A new technique for data compression

DhyaaAlrahman Latef Thajel* and Kadhim Mahdi Hashim**
University of Thi-Qar, college of education for pure science

*dhyaaalrahman1@gmail.com
**  Kadhimmehdi63@gmail.com

Abstract: Data compression has become more important than ever, due to the increasing demand for internet use and the exchange of a huge amount of images, videos, audio and documents as well as the growing demand for electronic archiving by government departments that produce thousands of documents per day. In this paper, the proposed technique for document compression will be presented. The proposed technique is a lossless and completed technique it is consists of two parts the compression part and decompression part. The compression part contains of some basic stages such as: pre-processing, blocks processing, run length encoding(RLE), replace maximum values by unused values, minimize levels, delta encoding, compression of ones values, encryption. After the encryption process is complete, the outputs are stored in two separate binary files with the extension of bmp; one of them is the header file and is considered a key for the second file which contains the compressed data. This technique applied on twenty documents and compared with other methods compression such as RLE, jpeg, tiff and png. The experimental results showed that the proposed technique gives a higher compression ratio than the rest of the methods.

Keywords: Data compression, run-length encoding (RLE), delta encoding, lossless.

1 Introduction

Data compression is the process of reducing the size of the original data in order to reduce the cost of transport and storage in memory and then return it to its original format when needed. Compression deals with techniques for reducing the storage required to save an image or the bandwidth to transmit it. Particularly in the uses of internet it is very necessary to compress data [1]. The field of image compression continues to grow at a rapid pace. As we look to the future, the need to store and transmit images will only continue to increase faster than the available capability to process all the data. Even with the rapid growth in computer power and the increase in Internet bandwidth, the ability to process and transmit the desired amount of image data continues to be problematic.

According to the comparative study of S Oswal, et al. In 2016 [2] between Deflate compression algorithms and Lempel-Ziv-Welch (LZW) data compression algorithm, the deflate algorithm is efficient in both compression rate as well as the speed at which the compression is done. Run Length Encoding in the worst case RLE generates the output data which is 2 times more than the size of input data. This is due to the fewer amount of runs in the source file.and the files that are compressed have very high values of compression ratio. This algorithm does not provide significant improvement over the original file [3].
2. Background

2.1 Run-length encoding (RLE) [4, 5]

Is one of the simplest forms of data compression methods. The principle of RLE is to exploit the repeating values in a source. This repeating string of characters is called a run. The algorithm counts the consecutive repeating amount of a symbol and uses that value to represent the run. In RLE, runs of data are stored as a single data value and count, rather than as the original run. This is most useful on data that contains many such runs: for example, simple graphic images such as icons, line drawings, and animations.

2.2 Lossless compression

Reduces bits by identifying and removing statistical redundancy. No information is lost in lossless compression. Lossless compression methods reduce size whilst preserving all of the original image information, and therefore without degrading the quality of the data [6]. Some of these techniques are like run length encoding, entropy encoding, Huffman encoding, arithmetic coding, Lempel–Ziv–Welch coding, deflation, chain codes, delta encoding and block coding.

2.3 Delta encoding

Delta encoding represents streams of compressed pixels as the difference between the current value and the previous value [7]. The first value in the delta encoded file is the same as the first value in the original image. All the following pixels in the encoded file are equal to the difference between the corresponding value in the input data, and the previous value in the input data [8].

2.4 BMP file format (.bmp)

The bitmap file format deal with graphic file related to microsoft windows OS. Normally these files are uncompressed so they are large. These files are used in basic windows programming [9]. BMP files always contain RGB data. The file can be 1-bit: 2 colors (monochrome), 4-bit: 16 colors, 8-bit: 256 colors, 24-bit: 16777216 colors, mixes 256 tints of Red with 256 tints of Green and Blue [10].

