Comparative Study of Data Compression Techniques

Anshul Anup Rajput
M.C.A. Department
TYMCA
Sardar Patel Institute of Technology Andheri, Mumbai, India

Ravi Ashok Rajput
M.C.A. Department
(TYMCA)
Sardar Patel Institute of Technology Andheri, Mumbai, India

Pooja Raundale, PhD
M.C.A. Department
(Head of Department)
Sardar Patel Institute of Technology Andheri, Mumbai, India

ABSTRACT
This document discusses data compression and some of the data compression techniques. Data Compression is a technique of reducing the amount of space data occupies, to ease the process of storage and communication. This involves but is not limited to interpretation and elimination of redundancy in data. The fundamental process of compression involves using a well drafted technique to convert the actual data into the compressed data (smaller size). Depending upon how well a compression technique works and how much data can be regenerated from the compressed data given by a certain technique, the technique is classified as either as a lossy data compression technique or lossless data compression technique.

Keywords
Data Compression, Lossless Data Compression.

1. INTRODUCTION
Data compression is a process of converting a particular data into another form of representation which has a lesser storage requirement. Data compression is the process of applying some compression algorithm on a data or information for changing it's bits representation in such way of it reduces the size of information while storing on disk. Data compression technique is used in the cloud computing mainly for the following two reasons:

1. Increase in the transfer speed of data.
2. Decrease the storage space required to store data.

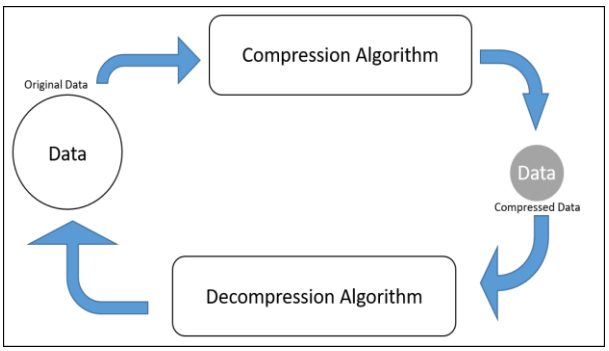

Figure 1: Pictorial Representation of Data Compression

In Lossless data compression, no actual information is lost because the bits are reduced by identifying the redundancy of data and eliminating only the redundancy. This technique is mostly applied for text document because any loss in a text document is acceptable. For example; zipping documents into one zip file compresses the data without any loss of a data while decompressing it. Lossless data compression technique is also known as a reversible compressing technique. Lossy compression reduces data size by eliminating the less valuable or unessential part of the data i.e. In a lossy data compression technique when data is compressed, some data or information is permanently removed. As a result, while doing decompression on the compressed data, we get back data which does not match the original data and the loss of data is clearly visible to the user. This technique is used for the compressing some large media file such as audio, video, jpeg file. Mp3 is yet another form of compression technique applied on audio**.** For e.g.: while sharing images clicked on our smartphones with the high mega pixels (size of pictures are high) and sharing them on WhatsApp application, the application decompresses the image to increase the transfer speed between the sender and receiver, thus resulting in a lower quality mega pixel image at the receiver's end.

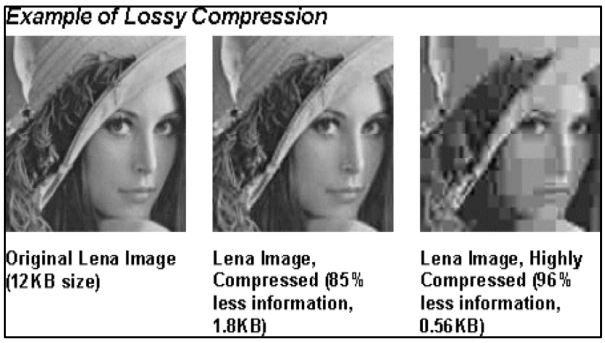

Figure 2: Levels of Lossy Compression

2. CLASSIFICATION OF LOSSLESS DATA COMPRESSION
Please Lossless data compression techniques can be classified further into Entropy based data compression technique and Dictionary based data compression technique.

In Entropy based data compression technique, the frequency of repeating data is first found and the repeating sequences are encoded.

In Dictionary based data compression, a dictionary is maintained and the encoder looks for any entry for every piece of data [4]. If there is an entry the encoder replaces the piece of data with the mapped encoding. If the entry is not present a new entry is made in the dictionary.

3. RUN-LENGTH ENCODING (RLE)
Please This is lossless data compression algorithm in which algorithm is applied on data in such way that large repeating pattern is caprice into small number of character of string, algorithm produce two bytes of output for each repeating character in data, one byte says that the total number character present in repeating pattern, and another bytes says that actual repeating character in string it's also called as run, (and its

count called as run count.)

