A Comparative Study of Entropy Encoding Techniques for Lossless Text Data Compression

P. RATNA TEJASWI[1]
Computer Science & Engineering
Swami Vivekananda Institute of Technology
Hyderabad, India

P. DEEPTHI[2]
Computer Science & Engineering
Swami Vivekananda Institute of Technology
Hyderabad, India

V.PALLAVI[3]
Computer Science & Engineering
Swami Vivekananda Institute of Technology
Hyderabad, India

D. GOLDIE VAL DIVYA[4]
Computer Science & Engineering
Swami Vivekananda Institute of Technology
Hyderabad, India

Abstract: *Data compression is the art of reducing the number of bits used to store or transmit information. Data compression reduces the size of data by removing excessive information from it. There are two major categories of compression algorithms: Lossy and Lossless. This paper examines entropy encoding techniques and compares their performance.*

Keywords: Data Compression, Lossy compression, Lossless compression, Shannon Fano, Huffman coding, Adaptive Huffman coding, Arithmetic coding.

I. INTRODUCTION

Data compression is the representation of an information source (a data file, a speech signal, an image, or a video signal) as accurately as possible using the fewest number of bits [1]. Data compression involves the development of a compact representation of information. Most of the information representations contain large amounts of redundancy. Redundancy can exist in various forms. It may exist in the form of correlation, context, Redundancy is "the part of the message that can be eliminated without the loss of essential information." Therefore, one aspect of data compression is redundancy removal.

After the redundancy removal process, the information needs to be encoded into a binary representation. At this stage we make use of the fact that if the information is represented using a particular alphabet some letters may occur with higher probability than others. In the coding step we use shorter code words to represent letters that occur more frequently, thus lowering the average number of bits required to represent each letter.

The design of a compression algorithm involves understanding the types of redundancy present in the data and then developing strategies for exploiting these redundancies to obtain a compact representation of the data. However, more popularly, compression schemes are divided into two main groups: lossless compression and lossy compression. Lossless compression preserves all the information in the data being compressed, and the reconstruction is identical to the original data. In lossy compression some of the information contained in the original data is irretrievably lost. The loss in information is, in some sense, a payment for achieving higher levels of compression.

II. TYPES OF DATA COMPRESSION

Data compression techniques are broadly classified into two types. They are

a. Lossless Data Compression
b. Lossy Data Compression

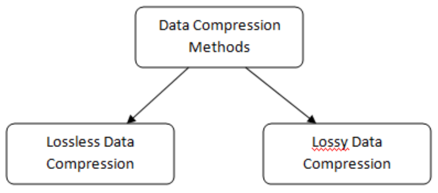


Fig 1. Types of Data compression techniques

A. Lossless Data Compression

Lossless data compression can recover exactly the original data from the compressed data after a compress/expand cycle. Lossless compression is generally used for discrete data, such as database records, spreadsheets, word-processing files, and even some kinds of image and video information [2]. Lossless data compression is used ubiquitously in computing, from saving space on your personal computer to sending data over the web, communicating over a secure shell, or viewing a PNG or GIF image.

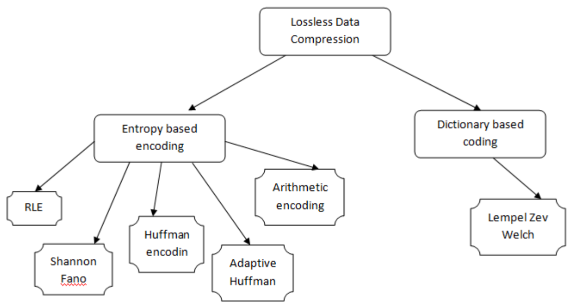


Fig 2: Types of Lossless Data Compression

1. Entropy based encoding

Entropy coding creates and assigns a unique prefix-free code to each unique symbol that occurs in the input. These entropy encoders then compress data by replacing each fixed-length input symbol with the corresponding variable-length prefix-free output codeword.

