Loss Less compression Algorithms and their comparison

Mitrajeet Golsangi*, Milind Kulkarni†,
dept. of Computer Science
Vishwakarma Institute of Technology
Pune, India
Email : *mitrajeet.golsangi20@vit.edu, †milind.kulkarni@vit.edu

***Abstract*—The paper discusses three of the many lossless compression algorithms. It discusses the Huffman coding, run length coding (RLE) and the Arithmetic coding in detail. The main aim of the paper is to make the readers understand these 3 algorithms in detail. The paper further discusses the differences in each of the algorithms and compares them to each other. The aim of the paper is not to give the readers a good lossless compression algorithm, but to make the reader understand the different types of compression algorithms and then decide the best one required for the application on their own accord.**

***Index Terms*—compression, Huffman coding, optimal merging, Arithmetic Coding, Run Length Coding, Data compression, Entropy compression**

I. Introduction

Since ancient times the migration of data from one place to another has been a difficulty, and one of the main concerns in it was its size of it. If the data, say a hunted elephant was to be transported it was nearly impossible to pick it up whole. Thus, came the concept of data compression. Data compression is a technique of changing the data in order to make it more transportable. Like in the previous analogy a hunted elephant can be cut into more sizable pieces in order to transport it. Now extending this analogy to an elephant puzzle, even though the puzzle can be easily dismantled and transported from one place to another the person on the other end needs to know the exact order in which the pieces are to be re assembled in order to get the original elephant back. This process can be called as decoding.

In Computer terminologies data compression plays an important role. In today's world the data sizes are constantly increasing [1] and the devices on which it is accessed have a relatively lower storage space due to the drive in creating smaller devices. Data compression can be said to be a puzzle that everyone is trying to solve. It was not until a scientist David Huffman [2], in 1952 proved mathematically that the compression of a string when considering a single character is the most efficient when using a variable code length method which can be constructed using the optimal merging method. This ground breaking invention was a milestone in the string compression era and set the world on a new path.

This first started in 1951 when Huffman was a college student in MIT and told his idea to a number of people. This was seen in [3]. Huffman took it upon himself to solve the Fano problem and create a more efficient coding algorithm. After months of study and failures he was finally able to create the Huffman coding algorithm setting the Shannon-Fano coding as redundant and made the compression applications use Huffman coding instead. This is still continued on and there have been a lot of compression algorithms or techniques for achieving the size compression.

II. Literature Review

Before studying about encoding algorithms, it is important to know the meaning of encoding and types in which it can be performed. Strings in computers are interpreted as the binary equivalent of that character. Since the International Phonetic Alphabet (IPA) which was developed in 1888 [4] it uses the American Standard Code for Information Interchange (ASCII) to represent it's characters. The ASCII code is a 8-bit long binary string which encompasses a total of 128 characters [5] which is exactly the number of characters defined in the IPA. These 128 characters may include a wide alphabets, numerical characters and even some special characters. These ASCII values are used as a standard for interpreting the alphabets typed in by the user. Thus, each character has a 8-bit binary equivalent which is practically stored in the memory of the computer. Encoding a certain string means changing the regular ASCII code of that string into a smaller more memory efficient format. Every program in the world follows a certain generic string encoding rules, which are explained in detail in [6]. Like mentioned before encoding means assigning a more memory efficient value to the string. But this comes with a problem, if the server computer assigns a different code to the string the receiving computer cannot interpret that code, thus the message becomes useless. So in order for the receiving computer to successfully interpret the message the conversion table of the codes must be attached to the message that has been sent. For example, consider Tbl. I

If Tbl. I determines the encoding of ASCII characters into a more efficient format then Fig. 1 describes how the message must be sent in order for the receiving computer to interpret it properly

Now there are certain types of encoding in strings, they are broadly categorized into 2 main types. The first is the fixed code style, in which all the characters are given a fixed code length and the second is the variable code length, here each character is given variable code length according to certain

