A Research Paper on Lossless Data Compression Techniques

Prof. Dipti Mathpal
Assistant Professor
Department of Computer Engineering
Sardar Patel College of Engineering, Bakrol, India

Prof. Mittal Darji
Assistant Professor
Department of Information Technology
G. H. Patel College of Engineering and Technology, India

Prof. Sarangi Mehta
Assistant Professor
Department of Information Technology
Sardar Patel College of Engineering, Bakrol, India

Abstract

This research paper provides lossless data compression techniques and comparison between them. Data Compression is a process which reduces the size of data removing excessive information from it. It reduces the redundancy in data representation to decrease the storage required for that data and thus also reduces the communication cost by using the available bandwidth effectively. Data compression is important application in the area of file storage and distributed system. For different data formats like text, audio, video and image files there are different data compression techniques. Mainly there are two forms of data compression: - Lossy and Lossless. But in the lossless data compression, the integrity of data is to be preserved.

Keywords: Data Compression, Huffman Coding, Shannon-Fano Coding, Run Length Coding, Arithmetic Coding, LZ Algorithm

I. INTRODUCTION

Data compression is a process by which a file (Text, Audio, and Video) can be compressed, such that the original file may be fully recovered without any loss of actual information. This process may be useful if one wants to save the storage space. The exchanging of compressed file over internet is very easy as they can be uploaded or downloaded much faster. Data compression has important application in the area of file storage and distributed system. Data compression is used in multimedia field, text documents, and database table. Data Compression techniques can be classified in two major forms: Lossy Techniques & Lossless Techniques. The algorithm which removes some part of data is called lossy data compression. And the algorithm that achieve the same what we compressed after decompression is called lossless data compression.

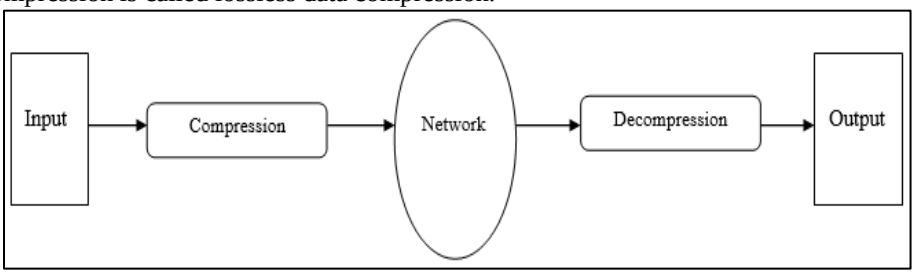


Fig. 1: Compression and Decompression

In short, Data Compression is the process of encoding data to fewer bits than the original representation so that it takes less storage space and less transmission time while communicating over a network. Data Compression is possible because most of the real world data is very redundant. A compression program is used to convert data from an easy-to-use format to one optimized for compactness. Likewise, a uncompressing program returns the information to its original form. Various lossless data compression algorithm have been proposed and used. Some of main techniques are Shannon-Fano, Huffman Coding, Run Length Encoding, and Arithmetic Encoding.

II. TYPES OF DATA COMPRESSION

There are two categories for compression of data used with digital graphics: lossy and lossless. While each uses different techniques to compress files, both have the same aim. To look for duplicate data in the graphic and use a much more compact data representation.

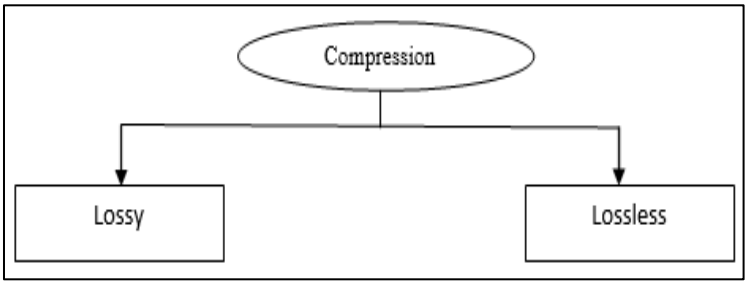


Fig. 2: Types of Compression

A. Lossy Compression

With some applications some loss of information is acceptable. In such cases lossy compression techniques are used. The best example is a videoconference where there is an acceptable amount of frame loss in order to deliver the image in real time. The loss may be in the form of color depth or graphic detail. Lossy compression looks for 'redundant' pixel information, and permanently discards it. Lossy compression isn't used for data such as text based documents and software, since they need to keep all their information. Lossy is only effective with media elements that can still 'work' without all their original data. These include audio, video, images and detailed graphics for screen design (computers, TVs, projector screens).

