Elinvo (Electronics, Informatics, and Vocational Education)
Vol 10, No. 1, 2025, pp. 63-77
ISSN 2580-6424 (printed) | ISSN 2477-2399 (online)

Optimizing 4-ary Huffman Trees and Normalizing Binary Code Structures to Minimize Redundancy and Level Reduction

Tonny Hidayat [1*], Hendra Kurniawan [2], Ali Mustopa [3], Jeki Kuswanto [4]

[1,2,3,4]Universitas Amikom Yogyakarta, Jl. Ring Road Utara, Condongcatur, Depok, Sleman, Daerah Istimewa Yogyakarta, Indonesia
[1] tonny@amikom.ac.id*; [2] ali.m@amikom.ac.id; [3] jeki@amikom.ac.id; [4] hendrakurniawan@amikom.ac.id

Article Info

Article history:
Received October 31, 2024
Revised February 24, 2025
Accepted March 4, 2025
Available Online July 26, 2025

Keywords:
Lossless; Compression; Huffman; Binary; Audio

Abstract

Since the present data expansion and increase is occurring at an increasingly rapid pace, the solution of adding storage space is not sustainable in the long run. The growing need for storage media can be addressed with lossless compression, which reduces stored data while allowing full restoration. Huffman remains a potent method for data compression, functioning as a "back end" process and serving as the foundational algorithm in applications among others Monkey' PKZIP, WinZip, 7-Zip, and Monkey's Audio. Lossless compression of 16-bit audio requires binary structure adjustments to balance speed and optimal compression ratio. The use of a 4-ary Huffman tree (4-ary) branching procedure to generate binary code generation and to insert a maximum of 2 dummy data symbol variables that are given a binary value of 0 with the condition that if the number of MOD 3 data variables = remaining 2, then 2 dummy data are added, if the result is the remainder 0 = 1 dummy data, and if the remainder = 1 then it is not required. This process effectively maintains a high ratio level while speeding up the 4-ary Huffman code algorithm's performance in compression time. The results show that the efficiency reaches 95.94%, the ratio is 38%, and the comparison is 1/3 of the Level based on calculations, testing, and comparison with other generations of the Huffman code. The 4-ary algorithm significantly optimizes archived data storage, reducing redundancy to 0.124 and achieving an entropy value of 2.91 across various data types.

Corresponding Author:
Email: tonny@amikom.ac.id

INTRODUCTION

The demand for storage media is expanding because, in today's digital age, everyone may quickly create, process, and reproduce their personal documentation in audio, text, and video formats [1]. As digital tools and technologies become more accessible, people are generating and sharing more data than ever before, it may cause the increase of digital content volume. This surge in data creation necessitates larger and more efficient storage solutions to accommodate the vast amounts of information being produced and stored by individuals and organizations alike.

One of the processes involved in digital file formatting is compression, which can be lossless or lossy [2]. The nature and requirements of lossless compression require that compressed files can be returned to their original condition with no damage or flaws . If data is not needed for an extended amount of time, it is normally archived (stored) [3]. In such circumstances, the solution is to use lossless

compression for archiving purposes, with the performance indication being the compression ratio and duration [4].

Audio compression is essential due to the large size of high-quality audio files (e.g., 16-bit or 24-bit), which consume significant storage space. Compression reduces file sizes, enabling efficient storage and transmission while preserving audio quality, crucial for music production, archival, and broadcasting [5]. Industries relying on audio compression include music and entertainment (e.g., recording studios, Spotify, Apple Music), broadcasting (radio, TV), and telecommunications (VoIP, video conferencing). By using audio compression, these sectors balance storage efficiency, cost savings, and audio quality, ensuring digital audio remains accessible and practical across applications.

One of the alghoritm that can be used is Huffman. This alghoritm is to compact without any loss since it can be utilized as a "back-end" operation with other techniques [6]. As of yet, no specific study has been conducted on the use of the Huffman technique to compact 16-bit of the audio data of WAV that strikes a compromise between compacting ratio and compacting time. The study provides a chance to investigate novel techniques to improving compression efficiency and speed for high-quality audio formats.

Almost all lossless compression software on the market, including WinZip, WinRar, and 7Zip, are based on the Huffman algorithm [7]. The goal now is for academics to build a binary data structure for a file that is both efficient in size and stable when altered. Achieving this would require innovative approaches to optimize the Huffman algorithm, such as exploring multi-level tree structures or hybrid compression techniques. Audio files in wav format are RAW files that are typically stored on a 16-bit file system with data symbol variations ranging from 0 to 65,536 [8]. This condition prevents each component of the audio file from being patterned in general, as well as compression from achieving its maximum ratio.

This study was to build the structure for a 16-bit data set using the Huffman tree algorithm to find the association between Compression Ratio (CR) and Compression Time (CT), that is required to improve the efficiency of the Huffman algorithm for data compression. This study investigates the intersection between the efficiency, length average, and also the variances and also the last result that will be compared with CR and CT.

RELATED WORK

An unique compression approach was introduced by Banetley et al. (1986) with the approach was structure that was identical to the process of the Huffman algorithm but has certain practical benefits; it compresses the data in a single traverse [9]. Huffman-based apporaches was also described by Amarsinghe and Kodituwakku (2010) and Sharma (2010), this introduction is to generate an optimal compact code [10]. The speed of this decoding is slow. Based on the ACW algorithm, Bahadili and Hussain (2010) established an alternative bit-level adaptive data compression approach that exceeded several widely employed compression methods in terms of compression ratio. [11].