Character	ASCII	Encoded
A	01000001	00
B	01000010	01
C	01000011	10
D	01000100	11

TABLE I
EXAMPLE OF DATA ENCODING

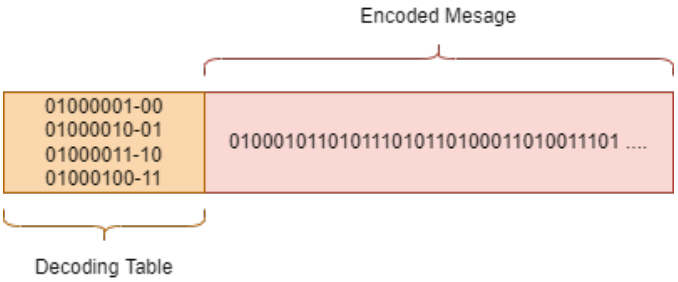

Fig. 1. Example of an Encoded message

conditions. Huffman coding follows the second approach and it more memory efficient than the first one.

A. Huffman Coding

In [2] David Huffman provides a method to encode a message using minimum redundancy. Here, redundancy removal follows two major rules

1) Identical coding digit arrangements are forbidden in any two messages
2) Message should be encoded in such a way that, once the starting point of sequence is known no additional indication may be required to decode the message

Thus, in order to achieve this we check if the given binary string satisfies the Eq. 1. If the given message satisfies the equation we say that it is potentially prefix-free, this was noted by L.T. Kraft. [7] and further developed by McMillan and Brockway [8].

$$K(T) = \sum_{i=0}^{n-1} 2^{-l_i} \leq 1 \tag{1}$$

Only knowing if the message was prefix free or not posed little importance, the real question was calculating the cost of a feasible code. This denoted by C(·, ·) is given by Eq. 2 [9]

$$C(W,T) = C(\langle w_i \rangle, \langle l_i \rangle) = \sum_{i=0}^{n-1} w_i \cdot l_i \tag{2}$$

Thus, David Huffman proposed a methodology which would be most efficient in providing such compressions. The method used a greedy optimal merging technique to provide this.

B. Run Length Coding (RLE)

The Run Length Encoding is a simple compression algorithm, patented by Hitachi. Similar to the Huffman encoding it also compresses the files based on the frequency of the appearing characters. Even though it is a general purpose encoding algorithm it is mainly used in the compression of images and 3D renders [10]. Back in 1967, it was an important compression technique in order to compress the television signal in transit [11]. Currently, this algorithm is used by many famous file formats like, JPEG, TIFF, PNG, etc. [12]

This method is particularly good when compressing the images and 3D models, this is because the more the repetitive data the higher the compression. This is further explained in the example.

C. Arithmetic Encoding

Arithmetic Encoding is a entropy based lossless data compression algorithm. [13] Basically, instead of calculating the frequency of each character, this algorithm calculates the probability, of each one. It is a superior method than the Huffman encoding method [2]. In this method all the data is represented within the real numbers 0 and 1 [14]. Here the value of each encoded character say x is $x \in [0, 1)$ Thus, a simple example of the same can be represented as sown in Tbl II

Character	Probability	Range
A	0.2	[0,0.2)
B	0.3	[0.2,0.4)
C	0.5	[0.4,0.6)
D	0.1	[0.6,0.8)
E	0.4	[0.8,0.9)
F	0.8	[0.9,1.0)

TABLE II
EXAMPLE OF ARITHMETIC ENCODING

Here, if we consider, L ≡ smallest binary value R ≡ product of probabilities then, L is replaced with

$$L = L + R \sum_{i=1}^{j-1} \tag{3}$$

and R is replaced with

$$R = R \times p_j \tag{4}$$

[15]

III. METHODOLOGY

A. Huffman Coding

The method is fairly straight forward and easy to explain with the help of an example. Consider a string

$$ABCCDEBBFDEBBCDAEGGDEDBBCDEFAG$$

The above string has a total length of 30 characters. Tbl III gives the respective ASCII representation and number of occurrences of characters in this string

Character	ASCII	count
A	01000001	3
B	01000010	7
C	01000011	4
D	01000100	6
E	01000101	5
F	01000110	2
G	01000111	3

TABLE III
ASCII REPRESENTATION OF STRING

Thus, the total length of the string in bits can be given as

$$\begin{aligned} len(S) &= \text{Number of characters} \times 8 \\ &= 30 \times 8 \\ \therefore len(S) &= 240bits \end{aligned}$$

Now, in order to assign optimized codes to the alphabets, we need to use the greedy optimized merging technique based on the count of the alphabets in the given string. Thus, arranging the alphabets in ascending order of their frequency we get Tbl. IV

