Digital Technologies, 2016, Vol. 2, No. 1, 9-13
Available online at http://pubs.sciepub.com/dt/2/1/2
© Science and Education Publishing
DOI:10.12691/dt-2-1-2

Noise Reduction in Data Communication Using Compression Technique

Brian E. Usibe[*], **Donatus E. Bassey, Julie Ogbulezie**

Department of Physics, University of Calabar, Nigeria
*Corresponding author: brianonics@yahoo.com

Abstract Noise is an ever present phenomenon while dealing with recording devices, be it digital or analog, be it specks in images or background hiss in music recordings. Therefore, this paper aims at ways of reducing the effects of these forms of noise on image and sound files that can be compressed and sent through a communication channel. This noise reduction mechanism is represented in the form of algorithms and combined with data compression to minimize the effects of noise, while images and sounds are transmitted with their original content undistorted.

Keywords: *data compression, noise reduction, compression algorithm, image smoothening, quantization*

Cite This Article: Brian E. Usibe, Donatus E. Bassey, and Julie Ogbulezie, “Noise Reduction in Data Communication Using Compression Technique.” *Digital Technologies*, vol. 2, no. 1 (2016): 9-13. doi: 10.12691/dt-2-1-2.

1. Introduction

Data Compression is the encoding of data to save storage space or transmission time. It is also a process of reducing the electronic space (data bits) used in representing a piece of information, by eliminating the repetition of identical sets of data bits (redundancy) in an audio, video, graphic or text file. White spaces in text and graphics, large blocks of the same color in pictures, or other continuously recurring data are reduced or eliminated by coding (encryption) with a program that uses a particular type of compression algorithm [7].

The same program is used to decompress (decrypt) the data so that it can be heard, read, or seen as the original data. Compression ratios of 1:10 to 1:20(or even much higher with emerging technologies) are achieved with common types of data, resulting in much smaller storage requirements or much faster communications.

Designing and selecting data compression schemes involves tradeoffs among factors such as the degree of compression, the amount of distortion introduced (for lossy compression schemes) and the computational resources required to compress and uncompress the data [3].

1.1. Data Compression Schemes

Morse Code, invented in 1838 for use in telegraphy, is an early example of data compression, based on using shorter code words for letters such as 'e' and ’t’ that are common in English. Modern works on data compression started in the late 1940s with the development of information theory [12].

In 1949, Claude Shannon and Robert Fano devised a systematic way to assign code words based on probabilities of blocks. An optimal method of doing this was developed by Huffman in 1951. Early implementations were typically done in hardware, with specific choice of code words being made as compromises between compression and error correction. In the mid-1970s, the idea emerged of dynamically updating code words based on actual data encountered using Huffman coding, online storage of text files also became common and software compression programs began to be developed almost all based on Huffman coding.

In 1977, Abraham Lempel and Jacob Ziv suggested the basic idea of a pointer based encoding, following further work by Terry Welch in 1978; the so called LZW algorithm rapidly became a method of choice for most general purpose compression systems, which was applied in modems.

In the late 1980s, digital images became more common and standards for compressing them emerged. In early 1990s lossy compression methods also began to be widely used [12].

2. Subject

Noise is an ever present part of all communication systems. Any receiver must contend with noise [8]. Noise can be defined as a spurious voltage of random nature with little or no periodicity. Noise is random undesirable electrical energy that enters the communication medium and interferes with the transmitted message. However, some noise is produced at the receiver. In analog systems, noise deteriorates the quality of the received signal, e.g. the appearance of ‘snow’ on the TV screen, or ‘static’ sounds in audio transmission. In digital communication systems, noise degrades the throughput (the average rate of successful message delivery over a communication channel) because it requires retransmission of data packets or extra coding to recover the data in the presence of errors [8].

There are two types of data compression techniques; lossless compression and lossy compression techniques.

