LICO: An Effective, High-Speed, Lossless Compressor for Images

Noushin Azami, Rain Lawson, and Martin Burtscher

Department of Computer Science
Texas State University
San Marcos, Texas, 78666, USA
`noushin.azami@txstate.edu`, `rgl37@txstate.edu`, `burtscher@txstate.edu`

Abstract

Due to the large and growing number of photographs taken with progressively higher resolution, the volume of image data being generated, stored, and processed increases steadily. Compressing images losslessly is important for enhancing storage efficiency, expediting transmission, and reducing energy consumption while preserving image quality. This paper presents *LICO*, a new lossless compression algorithm for color images. On the 30 CLIC'24 images, our serial and parallel implementations of *LICO* deliver a compression speed that is 13× to 46× and a decompression speed that is 14× to 135× higher than JPEG2000. Furthermore, *LICO* is both faster and compresses more than PNG, TIFF, BZIP2, GZIP, and Zstandard.

Introduction

People are taking more and more photographs of ever increasing resolution, resulting in a large industry centered around areas such as computer vision, image processing, and object detection [1, 2]. As a result, the compression of image data has become important for many reasons, including storage efficiency, faster transmission, and improved processing speed, especially in environments where system resources are limited. Some important domains such as medical imaging [3] and forensic science [4] require images to remain pristine, that is, they require lossless compression.

One use-case of fast lossless image compression is in handheld cameras [5], where high-resolution pictures need large amounts of memory. Compression can help but must be performed quickly on a small processor that runs off of a battery.

Another example where both high-speed and lossless compression is important is space-based imaging [6]. Images captured by space probes are used for mapping, navigation [7], Geographic Information Systems (GIS) [8], and even emergency response and disaster management, where the preservation of each pixel may be crucial. Space probes typically have onboard computers with limited memory and processing capacity that are powered by solar panels. Typically, the transmission of data to Earth requires a large amount of energy that is proportional to the size of the data.

Many satellites already use compression. However, lossless image compressors such as JPEG2000 and PNG might be too complex and slow. If the compression is too slow, images are either lost because they cannot be sent, which lowers the data return, or must be transmitted in uncompressed format, which may raise costs and energy consumption due to increased space-to-ground communication.

In this paper, we introduce Lossless Image COmpressor or "*LICO*", a new lossless image compression algorithm along with our serial and parallel implementations thereof that achieve not only a high compression ratio but also a higher compression and decompression speed than leading compressors from the literature. This paper makes the following contributions.

- A new lossless image compression algorithm as well as our high-speed serial and OpenMP-parallelized implementations.
- A detailed compression-ratio comparison of six widely-used lossless compressors on the CLIC'24 images.
- A detailed compression-throughput comparison of six widely-used lossless compressors on the CLIC'24 images.

The serial C++ and parallel OpenMP *LICO* code is freely available in open source [9].

Related Work

There are several widely-used formats for storing, exchanging, editing, and compressing images. The Tagged Image File Format (TIFF) [10] supports lossless compression using multiple approaches, including LZW [11] and ZIP [12]. Both approaches reduce the size by replacing repeating sequences of bytes with shorter sequences.

Potable Network Graphics (PNG) [13] is another widely-used image file format that supports lossless compression. It comprises a filtering stage followed by Deflate [14] and Huffman coding [15], the combination of which results in a high compression ratio. Huffman coding replaces frequently occurring values with shorter code words. Deflate is a combination of Huffman and LZ77 [16]. Since both Deflate and Huffman are slow in nature, PNG has a relatively low throughput.

JPEG2000 [17], which is both a compressor and an image format, offers several benefits over the original JPEG [18], including lossless compression and a higher image quality. It employs wavelet transforms [19] for both lossy and lossless compression with a single algorithm. This transform only uses integer coefficients, which eliminates the need for quantization, making it suitable for lossless compression. JPEG2000 employs various entropy-encoding methods to decrease redundancies.

There are also general-purpose compressors that are very popular. Bzip2 [20] is such a compressor that combines the Burrows-Wheeler Transform (BWT) [21] with Run-Length Encoding (RLE) [22] and Huffman. Gzip is another example that uses the above-mentioned Deflate approach. Whereas Bzip2 typically achieves a higher compression ratio, Gzip tends to be substantially faster. Zstandard [23] is a more recent general-purpose compressor. It not only achieves high compression ratios but also high throughputs. It includes an entropy coding stage that is based on Huff0 [24], a fast implementation of Huffman coding for modern CPUs. Zstandard further uses Finite State Entropy (FSE) [25], a high-speed compression algorithm based on Asymmetric Numeral Systems (ANS) [26]. The combination of these fast algorithms results in Zstandard's high compression ratio and speed.

