Lossless Data Compression Methodologies

Aayushi Agrawal ; Abhi Tanwar & Akshay Vyas
Dronacharya College of Engnn

Abstract:

Data compression involves encoding information using fewer bits than the original representation. Compression can be either lossy or lossless. The process of reducing the size of a data file is popularly referred to as data compression, although its formal name is source coding because coding is done at the source of the data before it is stored or transmitted. Data compression is important application in the area of file storage and distributed system because in distributed system data have to send from and to all system. So for speed and performance efficiency data compression is used. There are number of different data compression methodologies, which are used to compress different data formats like text, video, audio, image files. The paper presents various methodologies of data compression for lossless data. Huffman and arithmetic coding are compare according to their performances.

Keywords:
Data compression; lossy compression; lossless compression; Huffman Coding; Run Length Encoding ; Arithmetic Encoding

1. INTRODUCTION

Compression is useful because it helps reduce resource usage, such as data storage space or transmission capacity. Because compressed data must be decompressed to use, this extra processing imposes computational or other costs through decompression; this situation is far from being a free lunch. Data compression is subject to a space–time complexity trade-off. For instance, a compression scheme for video may require expensive hardware for the video to be decompressed fast enough to be viewed as it is being decompressed, and the option to decompress the video in full before watching it may be inconvenient or require additional storage. The design of data compression schemes involves trade-offs among various factors, including the degree of compression, the amount of distortion introduced *e.g.*, when using lossy data compression, and the computational resources required to compress and uncompress the data. 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 methods can be classified in several ways. One of the most important criteria of classification is whether the compression algorithms remove some part of data which cannot be recovered during decompression. 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. The lossy data compression algorithm is usually use when a perfect consistency with the original data is not necessary after decompression. Example of lossy data compression is compression of video or picture data. Lossless data compression is used in text file, database tables and in medical image because law of regulations.Various lossless data compression algorithm have been proposed and used. Some of main techniquesare Huffman Coding, Run Length Encoding, Arithmetic Encoding and Dictionary Based Encoding. In this paper we examine Huffman Coding and Arithmetic Encoding and give compression between them according to their performances.

2. DATA COMPRESSION

The theoretical background of compression is provided by information theory (which is closely related to algorithmic information theory) for lossless compression and rate–distortion theory for lossy compression. These areas of study were essentially forged by Claude Shannon, who published fundamental papers on the topic in the late 1940s and early 1950s. Coding theory is also related. The idea of data compression is deeply connected with statistical inference. There is a close connection between machine learning and compression: a system that predicts the posterior probabilities of a sequence given its entire history can be used for optimal data compression (by using arithmetic coding on the output distribution) while an optimal compressor can be used for prediction (by finding the symbol that compresses best, given the previous history). This equivalence has been used as justification for data compression as a benchmark for "general intelligence".Data compression can be viewed as a special case of data differencing: Data differencing consists of producing a difference given a source and a target, with patching producing a target given a source and a difference, while data compression consists of producing a compressed file given a target, and decompression consists of producing a target given only a compressed file. Thus, one can consider data compression as data differencing with empty source data, the compressed file corresponding to a "difference from nothing." This is the same as considering absolute entropy (corresponding to data compression) as a special case of relative entropy (corresponding to data differencing) with no initial data.

2.1 Lossy Data compression

Lossy data compression is the converse of lossless data compression. In these schemes, some loss of information is acceptable. Dropping nonessential detail from the data source can save storage space. Lossy data compression schemes are informed by research on how people perceive the data in question. For example, the human eye is more sensitive to subtle variations in luminance than it is to variations in color. JPEG image compression works in part by rounding off nonessential bits of information.[9] There is a corresponding trade-off between preserving information and reducing size. A number of popular compression formats exploit these perceptual differences, including those used in music files, images, and video. Lossy image compression can be used in digital cameras, to increase storage capacities with minimal degradation of picture quality. Similarly, DVDs use the lossy MPEG-2 Video codec for video compression. In lossy audio compression, methods of psychoacoustics are used to remove non-audible (or less audible) components of the audio signal. Compression of human speech is often performed with even more specialized techniques; speech coding, or voice coding, is sometimes distinguished as a separate discipline from audio compression. Different audio and speech compression standards are listed under audio codecs. Voice compression is used in Internet telephony, for example audio compression is used for CD ripping and is decoded by audio players.There are two basic lossy compression schemes: (1) in lossy transform codecs, samples of picture or sound are taken, chopped into small segments, transformed into a new basis space and quantized. The resulting quantized values are the entropy coded. (2) inlossy predictive codecs, previous and/or subsequent decoded data is used to predict the current sound sample or image frame. The error between the predicted data and real data, together with any extra information needed to reproduce the prediction is then quantized and coded.