2.1. Lossy Compression

Lossy compression is a type of data compression technique in which some data is deliberately discarded to achieve massive reductions in the size of the compressed file [2]. Higher compression ratios are achieved with lossy compression, than when compared with lossless compression technique. Lossy compression is mostly applied in graphic files in which the loss of data is not noticeable, a typical example is the Joint Photographic Experts Group (JPEG) compression scheme used for images [5].

2.2. Lossless Compression

Lossless data compression is a class of data compression algorithms that allows the original data to be perfectly reconstructed from the compressed data. Lossless compression is used in cases where it is important that the original and decompressed data be identical or where deviations from the original data could be harmful [10].

Typical examples are executable programs, text documents, as well as source code of some image file formats like Portable Network Graphics (PNG) that use only lossless compression.

Most lossless compression programs do two things in sequence, the first step generates a statistical model for the input data and the second step uses this model to map input data into bit sequences in such a way that frequently encountered data (probable data) will produce shorter output than improbable data.

3. Methods

3.1. Quantization

In lossy compression, quantization is at the basis of all compression schemes. Quantization is the process of representing the output of a source of a large alphabet with a small alphabet. It can also be defined as restricting a variable quantity to discrete values rather than to a continuous set of values [5]. It is a many-to-one mapping and is therefore irreversible. In the field of data compression, quantization is used in two ways:

1.) If the data to be compressed are in the form of large numbers, quantization is used to convert them to small numbers. Small numbers occupy less space than large ones. On the other hand, small numbers generally contain less information than large ones, so quantization results in lossy compression.

2.) If the data to be compressed are analog (i.e a voltage that changes with time), quantization is used to digitize it into small numbers.

There are two major ways of quantization:

i) Scalar Quantization: It is denoted by y= Q(x). It is the process of a quantization function (Q) to map a scalar (one-dimensional) input value (x) to a scalar output value [1].

ii) Vector Quantization: It is general idea of mapping a multidimensional space into a smaller set of messages S'. Vector quantization is typically implemented by selecting a set of representatives from the input space and then mapping all other points in the space to the closest representative.

3.2. Lossless Scheme

Lossless compression methods maybe categorized according to the type of data they are designed to compress, no lossless compression algorithm can efficiently compress all possible data. For this reason, many algorithms are designed either with a specific kind of data in mind, or with specific assumptions about what kinds of redundancy the uncompressed data are likely to contain. Some of the lossless compression schemes are:

a. HUFFMAN CODING-TEXT

Huffman Coding is an entropy encoding algorithm used for lossless data compression. It takes advantage of the disparities between frequencies & uses less storage for the frequently occurring characters at the expense of having to use more storage for each of the more rare characters [13].

Huffman Coding is the ideal compression technique used to compress text files for transmission. A case study, let us consider the below text or string of characters;

h a p p y h i p h o p

Now, we need to single out each of the characters and their number of occurrences as shown in Table 1 below:

Table 1. Huffman Encoding (showing Frequency of character occurrence)

C H A R A C T E R	F R E Q U E N C Y
P	4
H	3
S p a c e	2
A	1
I	1
O	1
U	1

Now we start building the encoding tree

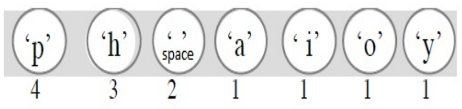

Figure 1a. Code tree for Huffman's code

Now, we choose the two smallest nodes. It could be in any order, it doesn't matter which two we pick, and combine them into a new tree whose root is a sum of the two nodes.

