Data Compression: Challenging the traditional algorithms and Moving one step closer to Entropy

Girik Malik
Battelle Center for Mathematical Medicine,
The Research Institute at Nationwide Children's Hospital,
The Ohio State University College of Medicine,
Columbus, OH, USA
Labrynthe, New Delhi, India

Abstract—Compression algorithms reduce the redundancy in data representation to decrease the amount of storage required. Data compression offers an attractive approach to reducing communication costs by using available bandwidth effectively. Over the last decade there has been an unprecedented explosion in the amount of digital data transmitted via the Internet, representing text, images, video, sound, computer programs, etc. With this trend expected to continue, there is a need for algorithms which can most effectively use available network bandwidth by maximally compressing data. It is also important to consider the security aspects of the data being transmitted while compressing it, as most of the text data transmitted over the Internet is vulnerable to a multitude of attacks. This paper is focused on addressing the problem of lossless compression of text files with an added security.

Lossless compression researchers have developed highly sophisticated approaches, such as Huffman encoding, arithmetic encoding, the Lempel-Ziv (LZ) family, Dynamic Markov Compression (DMC), Prediction by Partial Matching (PPM), and Burrows-Wheeler Transform (BWT) based algorithms. However, none of these methods have been able to reach the theoretical best-case compression ratio consistently, which implies a need for better algorithms. One approach for trying to attain better compression ratios is to develop new compression algorithms. An alternative approach, however, is to develop intelligent, reversible transformations that can be applied to a source text that improve an existing algorithm's ability to compress and also offer a sufficient level of security of the transmitted information. The latter strategy has been explored here. In this paper, we present a method of encoding the data into images and back and compare it with the compression of RAR and ZIP.

Index Terms—Data Compression, Security, Huffman Coding, Lempel-Ziv (LZ), Algorithm, Compression Ratio, RAR, ZIP

1 Introduction

THERE has been a lot of excitement in the field of Data Compression and everybody has been trying to reach the maximum limit of Data Compression, i.e., the entropy of Data [7]. To this day, nobody knows the single best technique for Data Compression as compression is highly data specific, some algorithms work best on some kind of data while the others work best on some other kind of data. People have also attempted to solve the problem by using various tools like *deflopt* and *zipmix* in order to create a hybrid combination of data compression techniques, by picking the best compression algorithms for different files and then combining them together into a single file which has maximum compression. One such example that we came across had a file of 700MB compressed to around 50MB. The image files also are internally compressed, using both lossy compression algorithms (for JPEG, WebP) and lossless algorithms (for PNG,PSD, RAW,

- *E-mail: girikmalik@gmail.com , malik.192@osu.edu*
- *Web: http://gmalik9.tk/*

TIFF) [3]. In this paper we change our approach of looking at how the data can be compressed and move one step further to achieve the data compression which is better than that achieved by widely used *RAR* and *ZIP* algorithms. We provide a method of lossless data compression by compressing the data into images (TIFF and PNG).

2 Existing Compression Algorithm

2.1 ZIP Archive

ZIP(.zip) is an archive file format that supports lossless data compression. The format was originally created in 1989 by Phil Katz, and was first implemented in PKWARE, Inc.'s PKZIP utility[6]

2.1.1 Design

.ZIP files are archives that store multiple files. .ZIP allows contained files to be compressed using many different methods, as well as simply storing a file without compressing it. Each file is stored separately, allowing different files in the same archive to be compressed using different methods. Because the files in a .ZIP archive are compressed individually it is possible to extract them, or add new ones, without applying compression or decompression to the entire archive. This contrasts with the format of compressed tar files, for which such random-access processing is not easily possible.