Hermassi et al. (2010) proved that an image can be encoded using multiple codewords of identical length [12]. An alternative of novel decoding technique was described by Chowdhury et al. (2002) for pseudo-static Huffman codes, exhibiting a highly fast representation of the Huffman header [13]. Suri and Goel (2011) investigated the utilization of ternary trees and developed a novel one-pass technique to describe adaptive Huffman codes [14]. Huffman (1952) suggested an algorithm as an encoder for reducing data and stated that no two signs would include the same sequence of codes and the code that results from the reconstitution of the data bits, and that no further settings need to be made to determine where the code starts and concludes once the starting point is known.[15]. Beginning with the publication of the Huffman coding technique and the achievement of a minimum redundancy value, this algorithm became quite popular for compressing text data before moving on to other forms of data, particularly images [16], [17]. Schack (1994) explains the length code in his study; the lengthy codeword for the Huffman and Shannon-Fano algorithms has the same interpretation [18]. Katona and Nemetz (1976) investigated the relationship between the self-information of the source symbol and the length of its codeword [18]

In another work, Hashemian (1995) proposed a new data compression strategy based on the clustering notion. The algorithm states that this new approach will use mini-mother storage while searching for symbols at a fast speed [19]. He found that that the clustering method approach is quite effective

in compressing video data in his experiments. . An array-based structure for data for Huffman networks involving a memory demand of 3n minus 2 was proposed by Chung (1997). This structure was decoded using a fast algorithm, which is accomplished by reducing the memory from 3n - 2 to 2n - 3, where n is the number of symbols. Chen et al. (1999) created a rapid alghoritm decoding with O (log n) time and ⌈3n/2⌉ + ⌈(N/2) log n⌉ + 1 memory space [17].

Fenwick (1995) stated that there was sometime inefficient during utilizing Huffman code [20]. It showed there was a fault during the chaning between the low with the higher extensions. Szpankowski (2011) and Baer (2006) state the minium size of the predicted length of fixed-to-variable lossless compression under prefix-out constraints [21], [22]. Kavousianos (2008) creates a variable-to-variable code by using the Huffman principle, sometimes referred to as fixed-to-variable codes. [23]. Vitter (1987) created an innovative method for online compression in networks in his paper [24]. Huffman codes were first used in database compression by Habib et al. (2013). [25]. Gallager (1978). He explained four properties of the Huffman code, including the symbol frequency property, the codeword length property, the sibling property, and the upper bound property. [26], Additionally, he demonstrated the flexible Huffman coding method. Welch (1984) and Lampel and Ziv (1977) created coding techniques for every type of source symbol. [27], [28]. In order to achieve efficient Huffman decoding, Lin et al. (2012) first converted the basic Huffman tree to a recursive Huffman tree. They then used the recursive Huff algorithm to decode several symbols simultaneously. [29]. It makes faster decoding possible. Zopfli, a compression tool that uses the Binary Huffman algorithm technique, was recently published by Google Inc. One of Google Inc.'s best compression algorithms, Zopfli, was released. According to Google, Zopfli's compression ratio is the best.
. [30].

Communication systems, statistics, and probability theory all make substantial use of information theory. To address this issue, lossless coding techniques are employed, such as the Huffman stationary binary algorithm.(Huffman, 1952), the Shannon method (Shannon & Weaver, 1949), arithmetic codes (Sayood, 2000), the Fano method (Hankerson, Harris, & Johnson, 1998), and the updated Fano-based technique for coding (Rueda & Oomen, 2004). These tactics' adaptive variations have been put forth and are explained in (Faller, 1973; Gallager, 1978; Ankerson et al., 1998; Knuth, 1985; Rueda, 2002; Sayood, 2018; Alakuijala et al., 2019).

Communication systems, statistics, and probability theory all make extensive use of information theory. The Huffman's static binary technique (Huffman, 1952), the Shannon approach (Shannon & Weaver, 1949), arithmetic codes (Sayood, 2000), the Fano method (Hankerson, Harris, & Johnson, 1998), as well as the improved Fano-based coding algorithm (Rueda & Oomen, 2004) are some of the lossless coding techniques used to solve this issue. (Faller, 1973; Gallager, 1978; Ankerson et al., 1998; Knuth, 1985; Rueda, 2002; Sayood, 2018; Alakuijala et al., 2019) have created and documented adaptive variants of these strategies. Developing an innovative structure for the length of a binary data bit utilizing the Huffman tree algorithm scheme on 16-bit data to strike a compromise between Compression Ratio and Compression Time in WAV format music files. . This study fills a vacuum in the literature by comparing the compression ratios and times of Huffman algorithm variations.

METHODS

The research employs an experimental approach to optimize the stability between file size reduction and compression speed in the processing of the 16-bit WAV audio data saved on the storage device.. The methodology involved collecting 16-bit WAV audio data to develop and test multiple Huffman-based code generation strategies. According to Ali et al. (2011), the mean of size is increasing at a pace of approximately 30% annually [31]. To address this challenge, this study focuses on designing and evaluating Huffman algorithm variations aimed at improving both the CR and CT, thereby enhancing storage efficiency and broader IT applications [32]. Several Huffman-based techniques will be implemented to construct binary trees that allocate data bits dynamically in response to increasing file size trends. The experimental design considers both the Compression Ratio and Compression Time for 16-bit data, which consists of 65,536 unique symbol variants, in contrast to 8-bit data, which has only 256 symbol variants [33].

ISSN 2580-6424 (printed) | ISSN 2477-2399 (online)

The expanded Huffman technique may be applicable to data formats other than audio data, which is the focus of this study. Conceptually, the technique can be used to text files, photos, and videos. like a reminder, like with music, the volume of data that must be kept for image and video data presents storage medium and infrastructural difficulties. Video and image data have already been compressed., but improvements such as enhanced Huffman algorithms that can increase Compression Ratio and Time Compression will provide an IT solution Addressing the challenge of preserving authenticity while expanding information retention on storage medium made especially for data archiving

.

