DATA COMPRESSION METHODS - PROJECT WORK
LZW, RLE, TANS ALGORITHMS

Ádám Burgert

Computer Science Engineering MSc

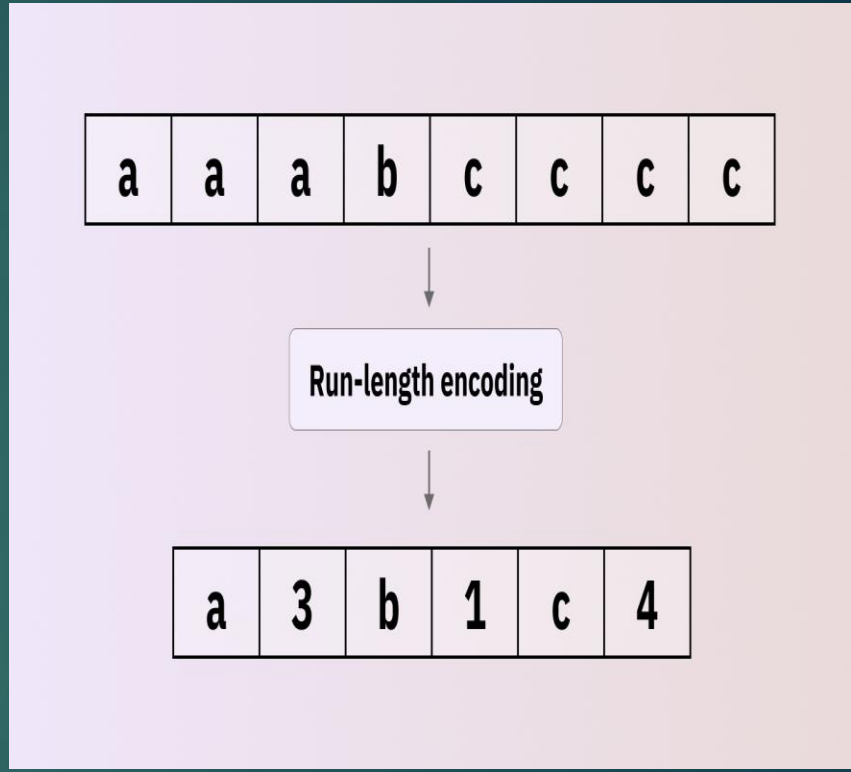
Pázmány Péter Catholic University

Github – Project

RLE

Run-Length Encoding (RLE) is a simple lossless data compression method.

- It compresses **runs of identical data values** (sequences where the same value repeats consecutively) by storing just one value plus a count, instead of the full run.
- Best suited for data with **many consecutive repeated elements**.
- Especially effective for compressing data with **repeated byte patterns**, shrinking the physical size of repeating characters.
- Works by replacing sequences of identical elements with a **pair: (element, count)**.
- **Advantages include:**
  - Reduced storage requirements
  - Improved data transfer efficiency
  - Faster data retrieval times
  - Particularly useful in environments with **limited bandwidth**.
  - Its simplicity makes it ideal for graphics and audiovisual data applications.
  - Main limitation: it's only effective on relatively simple repetitive **data patterns**, thus it has limited use with more complex or random data [1], [2].



Source: [3]

LZW

- Invented in 1984 by **Lempel, Ziv, and Terry Welch;** it's a form of lossless compression.
- It's a table-based lookup technique that removes duplicate data by compressing the original file into a smaller one.
- Also effective for compressing text and PDF files.
- Based on the **LZ78** algorithm (by Lempel and Ziv, 1978).
- Widely used for image files (GIF, TIFF), as well as PDF and TXT formats.
- Included as a feature in Unix file compression tools.
- Works by reading input symbols and combining them into strings.
- Uses a dynamically created dictionary to replace strings with shorter codes.
- Repeated sequences get replaced by shorter codes, reducing file size.
- The more repetition in data, the better the compression.
- **Greedy method:** it adds new sequences to the dictionary on the fly without pre-analysis, aiming to match and encode longer strings as it goes [4], [5], [6].