2. Dictionary based coding

A dictionary based coding operate by searching for matches between the text to be compressed and a set of strings contained in a data structure (called the 'dictionary') maintained by the encoder. When the encoder finds such a match, it substitutes a reference to the string's position in the data structure.

B. Lossy Data Compression

Lossy data compression original data is not exactly restored after decompression and accuracy of re-construction is traded with efficiency of compression. Lossy for images and sound where a little bit of loss in resolution is often undetectable, or at least acceptable.

III. Entropy Encoding Techniques

A. Run Length Encoding

Run-Length Encoding is a very simple compression technique that replaces a string of repeated symbols with a single symbol and a count (run length) indicating the number of times the symbol is repeated [3].

The following Input string:
MINIMUM
String can be encoded more compactly by replacing each repeated string of characters by a single instance of the repeated character and a number that represents the number of times it is repeated.
Output: 1M1N1I1M1U1M

B. Shannon Fanon Encoding

Shannon-Fano coding is developed by Claude Shannon at Bell Labs and R.M. Fano at MIT. It depended on simply knowing the probability of each symbol's appearance in a message [4]. This technique involves generating a binary tree to represent the probabilities of each symbol occurring. The symbols are ordered such that the most frequent symbols appear at the top of the tree and the least likely symbols appear at the bottom. The Shannon-Fano tree is built from the top down, starting by assigning the most significant bits to each code and working down the tree until finished.

The Shannon-Fano Algorithm:
Step 1:
For a given list of symbols, develop a corresponding list of probabilities or frequency counts so that each symbol's relative frequency of occurrence is known.
Step 2:
Sort the lists of symbols according to frequency, with the most frequently occurring symbols at the top and the least common at the bottom.
Step 3:
Divide the list into two parts, with the total frequency counts of the upper half being as close to the total of the bottom half as possible.
Step 4:
The upper half of the list is assigned the binary digit 0, and the lower half is assigned the digit 1. This means that the codes for the symbols in the first half will all start with 0, and the codes in the second half will all start with 1.
Step 5:
Recursively apply the steps 3 and 4 to each of the two halves, subdividing groups and adding bits to the codes until each symbol has become a corresponding code leaf on the tree.
Input String: MINIMUM

TABLE 1: SYMBOL FREQUENCIES

Symbol	**Count**
M	3
I	2
N	1
U	1

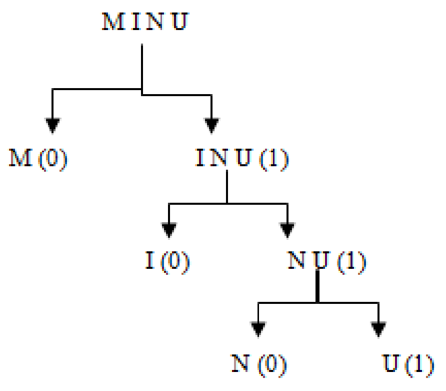
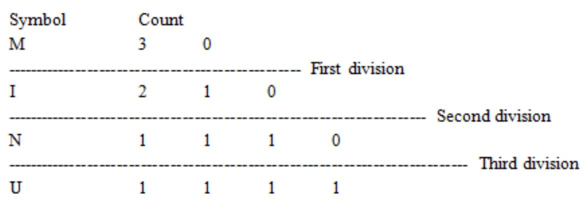


Fig 3: Shannon Fanon Tree



Symbol	Count				
M	3	0			
---------- First division					
I	2	1	0		
---------- Second division					
N	1	1	1	0	
---------- Third division					
U	1	1	1	1	

Fig 4: Shannon Fanon division

Putting the dividing line between symbols M and I assigns a count of 3 to the upper group and 4 to the lower. This means that M will have a code that starts with a 0 bit, and I, N and U are all going to start with a 1. Follow the same procedure for further divisions. After four division procedures, tables of codes are generated.