Character	ASCII	count
F	01000110	2
A	01000001	3
G	01000111	3
C	01000011	4
E	01000101	5
D	01000100	6
B	01000010	7

TABLE IV
ASCENDING ORDER OF FREQUENCY

Each character in this above table is treated as a leaf node and a optimal merging tree is created as follows. The two nodes with the minimum sum are summed and the sum is then appended in the table, the table is then sorted in ascending order and the procedure is repeated until the length of the string is reached. Thus, the table formed after optimal merging is shown in Tbl. V

This is better explained when a tree for the table is drawn as shown in fig 2

Now in order to generate codes one must assign 0 to the branch from the root node going left of it and 1 to the leaf node going right. This method gives variable codes to the alphabets given in the string. As the tree is made from summing the minimum frequency characters the minimum number of branches are required for the maximum frequency code and thus, the most occurring letter will have the least memory occupancy and vice versa. Thus, the codes generated for the given letters in the string are given in Tbl. VI

Character(s)	count
F	2
A	3
G	3
C	4
F, A	5
E	5
G, C	7
B	7
D	6
E, D	11
(F, A), (G, C)	12
(E, D), B	18
[(F, A), (G, C)], [(E, D), B]	30

TABLE V
ASCENDING ORDER OF FREQUENCY

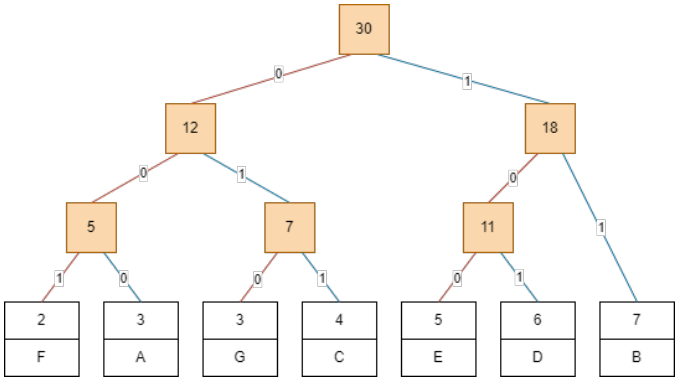

Fig. 2. Optimal Merge Tree

In Tbl. VI notice that the bits occupied by F which is the least frequent is 3 whereas the bits occupied by B which is the most frequent is 2. This is the importance of variable code lengths. Even though it might not seem much in a 30 character small string, it makes a huge difference when it comes to huge strings. Thus, the total size of the encoded string is given by Eq. 5

Character(s)	count	code
A	3	110
B	7	00
C	4	100
D	6	010
E	5	011
F	2	111
G	3	101

TABLE VI
ASCENDING ORDER OF FREQUENCY

$$len(S_{cmp}) = \sum_{i=1}^{n} D_i F_i \tag{5}$$

where,
$n \equiv$ Number of characters
$D_i \equiv$ Distance of Leaf from Root
$F_i \equiv$ Frequency of Character

$$\begin{aligned}
\therefore len(S_{cmp}) &= \sum_{i=1}^{7} D_i F_i \\
&= (3 \times 3) + (3 \times 2) + (3 \times 3) + \\
&\quad (3 \times 3) + (3 \times 3) + (3 \times 3) + (3 \times 3) \\
\therefore len(S_{cmp}) &= 60bits
\end{aligned}$$

Now, calculating the tree size

$$\begin{aligned}
len(Tree) &= 8n + \sum_{i=1}^{n} size(i) \\
&= (8 \times 7) + [(3) + (2) + (3) + (3) \\
&\quad + (3) + (3) + (3)] \\
&= 56 + 14 \\
\therefore len(Tree) &= 70bits
\end{aligned}$$

$$\begin{aligned}
\therefore total_size &= len(tree) + len(s_{cmp}) \\
&= 70 + 60 \\
\therefore total_size &= 130bits
\end{aligned}$$

Thus, the total size of the string which previously was 240 bits reduced to 130 bits. Meaning the Huffman compression made a 54.167% compression on the string

