%	81,27%	1,23	18,72%
sample 4.wav	87,27%	1,14	12,72%	90,28%	1,10	9,71%
sample 5.wav	93,35%	1,07	6,64%	96,20%	1,03	3,79%
sample 6.wav	82,14%	1,21	17,85%	85,81%	1,16	14,18%
sample 7.wav	94,61%	1,05	5,38%	97,10%	1,02	2,89%
sample 8.wav	92,02%	1,08	7,97%	95,81%	1,04	4,18%
sample 9.wav	45,63%	2,19	54,36%	55,48%	1,80	44,51%
sample 10.wav	95,82%	1,04	4,17%	99,24%	1,01	0,75%
Avg	73,81%	1,58	26,18%	79,35%	1,37	20,64%

Based on Table 6. above, the results show that the ratio of compression (Rc) of audio files using the Boldi-Vigna ζ2 algorithm is smaller than using the End-Tagged Dense Code algorithm. This proves that the Boldi-Vigna ζ2 algorithm has a smaller compressed file size than the End-Tagged Dense Code algorithm.

Journal of Theoretical and Applied Information Technology
30th June 2023. Vol.101. No 12

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

The compression ratio (Cr) of audio files using the Boldi-Vigna ζ2 algorithm is greater than using the End-Tagged Dense Code algorithm. This proves that the Boldi-Vigna ζ2 algorithm compresses audio files with *.wav extension better than the End-Tagged Dense Code algorithm.

And the Space Savings (Ss) of audio file compression using the Boldi-Vigna ζ2 algorithm is greater than using the End-Tagged Dense Code algorithm. This proves that the Boldi-Vigna ζ2 algorithm saves storage space better than the End-Tagged Dense Code algorithm.

8.3. Compression Time Comparison

Compression time comparison can be seen in Table 7.

Table 7. Compression Time Comparison

File Name	Running Time (*Seconds*)	
	Boldi-Vigna ζ2	End-Tagged Dense Code
sample 1.wav	1,24	1,54
sample 2.wav	4,53	4,10
sample 3.wav	2,54	1,86
sample 4.wav	0,95	0,96
sample 5.wav	0,70	0,70
sample 6.wav	0,74	0,77
sample 7.wav	0,10	0,11
sample 8.wav	0,42	0,37
sample 9.wav	0,05	0,03
sample 10.wav	0,04	0,03
Average	1,13	1,05

Based on Table 7. above, audio file compression time using the End-Tagged Dense Code algorithm is smaller than using the Boldi-Vigna ζ2 algorithm. This proves that the End-Tagged Dense Code algorithm is faster in performing the compression process than the Boldi-Vigna ζ2 algorithm.

8.4. Comparison Graph of Compression Results

Comparison of the performance of the two algorithms based on the parameters Ratio of Compression (Rc), Compression Ratio (Cr), Space Savings (Ss), and Compression Time, depicted with a graph in Figure 3., Figure 4., Figure 5., and Figure 6.