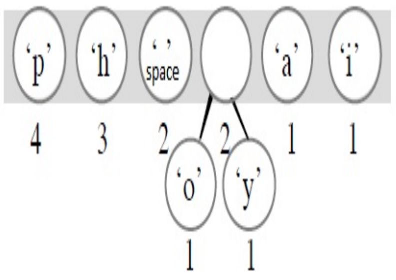

Figure 1b. First tree root of Huffman's code

Now we pick out the remaining nodes and combine

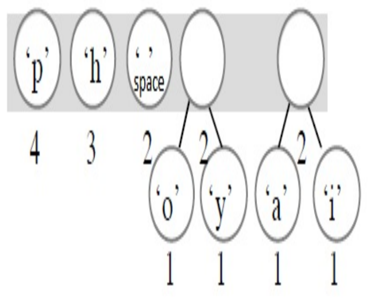

Figure 1c. Second tree root of Huffman's code

Again we pull out the two smallest nodes and build a tree of weight "4"

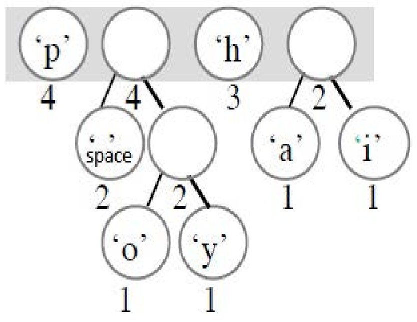

Figure 1d. Weight "4" root of Huffman's code

Further combination results to:

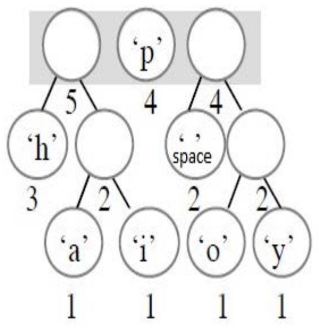

Figure 1e. Combined Weights "4"

Combining the two 4's gives us a tree of character weight "8".

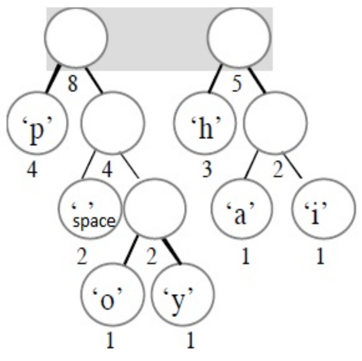

Figure 1f. Tree character for weight "8" Huffman's code

Finally, we combine the last two to get our final tree. The root node of the final tree will always have a weight equal to the number of characters in the input file.

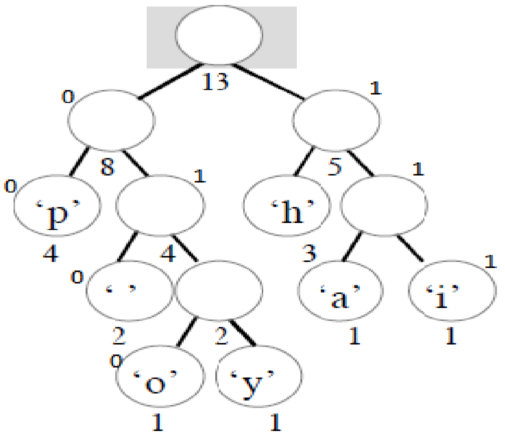

Figure 1g. Huffman's encoded character

From the last sketch, we assign bits to each of the nodes (excluding the root node) a zero (0) to the left and a 1 to the right. So the root node is 13, just the same as the number of characters in the text.

happy hip hop = 13 characters

Table 2. Huffman Encoding (Showing Codes of Character)

C H A R A C T E R	C O D E
P	0 0
H	1 0
S p a c e	0 1 0
A	1 1 0
I	
O	
Y	

Happy hip hop = *10110 00000 111010101110010011000* = **31 bits of data**

For this text to be encoded, it has to be interpreted in bits. We have achieved that in 31 bits of text as shown above. ASCII encoding would have taken 8 × 13 bits, that is 104 bits.