B. Run Length Encoding

The Run Length Encoding method is an important compression algorithm in terms of repetitive input string compression. Consider a string

$$AAAAABBBBBBBBDDDDDDSSSSSSSS$$

Now, Applying the run length compression on Str, the program will read the string from left to right. For each encounter of the a duplicate symbol the program increments of the frequency of that character and scans the next character. The frequency of the character increases unit another character different from the current traversing character appears before the read head. Thus, after such a case the head writes down a pair of alphanumeric string, which first contains the frequency of the character followed by the actual character. Thus, after iterating for A the string will look like this

$$5ABBBBBBBBDDDDDDSSSSSSSS$$

This, process continues for the rest of the characters until all the characters are brought down to the frequency and character pair format. Thus, the final compressed string will be

$$5A8B5D8D$$

Now, in order to calculate the compression ratio, we can consider, that each number or character is of 8-bits, thus we get

$$\begin{aligned}
len(S) &= \text{No. of Characters} \times 8 \\
len(S_{uncmp}) &= 26 \times 8 \\
&= 208bits \\
len(S_{cmp}) &= 8 \times 8 \\
&= 64bytes \\
\therefore \text{Compression Percentage} &= \frac{len(S_{cmp})}{len(S_{uncmp})} * 100 \\
&= 30\%
\end{aligned}$$

Now, consider another string,

$$ABCDEASE$$

This string has no repetition of the characters. Thus after applying the run length encoding algorithm on it the encoded string obtained is

$$1A1B1C1D1E1A1S1E$$

Thus, calculating the length of each string we get

$$\begin{aligned}
len(S_{uncmp}) &= 8 \times 8 \\
&= 65bits \\
len(S_{cmp}) &= 16 \times 8 \\
&= 128bits
\end{aligned}$$

Thus, here after application of the run length encoding on a string the size of the string is increased instead of being reduced. This is a major drawback in RLE, and one of the mak=jor reasons why it is widely used in the image sector for compression instead of text.

C. Arithmetic Encoding

Now, to understand the working of arithmetic encoding, the paper assumes the following

Prerequisites:

1) Basics of probability
2) Basic knowledge of computation activities

As done before consider a string

$$WENT.$$

Here the string contains 5 characters which may be denoted by the list S', where S' = { "W", "E", "N", "T", "." }

Thus, now considering the total length of the list i.e. 5 and the number of non-repeating characters will give the probability for each character.

Thus, we divide the interval from 0 to 1 in 5 parts, where each division will be the range for the given symbol. This is shown in Fig. **??**

Now, in the list S' we have the first element as "W", so now we expand the range [0.8, 0.9) and then further subdivide it

Character(s)	count	Probability
W	1	0.3
E	1	0.3
N	1	0.2
T	1	0.1
.	1	0.1

TABLE VII
ASCENDING ORDER OF FREQUENCY

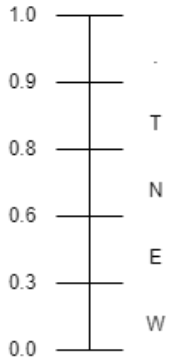

Fig. 3. Probability Distribution of Characters

into 5 parts. This division or range of symbols is calculated using the formula

$$Range_j = \text{lower limit} : \text{lower limit} + (\text{upper bound} - \text{lower bound})p_j \tag{6}$$

Thus, substituting the values for e we get

$$Range_E = 0.8 : 0.8 + (0.9 - 0.8) \times 0.3$$
$$Range_E = 0.8 : 0.83$$

This is continued till we get the last character. Thus the figurative representation of the same is given in fig 4

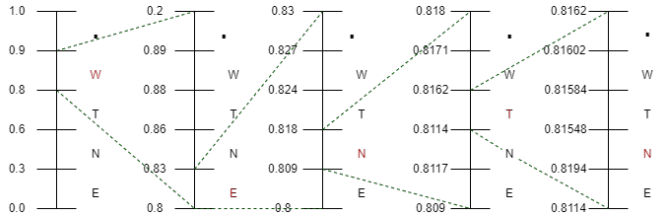

Fig. 4. Probabilistic splitting of characters

Now, that all the probabilities are created and every symbol has a range of probabilities, the arithmetic code word and tag are created.