Figure 3. Ratio of Compression (Rc) Comparison Graph

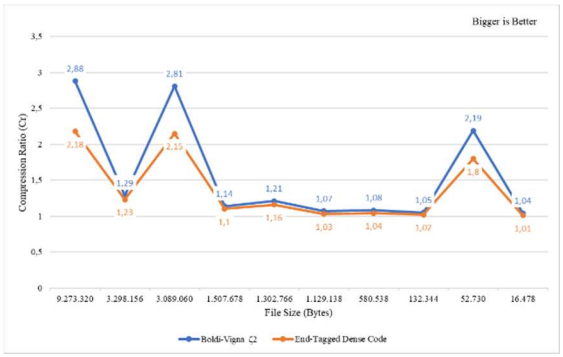

Figure 4. Compression Ratio (Cr) Comparison Graph

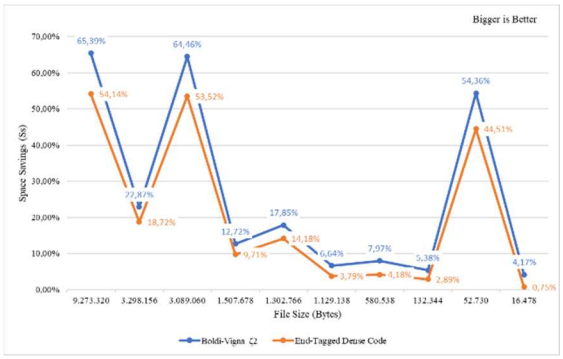

Figure 5. Space Savings (Ss) Comparison Graph

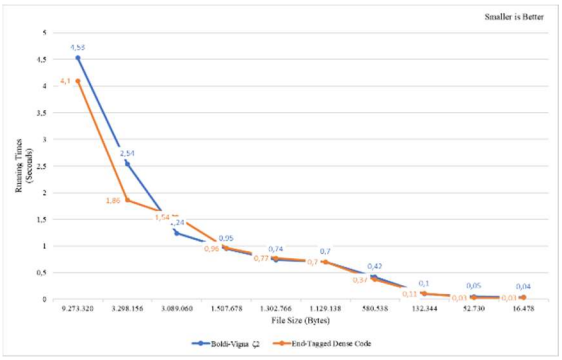

Figure 6. Running Time Comparison Graph

9. CONCLUSION

9.1. Conclusion

Based on testing in this study, it was found that the Boldi-Vigna ζ2 algorithm is better in the audio file compression process with an average Ratio of Compression of 73.81%, an average Compression Ratio of 1.58, and an average Space Savings of 26.18%. While the End-Tagged Dense Code algorithm has a better compression time with an average compression time of 1.05 seconds. It can be concluded that the Boldi-Vigna ζ2 algorithm is better at compressing audio files, while the End-Tagged Dense Code algorithm is faster at performing the compression process. The compression results obtained are influenced by the number of the same digital values contained in the wav audio file that needs to be compressed. Both algorithms can restore the whole audio file like the original audio file through the decompression process.

9.2. Future Research

For future research, it is hoped that the system created can perform compression on audio files in 16-bit format, the system can use audio files with other extensions such as *.mp3, *.flac, *.aac, *.aiff, and so on, then the system can be more effective; so that the compression result file has a smaller size. and the system can be run on various platforms such as desktop, web, and IOS.

REFERENCES

[1] O. Manz, *Well Packed – Not a Bit Too Much: Compression of Digital Data Explained in an Understandable Way*. Germany: Springer Fachmedien Wiesbaden, 2021.

[2] J. Hashim, A. Hameed, M. J. Abbas, M. Awais, H. A. Qazi, and S. Abbas, "LSB Modification based Audio Steganography using Advanced Encryption Standard (AES-256) Technique," *12th Int. Conf. Math. Actuar. Sci. Comput. Sci. Stat. MACS 2018 - Proc.*, pp. 1–6, 2019, doi: 10.1109/MACS.2018.8628458.

[3] H. Karimi, P. Roy, S. Saba-Sadiya, and J. Tang, "Multi-Source Multi-Class Fake News Detection," *Proc. 27th Int. Conf. Comput. Linguist.*, pp. 1546–1557, 2018, [Online]. Available: https://aclanthology.coli.uni-saarland.de/papers/C18-1131/c18-1131

[4] J. Lee and F. Liu, "An Efficient Graph Compressor Based on Adaptive Prefix Encoding," *31st Int. Conf. Sci. Stat. Database Manag. (SSDBM '19)*, 2019.

[5] J. Mancebo, C. Calero, F. García, N. R. Brisaboa, A. Fariña, and Ó. Pedreira, "Saving Energy in Text Search Using Compression," no. c, pp. 1–7, 2019.

[6] K. Sayood, *Introduction to Data Compression. 5th Edition*, 5th ed. Morgan Kaufmann, 2018.

[7] D. Rachmawati, M. A. Budiman, and M. A. Subada, "Comparison study of Fibonacci code algorithm and Even-Rodeh algorithm for data compression," *J. Phys. Conf. Ser.*, vol. 1321, no. 3, 2019, [Online]. Available: IOP Publishing.

[8] T. Grzes, *Lossless and Lossy Audio Codecs for Low-Performance Microcontrollers for Use in IoT*. Berlin, Heidelberg: Springer-Verlag, 2019. doi: 10.1007/978-3-030-28957-7_36.

[9] M. A. Budiman and D. Rachmawati, "On Using Goldbach G0 Codes and Even-Rodeh Codes for Text Compression," *IOP Conf. Ser. Mater. Sci. Eng.*, vol. 180, 2017, [Online]. Available: IOP Publishing.

[10] P. Boldi and S. Vigna, "Codes for the World Wide Web," *Internet Math.*, vol. 2(4), no. 407–429, 2005.

[11] N. R. Brisaboa, A. Fariña, G. Navarro, and J. R. Paramá, "Lightweight natural language text compression," *Inf. Retr. Boston.*, vol. 10(1), pp. 1–33, 2007.

[12] G. Buehrer and K. Chellapilla, "A Scalable Pattern Mining Approach to Web Graph Compression with Communities," *Proc. 2008 Int. Conf. Web Search Data Min. (WSDM '08)*, pp. 95–106, 2008.

[13] F. Claude and G. Navarro, "Fast and

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Compact Web Graph Representations," *ACM Trans. Web*, vol. 4, no. 4, 2010.

[14] T. Valencia, L. O. Cerdeira, E. L. Iglesias, and F. J. Rodríguez, "Translation Table Compression under End-Tagged Dense Code," pp. 0–5, 2010.

[15] N. Lin, K. V. A, and Y. N. Soe, "Text Compression for Myanmar Information Retrieval," *Proc. 2019 3rd Int. Conf. Nat. Lang. Process. Inf. Retr.*, pp. 62–67, 2019.