Symbol	Code	
M	0	
I	10	
N	110	
U	111	

Tab 2: Shannon Fanon Symbol codes

C. *Huffman Encoding*

Huffman coding, an algorithm developed by David A. Huffman published in the 1952 paper "A Method for the Construction of Minimum-Redundancy Codes". Huffman coding shares most characteristics of Shannon-Fano coding. It creates variable-length codes that are an integral number of bits. Symbols with higher probabilities get shorter codes. Two families of Huffman Encoding have been proposed: Static Huffman Algorithms and Adaptive Huffman Algorithms. Static Huffman Algorithms calculate the frequencies first and then generate a common tree for both the compression and decompression processes [5]. Details of this tree should be saved or transferred with the compressed file. The Adaptive Huffman algorithms develop the tree while calculating the frequencies and there will be two trees in both the processes.

A binary tree is created using the symbols as leaves according to their probabilities and paths of those are taken as the code words. Huffman codes are built from the bottom up, starting with the leaves of the tree and working progressively closer to the root.

The tree is then built with the following steps:
Step 1:
The two free nodes with the lowest weights are located.
Step 2:
A parent node for these two nodes is created. It is assigned a weight equal to the sum of the two child nodes.
Step 3:
The parent node is added to the list of free nodes, and the two child nodes are removed from the list.
Step 4:
One of the child nodes is designated as the path taken from the parent node when decoding a 0 bit. The other is arbitrarily set to the 1 bit.
Step 5:
The previous steps are repeated until only one free node is left. This free node is designated the root of the tree.

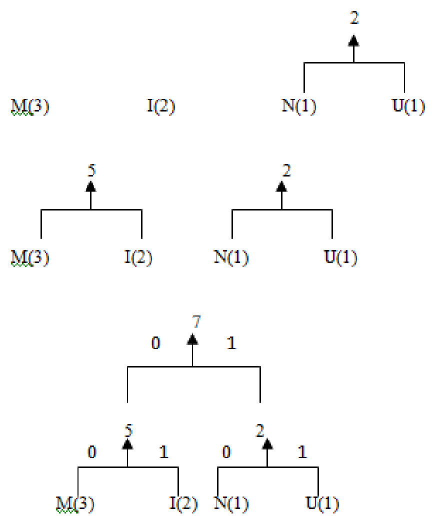


Fig 5: Huffman Encoding Tree

TABLE 3: HUFFMAN SYMBOL CODES

Symbol	Code
M	00
I	01
N	10
U	11

D. *Adaptive Huffman Encoding*

Adaptive Huffman coding was first generated by Faller in 1973 and Gallager in 1978. Knuth assisted improvements in the original algorithm in 1985 and the resulting algorithm is known as algorithm FGK. A more current version of this coding is described by Vitter in 1987 and called as algorithm [6].

```
ENCODER
-----------------
Initialize_model();
while ((c = getc (input)) != eof)
  {
    encode (c, output);
    update_model (c);
  }
}
```

```
DECODER
---------------
Initialize_model();
while ((c = decode (input)) != eof)
{
        putc (c, output);
        update_model (c);
}
```

The key is to have both encoder and decoder to use exactly the same initialization and update_model routines. Update_model does two things: (a) increment the count, (b) update the Huffman tree.

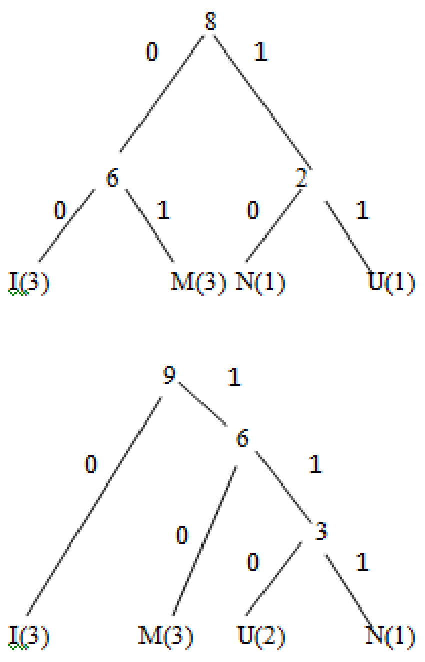


Fig 6: Adaptive Huffman Encoding

During the updates, the Huffman tree will be maintained its sibling property, i.e. the nodes (internal and leaf) are arranged in order of increasing weights. When swapping is necessary, the farthest node with weight W is swapped with the node whose weight has just been increased to W+1. The Huffman tree could look very different after node swapping.