2.1.2 Structure

A .ZIP file is correctly identified by the presence of an end of central directory record which is located at the end of the archive structure in order to allow the easy appending of new files. If the end of central directory record indicates a non-empty archive, the name of each file or directory within the archive should be specified in a central directory entry, along with other metadata about the entry, and an offset into the .ZIP file, pointing to the actual entry data. This allows a file listing of the archive to be performed relatively quickly, as the entire archive does not have to be read to see the list of files. The entries within the .ZIP file also include this information, for redundancy, in a local file header. Because zip files may be appended to, only files specified in the central directory at the end of the file are valid. Scanning a ZIP file for local file headers is invalid (except in the case of corrupted archives), as the central directory may declare that some files have been deleted and other files have been updated.

2.1.3 Compression Method

The .ZIP File Format Specification documents the following compression methods: Store (no compression), Shrink, Reduce (levels 1-4), Implode, Deflate, Deflate64, bzip2, LZMA (EFS), WavPack, and PPMd. The most commonly used compression method is **DEFLATE**, which is described in IETF RFC 1951[4].

Compression methods mentioned, but not documented in detail in the specification include: PKWARE Data Compression Library (DCL) Implode, IBM TERSE, and IBM LZ77 z Architecture (PFS). A "Tokenize" method was reserved for a third party, but support was never added.

2.1.4 Encryption

.ZIP supports a simple password-based symmetric encryption system which is documented in the .ZIP specification, and known to be seriously flawed. In particular it is vulnerable to known-plaintext attacks which are in some cases made worse by poor implementations of random number generators.[8]

New features including new compression and encryption (e.g. AES) methods have been documented in the .ZIP File Format Specification since version 5.2. A WinZip-developed AES-based standard is used also by 7-Zip, Xceed, and DotNetZip, but some vendors use other formats.[1] PKWARE SecureZIP also supports RC2, RC4, DES, Triple DES encryption methods, Digital Certificate-based encryption and authentication (X.509), and archive header encryption.[2]

3 The New Algorithm

The new algorithm(pkd) supports lossless data compression and works by converting the data into image

3.1 Design

The new algorithm allows the files to be compressed using many different methods and also has the security feature built into it. The first method is by converting the data into the corresponding RGB values and then storing them onto a figure. The second method works by converting the data into its corresponding floating point value and then storing it as a float image, where every pixel holds the floating point value. The third method is by converting the file into another described format called GIPA(.gipa) format, which adds another layer of encryption for security and then stores in into the image using the two aforementioned methods.

3.2 Structure

The image, by default, has to be two dimensional and can be easily represented by a two-dimensional array or a list of lists. This however does not apply to the data and the data need not be present in the same format or can be easily re-structured. So it becomes necessary to structure the data in the form of a matrix. One such method used here is by adding additional bits to the data in order to make it reach the closest integer size for the two-dimensional array to be complete.

3.3 Encryption

The new algorithm supports the encryption to the fullest as the data generated after the compression is not any ordinary text data, its a photograph which cannot be easily encoded using the normal Pattern Recognition Algorithms as the data after the pattern recognition will again be encrypted by either the normal existing techniques or by another unique technique called GIPA(.gipa). GIPA currently depends on the addressing of the characters in the file using a variable-length dictionary, which is not fixed and changes dynamically with the file.

4 The conversion to RGB

The first method of compression is the conversion of the Data to RGB. In this method, the data is read character by character and stored into an array where each element of the array represents a single data character. The elements are then converted to their corresponding ASCII Value. In the next step, the data, in form of ASCII values, is clubbed into tuples of size 3, where each entry of the tuple corresponds to the individual Red, Green and Blue values of the RGB respectively. This reduces the data by almost $\frac{1}{3}^{rd}$ of the original size in terms of the tuples. The tuples are then stored to the image file pixel by pixel. It can also be written as a matrix directly, reducing the computation time.