To ensure that an algorithm works optimally, a mathematical calculation can be performed to ensure its performance before entering the programming and assembly stages. Entropy coding includes the Huffman algorithm, which allows for a calculation to be performed in phases using the following parameters. The mathematical calculation steps used to analyze the Huffman compression process are shown in Fig. 1.

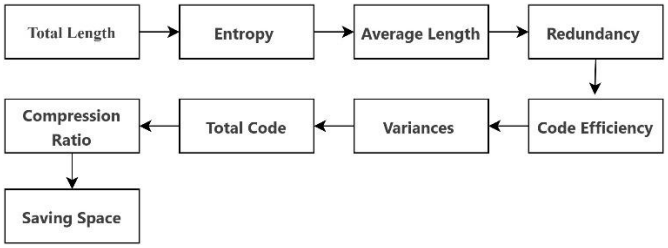

Figure 1. Step by step for a calculation

a. Total Length (TL)

The variables that must be set are the numerous symbol variants, their frequencies, and probabilities, to calculate the Total Length (number of bits) as the size of the uncompressed data (source).

$$TL = \sum_{i=1}^{n}(fi \times li) \quad \text{..... (1)}$$

where:
fi = frequency of symbol *i*
li = Huffman length code assigned to symbol *i*
n = total number of unique symbols in the datase

b. Entropy (H)

The entropy calculation method can be used to determine the data encoding value in terms of bits per symbol (bps), as the closer it is to bps, the more efficient the compression algorithm.

$$H = -\sum_{i=1}^{m}(P_i \times log_2(P_i)) \quad \text{..... (2)}$$

where:
Pi = i-symbol probability
$log_2(Pi)$ = logarithm base 2 of the probability of symbol

c. Average Length (L_{av})

The length of a message must be equalized and averaged in order to normalize the length of each data symbol character.

$$L_{av} = -\sum_{i=1}^{n} P_i \times n_i \quad \text{..... (3)}$$

where:
n_i = length of the Huffman code assigned to symbol *i*

d. Redundancy (R)

The difference of length between L_{av} and H will produce a value where the smaller the result, the more optimal the performance of the algorithm.

$$R = L_{av} - H(X) \quad \text{..... (4)}$$

e. Code Efficiency (CE)

To ensure the advanced level of the optimization process, an efficiency calculation process can be carried out on the influence of the length of the binary code variable.

$$CE = H(X)/L_{av} \times 100\% \quad (5)$$

f. Variances (V)

The variance means the wide of number set and variance is used to measure the average length. The value of zero means identical for all values. There is no negative variance and low variance means that the data is near the mean and others. The large variance indicates the far of the mean from other numbers.

$$V = \sigma^2 = E(\alpha_i - A)^2 = \frac{1}{n}\sum_{i=1}^{n}(\alpha_i - A)^2 \quad (6)$$

g. Total Code (TC)

Following the compression process, it is required to determine how much total code is created by looking at the frequency value of each symbol as well as the code length.

$$TC = \sum_{i=1}^{n} tl' \times f \quad (7)$$

h. Compression Ratio (CR)

In the final process, the most common main performance indicator is calculating the comparison ratio before and after compression.

$$CR = \frac{TC}{TL} \times 100\% \quad (8)$$

i. Saving Space (SS)

Another calculation to facilitate descriptive assessment is how much savings are generated by calculating saving space

$$SS = \frac{TL-TC}{TL} \times 100\% \quad (9)$$

PROPOSED HUFFMAN WITH 4-ARY

In binary, there are several levels of branches that can be used, known as code generation. Branching 4, also known as 4-ary in this paper, will be ensured to be optimal for processing 16-bit data in order to achieve a balanced result (high fixed ratio with fast compression process) between CR and CT. Based on related research and consideration of mathematical calculations, it needs to be proven by the following simulation process.

As an initial calculation, it can be done by simulation using sample data in Table 1. with a 16-bit file system.

Table 1. Sample Data

Symbol	Freq	Prob
Sy1	35	0.35
Sy2	21	0.21
Sy3	15	0.15
Sy4	8	0.08
Sy5	5	0.05
Sy6	3	0.03
Sy7	2	0.02
Sy8	1	0.01
Sy9	1	0.01
Sy10	1	0.01
Sy11	1	0.01
Sy12	1	0.01

Sy13	1	0.01
Sy14	1	0.01
Sy15	1	0.01
Sy16	1	0.01
Sy17	1	0.01
Sy18	1	0.01
Total	100	1

TL = 35x8 + 21x8 + 15x8 + 8x8 + 5x8 + 3x8 + 2x8 + 1x8 + 1x8 + 1x8 + 1x8 + 1x8 + 1x8 + 1x8 + 1x8 + 1x8 + 1x8
= 800 bit

H = - (0.35log20.35 + 0.21log20.21 + 0.15log20.15 + 0.08log20.08 + 0.05log20.05 + 0.03log20.03 + 0.02log20.02 + 0.01log20.01 + 0.01log20.01 + 0.01log20.01 + 0.01log20.01 + 0.01log20.01 + 0.01log20.01 + 0.01log20.01 + 0.01log20.01 + 0.01log20.01 + 0.01log20.01)
= 2.916541606 bits/symbol

To optimize the organization of the sample data table in Table 1. The frequency is then normalized by situating it with the value that has been added to the top position against the same value; this procedure is repeated, as shown in Fig. 2.