An arithmetic code word is the key by which the message is converted into the encoded format. Thus, in our example the range of the code word is given by

$$0.81602 < codeword < 0.8162$$

Thus, after the code word's range is found we move on to the creation of the arithmetic tag which is given by

$$Tag = \frac{\text{upper limit of codeword} + \text{lower limit of codeword}}{2} \tag{7}$$

Thus, in our example,

$$Tag = \frac{0.81602 + 0.8162}{2} = 0.81611$$

Then, the complete range undergoes binary division, until the divided chunk lies completely inside the obtained interval. This is done using the binary expansion

$$0.\beta_1 \dots \beta_m = \sum_{i=1}^{m} \frac{\beta_i}{2^i} \tag{8}$$

Thus, the diagramatic representation of the same can be shown in Fig. 5

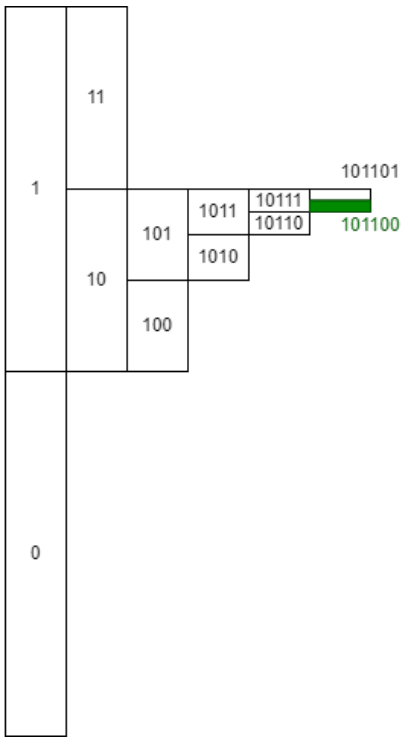

Fig. 5. Airthmetic Binary Expansion

Thus, the final encoding of the word "WENT." is converted into its binary equivalent as 101100, this is because

$$\begin{aligned} .101100 &= \frac{1}{2^1} + \frac{1}{2^3} + \frac{1}{2^4} \\ &= \frac{1}{2} + \frac{1}{8} + \frac{1}{16} \\ &= 0.5 + 0.125 + 0.0625 \\ .101100 &= 0.6875 \end{aligned}$$

Which is inside the interval of

Similarly, we can decode the program using the backtracking of the same method, i.e. we find out the interval in which

IV. Comparison

Till this point the paper has discussed the various compression algorithms. This might be a bit overwhelming for the reader, thus the paper will now discuses the key differences between each approach and give the reader the best possible approach for file compressions

A. Huffman Encoding vs Run Length Encoding

Both the RLE compression technique and Huffman Encoding are frequency base encoding techniques. The major difference between RLE and Huffman Encoding is the way of codes generated. Huffman encoding generates a code using a optimized merging technique in order to generate a code tree, whereas RLE simply converts the frequency of each character into a $< frequency >< character >$ tuple [16]. The major usage of RLE is done inside the image and 3D object compression as these files have a lot of repetitive values, and as explained in the previous examples, the RLE compression for non repeating characters may not be necessarily lesser in size than the original File. Huffman coding provides a time complexity of $O(n \log n)$, this is because the encoding of each unique character has a time complexity of $O(n \log n)$ and the extraction of the minimum frequency from the priority queue takes place $2 \times (n-1)$ times with each operation having the time complexity $O(\log n)$ [2] Run Length Encoding provides a time complexity of $O(n)$ [10], this is because the algorithm simply traverses the string and then converts the same repeating characters into tuples. Thus, compared to Huffman encoding the RLE compression algorithm is computationally less expensive but provides a poor compression ratio for non repeating strings.

B. Huffman Encoding vs Arithmetic Encoding

Arithmetic encoding is said to be an advancement on Huffman Encoding. Arithmetic encoding performs exceptionally well on the large datasets. It basically assigns a sequence of bits to a message, thus the Arithmetic coding algorithm uses a set of probabilistic model. This in Huffman coding is obtained by making a optimized merged tree, which is relatively more computationally extensive. [17] The time complexity for $O(|\Sigma| + n)$ whereas the overall time complexity for Huffman encoding is $O(n \log n)$ [2]. Overall arithmetic encoding can be said to be an improvement on Huffman coding. The arithmetic coding overcomes some difficulties faced in the Huffman encoding algorithm.