4.1 Analysis

As previously mentioned, the reshaping of the data into a matrix of Red, Green and Blue values, reduces the size to almost $\frac{1}{3}^{rd}$ of the original data. There is no such best image format for holding the data but in general there is a race between JPEG(.jpeg or .jpg) and PNG(.png), where one outperforms the other based on the type of data. Then comes GIF(.gif), following which is TIFF(.tiff or .tif). In this case, the top performing ones, i.e., the JPEG and PNG file formats outperform both 7ZIP(.7z) and RAR compression whereas the second best of them is unable to perform better than ZIP archive which follows the best one and is followed by the second best image format. Another interesting thing to mention would be the comparison of various colours of the pixel and their respective file sizes. A block of 146x150 pixels was taken for each of the colours and it was found that the block with Green colour occupied the least space while the one with Red colour occupied most space. Results in Table 1

5 Conversion to Float Values

The second method of compression is the conversion of the Data to Floating point values and then storing it in an image of 32-bit floating point [5]. The data in this case need not be

TABLE 1
COMPARISON OF VARIOUS COLOURS ON BLOCK OF SIZE 146x150 PIXELS

Colour of Block	**Size** ***(in bytes)***
Green	453
Blue	454
White	513
Black	515
Red	516

grouped into tuples of size 3 and only needs to be reshaped into a matrix. Floating point image here should not be confused with High Dynamic Range (HDR) Imaging[9].

5.1 Analysis

The analysis for this particular algorithm was done with a matrix of 3000x3000 floating point numbers (upto 8 places of decimal) and it can hold data correctly upto 16 places of decimal. The image file format for storage in this case is TIFF as the other formats are inefficient for holding floating point values but the figure can be converted to GIF if required and can still be mapped to the original image. The compression achieved in this case is much better and faster for large data than the traditional ZIP, RAR and 7Z, in which 7Z is the best but slowest of them all.

6 Results

The book *A Brief History of Time* by Stephen Hawking was appended nine time, one after another, in text format along with images. The images used here were not in the jpeg format, they were as well compressed by encoding them to *base64*, a method generally used for viewing images over the internet. The file size came out to be 12.6MB.

Another file was converted into GIPA format, sized 113KB. The plain text file of which was sized 18KB and contained the text *Hi this is just an IT Project* 624 times.

The results are not fixed for any particular file format and different file formats provide different compressions. In some cases JPEG proves better, while in some other cases PNG provides a better compression. Another inherent feature in this method is data security. The images can be displayed publicly and distributed freely, people can easily see the images without any additional softwares but cannot decode it easily. It is one of the few algorithms that provide both encryption as well as compression. The results are compared below.

6.1 RGB Method

The book, *A Brief History of Time* appended nine times, one after another was converted first. The best compression was achieved by JPEG while the worst performance was that of TIFF, the other ones being in between. When compressed the same file using the existing algorithms of RAR and ZIP, the best compression was achieved by 7Z while the worst was that of ZIP.

Further, the conversion of GIPA file of plain text, as mentioned above, was best compressed in JPEG, while the worst performance was that of TIFF. It is quite evident that there are some repeating patterns in the figures of the GIPA files, which can be further compressed using some standard pattern recognition algorithms and combining them with machine learning. In both the cases, the conversion to JPEG gave the best compression, even better than ZIP and RAR, but the conversion to JPEG comes with an added overhead of loosing a few bits of information due to its inner structure and how it is stored on the disk. Even leaving out the lossy storage of JPEG, the second best storage of PNG gives a significant amount of compression which is comparable and in some cases even better than the existing algorithms. The results are summarised in Tables 2 and 3. The compressed photographs of the book are given in Figure 1 and 2. The results for the compressed GIPA file are given in Figure 3 and 4. The GIF and TIFF of both could not be attached because of restrictions in LaTeX.

