Lossless Image Compression Algorithms

Alexandre Leopoldo
2019219929
DEI, FCTUC, Portugal

António Correia
2019216646
DEI, FCTUC, Portugal

João Monteiro
2019216764
DEI, FCTUC, Portugal

***Abstract*—Through the years the need for space has increased, the solution is the compression of data. Compression of data can be lossless, which will be the one that will be explored, and lossy.**

Traditional solutions of lossless compression, such as RLE, Huffman, LZ77 gave rise to the more modern techniques like LZMA, Deflate and Bzip2. In this paper these methods will be explored.

Compression methods will be implemented and compared between each other and with PNG, to elect the best one. It was concluded that the best implementation was Bz2, a Bzip2 data compression algorithm.

Keywords—compression; lossless compression; compression speed; compression ratio;

I. Introduction

The more years we go through, the more data we produce at a faster rate. From a single 50x50 emoji to a 4K 2-hour podcast, every media consumption website/app needs to store its files somewhere and those take up a lot of space. As so, we always have the urge for more storage space but, instead of trying to get more capacity, we can try reducing the size of our files. That's where compression comes in, more specifically image compression, which is the theme of this paper.

Compression techniques can be split into two: lossy and lossless, which will be the main focus of these paper. Lossy algorithms allow for the reduction of the original file size in exchange it will result in the loss of data, therefore lossy techniques will only result in an approximation of the the original file. Lossless compression, as the name suggests, uses algorithms that don't involve the destruction of data, it will guarantee that the information recovered will be the exact same as the original.

The transmitter sends the de compressed information using some type of compression algorithm, the receiver then decompresses the data.

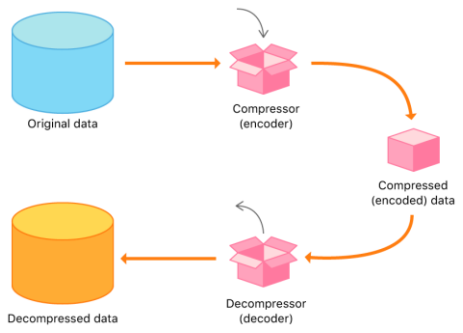


Fig. 1. Compression and Decompression

In this paper it will be explored lossless compression techniques.

II. Tradicional Compression Methods

Modern compression techniques are clever combinations of traditional methods. Therefore, the predominant methods need be taken into consideration before moving into modern solutions. There are types of lossless data compression methods, Entropy type, such as Arithmetic and Huffman, Dictionary type, like LZ77 and LZ78 and other types like RLE. These lossless methods are the following:

A. Run Length Encoding (RLE)

Run length Encoding uses a simple technique to compress data without the loss of information. This method makes use of symbol repetition, pixels in this case, where the pixel value, A, and its occurrences B, are denoted as such: {A, B}. RLE encoding is only effective if there are sequences of 4 or more repeating characters because RLE makes use of three characters to encode so coding two repeating characters would even lead to an increase in file size. RLE doesn't require much computational capacity and can be really effective in data with a lot of repetitive data.[1]

B. Huffman Encoding

Huffman Encoding is a type of optimal prefix code. It is based on the frequency of each character, and creates a binary tree, where the most used character is coded with the shortest binary code and the least used character is coded with the longest binary code. It is good for image but more effective on text since it produces an optimal prefix code.

C. Arithmetic Encoding

Arithmetic coding is a form of entropy encoding which results in a lossless data compression. What makes arithmetic coding very different from other forms of entropy encoding is that, instead of replacing each symbol with a code, arithmetic coding encodes the entire message into a single number, between 0 and 1. The process of encoding is simple. Start by dividing the interval [0,1] into parts, where each part corresponds to the probability of the character. Choose the interval corresponding to the first character of the sequence to be encoded and divide that interval again using the same method and repeat until the sequence is encoded. This type of encoding is much better when used in text, since there aren't too much different characters. When used in an RGB image, the performance of this algorithm is much lower.[2]

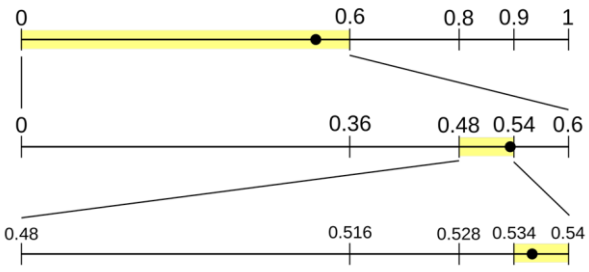


Fig. 2. Arithemetic Enconding example

D. LZ77 Encoding

LZ77 algorithm compresses information by replacing repeated occurrences of sequences of data. The technique makes use of a sliding window, made up of a look-ahead buffer and search buffer. The encoded sequence comes in the form of (offset, length of match, code word). As the sliding window moves it look for matches, if the sequence is already found on the search buffer this sequence is replaces it with its (o, l, c) and the sliding windows will move the length of the sequence + 1, it will do this until it reaches the end of file. One of the main limitations of the LZ77 algorithm is that it uses only a small window for previously seen text, which means it continuously throws away valuable dictionary entries because they slide out of the dictionary. A way to get around this issue is to increase the size of the search and look-ahead buffers. However, changing these parameters will drastically increase the CPU time needed for compression.