3. The proposed technique

3.1 Compression part

The proposed compression part consists of some stages as shown in figure (3.1).
3.1.1 Read image

The first step is reading a document image from its location in storage of the computer; the input image should be a binary image with bmp extension. If the image is of a gray or colored gradient type, it will be converted to a binary type.

3.1.2 Preprocessing (block dividing)

After reading the image file, the image is divided into blocks, the block size is selected (8x8) and then dividing the height of the image by the size of the block. If the height of the image is not divisible by the size of the block, we insert some white rows to complete the last partition. Then the image is resizes instead of (height * width) it will be (size of block *(width* number of partitions)). That means partitions are ordered in a sequential way and then the new image is divided into blocks.

3.1.3 Blocks processing

The blocks that contain only ones values (white blocks) are deleted and the rest of the blocks are placed in a matrix and their indexes in another matrix. That means from this step there are two outputs, matrix of non-white blocks and matrix of indexes. Matrix of non-white blocks is converted into a row logical vector while the delta algorithm will be applied to the matrix of indexes.

3.1.4 Apply the delta encoding to the indexes matrix

Delta encoding explained previously is applied on the increasing matrixes; the first pixel in the delta encoded is the same as the first pixel in the original matrix. All the following pixels in the encoded matrix are equal to the difference (delta) between the corresponding value and the previous value in the input matrix.

Example 3.1: Let \( A = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 250, 251, 
252, 253, 254, 255, 256, 257, 258, 259, 260] \).

\( A \) has 30 elements, the maximum value in \( A \) is 260 that takes 9 bits, that means the size of \( A \) is 30*9=270 bits and after applying the delta encoding on \( A \) the result will be:

\( A_1 = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1] \)

The maximum value in \( A_1 \) is 120 that takes 7 bits that means the size of \( A_1 \) is 30*7=210 bits. The result of applying delta encoding on \( A \) is reduced its size from 270 bits to 210 bits.

3.1.5 Compression the ones values

From previous step, it was found that the delta encoding produces a lot of ones and there is no zero in \( A_1 \) (This is because no two addresses have the same value), in this step ones will be compressed by setting zero and then the number of contiguous ones instead of the repetitive ones. The result of applying compression the ones on \( A_1 \) will be : \( A_2 = [0, 10, 110, 0, 9, 120, 0, 9] \). The maximum value in \( A_2 \) is 120 that takes 7 bits and it has 8 elements that mean the size of \( A_2 \) is 8*7=56 bits. There is a reduction in size from 210 bits to 56 bits.

3.1.6 Replace maximum values by unused values

This stage consists of many steps they will be explained in the following.
Step 1: Input the integer matrix: the output matrix from the previous step will be the input matrix in this step.

Step 2: Calculate maximum number of bits are needed and maximum value to represent data

Find the histogram of integer matrix, and find how many numbers (levels) are used in integer matrix. The length of the histogram represents the number of levels; it will be called \( L \). And then find how many bits are need to represent \( L \) by using flowing formula: \( B = \log_2(L) \). Where \( B \) is number of bits are need to represent \( L \).

If \( B \) is a fraction number, it is rounded to the larger integer. So the \((\text{ceil})\) function is used for rounding and the formula will be: \( B = \text{ceil}(\log_2(L)) \). And then calculate maximum value to represent data (max value) by flowing formula: \( \text{max value} = (2^B) - 1 \).

Example (3.2): Let \( A \) input integer matrix, after calculating the histogram of \( A \), the number of levels (the numbers used) was 280. The max bit and max value which in may be represented calculated as follows:

\[
L = 280; B = \log_2(280); B = 8.129 \quad \text{There is no 8.129 bit this value is rounded to 9 by \((\text{ceil})\) function.} \quad B = \text{ceil}(8.129) = 9 \text{ bits; max value} = 2^9 - 1 = 511 \text{ is maximum value to represent data.}
\]

Step 3: Search for unused values (levels) that are smaller than max value

Example (3.3): Let \( A \) input integer matrix and flowing table is represent the histogram information of \( A \).