6.2 Float Method

The Float method of compression was applied to a random matrix of dimension 500x500 con-

TABLE 2
COMPARISON OF VARIOUS METHODS OF COMPRESSION APPLIED TO THE BOOK *A BRIEF HISTORY OF TIME*

Type of File	**Size** *(in kilobytes)*
Original	13000
JPEG	1309
GIF	4893
PNG	10915
TIFF	12970
7Z	876
RAR	916
ZIP	8423

TABLE 3
COMPARISON OF VARIOUS METHODS OF COMPRESSION APPLIED TO THE GIPA FILE FORMAT

Type of File	**Size** *(in kilobytes)*
Original	113
JPEG	6
PNG	22
GIF	44
TIFF	114
7Z	18
RAR	33
ZIP	34

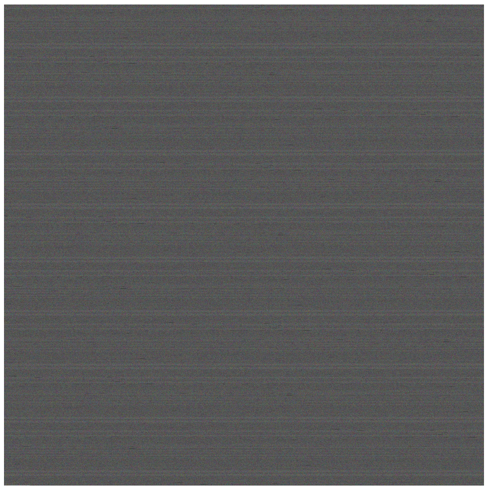
Fig. 1. Book Compressed in JPEG

sisting of floating point numbers upto 12 places of decimal. The numbers when written to a file in raw format occupy 5127KB while the compression algorithm described, compresses it to 976KB in TIFF format. The floating point images could be compressed to only TIFF format as the other formats cannot hold data in floating point numbers. The numbers taken in the matrix were in the range 0,1, but the method has also been tested for other greater