E. LZ78 Encoding

LZ78 was created 1 year after LZ77. In this compression algorithm, we make use of a dictionary to help us reduce the size of our file. Basically, the algorithm consists in going through our file symbol by symbol and, if it is already in our dictionary, we concatenate the next symbol and search our dictionary again. We do this until we have a sequence that is not in the dictionary. When that happens, we add it and, on our compressed file, a tuple gets written. This tuple is in the format (I, C), being I the index of the last sequence we found in the dictionary and C the character we added to that same sequence. This method isn't effective for image compression since it was created with the objective of compressing big texts. In the long term it relies on the appearance of similar words or character sets, which is not common in images.

III. Modern Compression Methods

The previous mentioned methods gave birth to the modern compression solutions. There are a lot of candidates of lossless image compression techniques. Some of those are:

A. Deflate

Deflate uses a combination of compression methods. It uses LZ77 to generate the symbols and then Huffman to encode them. A Deflate stream consists of a series of blocks. Each block is preceded by a 3-bit header. The first bit indicates if there are there are more blocks ahead, the second and third are used to indicate the encoding method for that block. Within compressed blocks, if a duplicate series of bytes is spotted (a repeated string), then a back-reference is inserted, linking to the previous location of that identical string instead. After that Huffman is used to replace common symbols with shorter representations and less commonly used symbols with longer representations. One popular use of this algorithm is in PNG.[6]

B. Lempel Ziv Markov Algorithm (LZMA)

LZMA uses a dictionary compression scheme similar to the LZ77 and produces a high compression ratio, while still maintaining a decent decompression speed. It uses a range encoder, to make a probability prediction of each bit. In LZMA compression, the compressed stream is a stream of bits, encoded using an adaptive binary range coder. The stream is divided into packets, each packet describing either a single byte, or an LZ77 sequence with its length and distance implicitly or explicitly encoded.

C. Bzip2

Bzip2 starts by applying RLE (Run-Length Encoding) to our information followed by a Burrow-Wheeler transformation. This transformation consists in taking every circular shift of our information, sorting them by lexicographic order and taking the last symbol of each one of them. After this, it proceeds to a Move-To-Front transformation. In this one, it starts with an empty sequence and a list which contains every byte from 0 to 256. It goes through the information symbol by symbol, replacing it with its index on the list and moving it to the front of the list. It continues doing this until it reaches the end of the file. It will be left with a sequence of indexes, to which it will apply RLE again but this time it will only replace 0's with a sequence of special codes, RUNA and RUNB. This new sequence of symbols will then be encoded with Huffman coding. Bzip2 is a pretty versatile method of compression, it can effectively compress a variety of file types, as long as they are single files. Its decompression is relatively fast, but the compression is not very efficient. This led to the creation of pbzip2, a version of this method that supports multi-threading. It also is free and open-source.

D. Fast, Efficient, Lossless Image Compression System (FELICS)

FELICS is a lossless image compression algorithm that achieves a similar compression ratio to original lossless JPEG however it performs 5-times faster.

The coding algorithm proceeds to output the first two pixels. Using a raster scan method, which consists in sweeping horizontally the pixels from left to right from on line and once it reaches its end, moves to the next line, it will encode each new pixel P, with the intensities of the two nearest neighbors of P that have already been coded.

These neighbors are the pixel above and the pixel to the left of the new pixel, if P is found on top or side of the image it will use different neighbors to make P's prediction (fig 3). We call the smaller neighboring value L and the larger value H, define Δ to be the difference between H and L. We treat Δ as the prediction of P.

It uses one bit to indicate whether P is in the range from L to H, an additional bit if needed, to indicate whether it is above or below the range, if P is out of range it is

reasonable to use exponential prefix codes, Golomb-Rice codes, to indicate how far out of range the value is. This technique allows for a good compression: the two nearest neighbors provide a prediction, and the image model implied by the algorithm closely matches the distributions found in real images. In addition, the method is very fast, since it uses only single bits and simple prefix codes. [3]