If we consider an image which has a lot of colors in it. A hypothetical line representing the pixel colors is as follows -

WWWWWWRRRGGGGGWWWWWBBBBBBWWW WWWWWRRRRR

Once the Run length data compression algorithm is applied to the above line it can be represented as -

6W3R5G5W6B8W5R

Hence the large string is compressed into a smaller string.

As evident in the above example that the data was compressed efficiently. But this is not always the case. In RLE algorithm the worst case situation can result into the output being double the size of the Input string. For example if WRGB is to be encoded using RLE than its output will be 1W1R1G1B which means that the size of encoded string is more than the original string.

This formula is incredibly simple to implement and doesn't need abundant mainframe power unit. RLE compression is merely economical with files that contain numerous repetitive information. These are often text files if they contain numerous areas for indenting however line-art pictures that contain Pieris brassicae or black area units are much more appropriate.

4. SHANNON-FANO ALGORITHM

This algorithm is mainly used to compress text files, it is similar to Huffman Coding(HC) algorithm; the only difference is that the SF algorithm uses the top-down approach while HC algorithm uses the bottom-up approach on the text. SF algorithm is rarely used as it is not useful for all the data formats as compared to Huffman Coding algorithm which is faster and better.

In SF algorithm, the letter from string is converted to its binary form from their respective ASCII value; then the probability of the occurrence of each letter is calculated and then the tree is created based on the frequency (probability value)(starting from the top and descending till only leaf nodes are left).

Following are steps in the technique:

1. It parses the input data string and find total occurrence of each character in given input.
2. Find out probability of each repeated character
3. Sort the probability in descending order
4. Spawn leaf node for each and every symbol
5. Partition the list into 2 parts so probability of the first part is almost equal to the second part of partition
6. Prepend 0 to first part and 1 to second part
7. Repeat step 5 and 6 to first and second partition of tree till each node in tree is leaf. It doesn't give guaranty of optimal code

Considering the same string that was used in run length algorithm:

WWWWWWRRRGGGGGWWWWWBBBBBBWWWWWW WWRRRRR

Calculating the total number of times each character has occurred and arranged it in descending order of their frequency.

Then a tree can be created for the characters:

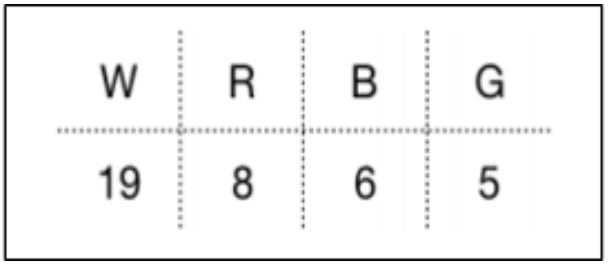

Figure 3: Frequency of letters in the considered sequence

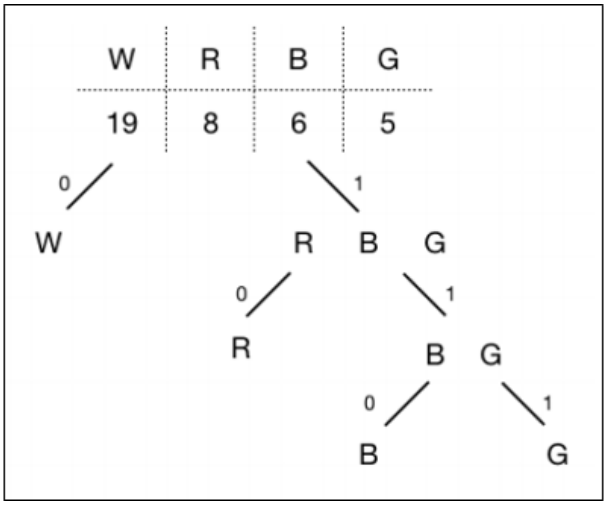

Figure 4: Tree for Shannon-Fano Encoding

The value of the characters becomes as follows:-

W - 0 R - 10 B - 110 G – 111.

Hence the encoded string becomes:-

00000010101011111111111111000001101101101101 10110000000001010101010.

The size of this output string is 68 bits i.e. ***8.5 bytes***

5. HUFFMAN CODING

The idea is founded on assigning variable length of code to input character length of code is depends on size frequency of character the highest repeatable code gets smallest code and smallest repeatable code get largest code. It is working is same as Shannon-Fanon Algorithm.[3] In Huffman coding the most repeatable word gets least binary number and least repeatable word get highest binary number. In Huffman coding the more often a symbol occurs in the original data the shorter the binary string used to represent it in the compressed data. Huffman coding requires two passes one to build a statistical model of the data and a second to encode it so is a relatively slow process. It is similar to Shannon Fano but it follows bottom up approach instead of top down approach.