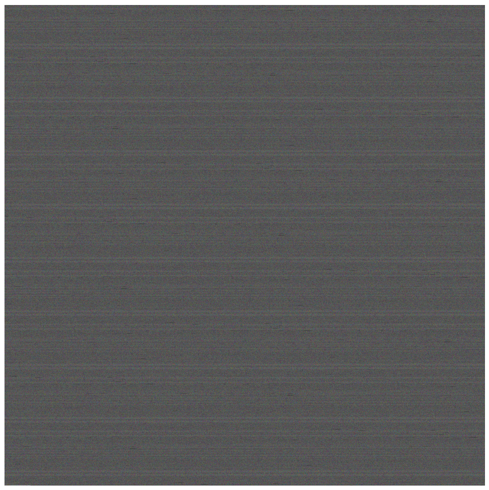
Fig. 2. Book Compressed in PNG

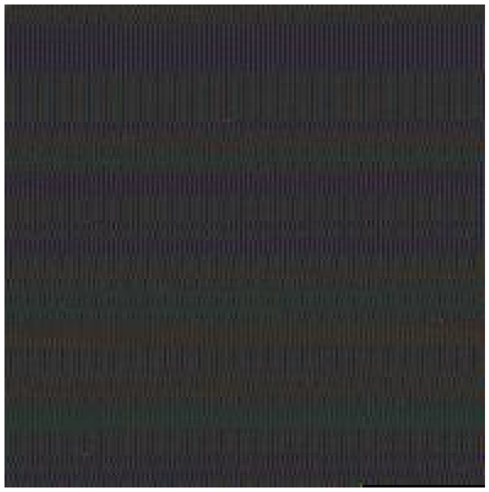
Fig. 3. GIPA file Compressed in JPEG

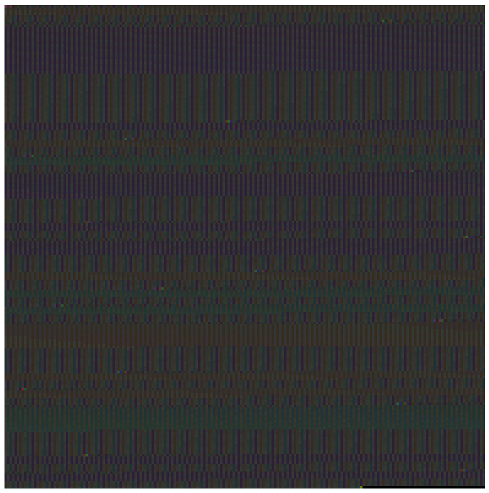
Fig. 4. GIPA file Compressed in PNG

numbers. The new algorithm pkd clearly out-

TABLE 4
COMPARISON OF VARIOUS METHODS OF COMPRESSION APPLIED TO THE FLOATING POINT MATRIX OF 500x500

Type of File	**Size** *(in kilobytes)*
Original	5127
TIFF	976
7Z	2021
RAR	2223
ZIP	2180

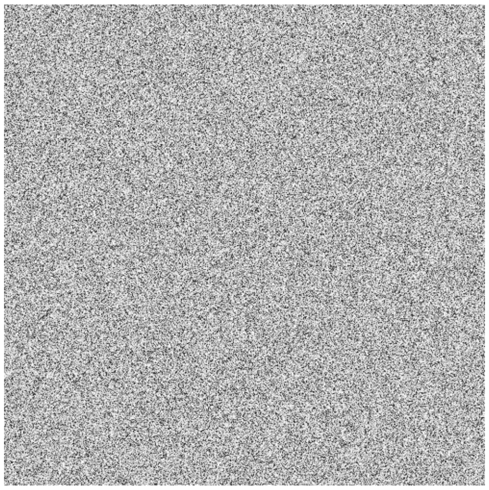
Fig. 5. Floating Point Matrix of 500x500 compressed in TIFF (Image in JPEG)

performs the existing algorithms by a large margin. The results are summarized in Table 4 and the image(converted to JPEG because of LaTeX graphics issues) is shown in Figure 5.

7 Bitwise Compression Algorithm

Another half developed algorithm presented here is the Bitwise Compression Algorithm. The algorithm is not yet fully developed, and needs more research and some advanced mathematics applied to it, but a basic idea is presented here. The idea is inspired from the Electrical Domain, keeping multiplexers in mind. Multiplexers are electronic devices which have 2^n inputs ($n \neq 0$) and n selection bits. The selection bits are given binary equivalent inputs, called the selection bits and the output is chosen according to the logic of selection bits.

The algorithm described here is inspired from the idea of selection bits of a Multiplexer. The idea is to maintain a single string of n bits for some or all the data. The data is then decoded according to the selection bits given. The selection string is applied to the longer string with some logical or other operation to get back the particular part of data, which could generally be a word, a character or a small string. We are still quite far from achieving such a compression as it is mathematically very complex and requires furthere developments in coding theory, that can be applied in order to come up with a generalized algorithm.

A small example is described here for compressing sequential numbers or some numbers in a particular range. If the starting number is 235 which is 11101011 in binary, we can start our selection bit from 0 (00000 in binary), then by subtracting 1(00001 in binary), we get 234 in decimal and adding one we get 236. Maintaining the number 235 in integer on any standard machine would have required 4 Bytes (24bits) and then maintaining a 1 in integer requires another 4 Bytes, in total we would have required 48bits, which using this algorithm is reduced to only 13bits.

Using a 5bit selection, we can produce upto 32 numbers on both sides, achieving significant compression. We might as well increase the number of bits and store even bigger numbers. A small demonstration is shown in Table 5

8 Conclusion

Comparing the new algorithm(pkd) with the already existing compression algorithms, used traditionally, the algorithm pkd described here is clearly found to be better both in terms of reduction in size as well as speed. The algorithm also gives as added advantage of data security due to its multi-level nature. The images produced by the algorithm can be open on any computer or mobile device without the requirement of any additional specific software. Another added advantage that comes with the algorithm is that of Data Analysis. The algorithm can easily be used for finding patterns in the data, something similar to the already existing heat maps. One such demonstration has already been done by adding the same data after one another and an emergent pattern could be seen from it.

