International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 4 Issue 03, March-2015

Various Approaches to Achieve Data Compression

Guide – Prof. Swati Ringe, Fr. C. R. C. E. Bandra, University of Mumbai

Mr. Dylan Andrades
Department of Computer Engineering
Fr. Conceicao Rodrigues College of Engineering
University of Mumbai
Mumbai, India

Mr. Hardik Agrawal
Department of Computer Engineering
Fr. Conceicao Rodrigues College of Engineering
University of Mumbai
Mumbai, India

***Abstract*— With the ever increasing growth seen in the field of computing, processing large sized files has become possible. The network technologies which are bound by the constraints involving the physical transfer medium have difficulties transferring such large sized files. This research is aimed at exploring various methods of data compression and implementing those methods. These techniques also provide a certain level of security to the compressed file. There are five image-based approaches and one statistical approach. These approaches usually convert any file into binary string, perform the compression operation on the binary string and then convert the binary string back to a file (compressed file). The reverse procedure is followed at the decompression side.**

Keywords*— Data compression, image-based compression, statistical redundancy removal

I. INTRODUCTION

Data compression algorithms [1], [2], [3] are designed to reduce the size of data so that it requires less space for storage and less bandwidth for transmission over communication channels of limited bandwidth. An additional benefit of compression is that it decreases the size of information to be transmitted, hence minimizing the errors. It can be of two types:

1. Lossless Data Compression, where the algorithm usually exploit statistical redundancy to represent data more concisely without losing information, so that the process is reversible. Though generally used in compression of text-based data, some lossless compression algorithms [5], [6] and standards [4] are also popular for image compression.
2. Lossy Data Compression, which is the converse of Lossless Data Compression. In such schemes, some loss of information is acceptable. Dropping non-essential details from the data source can save storage space.

The basic principle of compression involves transforming the data contained in a file into a format which requires lesser storage space than the original form. For efficient compression we need to implement an excellent data structure and an efficient algorithm to compress and decompress the source and compressed data respectively, thus optimizing the trade-off between the size reduction and the computation time.

The techniques explained in this article has a different touch to it. Various possibilities of converting a normal file to image file and applying compression to those image files are explored. Existing image compression algorithms have not been used to compress those image files. The compression algorithms are developed to provide lossless compression in images.

In this article, we introduce 6 different approaches to achieve compression. The first 5 approaches are image-based compression and the last approach is based on removing statistical redundancy in data. The basic idea behind compressing a file was to convert any file to its corresponding binary, applying the compression technique on the binary file and save the compressed binary as a compressed file. On the decompression end, read the compressed file and express it as binary, apply decompression algorithm on binary and convert the decompressed binary to the original file.

II. APPROACH – 1

The idea behind this approach is to create multiple matrices and fill it with the binary string obtained after conversion of the file to binary. These matrices can be represented as multiple images. This approach aims to superimpose all those images obtained from all the matrices and create one single image which will be very much smaller in size.

While theoretically implementing this algorithm, it was noted that adding bits in matrices was easy, but extracting the same bits from addition was complicated.

Example 1. If 0+1+1 = 2, then how do you identify whether 2 = 011 or 101 or 110

III. APPROACH – 2

In this approach, we divide the binary string in 24-bit strings, which are later represented as a 24-bit integer. Now, each 24-bit integer is used to represent an RGB color, where bits 16-23 represent the red component, bits 8-15 represent the green component and bits 0-7 represent the blue component.

These 24-bit strings are converted as RGB color and stored in a matrix. This matrix is then converted into a color image.

Example 2. The file used for testing is Vuze.exe which is 72008 bytes file. The following image was obtained after compression.

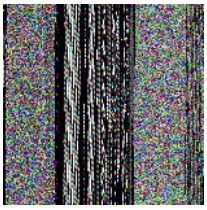

Figure 1 - Vuze.exe compressed using Approach 2

This compressed image is 60061 bytes file. It can be seen that the file was compressed up to 11947 bytes, which is about 16.59% compression. It was later noted that the approach did not work for larger files. This was because the file size of the image does not remain constant. The compression percent decreases as the file size goes on increasing.

IV. APPROACH – 3

The problem in Approach 1 was that the extraction of bits from the summation of bits was difficult. As a solution to that problem, it was decided that we create a set of rules to extract the bits from the summation.

In this approach, we define color schemes for 2 layers, where each bit {0,1} is colored.