Table (3.1) histogram information of \( A \)

| NO | 1    | 2    | 3    | 4    | 5    | 6    |
|----|------|------|------|------|------|------|
| Gray scale value | 3    | 4    | 5    | 7    | 510  | 1000 |
| Number of repetitions | 2000 | 1000 | 1000 | 2000 | 200  | 100  |

The values that were in \( A \) are 3, 4, 5, 7, 510, 1000, that mean \( L = 6 \); \( B = \log_2\). \( B = 2.584; B = \text{ceil}(2.584) = 3 \) bits; \( \text{max value} = 2^3 - 1 = 7 \). Searches for unused values that are less than 7 in \( A \), there are four values 0, 1, 2 and 6.

Step 4: Check the conditions

If maximum element in input matrix > max bit and unused numbers \( \neq [] \), then go to next step else go to end without any change. In the example (3.3) the maximum element in input matrix is 1000 greater than 7 and unused numbers \( = [0, 1, 2, 6] \), the condition is met then goes to the next step.

Step 5: replace maximum number in input data by minimum number in unused values

To complete the example (3.3) in this step the maximum value in the entered matrix will be replaced by the minimum unused value as follows: 1000 replaced by 0. This process is repeated as long as the condition in the previous step is met so 510 replaced by 1.

In this case, the number 7 will be the maximum of the entered matrix, which is equal to the max value, not smaller than it, and the condition will not be met and the process will stop. The histogram information of \( A \) will be as follows:
Table (3.2) histogram information of matrix (A) after replacement.

| NO  | 1  | 2  | 3  | 4  | 5  | 6  |
|-----|----|----|----|----|----|----|
| New values | 0  | 1  | 3  | 4  | 5  | 7  |
| Number of repetitions | 100 | 200 | 2000 | 1000 | 1000 | 2000 |

**Step6: Calculate the size of the replaced values plus the new data and the size of the input data**

The replacement process is not suitable for small arrays, and the size of the outputs may be greater than the size of the inputs, but it is very useful for large arrays. In this step, the size of the input will be compared with the size of the outputs to ensure that the replacement works only if there is a reduction in the size of the entered matrix.

To complete the example (3.3) and from the tables (3.1),(3.2) the size of the input and outputs are calculated as follows: A is the input matrix, number of its elements is 6300 and maximum value is 1000, this takes 10 bits so size of input will be 6300x10 = 63000 bits where maximum value after replacement is 7, this takes 3 bits so size of output will be 6300x3 = 18900 bits. And there are two values [1000,510] every one take 10 bits replaced by another values [0,1] every one take 1 bit This means that the cost of storing the replaced values costs 22bit, which is added to the size of the outputs and the comparison is as follows: Size of input: 63000 bits; Size of outputs: 18900 + 22 = 18922 bits; So: Size of input >> Size of outputs. In this case, go to the next step, and the outputs will be produced.

3.1.7 Apply the run length encoding (RLE) to the vector

In this step, run length encoding is applied to the logical vector which produced from the block processing stage. The output of this stage is a decimal vector representing the number of ones and the number of zeros in succession starting from the second position in the output vector where the first location is allocated to a sign in whether the input vector starts with ones or zeros. If it starts with ones, it takes zero and if it starts with zeros, takes one.

3.1.8 Separate one runs from zeros runs in two vectors.

Ones in the binary documents represent the background and zeros that represent the information and when applying the run length encoding ones produce decimal numbers greater than zeros this means the nature of numbers produced from ones differs from the nature of numbers produced from zeros, in this stage they will be separated into two matrices to increase the efficiency of the proposed data compression technique. In fact, the odd position will be separated from even position.

3.1.9 Replace maximum values by unused values

This stage of the compression technique has already been explained in detail. The inputs of this stage will be odd position matrix and even position matrix that were produced from the previous step.