tANS

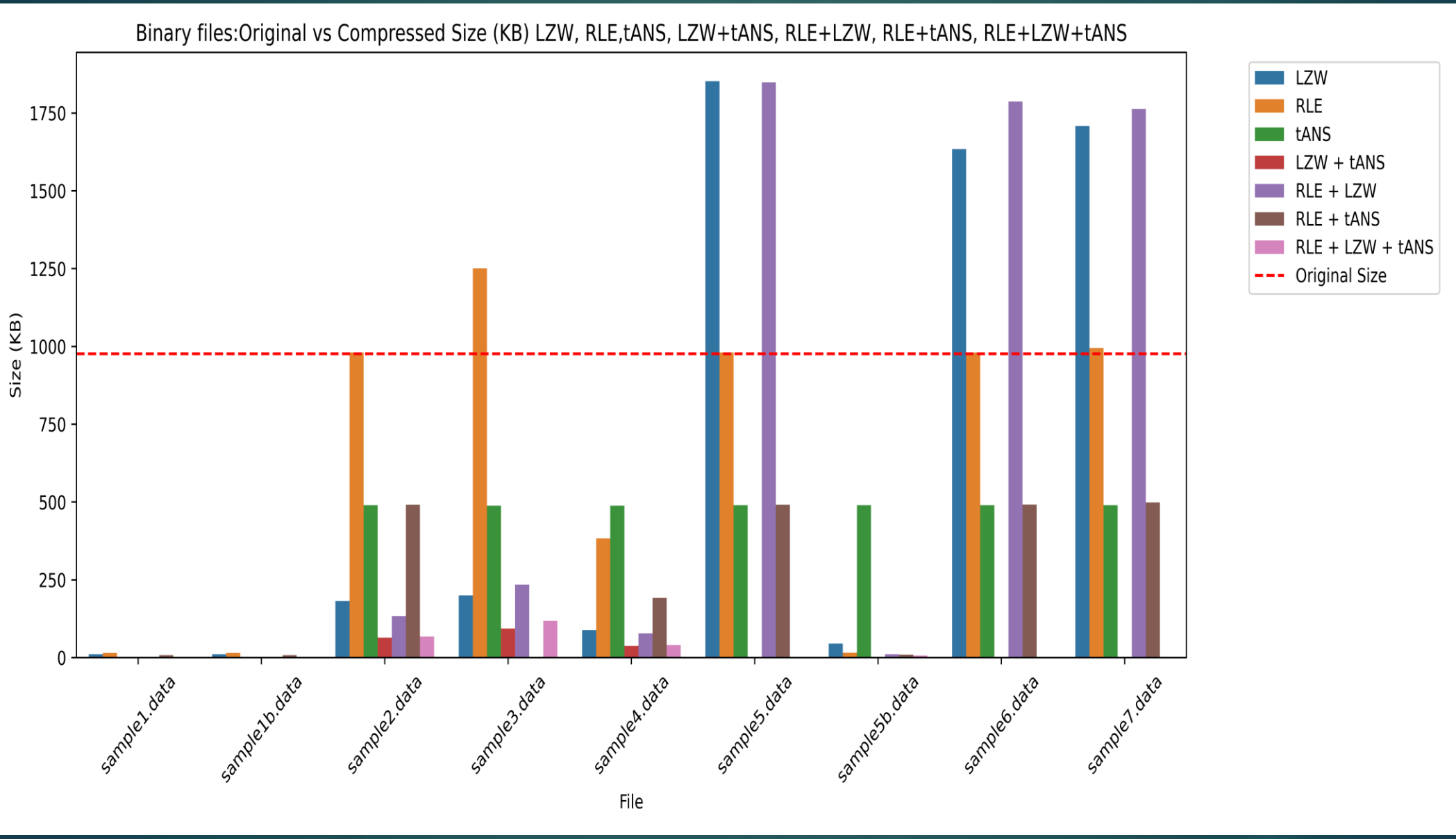
- **tANS (table-based Asymmetric Numeral Systems)** is a fast entropy coding method, invented by **Jarek Duda** in 2013.
- Designed to combine the high compression efficiency of arithmetic coding with the speed of **Huffman** coding.
- Unlike arithmetic coding, which uses fractional intervals, **tANS** uses a single integer state and precomputed tables for encoding/decoding.
- The algorithm updates this integer state based on the symbol's probability.
- It writes bits whenever the state exceeds a certain threshold during encoding.
- Decoding reverses this process using the same tables [7], [8].