Example 3. Consider the binary string "00010011". We write it in a matrix form as:

$$M1 = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix} \qquad M2 = \begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix}$$

Now, we define color scheme C1 for matrix M1 as

$$C1(0) = (1, 0, 1)$$
$$C1(1) = (1, 1, 0)$$

For matrix M2, we define color scheme C2 as a lighter version of color scheme C1

$$C2(0) = (5, 0, 5)$$
$$C2(1) = (5, 5, 0)$$

When we combine color schemes C1 and C2, we get a new color scheme C3 as follows

$$C3(00) = (6, 0, 6)$$
$$C3(01) = (6, 5, 1)$$
$$C3(10) = (6, 1, 5)$$
$$C3(11) = (6, 6, 0)$$

Combining the matrices M1 and M2 results in

$$M3 = \begin{bmatrix} 00 & 00 \\ 01 & 11 \end{bmatrix} = \begin{bmatrix} (6,0,6) & (6,0,6) \\ (6,5,1) & (6,6,0) \end{bmatrix}$$

This matrix is then converted to image.

At the decompression end, we subtract the obtained color from color scheme C2

$$(6, 0, 6) - (5, 0, 5) = (1, 0, 1)$$
$$(6, 0, 6) - (5, 5, 0) = (1, -5, 6)$$

Since, the color $(1, -5, 6)$ goes out of range, it is not valid. Hence, the decoded string will be 00, where both the colors are matched in C1 and C2.

The main idea behind this approach was to compress a file hierarchically, i.e. if the basic binary creates 16 matrices, then two color schemes will convert 2 matrices into 1 matrix and thus resulting in 8 matrices. This method will be followed till we get 1 matrix in the end, which will result in one image of very small size.

The problem with this approach was that maintaining and storing all color schemes was tedious. Also, after a certain level, the colors get exhausted after creating several schemes.

V. APPROACH – 4

In this approach, the binary file is split into S-bit string each. Then we create a matrix of size M, where

$$M = \sqrt{\frac{length\ of\ binary}{S}}$$

We convert each S-bit binary string into an integer and then we spread it in a color.

Example 4. Consider an S-bit integer as 36854.

$$R = (36854\ \%\ 256) = 246$$
$$G = \left(\left(\frac{36854}{256}\right)\%\ 256\right) = 143$$
$$B = \left(\left(\frac{36854}{256^2}\right)\%\ 256\right) = 0$$

Now, we can represent the S-bit number 36854 as a color component (246, 143, 0). With the help of these colors, we create an image.

At the decompression end,

$$S\ bit\ number = (R) + (256 * G) + (256^2 * B)$$

All the S-bit numbers are converted to strings and concatenated in proper order, giving us the original binary string, which is then converted to original file.

In case, the color exceeds (256*256*256), then we create multiple images, i.e. creating a new matrix.

Example 5. The file Vuze.exe of 72008 bytes was used to test this approach.

Figure 2 (a) **Figure 2 (b)**

Figure 2 - Vuze.exe
(a) compressed file – 1
(b) compressed file – 2

The compressed file 1 and 2 combined are 58219 bytes, which shows a compression of about 19.15%. But it was later noted that for larger files, the compression percent decreases.

VI. APPROACH – 5

This approach is a slight modification to Approach 4. The concept used in this approach is Java Advanced Imaging – Multiband Images [7]. These multiband images follow Portable Network Graphics (PNG) standard [4]. In a multiband image, a single image contains multiple bands. So, instead of having 3 separate images, we create 1 image with 3 bands, hoping it would reduce the file size.

But unfortunately, the file size doesn't decrease much. Although, it did help with reducing the number of files.

Example 6. Considering the same file used in Approach 4, the following images were obtained.

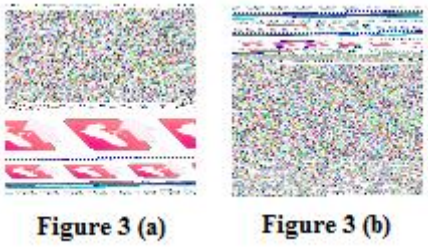

Figure 3 (a) Figure 3 (b)

Figure 3 - Vuze.exe
(a) compressed file – 1
(b) compressed file – 2

The compressed files 1 and 2 combined are of size 54993 bytes, which is 23.63% compression. Although, this holds true that this algorithm works better than Approach 4, it does not solve the problem of Approach 4. Approach 2, 4 and 5 all compress files, but the compression percentage decreases as the file size increases. It was later noticed that this was due to the headers of the image files. Thus, the approach style was migrated from image-based to statistic-based.

VII. APPROACH – 6

This approach is based on a property of ex-or and negation. Ex-or operation is performed on consecutive bits of the binary string. Whenever the result obtained is '1', the next position is stored. These positions hold the significance of negating the bit on the decompressing end. The first bit, last bit, total number of bits and the resultant bit along with the list of positions are sent as a compressed file.

On the decompression end, ex-or operation is performed between the last bit and the resultant bit and result is stored in the second-last bit. Now, if the second-last bit is stored in position list, then we negate the bit and store it as resultant bit. Otherwise, the resultant bit is same as second-last bit. Now, ex-or is performed between resultant bit and second-last bit. This process is done till it reaches the first bit. The result obtained is the track of all resultant bits.