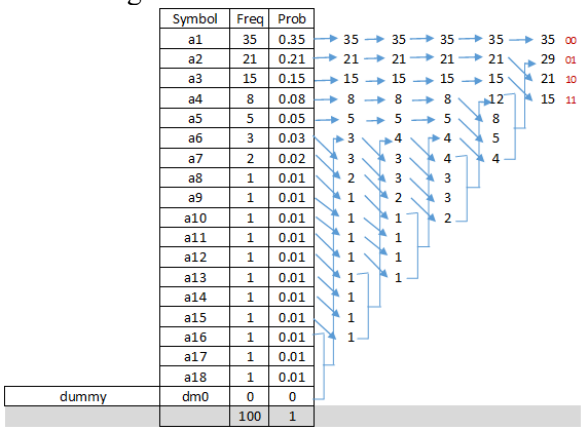

Figure 2. 6 4-ary Normalization

The above-mentioned reconstruction findings considerably improve the speed and structure of the new binary structure in Table 2. The lowest code is two codes long and represents the symbol with the highest frequency or likelihood. The largest code has a length of 8 codes and represents the symbol with the lowest frequency or likelihood. The longest code consists of eight codes and reflects the sign with the smallest of chance.

Table 2. List of Symbols and Code for Encoding

Symbol	Code	length	P(i)L(i)	Variances
Sy1	00	2	0.7	0.132496
Sy2	10	2	0.42	0.047699
Sy3	11	2	0.3	0.024336
Sy4	0101	4	0.32	0.005898
Sy5	0110	4	0.2	0.002304
Sy6	010010	6	0.18	0.007885
Sy7	010011	6	0.12	0.003505
Sy8	011100	6	0.06	0.000876
Sy9	011101	6	0.06	0.000876
Sy10	011110	6	0.06	0.000876
Sy11	011111	6	0.06	0.000876
Sy12	01000000	8	0.08	0.002460
Sy13	01000001	8	0.08	0.002460
Sy14	01000010	8	0.08	0.002460
Sy15	01000011	8	0.08	0.002460

Sy16	01000100	8	0.08	0.002460
Sy17	01000101	8	0.08	0.002460
Sy18	01000110	8	0.08	0.002460
dm0	01000111	8	0	0.000000
	Average Length L =		3.04	0.244849

The following steps of the process's calculation, after obtaining the aforementioned data, are:

L_{av} = 0.35 x 2 + 0.21 x 2 + 0.15 x 2 + 0.08 x 4 + 0.05 x 4 + 0.03 x 6 + 0.02 x 6 + 0.01 x 6 + 0.01 x 6 + 0.01 x 6 + 0.01 x 6 + 0.01 x 8 + 0.01 x 8 + 0.01 x 8 + 0.01 x 8 + 0.01 x 8 + 0.01 x 8 + 0.01 x 8 + 0 x 8
= 3.04 bits/symbol

R = 3.04 - 2.916541606
= 0.123458 bits/symbol

CE = (2.916541606 / 3.04) x 100%
= 95.93887 %

V = 0.35(2 – 3.46)2 + 0.21(2 – 3.46)2 + 0.15(2 – 3.46)2 + 0.08(4 – 3.46)2 + 0.05(4 – 3.46)2 + 0.03(6 – 3.46)2 + 0.02(6 – 3.46)2 + 0.01(6 – 3.46)2 + 0.01(6 – 3.46)2 + 0.01(6 – 3.46)2 + 0.01(6 – 3.46)2 + 0.01(8 – 3.46)2 + 0.01(8 – 3.46)2 + 0.01(8 – 3.46)2 + 0.01(8 – 3.46)2 + 0.01(8 – 3.46)2 + 0.01(8 – 3.46)2 + 0.01(8 – 3.46)2 + 0(8 - 3.46)2
= 0.244849

TC = 35x2 + 21x2 + 15x2 + 8x4 + 5x4 + 3x6 + 2x6 + 1x6 + 1x6 + 1x6 + 1x6 + 1x8 + 1x8 + 1x8 + 1x8 + 1x8 + 1x8 + 1x8 + 0x8
= 304 bit

CR = (*TC* / *TL*) x 100%
=(304 / 800) x 100 %
= 38 %

SS = ((*TL* – *TC*) / *TL*) x 100%
= ((800 – 304) / 800) x 100%
= 62%

Table 3 displays the results of a 4-ary tree with normalization. The findings reveal a Compression Ratio of 38%, which is nearly identical to employing a binary tree (37.25%), with a storage savings of 62%. The efficiency level is 95.93887%, which is similar to a binary tree with a gap of only 2.5%. As a result, this 4-ary should be improved and tested with a variety of compressed data types. The 4-ary is likewise updated using a reversed layout strategy.

Table 3. Compression Results with 4-ary / Modified Quad Tree with Tree Normalization

Information	Result
Length (TL) ASCII	800
Entropy (H)	2.916542
Average Length (Lav)	3.04
Total Code (TC)	304
Compression Ratio (CR)	38%
Saving Space (SS)	62%
Redundancy (R)	0.123458
Variances (V)	0.244849
Efficiency (CE)	95.93887
Level	5

RESULT AND DISCUSSION

The proposed design produces findings that are consistent with the objectives of this research, hence the following guidelines can be made:

1. Data on the number of variables and their corresponding frequency/probability values are then gathered from the input stream (16-bit WAV audio).

2. To find the quantity of fake data, measure the number of "Data variables" in the incoming data stream.

3. The data of "Data variables" is divisible by three (MOD 3), calculate quantity of fake data using the following guidelines.

   • For a mod result of two, there are two bogus data variables.
   • Fake data is counted as one when the mod result is zero.
   • Fake data is unnecessary if the mod result is 1. "ariables" is divisible by three (MOD 3).

4. To establish four fixed branches (after the root), add a dummy variable "dm" with a frequency of 0 and probability of 0.

5. Add the total data variables and the variety of simulated variables (dm).

6. Organize data and fictional variables by frequency or probability, from greatest to smallest.

7. To form a coding tree, the newly established node containing the exact same value will be placed above the previous node with the same number if the sum of the preceding nodes yields the equivalent frequency/probability ratio.