In this paper, we present serial and parallel CPU implementations of *LICO*. Our new algorithm is based on difference coding, bit and byte shuffling, and zero elimination, all of which are fast transformations. It achieves higher compression ratios than most of the above-mentioned compressors on a wide range of real-world images. Moreover, it delivers high compression and decompression throughputs, making it an attractive alternative to existing image compressors in environments where both speed and compression ratio are critical. Note that a high speed also means less energy consumption during compression and decompression.

Approach

LICO takes images in uncompressed BMP3 format as input. BMP is a widely used image file format that is supported by most editing, viewing, and conversion tools available on many operating systems. BMP3 is probably the most widely used version of BMP. It represents each pixel by three bytes, one each for red, green, and blue.

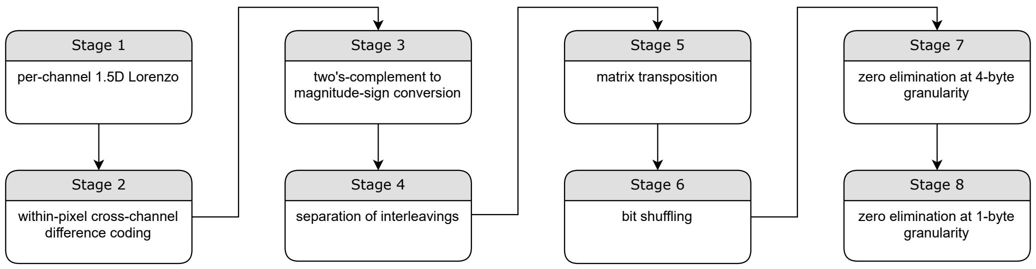
Stage 1
per-channel 1.5D Lorenzo
Stage 2
within-pixel cross-channel difference coding
Stage 3
two's-complement to magnitude-sign conversion
Stage 4
separation of interleavings
Stage 5
matrix transposition
Stage 6
bit shuffling
Stage 7
zero elimination at 4-byte granularity
Stage 8
zero elimination at 1-byte granularity

Figure 1: LICO algorithm pipeline stages

The *LICO* algorithm consists of the 8 data transformations outlined in Figure 1. Stage 1 exploits inter-pixel correlations by applying a variation of the Lorenzo predictor [27] to each color channel in the pixel matrix. We call this step "1.5-dimensional" (1.5D) Lorenzo. 1D Lorenzo predicts each pixel based on the neighboring pixel along one dimension. 2D Lorenzo combines the north, west, and northwest pixels' colors to form a prediction for the current pixel. In either case, the predicted color values are subtracted from the actual values, leaving a difference sequence. 2D Lorenzo tends to make more accurate predictions than 1D Lorenzo but is slower since it accesses three neighbors. As a compromise, our 1.5D Lorenzo stage uses the northern neighbor only for predicting the pixels in the first column and the western neighbor for predicting the remaining pixels. We found this approach to yield more accurate predictions than 1D Lorenzo and to be faster than 2D Lorenzo.

Stage 2 exploits intra-pixel correlations by computing another difference sequence. If neighboring pixels have the same color but different brightness, the result of the first stage will be three color-value differences per pixel that tend to be similar. This is why we subtract the first and third from the middle color channel for each pixel.

Stage 3 converts each value from two's-complement to magnitude-sign format. A difference sequence typically consists of values that cluster around zero. However, about half of these values are negative, which translates into many leading '1' bits in two's-complement representation. The positive values, in contrast, tend to have many leading '0' bits. Changing the representation to magnitude-sign format moves the sign bit to the least-significant position and creates many leading '0' bits for both positive and negative values, which increases the compressibility of the data.

Stage 4 separates the interleaved magnitude-sign differences of the three color channels into three separate non-interleaved matrices. We do this because only two of the three channels are intra-pixel decorrelated, meaning the channels exhibit different behaviors. Thus, grouping the values by channel improves compressibility.

Stage 5 transposes the three matrices. Since the 1.5D Lorenzo transformation in the first stage leaves some correlation between neighboring pixels in the same column unexploited, we transpose the three matrices so that the correlating values end up next to each other in memory (as the matrices are stored in row-major format).