Example 7. Consider an example where the data is {0110110110}

TABLE I. APPROACH 6 – COMPRESSION METHOD

Positions	1	2	3	4	5	6	7	8	9	10
	0	1	1	0	1	1	0	1	1	0
		1	1	↓	↓	↓	↓	↓	↓	↓
3	→	→	0	0	↓	↓	↓	↓	↓	↓
				0	1	↓	↓	↓	↓	↓
					1	1	↓	↓	↓	↓
6	→	→	→	→	→	0	0			
							0	1		
								1	1	
9	→	→	→	→	→	→	→	→	0	0
										0

First Bit = Last Bit = 0
Resultant Bit = 0
Total Bits = 10
Positions = 3, 6, 9

At the decompressing end, the information above is used to retrieve the original data back.

TABLE II. APPROACH 6 – DECOMPRESSION METHOD

		↓			↓			↓	
1	2	3	4	5	6	7	8	9	10
0	1	~~0~~	0	1	~~0~~	0	1	~~0~~	0
0	1	1	0	1	1	0	1	1	0

While implementing, it was observed that the positions need to be efficiently stored. Best case scenario is that no bits ever change. But, that would mean that all the data bits are 0, which is impossible. Worst case scenario is that if all the bits get changed, sending all the bit positions would be expanding the file rather than compressing it.

VIII. Statistical Analysis

The following table provides the statistics and comparison of all the approaches mentioned above.

TABLE III. Analysis of all the approaches

No.	Disadvantages	Compression Ratio
#1	Decompression could not be achieved	NA
#2	• Compression Ratio is irregular. • Technique is static. • Padding of bits is necessary when binary string is not a multiple of 24.	70.3 kB = 16.59 % 2.98 MB = 1.28 %
#3	• Compression Ratio irregular. • Storing of colour schemes at sender requires complex data structures. • Requires high computation time. • Color Scheme is limited to 256x256x256 colors. • Sending the database of color scheme stored in it to the receiver requires separate management.	70.3 kB = -20.07 % 2.98 MB = computationa lly expensive
#4	• Compression Ratio decreases with increase in file size. • This technique is very expensive computationally. • Extra padding needs to be added, which is irregular and varies depending on the distance between all the colours in the matrix.	70.3 kB = 19.15 % 2.98 MB = 1.6 %
#5	• This technique was aimed at reducing the number of extra images created due to padding. • The disadvantages of Approach 4 were still observed. • Anomalous compression ratio.	70.3 kB = 23.63 % 2.98 MB = 0.11 %
#6	• Sending positions in an efficient way is complicated. • More positions being changed means less compression and vice-versa. • Worst case scenario: If the last bit changes, then the file doesn't compress. This case happens often.	70.3 kB = 0% 2.98 MB = 0%

IX. Conclusion

Each of the proposed approaches can be further researched to develop full-fledged algorithms. Appropriate choice of data structures for storing and managing the color schemes for image-based approaches (1-5) has the potential to further improve their performance and decrease the high computational cost. Traditional compression algorithms can also be used along with these approaches in implementation of compression system to reduce the file size and thus help reduce the compression time.

Acknowledgment

We would like to thank Fr. Conceicao Rodrigues College of Engineering for providing the necessary resources. Our special thanks to Mr. Aakash Tiwari and Mr. George Cherian for help and support. We would also like to thank Mr. Sachin Balan for helping us with our research. Lastly, we would like to thank our parents for consistent emotional support.

References

[1] M. Crochemore & W. Rytter, "Text Algorithms," (2010) Available: http://monge.univ-mlv.fr/
[2] 'Universal lossless data compression algorithms' by Sebastian Deorowicz ,Silesian University of Technology ,Faculty of Automatic Control, Electronics and Computer Science, 2003
[3] 'Introduction to Data Compression' by Guy E. Blelloch, Computer ScienceDepartment, Carnegie Mellon University, January, 2013
[4] "ISO/IEC 15948:2004 - Information technology -- Computer graphics and image processing -- Portable Network Graphics (PNG): Functional specification", www.iso.org.
[5] 'Lossless Image Compression Based on Data Folding' by Suresh Yerva, Smita Nair, Krishnan Kutty
[6] Marcelo Weinberger, Gadiel Seroussi and Guilermo Sapiro, "The LOCO-I Lossless Image Compression Algorithm: Principles and Standardization into JPEG-LS," Computer Systems Laboratory, HPL Tech. Rep. 98-193, Nov 1998.
[7] "Programming in Java Advanced Imaging", Release 1.0.1, Sun Microsystems, Nov 1999.