8. Code assignment:
   - Left = 00
   - Left-Mid = 01
   - Right-Mid = 01
   - Right = 11

A large collection of 1,500 samples split into four categories—instruments, voices of humans, unplanned noise, and music—is used to evaluate the success of the rules developed. The following sample size formula is used to determine the sample size needed for findings that are statistically significant: $(\text{Z-score})^2 - \text{StdDev} \times (1-\text{StdDev})^3$ (Confidence Level)^2 = Required Sample SizeSize was calculated using a 90% confidence level (Z-Score 1.645), a 0.5 standard deviation, and an acceptable range of +/- 5% (0.05).

The range of confidence is the range of error plus or minus 5% in the given results. The Z-Score (Confidence Level) expresses how sure the findings will be within the intended margin of error. When a more exact standard deviation is unknown, the number 0.5 is used. This guarantees that the sample is large enough. Using this technique, it became clear that a sample size of 1,500 was necessary to get statistically significant results. Due to the huge amount of audio material available for testing, the smallest number of samples (500 each category) were chosen and examined in this study, totaling roughly 9,000.

The dataset sources differ in order to improve the diversity of audio kinds to be compressed; here is a description of the dataset sources used:

1. MUSAN: A Music, Speech, and Noise Corpus (David Snyder and Guoguo Chen and Daniel Povey) 2015 [34]. Link : http://www.openslr.org/17/
2. J. Bosch, R Marxer, and E. Gomez. Evaluation and combination of pitch estimation methods for melody extraction in symphonic classical music. Journal of New Music Research 2016 [35]. Link : https://zenodo.org/record/1289786#.XUubA-gzZqP
3. IRMAS-Bosch, J. J., Janer, J., Fuhrmann, F., & Herrera, P. "A Comparison of Sound Segregation Techniques for Predominant Instrument Recognition in Musical Audio Signals", in Proc. ISMIR (pp. 559-564), 2012 [36]. Link : https://zenodo.org/record/1290750#.XUujEegzZqM

4. FSDnoisy18k - Eduardo Fonseca, Manoj Plakal, Daniel P. W. Ellis, Frederic Font, Xavier Favory, and Xavier Serra, "Learning Sound Event Classifiers from Web Audio with Noisy Labels", arXiv preprint [37]. arXiv:1901.01189, 2019. Link : https://zenodo.org/record/2529934#.XUulsegzZqN
5. Music Track With CD quality- Robin Whittle. Lossless audio compression, 2000-2005 [16]. Link : http://www.firstpr.com.au/audiocomp/lossless/#rice

After analyzing it on the dataset, the audio kinds are classified into eight groups: music, mono music mono, art stereo, rips CD, speech, noise, and sounding instruments. To compare different variations of the Huffman technique, many categories are used.

The mean number obtained from the equation that comes next (15) is used for contrasting CR on various variants within the Huffman Tree scheme and the 4-ary approach, which is a development of the Huffman technique on every audio type, as shown in Table 4.

$$\overline{CR}_{i,j} = \frac{\sum X_{i,j}}{N_{i,j}} \quad \text{..... (10)}$$

Note:
$\overline{CR}$: The average CR value of i on j
i : Audio types
j : Huffman variance components
$\sum X_{i,j}$: The total of the component members of Huffman i in j
$N_{i,j}$: The number of component members i in j

It can be calculated such as below :.

Table 4. CR Huffman Tree Scheme Variants

Tipe Audio	*Average (all)*	*CR(%)*					
		Static	**Dynamic**	**Quad**	**Okta**	**Hexa**	**4-ary**
Music	54,44	49,92	50,74	54,45	57,04	60,44	54,03
Music mono	46,19	42,25	42,33	46,01	49,02	52,01	45,49
Music stereo	48,12	44,07	44,29	47,87	51,09	53,99	47,43
Ripping CD	55,32	45,59	45,63	58,06	60,61	64,13	57,86
Speech	48,23	41,38	41,77	49,45	52,52	55,46	48,85
Noise	46,36	41,74	42,42	46,49	49,67	52,23	45,59
Sound effects	37,84	32,84	33,49	38,06	41,64	44,30	36,72
Instruments	51,11	47,24	47,78	50,77	53,95	56,60	50,35

The difference between the final CR value and the average CR value, which is a function of the descriptive analysis results, indicates that the CR value sequence pattern works well for a variety of file types. Please refer to the following Table 5 for further information:

Table 5. Difference Between CR 4-ary and Average

Audio type	CR 4-ary (%)	Average CR (%)	Deviation
Music	54,03	54,44	-0,41
Music Mono	45,49	46,19	-0,70
Music Stereo	47,43	48,12	-0,69
Ripping CD	57,86	55,32	+2,55
Speech	48,85	48,23	+0,61
Noise	45,59	46,36	-0,87
Sound Effect	36,72	37,84	-1,12
Instruments	50,35	51,11	-0,76

The desired result is a decreased CR difference from the average CR value. According to the table above, there are six (six) types of audio files with a negative difference (-) and two (two) types of files with a positive difference value (+). The positive sign (-) indicates that the 4-ary application produces a lesser CR score than the median for the 6 (six) kinds of audio files. As a consequence, when compared, it is evident that the 4-ary software is quite effective at file compression.

To get an overall impression of the data as a whole based on the Huffman variance components, use the following Eq. (16) to calculate the average:

$$\overline{CR_j} = \frac{\sum X_j}{N_j} \quad \text{..... (11)}$$

Note:
$\overline{CR}$: The average CR value of j
j : Huffman variance components
$\sum X_j$: Total of values of the members j
N_j : Number of members j

In light of the overall CR value of the audio data compression findings, the 4-ary application (48.29%) is less efficient than the Static (43.13%) and Dynamic (43.56%) variations. For further information, check Table 6.