Project Implementation and System Design

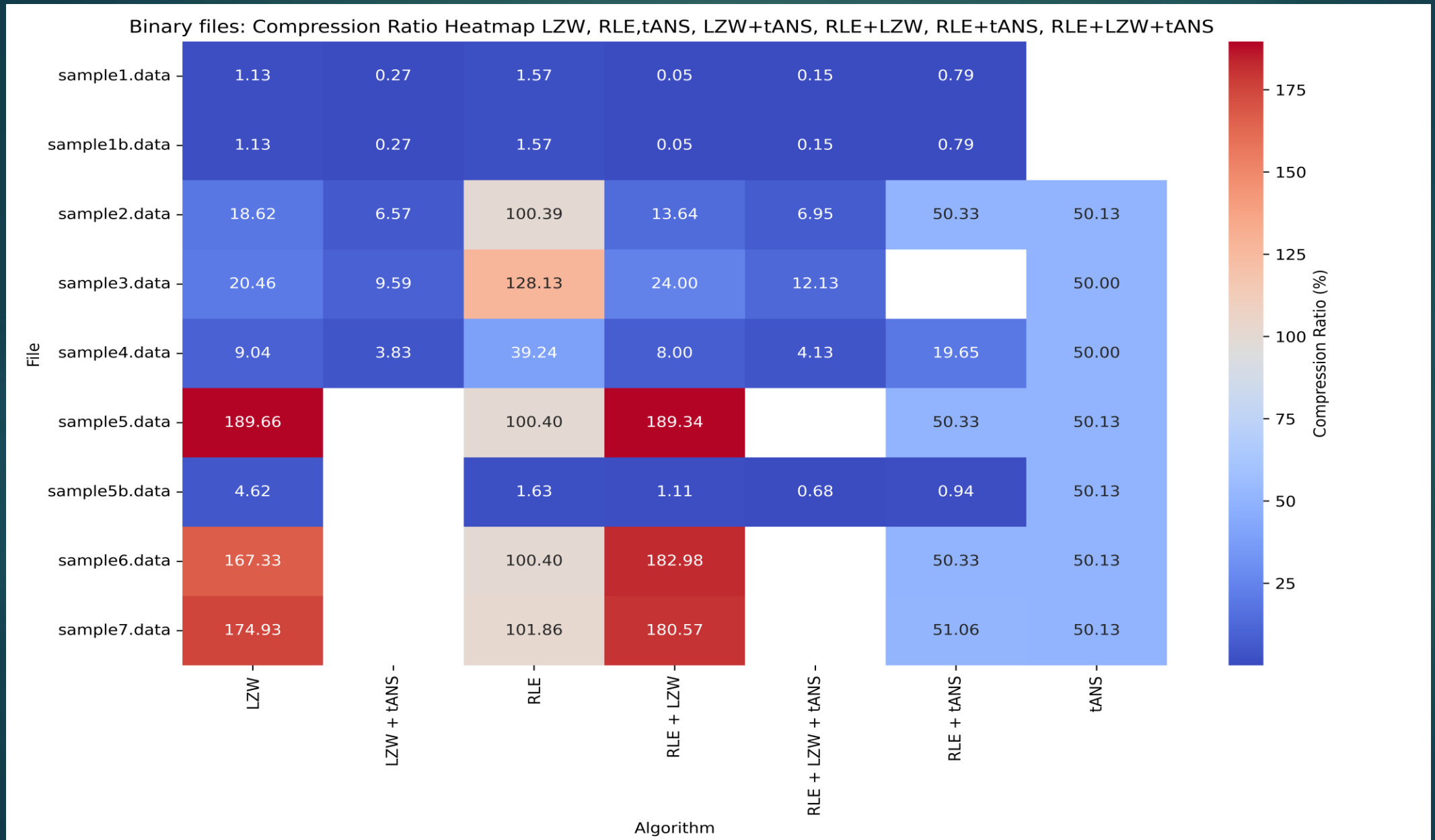
- **cli.py CLI for user interaction.**
  - python cli.py –compress –method tans
  - python cli.py –decompress –method tans
  - **Usage:**
    - cli.py [-h] [–compress] [–decompress] [–method lzw,rle,rle+lzw, tans, rle+tans, lzw+tans, rle+lzw+tans]
- **Files:**
  - compress.py
    - Compression python file utilizing LZW, RLE, tANS based compression
  - decompress.py
  - rle.py
  - lzw.py
  - tans.py
  - plot.py
    - Generates pdf reports of compression and decompression results visualizations Table, File, original size,
    - compressed size, compression ratio (%), space saved (%)

```python
def FileCompressor(InputPath: str, OutputPath: str, method: str = 'lzw'):
    try:
        with open(InputPath, 'rb') as f:
            data = f.read()

        if len(data) == 0:
            raise ValueError(f"File {InputPath} is empty and cannot be compressed.")

        with open(OutputPath, 'wb') as FileOut:
            if method == 'lzw':
                codes = list(LZWParallel(data))
                TotalCodeBytes = b''.join(code.to_bytes(2, 'big') for code in codes)
                FileOut.write(MAGIC_HEADERS[method])
                FileOut.write(len(TotalCodeBytes).to_bytes(4, 'big'))
                FileOut.write(TotalCodeBytes)

            elif method == 'rle':
                RleData = RLE_Encode(data)
                FileOut.write(MAGIC_HEADERS[method])
                ChunkedDataWriter(FileOut, RleData)

            elif method == 'rle+lzw':
                RleData = RLE_Encode(data)
                codes = LZW_Compress(RleData)
                TotalCodeBytes = b''.join(code.to_bytes(2, 'big') for code in codes)
                FileOut.write(MAGIC_HEADERS[method])
                ChunkedDataWriter(FileOut, TotalCodeBytes)

            elif method == 'tans':
                FreqTable, EncodedBits, FinalState, TableSize = TansEncode(data)
                FileOut.write(MAGIC_HEADERS[method])
                TansEncoded_Data_Writer(FileOut, FreqTable, EncodedBits, FinalState, len(data), TableSize=T

            elif method == 'rle+tans':
                RleData = RLE_Encode(data)
                FreqTable, EncodedBits, FinalState, TableSize = TansEncode(RleData)
                FileOut.write(MAGIC_HEADERS[method])
                TansEncoded_Data_Writer(FileOut, FreqTable, EncodedBits, FinalState, len(RleData), TableSiz
```