Stage 6 applies bit shuffling to the transformed values. This groups the most-significant bit of each value together, followed by all second-most-significant bits, and so on. For neighboring values that have many common leading '0' bits, this tends to produce long sequences of '0' bits, which are easy to compress.

Stage 7 operates on the resulting data at integer (i.e., 4-byte) granularity. This stage generates a bit map in which each bit specifies whether the corresponding word in the input is zero or not. It outputs the non-zero words, compresses the bitmap using a similar method, and then outputs the compressed bitmap.

Stage 8 is identical to Stage 7 except it operates at byte granularity. This eliminates zero bytes that the prior stage could not handle due to its larger word size.

Note that, perhaps with the exception of the last two stages, each stage performs a very simple transformation that requires little computation. To minimize memory accesses, we combine some transformations, i.e., we load a value, perform multiple transformations on it, and only then store the result back to memory. To further boost the performance, we store the result to a new location to implement both the separation of the data into three matrices and the transposition of those matrices.

The parallel version of *LICO* uses OpenMP to concurrently process the rows of pixels in the first five stages. The sixth stage is parallelized across sets of 8 bytes. The last two stages divide the data into chunks of 16 kilobytes and process them independently. The decoder employs the same parallelization strategy.

Methodology

We compare *LICO* to the three image and three general-purpose compressors listed in Table 1. To compile and run the programs, we followed the instructions in their source-code repositories. Our evaluation system is based on a 64-bit Intel Xeon Gold CPU with 2 sockets and 16 cores per socket. We compiled all codes with g++ version 12.2.1. For *LICO*, we used the $-O3$ and $-march = native$ flags. For the parallel version of *LICO*, we further used the $-fopenmp$ flag to enable OpenMP.

The general-purpose compressors Gzip, Bzip2, and Zstandard support multiple levels to trade off compression ratio and throughput. We show results for all 9 levels of Gzip and Bzip2 and all 22 levels of Zstandard. TIFF supports multiple lossless compression algorithms. We show results for TIFF-LZW, which uses the Lempel-Ziv-Welch algorithm, and TIFF-ZIP, which uses the Deflate algorithm.

Table 1: Lossless Compressors used for Comparison

Compressor	**Version**	**Source**	**Type**	**Ser/Par**
JPEG2000	2.5.0	jpeg.org/jpeg2000/	Image	Serial
PNG	1.6.40	libpng.org/pub/png/pngcode.html	Image	Serial
TIFF	4.6.0	simplesystems.org/libtiff/	Image	Serial
Bzip2	1.1.0	github.com/libarchive/bzip2	General-purpose	Serial
Gzip	1.13	gnu.org/software/gzip	General-purpose	Serial
Zstandard	1.5.1	github.com/facebook/zstd	General-purpose	Parallel

We utilize the full set of 30 images from the CLIC'24 challenge [28, 29] as inputs. The smallest image has a resolution of 2048×1152 pixels and the largest has a resolution of 2048×2048 pixels. Since these images are distributed in PNG format, we first converted them to the uncompressed BMP3 format and use the resulting BMP files as inputs. The sizes of these BMP files range from 7 to 12 megabytes.

We evaluate each compressor by its compression ratio as well as its compression and decompression throughput. We measure the compression ratio by dividing the initial file size by the compressed file size and the compression and decompression throughput by diving the initial file size by the compression or decompression time. This time does not include reading/writing the data from/to secondary storage since we only measure the actual compression and decompression time. To eliminate outliers, we use the median runtime of 9 identical runs for computing the throughputs.

Results

In this section, we compare the compression ratio, compression throughput, and decompression throughput of our serial and parallel *LICO* implementations to six widely-used lossless image and general-purpose compressors. Serial and parallel *LICO* yield the same compression ratio as they implement the same algorithm. In fact, their outputs are guaranteed to be bit-for-bit identical.

Figures 2 and 3 show scatter plots of the performance of each compressor. Figure 2 displays the compression ratio vs. compression throughput, and Figure 3 displays the compression ratio vs. decompression throughput. For all three metrics, a higher value indicates better performance. Hence, the higher-quality compressors are closer to the upper right corner of the charts. Due to the large range of throughputs, both figures have a linear y-axis but a logarithmic x-axis.