3.1.10 Minimize levels

Take the indexes of large numbers that have few repetitions and then reduce the levels to the middle. This process produces two outputs, the first output is the indexes matrix that is sent to the delta encoding and then compression ones and then the max replace, and the second output is the new matrix after reducing the levels in middle. The size of the inputs is compared to the size of the outputs. If the output size is less than the input size, this process is repeated automatically. The steps for this stage will be illustrated by the following example:
Example (3.4): Let (A) input integer(odd position or even position) matrix after applying max replace steps and following table is represent the histogram information of A.

Table (3.3) histogram information of A

| levels       | 0-7 | 8-15 | 16-31 | 32-63 | 64-127 | 128-137 |
|--------------|-----|------|-------|-------|--------|---------|
| repetitions  | 100000 | 200 | 100 | 50 | 70 | 10 |

To apply minimize levels stage the following steps are followed:

Step1: Calculate (max value), it has already been explained in the max replace stage.

\[ L = 138; B = \text{ceil}(\log_2(138)) = 8; \text{max value} = (2^8) - 1 = 255. \]

Step2: Search for numbers \( > ((\text{max value}-1)/2) \) subtract from \( ((\text{max value} + 1)/2) \) and take their indexes. \( (\text{max value}-1)/2 = (255-1)/2 = 127 \). From the table (3.3), there are 10 numbers that \( > 127 \) let this numbers are existed in following indexes:

| indexes | numbers |
|---------|---------|
| 500     | 128     |
| 1000    | 129     |
| 3000    | 130     |
| 5000    | 131     |
| 10000   | 132     |
| 20000   | 133     |
| 40000   | 134     |
| 60000   | 135     |
| 80000   | 136     |
| 100000  | 137     |

Subtract this numbers from \( ((\text{max bit} + 1)/2) = ((255 + 1)/2) = 128 \), after Subtracting will be as right table:-

| indexes | numbers |
|---------|---------|
| 500     | 0       |
| 1000    | 1       |
| 3000    | 2       |
| 5000    | 3       |
| 10000   | 4       |
| 20000   | 5       |
| 40000   | 6       |
| 60000   | 7       |
| 80000   | 8       |
| 100000  | 9       |

And the histogram information of A will be change as following table:

Table (3.4) histogram information of A after Minimize levels for one iteration

| levels       | 0-7 | 8-15 | 16-31 | 32-63 | 64-127 |
|--------------|-----|------|-------|-------|--------|
| Number of repetitions | 100000 | 202 | 100 | 50 | 70 |

Indexes matrix=[500,1000,3000,5000,10000,20000,40000,60000,80000,100000].

Step3: indexes matrix will be compressed by delta → compression ones → max replace. In this example the indexes matrix after compression will be as following:[500,500,2000,2000,5000,10000,20000,20000,20000,20000]. There has been no change in compression ones step (there is no ones to compression) and there has been no change in max replace part because it is not suitable for small matrixes.

Step4: calculate the size of outputs and inputs. From table (3.3) input matrix A has 100430 elements and maximum value was 137 it is take 8 bits. So size of \( A = 8 \times 100430 = 803440 \) bits. From table (3.4) maximum value of A after minimize levels was 127 it is take 7 bits. So size of output \( A = 7 \times 100430 = 703010 \) bits. And
indexes matrix after compression has 10 elements and maximum value was 20000 it is take 15 bits. So size of indexes matrix after compression = 15x10 = 150 bits. General size of outputs = 150 + 703010 = 703160 bits

**Step5:** Compare the size of the input with the size of the outputs if the outputs size is smaller than input size go to step (1) and repeat until the size of the outputs will be greater than input size then go to end and give the outputs of last iteration. In this example general size of outputs (703160 bits) < size of input (803440 bits). So go to step (1) and repeat all steps for second iteration.