Table 6. Average CR Huffman Tree Scheme Variant

Huffman Scheme	Average CR (%)	Overall Average (%)	Note
Static	43,13	48,45	Below average
Dynamic	43,56		Below average
Quad	48,90		Above average
Okta	51,94		Above average
Hexa	54,89		Above average
4-ary	48,29		Below average

The remaining space following compression is referred to as "space saving." As a result, the compressed file (CR) shrinks while the leftover space (SSc) grows. Using the following equation, a correlation study is performed to ascertain the link between CR and SSc.

$$r = \frac{N\sum XY - \sum X \sum Y}{\sqrt{(N\sum X^2 - (\sum X)^2)(N\sum Y^2 - (\sum Y)^2)}} \quad \text{..... (12)}$$

Note:
r : Correlation
N : The number of values
∑X : The sum of the X values
∑Y : The sum of the Y values
∑X^2 : The squared sum of each X value
∑Y^2 : The squared sum of each Y value
∑XY : The sum of each X value multiplied by each Y value
X : CR
Y : SSc

The results of the correlation analysis show a correlation value of -0.447 and a significant value of 0.000. Based on these values, it is known that CR and SSc are correlated quite well (between 0.400 - 0.600) and inversely proportional where the higher the CR value, the more significant the effect on the SSc value with the direction of change being smaller, and vice versa.

On the basis of the average value determined by the following formula, SSc on several variations of Huffman Tree designs and 4-ary procedures on each sound type is compared.

$$\overline{SSc}_{i,j} = \frac{\sum X_{i,j}}{N_{i,j}} \quad (13)$$

Note :

$\overline{SSc}_{i,j}$: The average SSc value of i on j

i : Types of audio (music, stereo, mono, ripping CDs, speech, noise, sound effects, and instruments)

j : Components of the Huffman variance (Static, Dynamic, Quad, Okta, Hexa, 4-ary)

$\sum X_{i,j}$: The total value of the component members i in j

$N_{i,j}$: The number of members of component i in j

Table 7 showed the average SSc values derived from audio data compression results utilizing the Huffman Tree scheme version and the 4-ary technique.

Table 7. SSc Huffman Tree Scheme Variants

Tipe Audio	Average (all)	SSc (%)					
		Static	Dynamic	Quad	Okta	Heksa	4-ary
Music	45,56	50,08	49,26	45,55	42,96	39,56	45,97
Music mono	53,82	57,75	57,67	53,99	50,98	47,99	54,51
Music stereo	51,88	55,93	55,71	52,13	48,91	46,01	52,57
Ripping CD	44,68	54,41	54,37	41,94	39,39	35,87	42,14
Speech	51,77	58,62	58,23	50,55	47,48	44,54	51,15
Noise	53,64	58,26	57,58	53,51	50,33	47,77	54,41
Sound Effect	62,16	67,16	66,51	61,94	58,36	55,70	63,28
Instruments	48,89	52,76	52,22	49,23	46,05	43,40	49,65

According to the comprehensive comparison, the SSc rating of the decompression results from the 4-Response application is higher than the Quad, Octa, and Hexa versions, but it is still lower than the capabilities of the Stability and Dynamic variants.

Table 8. Difference between 4-ary and average SSc

Audio Type	SSc 4-ary (%)	Average SSc (%)	Deviation
Music	45,56	45,97	+0,41
Music Mono	53,82	54,51	+0,69
Music Stereo	51,88	52,57	+0,69
Ripping CD	44,68	42,14	-2,55
Speech	51,77	51,15	-0,61
Noise	53,46	54,41	+0,77
Sound Effect	62,16	63,28	+1,12
Instruments	48,89	49,65	+0,76

The difference between the 4-ary SSc and the average SSc should be positive, as anticipated. There is a good indicator for the difference in six file formats. For a given audio type, the 4-ary generates a bigger space than the average SSc value, as indicated by the positive sign (+).SSc contrast between Huffman Tree Scheme variants based on the overall average value, using the following Equation (19):

$$\overline{SSc}_j = \frac{\sum X_j}{N_j} \quad (14)$$

Note:

$\overline{SSc}$: The average SSc value of j

j : Huffman variance components (Static, Dynamic, Quad, Okta, Hexa, 4-ary)

$\sum X_j$: The sum of the values of the members j

N_j : The number of members j

Table 9. Average SSc Calculation Results Based on Huffman Tree Scheme Variants

Huffman Scheme	Average SSc (%)	Overall Average (%)	Note
Static	56,87	51,55	Above average
Dynamic	56,44		Above average
Quad	51,10		Below average
Okta	48,06		Below average
Hexa	45,11		Below average
4-ary	51,71		Above average

Table 9 shows that the average value for the 4-ary variation is higher than the overall average value. However, the average SSc value of 4-ary remains lower than Static and Dynamic, implying that 4-ary's ability to provide remaining compression space is on par with Static and Dynamic schemes.

The amount of time needed for compression by the deflate application is known as Total Time Compression (TTC). An program is considered superior if it can compress files more quickly. TTC data processing is accomplished by figuring up how long it takes the program to compress each MB of the file (MBps). The following is the equation:

$$TTC_i = \frac{UF}{t} \quad (15)$$

Note :
TTC_i : Time to compress the file i
i : Audio types
UF : Size of file (megabyte – MB)
t : Time used to perform compression (second - s)

It is evident from Table 10's graph that, depending on the kind of audio, some component versions offer TCT benefits. The graph shows that the 4-ary application has TCT advantages on audio files of the Music mono type (18.37 MBps), Music Stereo (7.26 MBps), Noise (4.39 MBps), and Sound Effects (6.45 MBps), the Quad variant has speed advantages on Speech (6.09 MBps) and Instrument (6.20 MBps) type files, the Okta variant excels on Music (6.19 MBps) type files, and the Hexa variant excels on RippingCD (5.21 MBps) type files.