Each dot in the two figures represents a compressor's geometric-mean performance over the set of 30 images. For example, Figure 2 shows that serial *LICO* delivers a geometric-mean compression ratio of over 2.1 and a geometric-mean compression throughput of almost 165 MB/s. We color the results of Bzip2, Gzip, and Zstandard in magenta, red, and blue, respectively. The numbers indicate the level.

The black line highlights the Pareto front [30], that is, the best compressors in the 2D metric space. The compressors on this front outperform all other evaluated compressors in at least one of the two metrics.

The Pareto front in Figure 2 includes only JPEG2000 and parallel *LICO*. If we only considered serial codes, JPEG2000 and serial *LICO* would be the only two compressors on the Pareto front. Whereas JPEG2000 achieves a higher compression ratio, both *LICO* and parallel *LICO* outperform it in compression throughput. Since the x-axis is logarithmic, the seemingly small difference is actually large. *LICO* compresses almost 13× and parallel *LICO* almost 46× faster than JPEG2000.

Zstandard is the only compressor other than *LICO-par* that is parallel. While the parallel implementation of *LICO* outperforms all compressors in throughput, it is

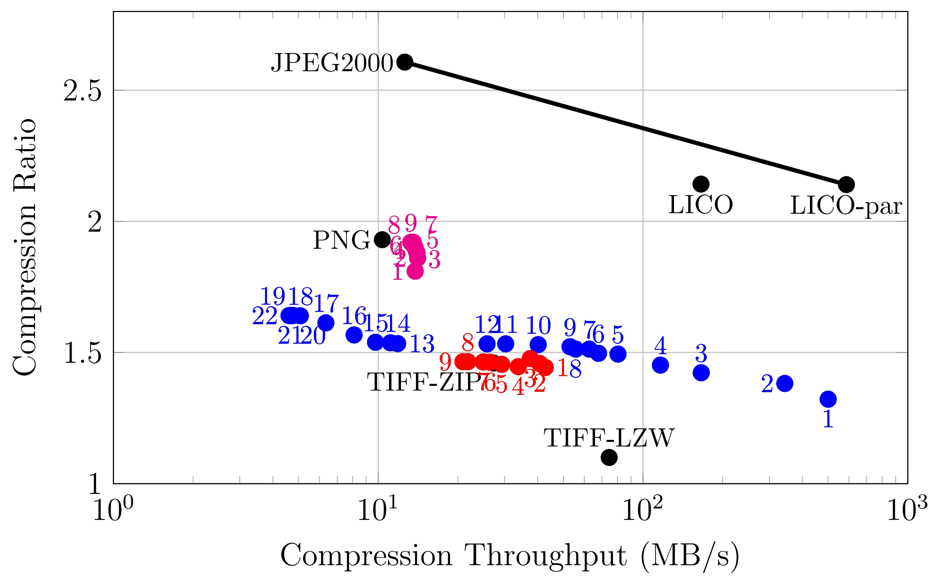

Figure 2: Compression Ratio vs. Compression Throughput (Bzip2 is magenta, Gzip is red, and Zstandard is blue; the numbers indicate the compression level)

worth noting that even the serial version of *LICO* is faster than all serial compressors and 20 of the 22 levels of parallel Zstandard. Furthermore, Zstandard achieves a much lower compression ratio than *LICO* for all 22 levels. *LICO*'s compression ratio is only second to JPEG2000's. It outperforms all other evaluated compressors in *both* compression ratio and compression speed. We also ran JPEG XL, the most recent version of JPEG, but even at its best-compressing lossless level, it only achieves a compression ratio of 1.53, which is lower than JPEG2000 and *LICO*.

Figure 3 shows similar results but for decompression. The Pareto front again only contains JPEG2000 and parallel *LICO*. The serial Pareto front would again only contain JPEG2000 and serial *LICO*. The decompression speed of parallel *LICO* is almost 135× faster than JPEG2000, which is serial, and almost 2× the speed of parallel Zstandard in its fastest mode. Serial *LICO* is faster than all serial compressors and nearly 14× faster than JPEG2000. Similar to compression, *LICO* also outperforms all evaluated compressors in speed during decompression.

Figures 2 and 3 only show geometric-mean results. To provide more detail, Table 2 lists the number of images on which each compressor is the fastest or best.

JPEG2000 reaches the highest compression ratio on the majority of the images. On the remaining images, Bzip2 in its best mode provides the highest compression. However, recall that the average compression ratio of *LICO* is higher than Bzip2's.