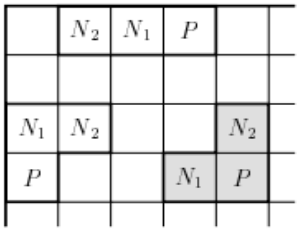


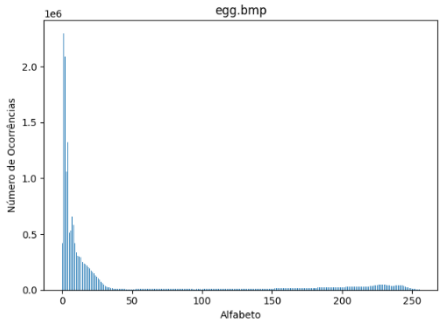
Fig. 3. Pixel P and its nearest neighbors N1 and N2

E. Context Adaptative Lossless Image Compression (CALIC)

CALIC encodes and decodes an image in raster scan order with a single pass through the image. For the purposes of context modeling and prediction, the coding process uses a neighborhood of pixel values taken only from the previous two rows of the image. Consequently, the encoding and decoding algorithms require a buffer that holds only two rows of pixels that immediately precede the current pixel. CALIC operates in two modes: binary mode and continuous-tone mode. In the binary mode, a context-based adaptive ternary arithmetic coder is used to code three symbols, including an escape symbol. In the continuous-tone mode, the system has four major integrated components: prediction, context selection and quantization, context-based bias cancellation of prediction errors, and conditional entropy coding of prediction errors. [4]

IV. Original Data Analisys

The following imagens show the statistical distribution of the original data set. Allowing for a better understanding of the compression methods results.



Entropy: 5.7242

Fig. 4. Statistical distribution egg.bmp

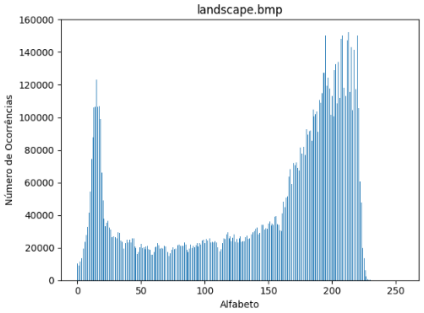


Fig. 5. Statistical distribution landscape.bmp

Entropy: 7.4205

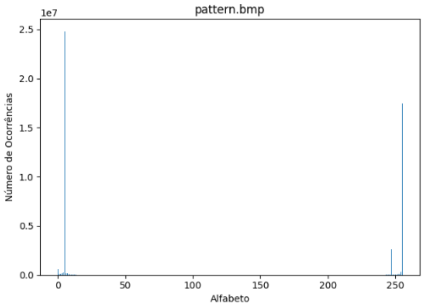


Fig. 6. Statistical distribution pattern.bmp

Entropy: 1.8292

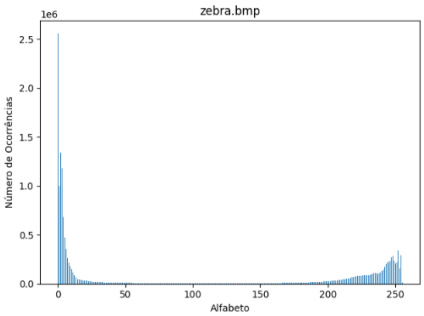


Fig. 7. Statistical distribution zebra.bmp

Entropy: 5.8312

V. Results

The next results and analysis are destined to help determine which compression method can be the most efficient on image compression.

The following tests were conducted on a Windows 10 machine with an i7-9750H processor and 16GB of RAM, and the results are the following:

TABLE I. BZ2

File	Original size (MB)	Final size (MB)	Comp. Ratio	Comp. Speed (MB/s)	Decomp. Speed (MB/s)
egg	16.92	4.53	3,74	18.23	30.87
landscape	10.50	3.33	3,15	14.71	25.39
pattern	45.78	1.72	26,55	90.69	130.15
zebra	15.96	5.36	2,98	16.35	28.51

TABLE II. LZMA

File	Original size (MB)	Final size (MB)	Comp. Ratio	Comp. Speed (MB/s)	Decomp. Speed (MB/s)
egg	16.92	4.60	3,68	9.92	47.05
landscape	10.50	3.08	3,41	1.97	47.59
pattern	45.78	1.84	24,86	9.54	160.63
zebra	15.96	5.40	2,96	1.68	40.12

TABLE III. DEFLATE

File	Original size (MB)	Final size (MB)	Comp. Ratio	Comp. Speed (MB/s)	Decomp. Speed (MB/s)
egg	16.92	6.30	2.68	57.90	228.15
landscape	10.50	4.24	2.48	46.42	249.75
pattern	45.78	2.69	19.03	228.77	369.34
zebra	15.96	7.26	2.20	52.26	245.91

TABLE IV. PNG

File	Original size (MB)	Final size (MB)	Comp. Ratio
egg	16.92	4.41	3.83
landscape	10.50	3.18	3.30
pattern	45.78	2.17	21.02
zebra	15.96	5.22	3.06