It is essential to be noted that the same tree must be used to do both encoding and decoding of your files. Since each Huffman tree creates a unique encoding of a particular file, you need to ensure that your decoding algorithm generates the exact same tree, so that you can get back the file [13].

b. LEMPEL-ZIV-WELCH CODING

LZW has its roots in the work of Jacob Ziv and Abraham Lempel in 1977, when they published a paper on "sliding window compression and followed it with another paper in 1978 on "dictionary" based compression. These algorithms were named LZ77 and LZ78, respectively. Then in 1984, Terry Welch made a modification to LZ78, which became very popular and was dubbed LZW coding [7].

Many text files have certain strings that repeat very often, for example "the"(with the spaces) takes 5 digits or 40 bits to encode. If we were to add the above 5 strings to the list of (conventional ASCII) characters after the last one at 256, so that whenever we encounter "the" we replace it with 256, compression takes place [10]. This is exactly the approach that LZW compression takes. It starts with a "dictionary" of all the single character with indices 0-255. It then starts to expand the dictionary as information gets sent through. Subsequently, redundant strings will be coded as a single bit and compression is said to have occurred. Below is the basic algorithm:

```
set
w= NIL
loop
read a character k
if wk exists in the dictionary
w=k
else
output the code for w
add wk to the dictionary
w=k
end loop
```

How it works:

The program reads one character at a time. If the code is in the dictionary, then it adds the character to the current work string and waits for the next one. If the work string is not in the dictionary (when the second character comes across) it adds the work string to the dictionary and sends over the work string without the new character, and then it sets the work string to the new character. The table below shows the compress process of LZW method of compression.

For decompression, the algorithm goes thus :

```
read a character k
output k
w=k
loop
read a character k
entry =dictionary entry for k
output entry
add w +first character of entry to the dictionary
w=entry
end loop
```

The nice thing is that the de-compressor builds its own dictionary on its side that matches exactly that of the compressor, so that only the codes need to be sent. The table below shows the decompression process.

Table 3. Decompression process using LZW

Input Codes: / W E D 256 E 260 261 257 B 260 T				
Input/NEW_CODE	OLD_CODE	STRING/Output	CHARACTER	New table entry
/	/	/		
W	/	W	W	256=W
E	W	E	E	257=WE
D	E	D	D	258=ED
256	D	/W	/	259=D
E	256	E	E	260=/WE
260	E	/WE	/	261=E/
261	260	E/	E	262=/WEE
257	261	WE	W	263=E/W
B	257	B	B	264=WEB
260	B	/WE	/	265=B
T	260	T	T	266=/WET

Note:
1.) The "base" dictionary makes use of ASCII character encoding.
2.) LZW compression replaces strings of characters with single codes. It doesn't do any analysis of the incoming text. Instead, it just adds any new character it sees to a table of strings. Compression occurs when a single code is output instead of a string of characters.
3.) It takes the stream of codes output from the compression algorithm and uses them to exactly recreate the input stream. The table can be built exactly as it was during compression using the input stream as data.

LZW compression is commonly used for (lossless) image compression. High compression ratios are achieved when LZW compression is used for images with plain solid colors. Compression ratios are lesser when the image has different gradients or layers. It is efficient in saving disk space while working and handling image files, it is also applied in Photoshop for saving images: a user has the option of saving as tiff (image) files with or without LZW compression.

3.3. Image Smoothening

A low pass filter is the basis for most smoothening methods. An image is smoothed by decreasing the disparity between pixel values by averaging nearby pixels. The algorithm makes an attempt to determine whether the actual differences in pixel values constitute noise or real photographic details and whether to average out the noise, or to preserve the photographic details [6].

One goal in image smoothening is to reduce the noise in such a way that the final image is still discernible. The simplest approach to image smoothening is by replacing each pixel by the average of the neighboring pixel values. From the Figure 2 below, we have a noisy image, if we smooth the pixels, we get Figure 3. Further smoothening will give us what we have in Figure 3.

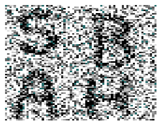

Figure 2.

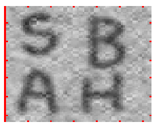

Figure 3.