E. *Arithmetic Encoding*

The main goal of Arithmetic coding is to replace a stream of input symbols with a single floating-point output number [7]. The algorithm begins with an interval of 0 and 1. After every input symbol from the alphabet is read, the interval is divided into a smaller interval in apposite to the input symbol's probability. This interval starts the new interval and it is divided into parts according to probability of symbols the input alphabet. This is repeated for every input symbol [8]. Unlike other statistical methods, it doesn't create tree or code for every symbol; instead it creates a code called as 'tag' for entire message.

TABLE 4: SYMBOL PROBABILITIES

Symbol	**Probability**
M	3/7 =0.42
I	2/7 = 0.28
N	1/7 = 0.14
U	1/7 = 0.14

TABLE 5: LOW AND HIGH VALUES

Symbol	**Low**	**High**
M	0	0.42
I	0.42	0.7
N	0.7	0.84
U	0.84	1

TABLE 6: ARITHMETIC ENCODING PROCESS

Symbol	**Low**	**High**	**Range**
Initial	0	1	1
M	0	0.42	0.42
I	0.1764	0.294	0.1176
N	0.25872	0.27518	0.01646
I	0.26563	0.27024	0.00461
M	0.26563	0.26756	0.00193
U	0.26725	0.26756	0.00031
M	0.26725	0.26738	

Here
New low = old low + (range * low of symbol)
New high = old low + (range * high of symbol)
Range = High – Low

Using the Tag value, we can decode the string.
Here tag value is 0.26725
Or
Tag = (0.26725+ 0.26738) / 2
= 0.267315

IV. Measuring Entropy Encoding Performances

This paper introduces the comparison of performances of above algorithms, based on different factors [9]. There are many different ways to measure the performance of a compression algorithm. The main concern is space and time efficiency, while measuring the performance. Following are some factors used to evaluate the performances of the lossless algorithms.

Compression ratio: compression ratio is the ratio between size of compressed file and the size of source file.

$$\text{Compression ratio} = \frac{\text{Size after compression}}{\text{Size before compression}}$$

Compression factor: compression factor is the inverse of compression ratio. That is the ratio between the size of source file and the size of the compressed file.

$$\text{Compression factor} = \frac{\text{Size before compression}}{\text{Size after compression}}$$

Saving percentage calculates the shrinkage of the source file as a percentage.

Saving percentage =

$$\frac{\text{Size before compression} - \text{Size after compression}}{\text{Size before compression}}$$

Compression Time can be defines as time taken to compress particular file. Time taken for the compression and decompression should be considered separately [10]. For a particular file, if the compression and decompression time is less and in an acceptable level, it means that algorithm is acceptable with respect to time.

Entropy can be used as a performance factor, if the compression algorithm is based on statistical information of the source file [11]. Let set of event be S= $\{s_1,s_2, s_3,\dots s_n\}$for an alphabet and each s_j is a symbol used in this alphabet. Let the occurrence probability of each event be p_j for event s_j. Then the self-information I(s) is defined as follows:
$I(s) = \log_b 1/p_j$ or $I(s) = -\log_b 1/p_j$

The first order Entropy value H(P) can be calculated as follows:

$$H(P) = \sum_{i=0}^{n} P_i \log_2 (^1/P_i)$$

After performing RLE, Shannon Fanon, Huffmon Encoding techniques on the string MINIMUM, the results are as follows:

TABLE 7: DATA COMPRESSION USING SF ENCODING

Symbol	**Count**	**$\log_2(^1/P_i)$**	**Shannon fanon size**	**Shannon fanon Bits**
M	3	1.2226	1	3
I	2	1.8074	2	4
N	1	2.8079	3	3
U	1	2.8079	3	3