The above compression methods besides PNG, were obtained from python implementations. Both Bz2, a Bzip2 algorithm, and LZMA were already part of python's libraries, deflate, that handles compression and decompression of Gzip data, was installed externally [6].
The following formulas were used to determine compression ratio and compression / decompression speed:

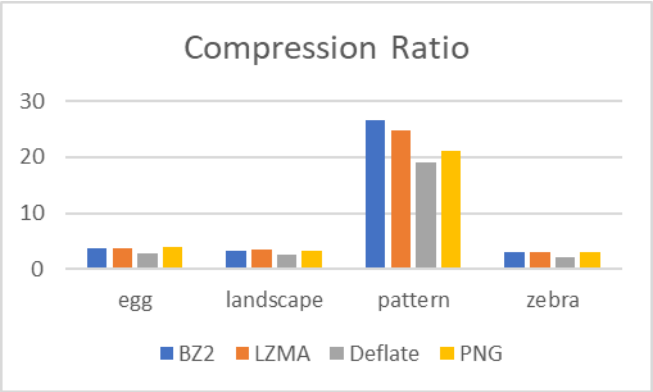
$$\text{Compression Ratio} = \frac{\text{Uncompressed Size}}{\text{Compressed Size}}$$

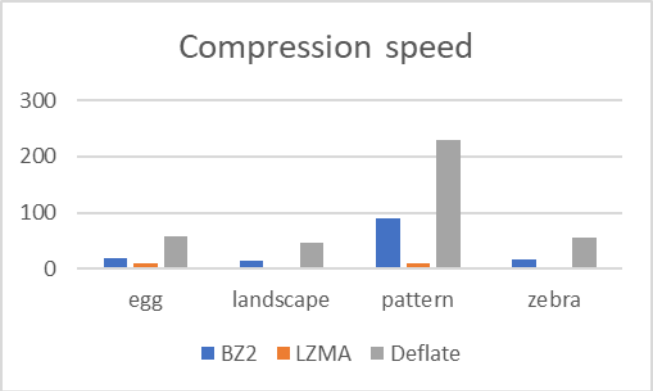
Decompression Speed

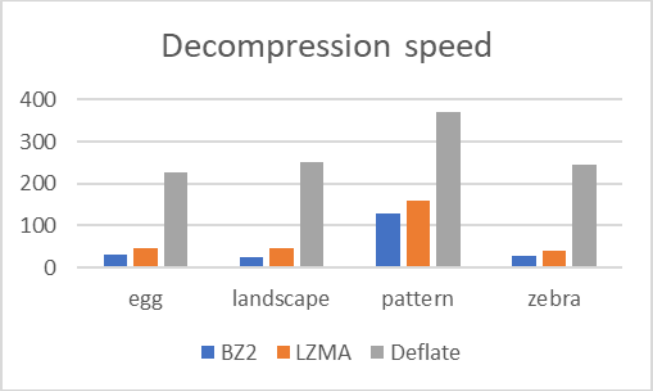
$$\text{speed} = \frac{\text{Uncompressed bits}}{\text{seconds to decompress}}$$

Compression Speed

$$\text{speed} = \frac{\text{Uncompressed bits}}{\text{seconds to compress}}$$







VI. FINAL SOLUTION

After a careful observation of the results, it can be assumed that the best compression implementation that provides both a good compression ratio and speed, compared to PNG is Bz2.
LZMA provides a good compression ratio similar to Bz2. However, its compression speed is to low compared to the other two methods. Deflate has a compression and decompression speed much higher than the other implementations, but it scored the lowest on compression ratio. For this reason, we believe that the best candidate to provide the best results compared to the PNG compression algorithm is indeed Bz2, a Bzip2 algorithm.

VII. Conclusion

It can be concluded that there is no perfect compression method, it depends on the user's preferences on compression speed and/or ratio and on the type of data that is being compressed.

References

[1] Ms. Ijmulwar, D. Kapgate, A Review on - Lossless Image Compression Techniques and Algorithms, International Journal of Computing and Technology, Volume 1, Issue 9, October 2014

[2] https://en.wikipedia.org/wiki/Arithmetic_coding

[3] Fast and Efficient Lossless Image Compression Paul G. Howard and Jeffrey Scott Vitter appears in the proceedings of the IEEE Computer Society/NASA/CESDIS Data Compression Conference,Snowbird, Utah, March 30-April 1, 1993, pages 351-360.

[4] X. Wu and N. Memon. Context-based, adaptive, lossless image coding. IEEE Trans. Commun., 45:437–444, 1997.

[5] https://en.wikipedia.org/wiki/DEFLATE

[6] https://pypi.org/project/deflate/

[7] https://en.wikipedia.org/wiki/LZ77_and_LZ78#LZ77

[8] https://en.wikipedia.org/wiki/Bzip2