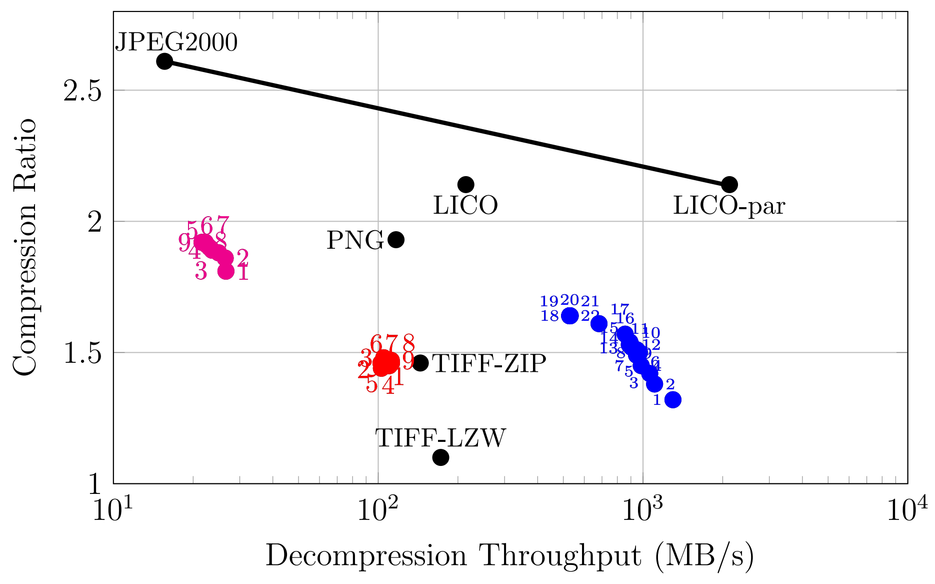

Figure 3: Compression Ratio vs. Decompression Throughput (Bzip2 is magenta, Gzip is red, and Zstandard is blue; the numbers indicate the compression level)

Parallel *LICO* dominates in throughput. It delivers the fastest compression and especially decompression on the majority of the images. Zstandard is the only other compressor that is sometimes the fastest, but its compression ratio is much lower.

Summary and Conclusion

In this paper, we present *LICO*, a new lossless compression algorithm for real-world images. It is based on 8 data transformations, all of which are surprisingly simple. Yet, the resulting algorithm compresses well and is fast. In fact, our evaluation of the 30 images from the CLIC'24 contest shows that *LICO* compresses and decompresses faster than JPEG2000, PNG, TIFF, Bzip2, Gzip, and Zstandard. Moreover, it reaches a higher geometric-mean compression ratio than all of these compressors except JPEG2000. The combination of efficiency and effectiveness makes it an attractive option in settings where speed, compression ratio, and energy consumption are critical, such as in hand-held cameras and space probes.

Acknowledgements

This work has been supported in part by the Department of Energy, Office of Science under Award Number DE-SC0022223.

Table 2: Number of images on which compressors achieve fastest compression, fastest decompression, and highest compression ratio

Compressor	**Fastest Compression**	**Fastest Decompression**	**Highest Compression Ratio**
LICO	0	0	0
LICO-par	20	29	0
JPEG2000	0	0	25
PNG	0	0	0
TIFF-LZW	0	0	0
TIFF-ZIP	0	0	0
Gzip-best	0	0	0
Gzip-fast	0	0	0
Bzip2-best	0	0	5
Bzip2-fast	0	0	0
Zstd-best	0	0	0
Zstd-fast	10	1	0

References

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2012.

[2] James E. Fowler and Qian Du, "Reconstructions from compressive random projections of hyperspectral imagery," in *Optical Remote Sensing: Advances in Signal Processing and Exploitation Techniques*, Saurabh Prasad, Lori M. Bruce, and Jocelyn Chanussot, Eds., chapter 3, pp. 31–48. Springer, 2011.

[3] Tim Bruylants, Adrian Munteanu, and Peter Schelkens, "Wavelet based volumetric medical image compression," *Signal processing: Image communication*, vol. 31, pp. 112–133, 2015.

[4] Matthew C Stamm and KJ Ray Liu, "Anti-forensics of digital image compression," *IEEE Transactions on Information Forensics and Security*, vol. 6, no. 3, pp. 1050–1065, 2011.