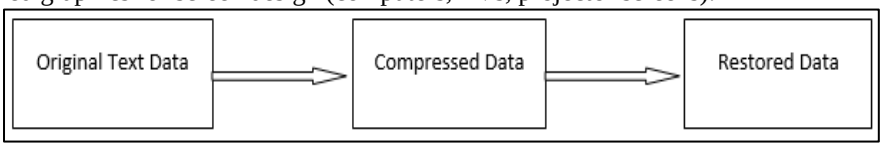


Fig. 3: Lossy Compression

B. Lossless Compression

With lossless compression, data is compressed without any loss of data. Lossless compression algorithms reduce file size with no loss in image quality. When the file is saved it is compressed, when it is decompressed (opened) the original data is retrieved. The file data is only temporarily 'thrown away', so that the file can be transferred. This type of compression can be applied not just to graphics but to any kind of computer data such as spreadsheets, text documents and software applications. This is why the term *lossless* is used, as no data is lost. While the advantage of this is that it maintains quality the main disadvantage is it doesn't reduce the file size as much as lossy compression.

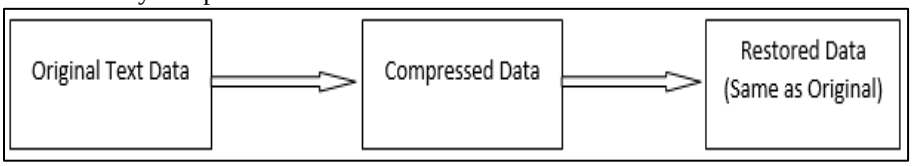


Fig. 4: Lossless Compression

III. LOSSLESS DATA COMPRESSION TECHNIQUES

A. Huffman Coding

A binary code tree is generated in Huffman Coding for given input. A code length is constructed for every symbol of input data based on the probability of occurrence. Huffman codes are part of several data formats as ZIP, GZIP and JPEG. Normally the coding is preceded by procedures adapted to the particular contents. For example the wide-spread DEFLATE algorithm as used in GZIP or ZIP previously processes the dictionary based LZ77 compression. It is a sophisticated and efficient lossless data compression technique. The symbol with highest probability has the shortest binary code and the symbol with lowest probability has the longest binary code.

There are mainly two major parts in Huffman Coding

- Build a Huffman Tree from input characters.
- Traverse the Huffman Tree and assign codes to characters.

A:	30
B:	70
C:	35
D:	14
E:	11
F:	77
G:	25

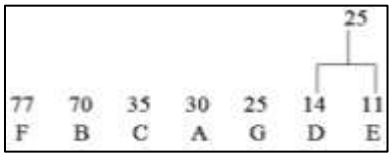
Fig. 5: Character with Frequencies.

The characters and their respective frequencies are given.

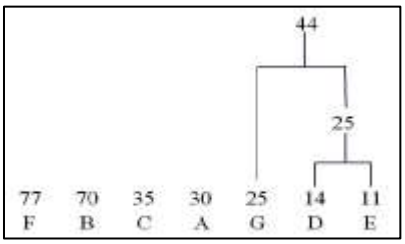
1) Step 1: In first step of building a Huffman Code arrange the symbols in decreasing order of their probabilities.

77	70	35	30	25	14	11
F	B	C	A	G	D	E

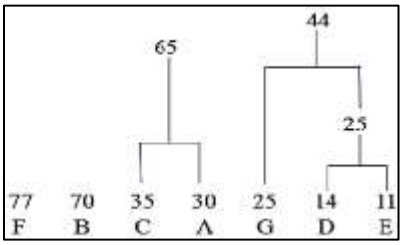
2) Step 2: In second step of building a Huffman code we take two least-frequent characters and logically grouped them together, and then their frequencies are added. In our example, the D and E characters have grouped together and we have combined frequency are 21:



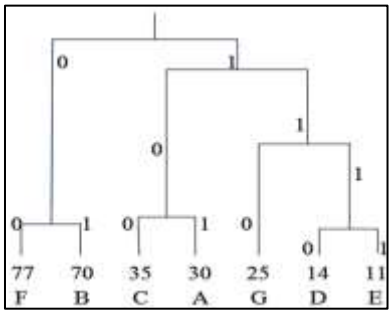
This begins the construction of a "binary tree" structure. Now we again select the two elements with the lowest frequencies, and the lowest frequency is D-E combination and G. we group them together and add their frequencies. This is new combination of frequency 44:



Continue in the same way to select the two elements with the lowest frequency, group them together, and then add their frequencies, until we reach to all elements and remains only one parent for all nodes which is known as root. In third iteration, the lowest frequency elements are C and A:



3) Step 3: In third step we do labeling the edges from each parent to its left child with the digit 0 and the edge to right child with 1. The code word for each source letter is the sequence of labels along the path from root to leaf node representing the letter. Now final binary tree will be as follows:



Tracing down the tree gives the "Huffman codes", with the shortest codes assigned to the character with greater frequency shown in figure 6:

F:	**00**
B:	**01**
C:	**100**
A:	**101**
G:	**110**
D:	**1110**
E:	**1111**

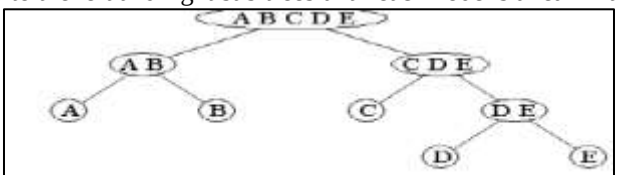
Fig. 6: Huffman codes with shortest codes

The Huffman codes won't get confused in decoding. The best way to see that this is so is to envision the decoder cycling through binary tree structure, guided by the encoding bits it reads, moving top to bottom and then back to the top.

B. Shannon Fano Coding

This is one of an earliest technique for data compression that was invented by Claude Shannon and Robert Fano in 1949.In this technique, a binary tree is generated that represent the probabilities of each symbol occurring. The symbols are ordered in a way such that the most frequent symbols appear at the top of the tree and the least likely symbols appear at the bottom. The algorithm for Shanon-Fano coding is:

1) Parse the input and count the occurrence of each symbol.
2) Determine the probability of occurrence of each symbol using the symbol count.
3) Sort the symbols according to their probability of occurrence, with the most probable first.
4) Then generate leaf nodes for each symbol.
5) Divide the list in two while keeping the probability of the left branch roughly equal to those on the right branch.
6) Prepend 0 to the left node and 1 to the right node codes.
7) Recursively apply steps 5 and 6 to the left and right sub trees until each node is a leaf in the tree.



		Step 1		Step 2		Step 3	
Char	*f*	S	C	S	C	S	C
A	24	24	0	24	00	-	-
B	12	36	0	12	01	-	-
C	10	26	1	10	10	-	-
D	8	16	1	16	-	16	110
E	8	8	1	8	-	8	111

Fig. 7: Shannon Fano Coding

C. Run Length Encoding

Data often contains sequences of identical bytes. Replacing these repeated byte sequences with the number of occurrences, a reduction of data can be achieved. This is known as Run-length Encoding. RLE basically compresses the data by reducing the physical size of a repeating string of characters. This repeating string is called a run which is typically encoded into two bytes where the first byte represents the total number of characters in the run and is called the run count and it replaces runs of two or more of the same character with a number which represents the length of the run which will be followed by the original character and single characters are coded as runs of 1. The blank (space) character in text is such a symbol; single blanks or pairs of blanks are ignored. RLE is useful where redundancy of data is high or it can also be used in combination with other compression techniques also.

1) Example of RLE:

- Input: YYYBBCCCCDEEEEEERRRRRRRRRR
- Output: 3Y2B4C1D6E10R

The drawback of RLE algorithm is that it cannot achieve the high compression ratios as compared to another advanced compression methods, but the advantage of RLE is that it is easy to implement and quick to execute thus making it a good alternative for a complex compression algorithm.

Generally, Shannon-Fano coding does not guarantee the generation of an optimal code. Shannon – Fano algorithm is more efficient when the probabilities are closer to inverses of powers of 2

D. Arithmetic Coding

Arithmetic encoding is the most powerful compression techniques. This converts the entire input data into a single floating point number. A floating point number is similar to a number with a decimal point, like 4.5 instead of 41/2. However, in arithmetic

coding we are not dealing with decimal number so we call it a floating point instead of decimal point. Let's take an example we have string:

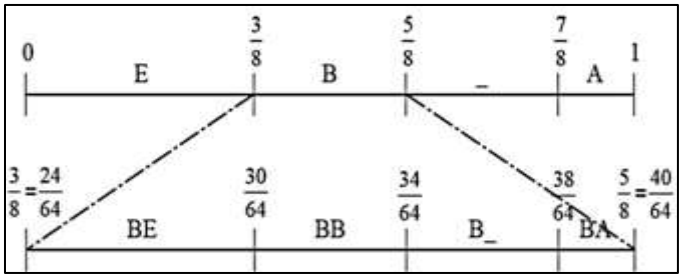
BE_A_BEE

And we now compress it using arithmetic coding.