Results: Binary files – Original vs Compressed size (KB)



Results: Binary files - Compression Ratio (%)



Results : Binary files - Space saved (%)

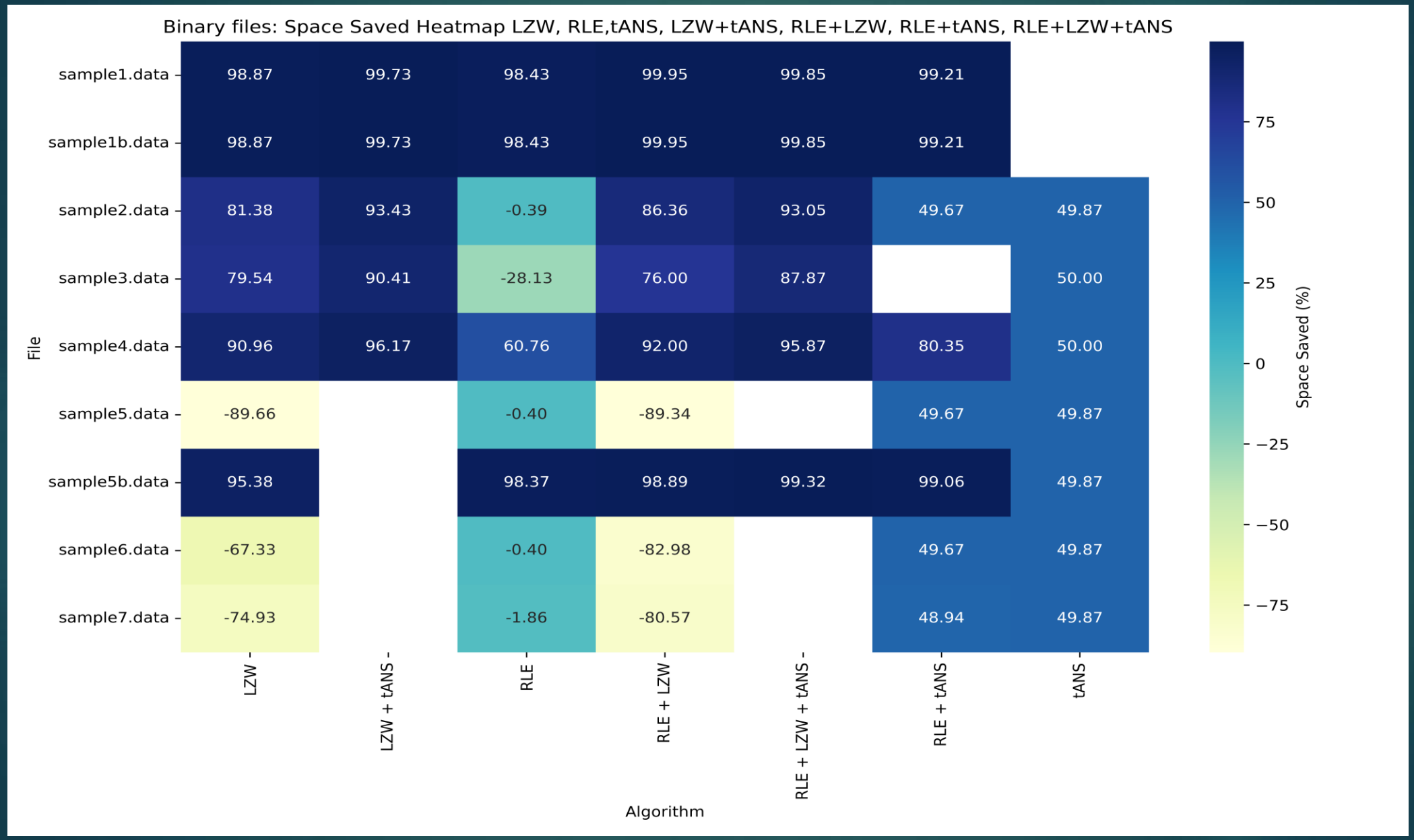


Image files:

apple.jpg

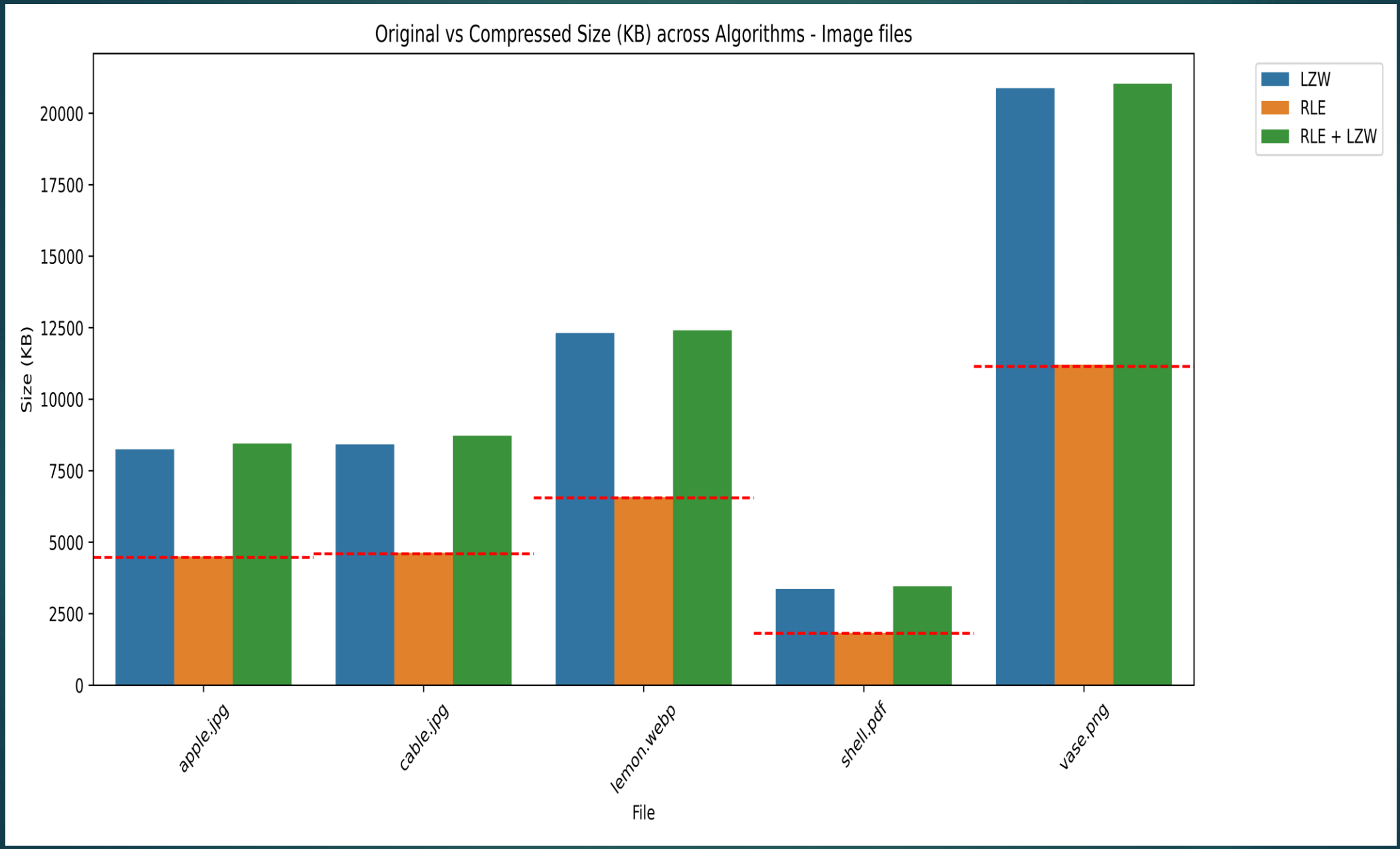
cable.jpg

vase.png