Considering the same string that was used in run length algorithm:

WWWWWWRRRGGGGGWWWWWBBBBBBWWWWWW WWRRRRR

Firstly the total number of times each character has occurred

should be counted and the frequencies should be arranged in descending order. Refer figure 3 for the character count.

We then create the tree for the characters

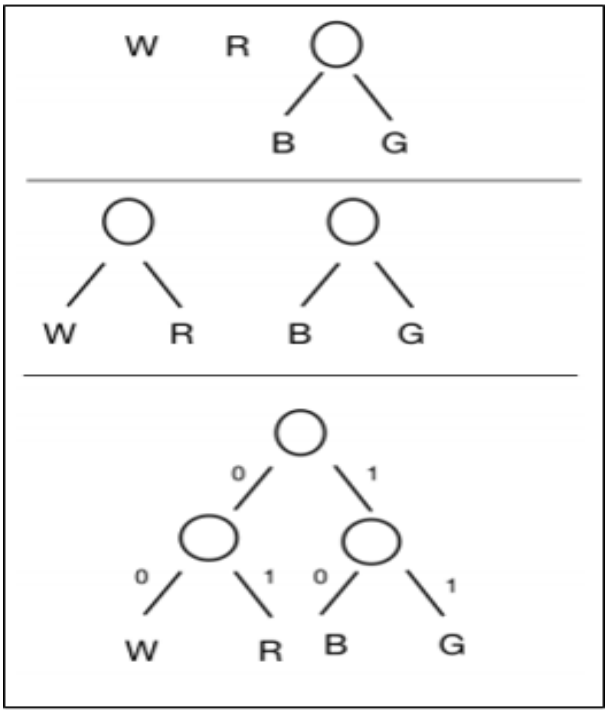

Figure 5: Huffman Tree

The value of the characters becomes as follows :-

W - 00 R - 01 B - 10 G - 11

Hence the encoded string becomes –

000000000000010101111111111100000000001010101010100000000000000000101010101

The size of this output string is 76 bits i.e. ***9.5 bytes***

6. LEMPEL-ZIV-WELCH

This algorithm is used in PDF and TIFF (Tagged Image File Format**),** it's mostly used in UNIX file compression utility and is used in GIF Format [1]. LZW reads a sequence of symbol and performs grouping of the symbols into string and then converts into code, so code takes less space than actual string.

In this algorithm, as the input data being processed, a dictionary keeps track of all the corresponding words and where they are encountered and converts them into a code value. The words are replaced by their corresponding codes and so the input file is compressed and decompression creates the same string table and by analyzing the input stream it decodes and translates the code back to the original string or text.

LZW Compression

Start with first character in the string

Dictionary = store characters together to check if the sequence is repeated again and remove redundant values

Current = 1st input character

Till string is not over

Next = 2nd input character

If current + next = string in table

Group them together

Else

Add current value to output

Add current + next to dictionary

Current = next

Next = next + 1

Output for current = reduced bits of data in numerical form

LZW Decompression

Start from the first value in the output array (compressed data)

Current = 1st value

Till output string is not over

Next = 2nd value

If Current +Next not present in dictionary

Add = Current + Next

Add to Dictionary

Add column ASCII value begins from 256 (255 is the last ASCII value + 1)

Output current

Else Search in dictionary for ASCII value (Current + Next) & add to dictionary after current (Current + value from dictionary) Current searched value ASCII value +1 Compare with next value. Output table = string.

Considering the same string as example:

WWWWWWRRRGGGGGWWWWWBBBBBBWWWWWWWWWRRRRR

Create Small Table/ Sample Table as follows. This table defines codes for each distinct character in the data. Small table is used as as initializer of the process.

Character	Code
W	1
R	2
G	3
B	4

Figure 6: Sample Table for Lempel-Ziv-Welch

Then, a complete dictionary is made as in the figure 7.

Encoded Output	Index	Pattern
	1	W
	2	R
	3	G
	4	B
1	5	WW
5	6	WWW
6	7	WWWR
2	8	RR
8	9	RRG
3	10	GG
10	11	GGG
10	12	GGW
5	13	WWWW
5	14	WWB
4	15	BB
15	16	BBB
16	17	BBBW
13	18	WWWWW
13	19	WWWWR
8	20	RRR
8	-	-

Figure 7: Dictionary for Lempel-Ziv-Welch

Encoded Output is:

1,5,6,2,8,3,10,10,5,5,4,15,16,13,13,8.