2.2 Lossless Data compression

Lossless data compression algorithms usually exploit statistical redundancy to represent data more concisely without losing information, so that the process is reversible. Lossless compression is possible because most real-world data has statistical redundancy. For example, an image may have areas of colour that do not change over several pixels; instead of coding "red pixel, red pixel, ..." the data may be encoded as "279 red pixels". This is a basic example of run-length encoding; there are many schemes to reduce file size by eliminating redundancy.The Lempel–Ziv (LZ) compression methods are among the most popular algorithms for lossless storage.[6] DEFLATE is a variation on LZ optimized for decompression speed and compression ratio, but compression can be slow. DEFLATE is used in PKZIP, Gzip and PNG. LZW (Lempel–Ziv–Welch) is used in GIF images. Also noteworthy is the LZR (Lempel-Ziv–Renau) algorithm, which serves as the basis for the Zip method. LZ methods use a table-based compression model where table entries are substituted for repeated strings of data. For most LZ methods, this table is generated dynamically from earlier data in the input. The table itself is often Huffman encoded (e.g. SHRI, LZX). A current LZ-based coding scheme that performs well is LZX, used in Microsoft's CAB format.The best modern lossless compressors use probabilistic models, such as prediction by partial matching. The Burrows–Wheeler transform can also be viewed as an indirect form of statistical modelling. The class of grammar-based codes are gaining popularity because they can compress highly repetitive text, extremely effectively, for instance, biological data collection of same or related species, huge versioned document collection, internet archives, etc. The basic task of grammar-based codes is constructing a context-free grammar deriving a single string. Sequitur and Re-Pair are practical grammar compression algorithms for which public codes are available.In a further refinement of these techniques, statistical predictions can be coupled to an algorithm called arithmetic coding. Arithmetic coding, invented by JormaRissanen, and turned into a practical method by Witten, Neal, and Cleary, achieves superior compression to the better-known Huffman algorithm and lends itself especially well to adaptive data compression tasks where the predictions are strongly context-dependent. Arithmetic coding is used in the bi-level image compression standard JBIG, and the document compression standard DjVu. The text entry system Dasher is an inverse arithmetic coder.

3. Huffmann coding for data compression

A Huffman Coding is more sophisticated and efficient lossless data compression technique. In Huffman Coding the characters in a data file are converted to binary code. And in this technique the most common characters in the file have the shortest binary codes, and the least common have the longest binary code. To check Huffman Coding‟s work we assume that we have a text file and we have to compress it through Huffman Coding, the characters in the file have the following frequencies shown in figure 1:

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

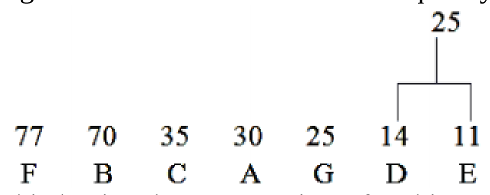
Fig1: Frequencies characters in the file

We have characters from A to G and corresponding frequencies.

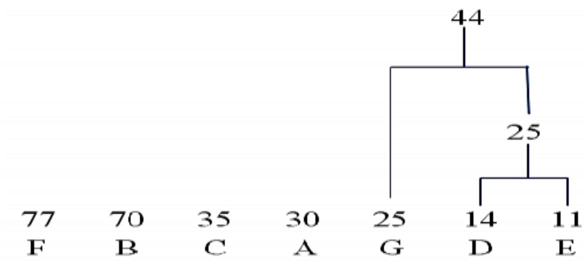
Step 1: In first step of building a Huffman Code order the characters from highest to lowest frequencies of occurrence as follows:

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

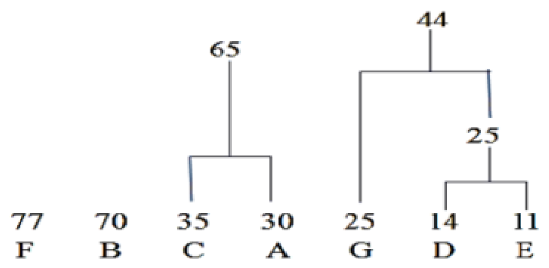
Step 2: In second step of building a Huffman code we take two least-frequent characters and logically grouped them together, and then their frequencies are added. In our example, the D and E characters have grouped together and we have combined frequency are 21:



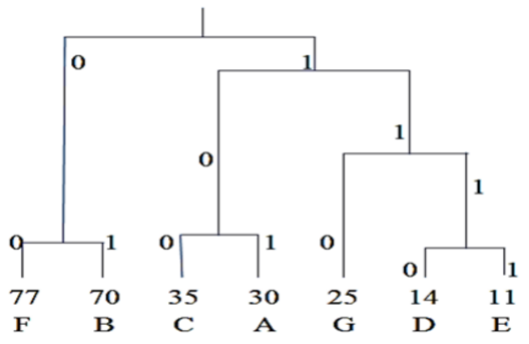
This begins the construction of a "binary tree" structure. Now we again select the two elements with the lowest frequencies, and the lowest frequency is D-E combination and G. we group them together and add their frequencies. This is new combination of frequency 44:



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:



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 2:

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

Fig 2: Huffman codes with the 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.

