Lossless compression of bilevel images with JPP

channoti@ulb.ac.be
Cédric Hannotier

mpetitje@ulb.ac.be
Mathieu Petitjean

I. Introduction

This report presents the implementation of a lossless coder for bilevel image compression. Several techniques are developed and their performances are compared.

II. Techniques description

The different coding schemes that have been implemented rely on several transforms and entropy coders that are introduced here.

A. Run-Length Encoding (RLE)

The RLE transforms a bilevel image into a sequence of runs. A run is the number of successive occurrences of the same value in the file. For example, the sequence $[0, 0, 0, 1, 1, 1, 1]$ would be coded $[3, 4]$ by a coder assuming the first received symbol is a 0.

B. Move to Front transform (MTF)

The purpose of the MTF is to reduce the entropy of the image [1]. It is also based on the concept of runs but replaces each symbol of a run by 0 except the first one. The first one is replaced by the index of the symbol in the alphabet of the source. For example, for a sequence $[1, 1, 1, 0, 0, 0, 0]$, the alphabet is $[0, 1]$. The index of 1 in the alphabet is 1, so that the sequence becomes $[\mathbf{1}, \mathbf{0}, \mathbf{0}, 0, 0, 0, 0]$. The 1 is moved to the front of the alphabet (hence the name of the MTF), and the sequence of 0's now needs to be encoded. The alphabet is now $[1, 0]$, so that the index of 0 is 1. The final sequence is $[1, 0, 0, 1, 0, 0, 0]$ and the final alphabet is $[0, 1]$.

C. Two-Role Encoder (TRE)

The TRE is a variant of the RLE which is more suited when one symbol is more probable than the other [1], so that it is meant to be used after the MTF. Each non-zero element is shifted by 8 bits, while the sequence of zeros is simply encoded as the run length. To follow the previous example, the sequence $[1, 0, 0, 1, 0, 0, 0]$ would be encoded as $[256, 2, 256, 3]$. Here, the assumption that no run length will exceed 256 is done. The stream can be decoded knowing that a symbol that is repeated is encoded with a number greater than 255, while a run cannot be greater than 255.

D. Exp-Golomb coding

The Exp-Golomb coding is a variable length prefix code meant for entropy coding. The i^{th} symbol is mapped to the binary representation of $i + 1$, preceded by M zeros with $M = \lfloor \log_2(i) \rfloor$. When the short codewords are assigned to most probable symbols, compression can be achieved [2].

E. Arithmetic coding

An arithmetic encoder converts a sequence of data symbols into a single fractional number and can approach the optimal fractional number of bits required to represent each symbol. In general, it outperforms prefix codes [2]. An integer version of the coder was implemented to cope with finite precision problems. When the interval corresponding to the symbol stream becomes too small, some bits are already extracted and the interval is made wider again.

III. Compression schemes

Various combinations of the previously detailed techniques are implemented in the codec. The encoder will try for all possible techniques and write to file the most advantageous method. They are listed here:

- **RLE-Gb**: the image is RLE-encoded then compressed with Exp-Golomb. A LUT also needs to be transmitted to allow the mapping of shortest codewords to most probable symbols.
- **RLE-Ath**: follows the same process as the RLE-Gb but the Golomb code is replaced by the arithmetic encoder. The symbols and their probabilities also need to be transmitted in a LUT.
- **MTF-Ath**: the image is transformed using the MTF and then is compressed by the arithmetic encoder. The dictionnary of the MTF and the probabilities for the arithmetic coder need to be added to the file.
- **Benzid**: the method is inspired from the one proposed in [1]. The image is transformed using the MTF then the TRE and compressed using the arithmetic coder.

Moreover, all the techniques are tested on the image in two scanning orders: horizontal and vertical.

In the compressed file, a flag identifies the chosen method. For some of them, additional size information is also needed to correctly parse the file while decoding. The LUTs are always transmitted with fixed size numbers (8 or 32 bits numbers depending on the case). The typical structure of the compressed file is shown below:

Header		Data	LUT
Method flag	Size of data	Compressed image	Dictionnaries for reconstruction

IV. Performance analysis

The various methods have been tested on several test images. The results are summarized in Table I, where the compression ratio is shown.