3.1.11 **Convert all output vectors to binary and then encrypted and stored securely**

**Step1:** Convert vectors to binary form and then convert them to logical row vectors.

**Step2:** The row vectors are connected together and the first final file is produced and stored.

**Step3:** Collect header information in one decimal vector.

**Step4:** Convert decimal header vector to binary form and then convert it to logical row vectors.

**Step5:** The header file key is the number of elements of the header file vector before converting from decimal into binary, the key is converted to a binary form and repeated along the header file vector and then the XOR process is applied between the two vectors and the result is stored in separate file.

3.2.2 **Decompression parts:** The decompression part reverses the compression stages completely.

4. Experimental Results

The proposed compression technique was applied to the set of twenty testing images by different sizes of blocks. The best result was by using 8x8 size block and the average CR of proposed compression technique as shown in table (4.1).

| Size blocks(pixel) of 4x4 | 8x8 | 12x12 | 16x16 |
|-------------------------|-----|-------|-------|
| Average of CR            | 12.9| 13.34 | 13.12 | 12.87 |

When using Block 8x8, the average of compression time of the test images was 0.4225 second and average of decompression time was 0.6675 second. The compressed files that were produced by the proposed compression technique are compared with RLE, jpeg, tiff and png, as shown in the table (4.2).

| No  | Orig -nal Size (KB) | CR of compressed files | Propos ed techniq ue |
|-----|---------------------|------------------------|----------------------|
|     | RLE                | jpeg | tiff | png |                     |
| Image 1 | 493                | 1.16 | 0.5  | 3.3 | 3.6 | 3.73               |
| Image 2 | 493                | 1.45 | 0.6  | 4.0 | 4.4 | 4.65               |
| Image 3 | 493                | 2.10 | 0.9  | 7.6 | 7.4 | 7.71               |
After decompression, the totals mean error between decompressed files and original files are calculated and the results of them were zeros.

**Result analysis:** through experimental results, it was observed that the proposed technique gives a higher compression ratio (CR) than other methods because the stages as block processing, replace maximum values by unused values and minimize levels are aims to break down large numbers and convert them into small numbers and represent them with the fewest bits and all the remaining stages aim to reduce data in different ways.

5. Conclusions

1- The results of the proposed technique were compared with different compression methods such as RLE, jpeg, tiff and png.
2- Through experimental results, it was observed that the proposed technique gives a higher compression ratio (CR) than other methods.

3- The files that were decompressed are exactly the same as the original files, because the mean total error is zero.

4- The jpeg method, although it is a lossy, but it gave the worst compression ratio and in some cases produces much larger files than the original files.

5- The bit rate of the proposed technique is the lowest compared to the rest of the methods.

6- The best size for the block used in the proposed technique is 8x8.

7- In addition to the high compression ratio, the proposed technique produces encrypted files securely, which distinguishes it from other methods.

References

[1] Abhishek Yadav & Poonam Yadav, Digital image processing.

[2] S Oswal, A Singh, K Kumari, “Deflate Compression Algorithm”, International Journal of Engineering Research and General Science, January-February, 2016.

[3] A J Maan, “Analysis and Comparison of Algorithms for Lossless Data Compression”, International Journal of Information and Computation Technology, 2013.

[4] F. Semiconductor, “Using the Run Length Encoding Features on the MPC5645S”, 2011.

[5] S. Gaurav Vijayvargiya and R. P. Pandey, “A Survey: Various Techniques of Image Compression”, 2013.

[6] A. H. Hussein, S. Sh. Mahmud, R. J. Mohammed, Image Compression.

[7] M. Dipperstein, “Adaptive Delta Coding Discussion and Implementation, Online available: http://michael.dipperstein.com/delta/index.html.

[8] S. W. Smith, “Data Compression, Sci. Eng. Guid. to Digit. Signal Process, 1997.

[9] Shubham Venayak, Sukhwinder Singh, “A Study of 2-D Image Compression Technique”, International Journal of Enhanced Research in Management & Computer Applications, Vol. 3 Issue 3, March-2014.

[10] Scott E Umbaugh, Digital image processing and analysis, 2011.