TABLE 8: DATA COMPRESSION USING HUFFMAN ENCODING

Symbol	**Count**	**$\log_2(^1/P_i)$**	**Huffman size**	**Huffman Bits**
M	3	1.2226	2	6
I	2	1.8074	2	4
N	1	2.8079	2	2
U	1	2.8079	2	2

Entropy Calculation:

$$\text{Entropy} = \sum_{i=0}^{n} P_i \log_2 (^1/P_i)$$

H(P)=0.4285*1.2226+0.2857*1.8074+0.1428*2.8079+0.1428*2.8079
= 1.84219

Default ASCII encoding uses 8bits per symbol. Result after SF & Huffman encoding is 1.84219bits per symbol.

TABLE 9: PERFORMANCE OF DIFFERENT ENTROPY ENCODING TECHNIQUES

Entropy encoding methods	**Compression Ratio**	**Compression Factor**	**Saving percentage**
Run Length Encoding	1.333	0.7	42%
Shannon Fanon Encoding	0.2321	4.30	76%
Huffman Encoding	0.25	4	75%

Arithmetic Encoding algorithm has an Underflow problem, which gives an erroneous result after few numbers of iterations. Therefore it is not suitable for comparison. Huffman Encoding and Shannon Fano algorithm shows similar results. Shannon Fanon algorithm has more saving percentage than Huffman Encoding, so this factor can be used to determine the more efficient algorithm from these two.

While considering the major performance factors like compression ratio, compression factor and saving percentages of the all the selected algorithms. The Huffman encoding is considered as the most efficient algorithm, as the values of this algorithm lies acceptable range and it also shows better results.

V. Conclusion

This study mainly focuses on various entropy encoding techniques. In entropy encoding techniques, Shannon Fanon and Huffman encoding algorithm are the two methods which are better than RLE algorithm. With the help of various performance factors, it is easy to choose algorithms that are more efficient. This paper demonstrates that if we use the right data compression techniques, it will certainly be helpful in reducing the storage space and the computational resources.

References

[1] Introduction to Data Compression, Khalid Sayood, Ed Fox (Editor), March 2000.

[2] Universal lossless data compression algorithms, Sebastian Deorowicz, [s.n] publisher,2003

[3] Amarjit Kaur, Navdeep Singh Sethi, Harinderpal Singh, "A Review on Data Compression Techniques", IJARCSSE, Volume 5, Issue 1, January 2015, ISSN:2277 128X

[4] Christine Lamorahan, Benny Pinontoan, Nelson Nainggolan, "Data Compression Using Shannon-Fano Algorithm" Jdc, Vol. 2, No. 2, September, 2013

[5] Haroon Altarawneh and Mohammad Altarawneh, "Data Compression Techniques on Text Files:A Comparison Study" International Journal of Computer Applications (0975 – 8887) Volume 26– No.5, July 2011

[6] C.Kailasanathan, R.Safavi Naini and P.Ogunbona, "Secure Compression using Adaptive Huffman Coding"
IEEE, pp. 336-339

[7] Paul G. Howard and Jeffrey Scott Vitter, "Practical Implementations of Arithmetic Coding", Technical Report No. 92-18, Revised version, April 1992

[8] G.G. Langdon, "An Introduction to Arithmetic Coding", IBM Journal of Research and Development, Volume 28,Issue 2, April 2010, pp 135-149

[9] P.Ravi, Dr.A.Ashokkumar, "A Study of Various Data Compression Techniques", IJCSC, Volume 6, Issue 2, April-September 2015, ISSN: 0973-7391

[10] S.R. Kodituwakku and U. S. Amarasinghe "Comparison of lossless data compression algorithms For text data" Indian Journal of Computer Science and Engineering Vol 1 No 4 416-425

[11] Senthil Shanmugasundaram and Robert Lourdusamy, "A Comparative Study Of Text Compression Algorithms". International Journal of Wisdom Based Computing, Vol. 1 (3), December 2011