[5] Dharmpal Takhar, Jason N Laska, Michael B Wakin, Marco F Duarte, Dror Baron, Shriram Sarvotham, Kevin F Kelly, and Richard G Baraniuk, "A new compressive imaging camera architecture using optical-domain compression," in *Computational Imaging IV*. SPIE, 2006, vol. 6065, pp. 43–52.

[6] Lilian N Faria, Leila MG Fonseca, and Max HM Costa, "Performance evaluation of data compression systems applied to satellite imagery," *Journal of Electrical and Computer Engineering*, vol. 2012, pp. 18–18, 2012.

[7] Anil N Joglekar and J David Powell, "Data compression in recursive estimation with applications to navigation systems," *Journal of Aircraft*, vol. 12, no. 1, pp. 58–64, 1975.

[8] Haijun Zhu and Chaowei Phil Yang, "Data compression for network gis.," 2008.

[9] Noushin Azami and Martin Burtscher, "Lico code," `https://github.com/burtscher/LICO/`, 2023, Accessed: 2023-11-05.

[10] Simple Systems, “Libtiff,” `http://www.simplesystems.org/libtiff/`, 2023, Accessed: 2023-11-05.

[11] Terry A. Welch, “A technique for high-performance data compression,” *IEEE Computer*, vol. 17, no. 6, pp. 8–19, 1984.

[12] PKWARE, Inc., *APPNOTE - .ZIP File Format Specification*, 2007.

[13] W3C, *Portable Network Graphics (PNG) Specification (Second Edition)*, 2003.

[14] Peter Deutsch, “Deflate compressed data format specification version 1.3,” Tech. Rep., 1996.

[15] David A. Huffman, “A method for the construction of minimum-redundancy codes,” *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098–1101, 1952.

[16] Abraham Lempel and Jacob Ziv, “A universal algorithm for sequential data compression,” *IEEE Trans. on Information Theory*, vol. 23, no. 3, pp. 337–343, 1977.

[17] Athanassios Skodras, Charilaos Christopoulos, and Touradj Ebrahimi, “The jpeg 2000 still image compression standard,” *IEEE Signal processing magazine*, vol. 18, no. 5, pp. 36–58, 2001.

[18] Gregory K Wallace, “The jpeg still picture compression standard,” *IEEE transactions on consumer electronics*, vol. 38, no. 1, pp. xviii–xxxiv, 1992.

[19] Wim Sweldens, “The lifting scheme: A custom-design construction of biorthogonal wavelets,” *Applied and comput. harmonic analysis*, vol. 3, no. 2, pp. 186–200, 1996.

[20] Julian Seward, “bzip2 and libbzip2,” *avaliable at http://www. bzip. org*, 1996.

[21] “Burrows-wheeler transform,” `https://en.wikipedia.org/wiki/Burrows%E2%80%93Wheeler_transform`, Accessed: July 31, 2023.

[22] “Run-length encoding (rle),” `https://en.wikipedia.org/wiki/Run-length_encoding`, Accessed: July 31, 2023.

[23] Facebook, Inc., “Zstandard - fast real-time compression algorithm,” `https://facebook.github.io/zstd/`, 2023, Accessed: 2023-11-05.

[24] Klaus Post, “klauspost/compress github repository,” `https://github.com/klauspost/compress/tree/master/huff0`, 2023, Accessed: 2023-11-05.

[25] Jarek Duda, “Asymmetric numeral systems: entropy coding combining speed of huffman coding with compression rate of arithmetic coding,” *arXiv preprint arXiv:1311.2540*, 2013.

[26] Jarosław Duda, “Asymmetric numeral systems: entropy coding combining speed of huffman coding with compression rate of arithmetic coding,” in *Proceedings of the Data Compression Conference (DCC)*, 2014, pp. 309–318.

[27] Lawrence Ibarria, Peter Lindstrom, Jarek Rossignac, and Andrzej Szymczak, “Out-of-core compression and decompression of large n-dimensional scalar fields,” in *Computer Graphics Forum*. Wiley Online Library, 2003, vol. 22, pp. 343–348.

[28] Compression.cc, “Image Compression Task,” `https://compression.cc/tasks/#image`, 2023, Accessed: 2023-11-05.

[29] Unsplash, “Unsplash,” `https://unsplash.com/`, 2023, Accessed: 2023-11-05.

[30] Kaisa Miettinen, “Theoretical analysis of multiobjective optimization,” *Doctoral Dissertation, Acta Polytechnica Scandinavica, Mathematics and Computing Series, No. 67*, 1998.