1) Step 1: in the first step we do is look at the frequency count for the different letters:

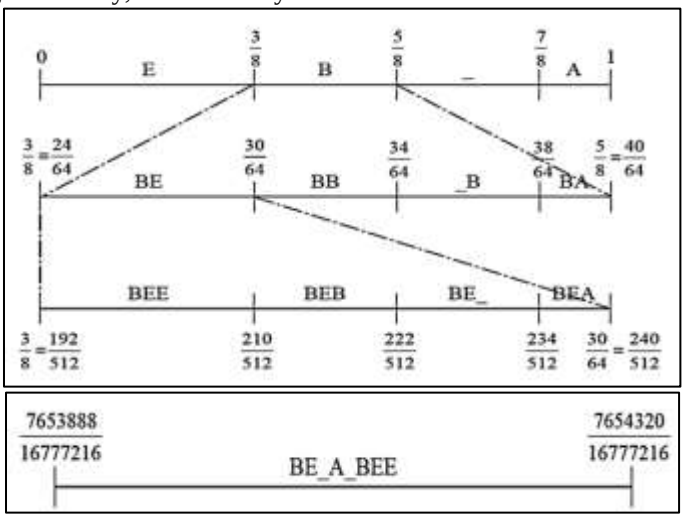
E	B	_	A
3	2	2	1

2) Step 2: In second step we encode the string by dividing up the interval [0, 1] and allocate each letter an interval whose size depends on how often it count in the string. Our string start with a 'B', so we take the 'B' interval and divide it up again in the same way:



The boundary between 'BE' and 'BB' is 3/8 of the way along the interval, which is itself 2/3 long and starts at 3/8. So boundary is 3/8 + (2/8) * (3/8) = 30/64. Similarly the boundary between 'BB' and 'B' is 3/8+ (2/8) * (5/8) = 34/64, and so on.

3) Step 3: In third step we see next letter is now 'E', so now we subdivide the 'E' interval in the same way. We carry on through the message and, continuing in this way, we eventually obtain:



So we represent the message as any number in the interval [7653888/16777216, 7654320/16777216]

REFERENCES

[1] Introduction to Data Compression, Khalid Sayood, Ed Fox (Editor), March 2000.
[2] Ken Huffman. Profile: David A. Huffman, Scientific American, September 1991, pp. 54–58.
[3] Cormak, V. and S. Horspool, 1987. Data compression using dynamic Markov modeling, Comput. J., 30: 541–550.
[4] Cleary, J., Witten, I., "Data Compression Using Adaptive Coding and Partial String Matching", IEEE Transactions on Communications, Vol. COM-32, No. 4, April 1984, pp 396-402.
[5] Mahoney, M., "Adaptive Weighting of Context Models for Lossless Data Compression", Unknown, 2002.
[6] P. Kumar and A.K Varshney, " Double Huffman Coding " International Journal of Advanced Research in Computer Science and Software Engineering (IJARCSSE) Volume 2, Issue 8, August 2012
[7] S.Shanmugasundaram and R. Lourdusamy, " IIDBE: A Lossless Text Transform for Better Compression " International Journal of Wisdom Based Computing, Vol. 1 (2), August 2011
[8] M. Sharma, "Compression using Huffman Coding " IJCSNS International Journal of Computer Science and Network Security, VOL.10 No.5, May 2010
[9] K. Rastogi, K. Sengar, "Analysis and Performance Comparison of Lossless Compression Techniques for Text Data" International Journal of Engineering Technology and Computer Research (IJETCR) 2 (1) 2014, 16-19
[10] A.J Mann, "Analysis and Comparison of Algorithms for Lossless Data Compression" International Journal of Information and Computation Technology, ISSN 0974-2239 Volume 3, Number 3 (2013), pp. 139-146.
[11] D. S. Taubman and M. W. Marcellin, JPEG2000: Image Compression Fundamentals, Practice and Standards, Kluwer Academic Publishers, Massachusetts, 2002.
[12] A. Singh and Y. Bhatnagar, " Enhancement of data compression using Incremental Encoding" International Journal of Scientific & Engineering Research, Volume 3, Issue 5, May-2012