V. Drawbacks

Further the paper will discuss the drawbacks of each of the algorithms. A common drawback for all the discussed compression algorithms is the compression ratio. As compared to the lossy compression algorithms.

A. Huffman Encoding

Since, Huffman encoding creates an entire tree structure for the given input, it has a really computationally expensive as compared to the other compression techniques

B. Run Length Encoding

Even though it is really computationally inexpensive the algorithm only compresses the repetitive occurring strings. If teh string fails to fit in this criterion the RLE compressed file may even be bigger than the original file.

C. Arithmetic Encoding

Since, arithmetic encoding completely relies on the probability of the occurring characters, the complete codeword is needed in order to decode the message. This means that even a single corrupt bit may result in the corruption of the entire message.

VI. Conclusion

Thus, after all the detailed discussion about various compression techniques, and comparing them with each other, the conclusion question is, which algorithm to chose ? The answer to that question depends on the readers needs. This is because if the user wants a general purpose small text file compression Huffman coding will give the best results. Whereas if the user wants to compress 3D models or images the run length encoding gives the perfect outcome. But, if the user has a large dataset containing a general purpose data, the arithmetic encoding gives the best results. Thus, overall the lossless compression algorithms provide a way to make the transfer of heavy data easy without losing any essential information.

References

[1] K. Sayood, *Introduction to data compression*. Morgan Kaufmann, 2017.

[2] D. A. Huffman, "A method for the construction of minimum-redundancy codes," *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098–1101, 1952.

[3] G. Stix, "Profile: Information theorist david a. huffman." *Scientific American*, vol. 3, no. 265, pp. 54–58, 1991.

[4] J. L. Hieronymus, "Ascii phonetic symbols for the world's languages: Worldbet," *Journal of the International Phonetic Association*, vol. 23, p. 72, 1993.

[5] D. Salomon, "The ascii code," in *Data Compression*. Springer, 1998, pp. 301–303.

[6] S. Legg, "Generic string encoding rules (gser) for asn. 1 types," *Internet proposed standard RFC*, vol. 3641, 2003.

[7] L. G. Kraft, "A device for quantizing, grouping, and coding amplitude-modulated pulses," Ph.D. dissertation, Massachusetts Institute of Technology, 1949.

[8] B. McMillan, "Two inequalities implied by unique decipherability," *IRE Transactions on Information Theory*, vol. 2, no. 4, pp. 115–116, 1956.

[9] A. Moffat, "Huffman coding," *ACM Computing Surveys (CSUR)*, vol. 52, no. 4, pp. 1–35, 2019.

[10] D.-H. Xu, A. S. Kurani, J. D. Furst, and D. S. Raicu, "Run-length encoding for volumetric texture," *Heart*, vol. 27, no. 25, pp. 452–458, 2004.

[11] A. H. Robinson and C. Cherry, "Results of a prototype television bandwidth compression scheme," *Proceedings of the IEEE*, vol. 55, no. 3, pp. 356–364, 1967.

[12] D. A. M. A. Ibrahim and D. M. E. Mustafa, "Comparision between (rle and huffman) algorithms for lossless data compression," *Internation Journal of Inovation Technology and Research*, vol. 3, pp. 1808–1812, 2015.

[13] Z.-N. Li, M. S. Drew, and J. Liu, "Social media sharing," in *Fundamentals of multimedia*. Springer, 2014, pp. 617–643.

[14] I. H. Witten, R. M. Neal, and J. G. Cleary, "Arithmetic coding for data compression," *Communications of the ACM*, vol. 30, no. 6, pp. 520–540, 1987.

[15] A. Moffat, R. M. Neal, and I. H. Witten, "Arithmetic coding revisited," *ACM Transactions on Information Systems (TOIS)*, vol. 16, no. 3, pp. 256–294, 1998.

[16] M. Stabno and R. Wrembel, "Rlh: Bitmap compression technique based on run-length and huffman encoding," *Information Systems*, vol. 34, no. 4-5, pp. 400–414, 2009.

[17] A. Shahbahrami, R. Bahrampour, M. S. Rostami, and M. A. Mobarhan, "Evaluation of huffman and arithmetic algorithms for multimedia compression standards," *International Journal of Computer Science Engineering and Applications*, vol. 11, no. 25, 2016.