Image	RLE-Gb	RLE-Ath	MTF-Ath	Benzid
256x256	2.03	1.9	**2.42**	1.65
noisy, 256x256	0.96	**1.04**	0.99	0.88
512x512	1.62	**1.68**	1.64	1.42
1024x1024	2.34	**2.34**	2.09	2.17
2048x2048	6.39	**7.09**	6.55	6.47

Table I: Performance of the different schemes on images with various sizes. The best compression ratio is bolded for each image.

It appears that the MTF-Ath is the best performer for small images, while the RLE-Ath is the best for bigger or noisy images. It is easy to understand by looking at Figure 1 and Figure 2, which highlight the impact of the header, compressed data and dictionaries on the final size.

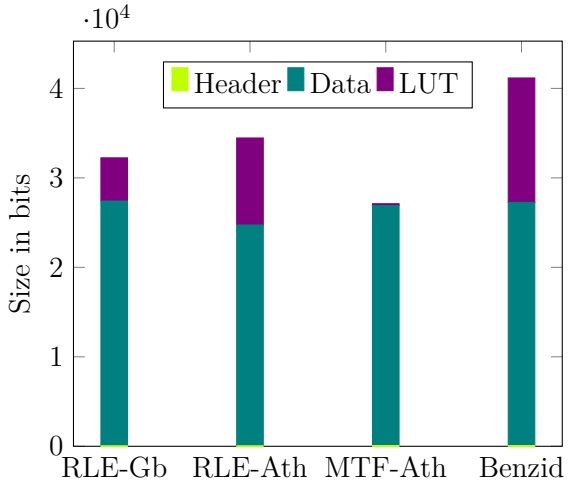

Figure 1: Performance of the techniques on a 256x256 image (earth)

It appears that even in the small image, the best data compression is done with RLE-Ath.

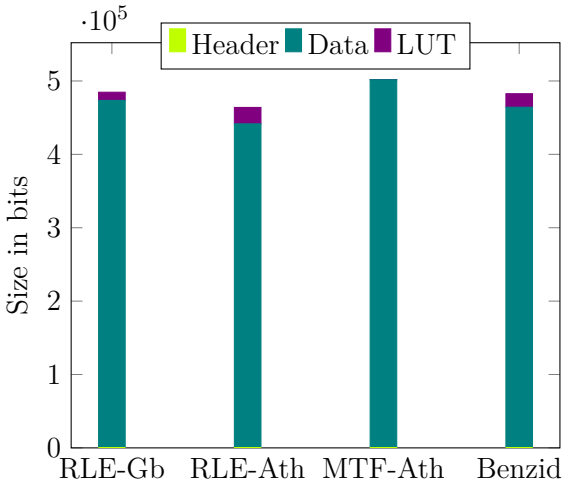

Figure 2: Performance of the techniques on a 1024x1024 image (airport)

However, the overhead needed to decode the image is very important. The MTF-Ath only needs few bytes of dictionaries while using the powerful arithmetic coder, making it the most efficient technique.

When the size of the image increases, the size of the dictionaries becomes less significant with respect to the size of the compressed data. RLE-Ath is then the best performer.

V. Conclusion

The encoder makes use of the powerful arithmetic coder to reach decent compression ratios, with significant improvements over the initial RLE-Gb scheme. Because it tries all the possible schemes, it allows for the best available compression regardless of the image size.

To go one step further, it could have been possible to try an adaptive arithmetic encoder. The compression of the data would be less ideal than with the one that was implemented (and relies on the knowledge of the probability of each symbol) but would reduce the size of the necessary dictionary.

References

[1] R. Benzid. "Lossless compression of binary image using move-to-front transform and two-role encoder". In: *Electronics Letters* 47.2 (2011), pp. 104–105. DOI: 10.1049/el.2010.2362.

[2] Iain E. Richardson. *The H.264 Advanced Video Compression Standard*. 2nd. Wiley Publishing, 2010.