Table 10. TCT Calculation Results of Huffman Tree Scheme Variants

Tipe Audio	Average (all)	TTC (mbps)					
		Static	Dynamic	Quad	Okta	Heksa	4-ary
Music	5,99	5,76	0,08	6,04	6,19	5,93	6,05
Music mono	10,25	11,19	0,14	7,38	7,20	7,13	18,37
Music stereo	7,08	6,92	0,11	7,22	6,76	7,22	7,26
Ripping CD	5,00	5,10	0,11	4,81	5,09	5,21	4,76
Speech	5,80	5,71	0,14	6,09	5,56	5,78	5,84
Noise	4,34	4,11	0,10	4,50	4,33	4,39	4,39
Sound Effect	6,28	6,09	0,16	6,32	6,31	6,22	6,45
Instruments	5,56	4,57	0,09	6,20	5,93	5,94	5,18

Using the following formula, TTC is compared between Huffman Tree Scheme :

$$\overline{TTC_j} = \frac{\sum X_j}{N_j} \quad (16)$$

Note :
$\overline{TTC}$: The average TTC value of j
j : Huffman variance components

$\sum X_j$: Sum of the values of the members j
N_j : Number of members j

The calculation results can be seen in the following Table 11.

Table 11. Average TCT Calculation Results Based on The Huffman Tree Scheme Variant

Scheme	Average TTC (MBps)	Overall Average (MBps)	Note
Static	6,18	6,29	Below average
Dynamic	<0,2		
Quad	6,07		
Okta	5,92		
Hexa	5,98		
4-ary	7,29		Above average

The 4-ary application's TTC value (7.29 MBps) is higher than typical, allowing it to compress files faster than other Huffman variations. As a result, when compared to other Huffman Tree scheme versions, 4-ary is the most efficient compression strategy for deflating data in terms of time.

CONCLUSION

The primary objective of this research is to optimize the compression of 16-bit audio data in WAV format for archiving purposes, with a focus on two critical parameters: compression ratio and compression time. In this study, we introduce many types of generation code that can be used to process data bits in the future. The main design of the method we present is called 4-arry / Quad Modif by adding normalization or a mechanism, the first of which is dynamic Huffman, which creates a procedure in which if there is a value with the same frequency / probability, the new value is positioned at the top among the same values. The second adds a method from the Huffman Adaptive extension, namely the use of additional data symbols whose frequency / probability value is 0, known as NYT (Not Yet Transmitted); the difference with the architecture we suggest is that we utilize two types of 0 files named 'dummy'. According to the findings of the research, the 4-ary successfully balances compression ratio and compression time, which is extremely useful if this technique is used to archive files in a lossless way. Based on the ratio, the results are similar to those of the binary tree, but with a large increase in processing speed (compression) and space requirements. Aside from the ratio, the main aspect to consider while archiving files is compression speed. Thus, the suggested 4-ary algorithm compresses 16-bit data by balancing compression ratio and compression time.

ACKNOWLEDGMENT (11pt)

This study was supported by Kemendikbud Ristek Dikti Republic of Indonesia No. 107/E5/PG.02.00.PL/2024, LLDIKTI V No. 0609.20/LL5-INT/AL.04/2024 and Universitas Amikom Yogyakarta No. 037/KONTRAK-LPPM/AMIKOM/VI/2024.

REFERENCES

[1] D. Groppi, A. Pfeifer, D. A. Garcia, G. Krajačić, and N. Duić, "A review on energy storage and demand side management solutions in smart energy islands," *Renew. Sustain. Energy Rev.*, vol. 135, p. 110183, Jan. 2021.

[2] F. Mentzer and M. Tschannen, "Learning better lossless compression using lossy compression," *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, pp. 6638--6647, 2020.

[3] T. Hidayat, M. H. Zakaria, and A. N. C. Pee, "Survey of Performance Measurement Indicators for Lossless Compression Technique based on the Objectives," in *2020 3rd International Conference on Information and Communications Technology (ICOIACT)*, 2020, pp. 170–175.

[4] T. Hidayat, M. H. Zakaria, and A. N. C. Pee, "Increasing the Huffman generation code algorithm to equalize compression ratio and time in lossless 16-bit data archiving," *Multimed. Tools Appl.*, vol. 82, no. 16, pp. 24031–24068, Jul. 2023.

[5] J. C. Cavalcanti, M. Englert, M. Oliveira, and A. C. Constantini, "Microphone and Audio

Compression Effects on Acoustic Voice Analysis: A Pilot Study," *J. Voice*, vol. 37, no. 2, pp. 162–172, Mar. 2023.

[6] S. Congero and K. Zeger, "Competitive Advantage of Huffman and Shannon-Fano Codes," *IEEE Trans. Inf. Theory*, pp. 1–1, 2024.

[7] U. Jayasankar, V. Thirumal, and D. Ponnurangam, "A survey on data compression techniques: From the perspective of data quality, coding schemes, data type and applications," *J. King Saud Univ. - Comput. Inf. Sci.*, vol. 33, no. 2, pp. 119–140, Feb. 2021.

[8] T. Hidayat, M. H. Zakaria, and N. Che Pee, "Comparison of Lossless Compression Schemes for WAV Audio Data 16-Bit Between Huffman and Coding Arithmetic," *Int. J. Simul. Syst. Sci. Technol.*, vol. 19, no. 6, pp. 36.1-36.7, Feb. 2019.

[9] J. L. Bentley, D. D. Sleator, R. E. Tarjan, and V. K. Wei, "A locally adaptive data compression scheme," *Commun. ACM*, vol. 29, no. 4, pp. 320–330, Mar. 1986.

[10] P. Sarker and M. L. Rahman, "Introduction to Adjacent Distance Array with Huffman Principle: A New Encoding and Decoding Technique for Transliteration Based Bengali Text Compression," 2021, pp. 543–555.