The binary encoding can be written as:

00001,00101,00110,00010,01000,00011,01010,01010,00101, 00101,00100,01111,010000,01101,01101,01000

Thus, the compression converts the string into a sequence of **80 bits i.e. 10 bytes**

The idea of this technique is such that, as the number of long repetitive sequences increase the efficiency of the algorithm increases.

7. COMPRESSION PERFORMANCE

While measuring the compression performance many factors have to be taken into consideration but the main two factors are space and time efficiency. By finding the compression ratio, compression factor, data saving percentage, compression time and the code efficiency of each algorithm we can compare them and measure their performance.

Lossless data compression is categorized into two parts

Dictionary based and entropy based encoding

Measurement parameters

Compression ratio = $\frac{Size\ of\ Uncompressed\ Data}{Size\ of\ Compressed\ Data}$

Compression factor = $\frac{1}{Compression\ Ratio}$

Saving percentage is given by the following formula,

$\frac{Uncompressed\ Data - Compressed\ Data}{Uncompressed\ Data}$ %

Compression time is the time (in milliseconds) which an algorithm takes to compress a file

Decompression time is the time (in milliseconds), which an algorithm takes to retrieve the original file from the compressed file.

The factors like compression time and decompression time are relative since different CPU speed can cause variations in performance results and hence cannot be recorded as standard results.

Size of Uncompressed Data is 38 bytes which is same for all the techniques since the string sequence used to study compression is same for all techniques.

Compression ratios for different Compression techniques are as in the table 1.

Table 1: Compression Ratios of different compression techniques

Technique	Compressed Data Size (bytes)	Compression Ratio
Run Length Encoding	14	2.7
Shannon Fano	8.5	4.5
Huffman Coding	9.5	4
Lempel Ziv Welch	10	3.8

The compression factor for the studied techniques is as in table 2:

Table 2: Compression Factor for different compression techniques

Technique	Compression Factor
Run Length Encoding	0.37
Shannon Fano	0.22
Huffman Coding	0.25
Lempel Ziv Welch	0.26

The saving percentage for the studied techniques is as in table 3.

Table 3: Saving Percentage for different compression techniques

Technique	Saving Percentage
Run Length Encoding	63.1
Shannon Fano	77.6
Huffman Coding	75.0
Lempel Ziv Welch	73.6

8. CONCLUSION

Various lossless data compression techniques studied in depth from various research papers and authors thus resulting that text data can be compressed more easily and efficiently by various techniques. For sequences with different levels of repetition and for different lengths of repeating sequences, different algorithms work differently. For instance, Shannon-Fano algorithm's compression and decompression time for text files is less as compared to Huffman coding. Run-length encoding works around consecutively repeating bits of data but does not consider all the redundancy in the data. Also, Shannon-Fano algorithm is better in terms of performance as compared to RLE but reliability is low since it can generate two different encoding sequence for same data. Hence, output is inconsistent. Huffman encoding requires two passes and hence is slow as compared to LZW which can do the encoding in a single pass. Also, one more fact can be inferred that is the data compression algorithms work on redundancy reduction by encoding redundant data. Employing better algorithms for pattern deduction or redundancy deduction using machine learning can help improve compression ratios even better.

9. REFERENCES

[1] Dea Ayu Rachesti, Tito Waluyo Purboyo, Anggunmeka Luhur Prasasti, "Comparison of Text Data Compression Using Huffman, Shannon-Fano, Run Length Encoding" , Lecturer, Faculty of Electrical Engineering, Telkom University, Bandung, Indonesia.

[2] M R Hasan , M I Ibrahimy , S M A Motakabber , M M Ferdaus and M N H Khan, Comparative data compression techniques and multi compression results, Dept. of Electrical and Computer Engineering, International Islamic University Malaysia, Gombak, Malaysia.

[3] S.R. KODITUWAKKU, U. S.AMARASINGHE, COMPARISON OF LOSSLESS DATA COMPRESSION ALGORITHMS FOR TEXT DATA, Department of Statistics & Computer Science, University of Peradeniya, Sri Lanka.

[4] Arup Kumar Bhattacharjee ,Tanumon Bej, Saheb Agarwal, Comparison Study of Lossless Data Compression Algorithms for Text Data, Comparison Study of Lossless Data Compression Algorithms for Text Data.

[5] Data compression using dynamic Markov modeling, Cormak, V. and S. Horspool, 1987. Comput. J., 30: 541–550

[6] Mohammad Hosseini, "A Survey of Data Compression Algorithms and their Applications", Applications of Advanced Algorithms, At Simon Fraser University, Canada, January 2012

[7] Blelloch, E., 2002. Introduction to Data Compression, Computer Science Department, Carnegie Mellon University.