4. Arithmetic Encoding for Data Compression

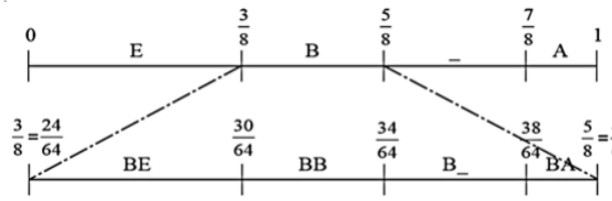
Arithmetic encoding is the most powerful compression techniques. This converts the entire input data into a single floating point number. A floating point number is similar to a number with a decimal point, like 4.5 instead of 41/2. However, in arithmetic coding we are not dealing with decimal number so we call it a floating point instead of decimal point. Let‟s take an example we have string:

BE_A_BEE

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

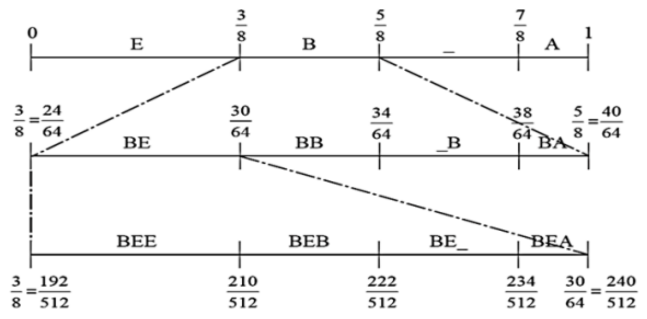
E	B	_	A
3	2	2	1

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 starts with a ‘B’, so we take the ‘B’ interval and divide it up again in the same way:

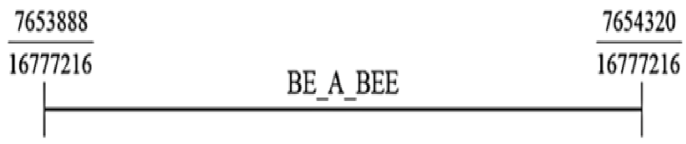


The boundary between ‘BE’ and ‘BB’ is 3/8 of the way along the interval, which is itself 2/3 long and starts at 3/8. So boundary is 3/8 + (2/8) * (3/8) = 30/64. Similarly the boundary between ‘BB’ and ‘B_’ is 3/8+ (2/8) * (5/8) = 34/64, and so on.

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:



and continuing in this way, we obtain:



So we represent the message as any number in the interval

[7653888/16777216, 7654320/16777216]

However, we cannot send numbers like 7654320/16777216 easily using computer. In decimal notation, the rightmost digit to the left of the decimal point indicates the number of units; the one to its left gives the number of tens: the next one along gives the number of hundred, and so on.

So

7653888 = (7*106) + (6*105) + (5*104) + (3*103) + (8*102) + (8*10) + 8

Binary numbers are almost exactly the same, only we deal with powers of 2 instead of power of 10. The rightmost digit of binary number is unit (as before) the one to its left gives the number of 2s, the next one the number of 4s, and soon.

So

110100111 = (1*28) + (1*27) + (0*26) + (1*25) + (0*24) + (0*23) + (1*22) + (1*21) + 1

= 256 + 128 + 32 + 4 + 2 + 1 = 423 in denary (i.e. base 10).

5. Conclusion

Performance measure is used to find which technique is good according to some criteria. Depending on the nature of application there are various criteria to measure the performance of compression algorithm. When measuring the performance the main thing to be considered is space efficiency and the time efficiency is another factor.Since the compression behavior depends on the redundancy of symbols in the source file, it is difficult to measure performance of compression algorithm in general. The performance of data compression depends on the type of data and structure of input source. The compression behavior depends on the category of the compression algorithm: lossy or lossless. In this paper we have find out that arithmetic encoding methodology is very powerful over Huffman encoding methodology. In comparison we came to know that compression ratio of arithmetic encoding is better. And furthermore arithmetic encoding reduces channel bandwidth and transmission time.

6. References

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

2 Burrows M., and Wheeler, D. J. 1994. A Block-Sorting Lossless Data Compression Algorithm. SRC Research Report 124, Digital Systems Research Center.

3 Ken Huffman. Profile: David A. Huffman, Scientific American, September 1991, pp. 54–58.
4Blelloch, E., 2002. Introduction to Data Compression, Computer Science Department, Carnegie Mellon University.

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

6 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.

7 Mahoney, M., "Adaptive Weighting of Context Models for Lossless Data Compression", Unknown, 2002.