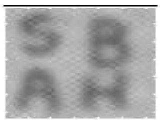

Figure 4.

Source: [11]

From Figure 4, much of the spotty noise has been muted out. On the downside, the sharp boundaries that make up the letters have been smeared due to excess averaging. While other sophisticated approaches exist, the goal is to minimize the noise, and to keep the real images sharp. The trick is not to do much, and also to know when to stop.

4. Result

Combining image quantization of lossy compression & image smoothening in one code, we can successfully compress an image file and smoothen it. The smoothening makes the noise less visible. This method of smoothening (unweight sliding - average smooth) replaces each pixel in the image with the average of 'm' adjacent pixels where 'm' is a positive integer called smooth width and it is always an odd number.

Below is a typical algorithm for smoothening images for compression:

```
001- Import the image.
002- Create a window and display original image.
003- Create another window and display the image as a surface.
004- Smooth the image with the SMOOTH function, which uses the average value of each group of pixels affected by any specified kernel (or smooth width e.g. 3×3, 5×5) applied to the image.
005- Create another window to display the smoothed image as a surface.
006- Create another window and display the smoothed image.
007- GO TO 002
008- ELSE end.
```

5. Conclusion

Noise reduction in communication and data compression are two different concepts, but could be complementary in data communication as data compression techniques have been shown to reduce noise. Noise reduction algorithms have been imbibed into data compression techniques such that as data (image or sound) is compressed, to be encoded, the noise reduction scheme is applied before it is sent through the communication channel.

Quantization Lossy compression technique and Smoothening of images can strip off unnecessary features (including noise from images and sound clips respectively) from data files, lowering the file sizes & bandwidth it would have consumed. This algorithm is set to perform the double functions of compressing specific data type (image or sound) and subsequently reduce traces of noise found in the data. This algorithm is a model that can be applied in multimedia studios, telecommunication industries, etc.

References

[1] Belloch, E.G. (2013). Introduction to Data Compression. Computer Science Department, Carnegie Mellon University. [Online]. Available: http://blellochcs.cmu.edu. [2013 , August 17].

[2] Haas, J (2013). Lossy Compression: About.com Guide. [Online]. Available:http://linux.about.com/cs/linux101/g/Lossy_Compression.htm. [2013, August 15].

[3] Hodgson, J. (2010). Understanding Records: A Field to Recording Practice. p.86.

[4] Karla, K.P. Data Compression Techniques for E-Learning. E-Learning Division, Department Of Information. Technology, Ministry of Communications & Information Technology, Government of India [cited 2013 June 20].

[5] Khalid, S. (2005). Introduction to Data Compression, Third Edition. Morgan Kaufmann Series: Pg 432.

[6] Leslie S., D. Z .Richard (1995). The Focal Encyclopedia of Photography. Focal Press: Pg 507.

[7] Nelson, M. (1989). LZW Data Compression. [Online]. Available: http://www.eg.bucknell.edu/~cs379/Algorithms/S01/lzw.html. [2013, August 13].

[8] Niknejad, A.M. Noise in Communication Systems. UC Berkeley 2013.

[9] Rouse, M. (2005) What is audio noise? [Online]; Available from URL: http://whatis.techtarget.com/definition/audio-noise. (2013, July 22).

[10] Salomon, D. and Bryant, D., G. Motta.(2009) Handbook of Data Compression. 5th Edition. Springer. Pg 16-18

[11] Sethian, J.A. (2010). Noise Removal in Images. [Online]. Available: http://math.berkeley.edu/~sethian/2006/Applications/ImageProcessing/noiseremoval.html. [2013 , Feb 15].

[12] Wolfram, S.: A New Kind of Science (Wolfram Media 2002) Pg. 1069 Retrieved from http://www.wolframscience.com/refrences/1069b

[13] Zelenski, J. (2012). Data Compression and Huffman Encoding. Handout 31: Pg 3-9.