TABLE 5
DATA COMPRESSION USING BITWISE COMPRESSION ALGORITHM BY PERFORMING BITWISE SUBTRACTION STARTING FROM 235

Selection Bits		**Number**	
Binary	**Decimal**	**Binary**	**Decimal**
00000	0	11101011	235
00001	1	11101010	234
00010	2	11101001	233
00011	3	11101000	232
00100	4	11100111	231
00101	5	11100110	230
00110	6	11100101	229
00111	7	11100100	228
01000	8	11100011	227
01001	9	11100010	226
01010	10	11100001	225
01011	11	11100000	224
01100	12	11011111	223
01101	13	11011110	222
01110	14	11011101	221
01111	15	11011100	220
10000	16	11011011	219
10001	17	11011010	218
10010	18	11011001	217
10011	19	11011000	216
10100	20	11010111	215
10101	21	11010110	214
10110	22	11010101	213
10111	23	11010100	212
11000	24	11010011	211
11001	25	11010010	210
11010	26	11010001	209
11011	27	11010000	208
11100	28	11001111	207
11101	29	11001110	206
11110	30	11001101	205
11111	31	11001100	204

APPENDIX A PSEUDOCODES

A.1 Reshaping Data

The reshaping of the data, as described in the main paper can be done as follows:

//Assuming the data has already been read, is stored in a list in tuples or otherwise and length calculated

Step 1: *Calculate square-root of the length of the list (or number of tuples)*

Step 2: *Take its integer part in a variable and assign the next integer to another variable*

Step 3: *Take their product in a third variable*

Step 4: *Calculate the difference between the product and the length of the list*

Step 5: *If the difference is less than zero, then increment the first variable(the integer part of square root) by 1*

Step 6: *Calculate the new product of the variables in the third variable again*

Step 7: *Append the zeros(or any other character) equal to the difference of product and length of the list*

The list can now be easily reshaped into the two variables

A.2 Converting Data to RGB

The Data to RGB conversion as described in the paper

Step 1: *Open the file to be read*

Setp 2: *Read the data from the file and store it in an array or list character wise*

Step 3: *Take the elements from the array in groups of 3 and club them to form a tuple*

Step 4: *Add additional bits(or tuples) to the existing list of tuples in order to reshape the matrix (as described in A.1)*

Step 5: *Create an empty image of the size of the matrix in A.1*

Step 6: *Put the data from the matrix on to the image (pixel wise or otherwise)*

Step 7: *Save the image on to the disk*

A.3 Converting Data to Float

The Data to 32-bit Floating Point Image conversion as described in the paper

Step 1: *Open the file to be read*

Setp 2: *Read the data from the file and store it in an array or list character wise*

Step 3: *Add additional bits to the existing list of values in order to reshape the matrix (as described in A.1)*

Step 4: *Create an empty image of the size of the matrix in A.1, this time of float type*

Step 5: *Put the data from the matrix on to the image (pixel wise or otherwise)*

Step 6: *Save the image on to the disk*

REFERENCES

[1] Aes encryption information: Encryption specification ae-1 and ae-2. *http://www.winzip.com/aes_info.htm/*.

[2] Application note on the .zip file format. *http://www.pkware.com/support/zip-app-note/*.

[3] 2011 Common Image File Format Comparisons. https://socialcompare.com/en/comparison/image-file-formats. 2011.

[4] L Peter Deutsch. Deflate compressed data format specification version 1.3. 1996.

[5] Pillow (PIL fork). Read the docs. *http://pillow.readthedocs.org/en/latest/reference/ImageMath.html/*.

[6] Phillip Katz. Computer software pioneer, 37. *The New York Times*, 2009.

[7] Claude Elwood Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 1948.

[8] Michael. Stay. Zip attacks with reduced known plaintext. *http://math.ucr.edu/ mike/zipattacks.pdf*.

[9] Wikipedia. High dynamic range imaging. *http://en.wikipedia.org/wiki/High-dynamic-range_imaging/*.