[11] H. Al-Bahadili and S. M. Hussain, "A bit-level text compression scheme based on the ACW algorithm," *Int. J. Autom. Comput.*, vol. 7, no. 1, pp. 123–131, 2010.

[12] H. Hermassi, R. Rhouma, and S. Belghith, "Joint compression and encryption using chaotically mutated Huffman trees," *Commun. Nonlinear Sci. Numer. Simul.*, vol. 15, no. 10, pp. 2987–2999, Oct. 2010.

[13] R. A. Chowdhury, M. Kaykobad, and I. King, "An efficient decoding technique for Huffman codes," *Inf. Process. Lett.*, vol. 81, no. 6, pp. 305–308, 2002.

[14] P. R. Suri and M. Goel, "Ternary Tree and Memory-Efficient Huffman Decoding Algorithm," *Int. J. Comput. Sci. Issues*, vol. 8, no. 1, pp. 483–489, 2011.

[15] D. Huffman, "A Method for the Construction of Minimum-Redundancy Codes," *Proc. IRE*, vol. 40, no. 9, pp. 1098–1101, Sep. 1952.

[16] K. Chung, "Efficient Huffman decoding," *Inf. Process. Lett.*, vol. 61, pp. 97–99, 1997.

[17] R. Schack, "The length of a typical Huffman codeword," *IEEE Trans. Inf. Theory*, vol. 40, no. 4, pp. 1246–1247, Jul. 1994.

[18] G. Katona and O. Nemetz, "Huffman codes and self-information," *IEEE Trans. Inf. Theory*, vol. 22, no. 3, pp. 337–340, May 1976.

[19] R. Hashemian, "Memory efficient and high-speed search Huffman coding," *IEEE Trans. Commun.*, vol. 43, no. 10, pp. 2576–2581, 1995.

[20] P. M. Fenwick, "Huffman code efficiencies for extensions of sources," *IEEE Trans. Commun.*, vol. 43, no. 2/3/4, pp. 163–165, Feb. 1995.

[21] W. Szpankowski and S. Verdu, "Minimum Expected Length of Fixed-to-Variable Lossless Compression Without Prefix Constraints," *IEEE Trans. Inf. Theory*, vol. 57, no. 7, pp. 4017–4025, Jul. 2011.

[22] M. B. Baer, "A general framework for codes involving redundancy minimization," *IEEE Trans. Inf. Theory*, vol. 52, no. 1, pp. 344–349, Jan. 2006.

[23] X. Kavousianos, E. Kalligeros, and D. Nikolos, "Test Data Compression Based on Variable-to-Variable Huffman Encoding With Codeword Reusability," *IEEE Trans. Comput. Des. Integr. Circuits Syst.*, vol. 27, no. 7, pp. 1333–1338, Jul. 2008.

[24] J. S. Vitter, "Algorithm 673: Dynamic Huffman coding," *ACM Trans. Math. Softw.*, vol. 15, no. 2, pp. 158–167, 1987.

[25] A. Habib, A. S. M. L. Hoque, and M. R. Hussain, "H-HIBASE: Compression Enhancement of HIBASE Technique Using Huffman Coding," *J. Comput.*, vol. 8, no. 5, pp. 1175–1183, May 2013.

[26] R. Gallager, "Variations on a theme by Huffman," *IEEE Trans. Inf. Theory*, vol. 24, no. 6, pp. 668–674, Nov. 1978.

[27] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Trans. Inf. Theory*, vol. 23, no. 3, pp. 337–343, May 1977.

[28] Welch, "A Technique for High-Performance Data Compression," *Computer (Long. Beach. Calif).*, vol. 17, no. 6, pp. 8–19, Jun. 1984.

[29] Y. K. Lin, S. C. Huang, and C. H. Yang, "A fast algorithm for Huffman decoding based on a recursion Huffman tree," *J. Syst. Softw.*, vol. 85, no. 4, pp. 974–980, 2012.

[30] J. Alakuijala *et al.*, "Brotli," *ACM Trans. Inf. Syst.*, vol. 37, no. 1, pp. 1–30, Jan. 2019.
[31] K. Sayood, "Huffman Coding," *Introd. to Data Compression*, pp. 41–88, 2018.
[32] H. Chen, W. Xie, A. Vedaldi, and A. Zisserman, "Vggsound: A Large-Scale Audio-Visual Dataset," in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020, pp. 721–725.
[33] N. Griffioen, W. Sterkens, W. Dewulf, and J. Peeters, "Enhancing Battery Detection in X-Ray Imaging in WEEE with a 16 bit Deep Learning Pipeline," in *2024 Electronics Goes Green 2024+ (EGG)*, 2024, pp. 1–6.
[34] D. Snyder, G. Chen, and D. Povey, "MUSAN: A Music, Speech, and Noise Corpus," no. October, Oct. 2015.
[35] J. J. Bosch, R. Marxer, and E. Gómez, "Evaluation and combination of pitch estimation methods for melody extraction in symphonic classical music," *J. New Music Res.*, vol. 45, no. 2, pp. 101–117, Apr. 2016.
[36] J. J. Bosch, J. Janer, F. Fuhrmann, and P. Herrera, "A comparison of sound segregation techniques for predominant instrument recognition in musical audio signals," *Proc. 13th Int. Soc. Music Inf. Retr. Conf. ISMIR 2012*, no. Ismir, pp. 559–564, 2012.
[37] E. Fonseca, M. Plakal, D. P. W. Ellis, F. Font, X. Favory, and X. Serra, "Learning Sound Event Classifiers from Web Audio with Noisy Labels," *ICASSP, IEEE Int. Conf. Acoust. Speech Signal Process. - Proc.*, vol. 2019-May, no. 688382, pp. 21–25, 2019.