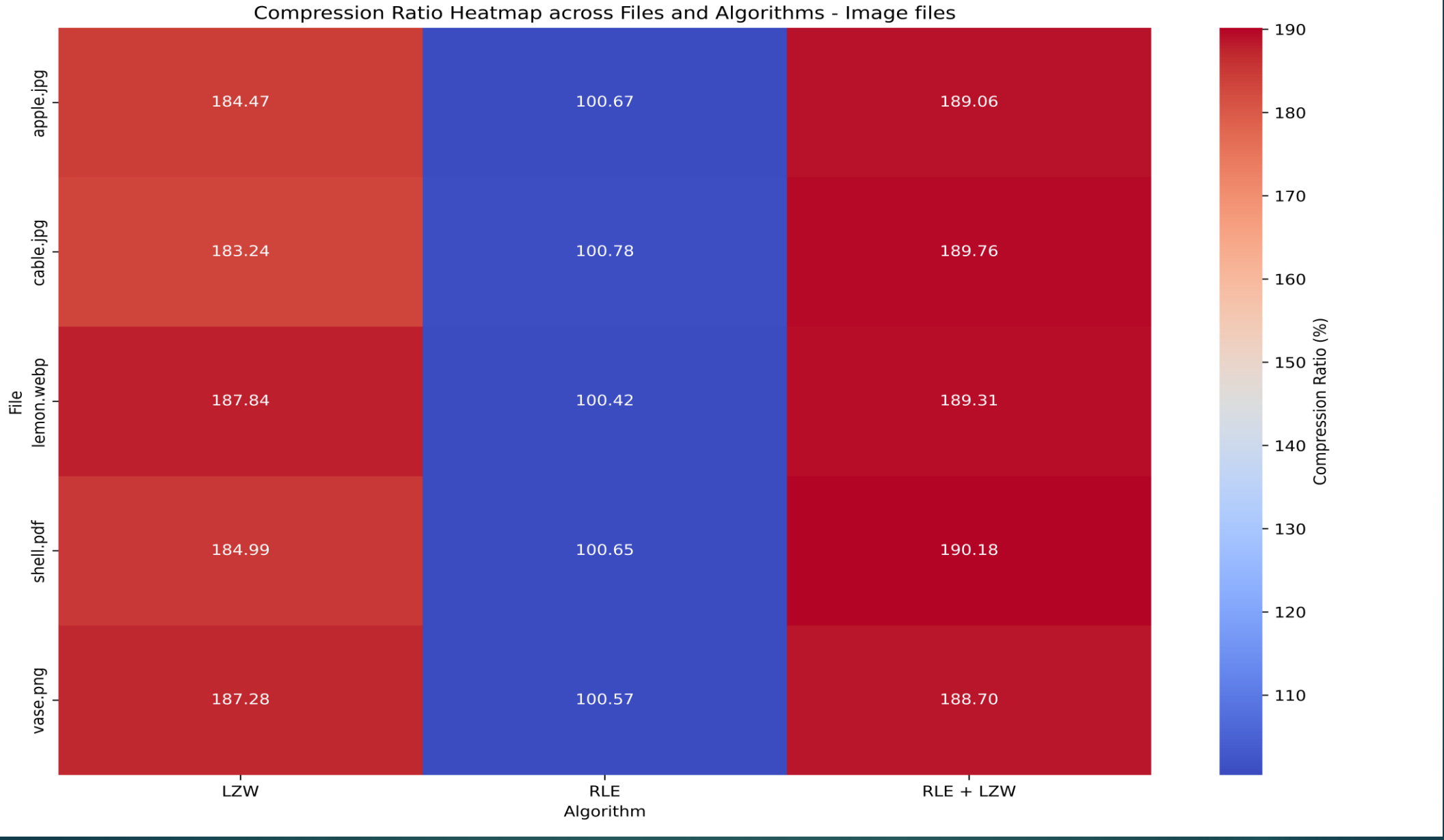
shell.pdf

lemon.webp

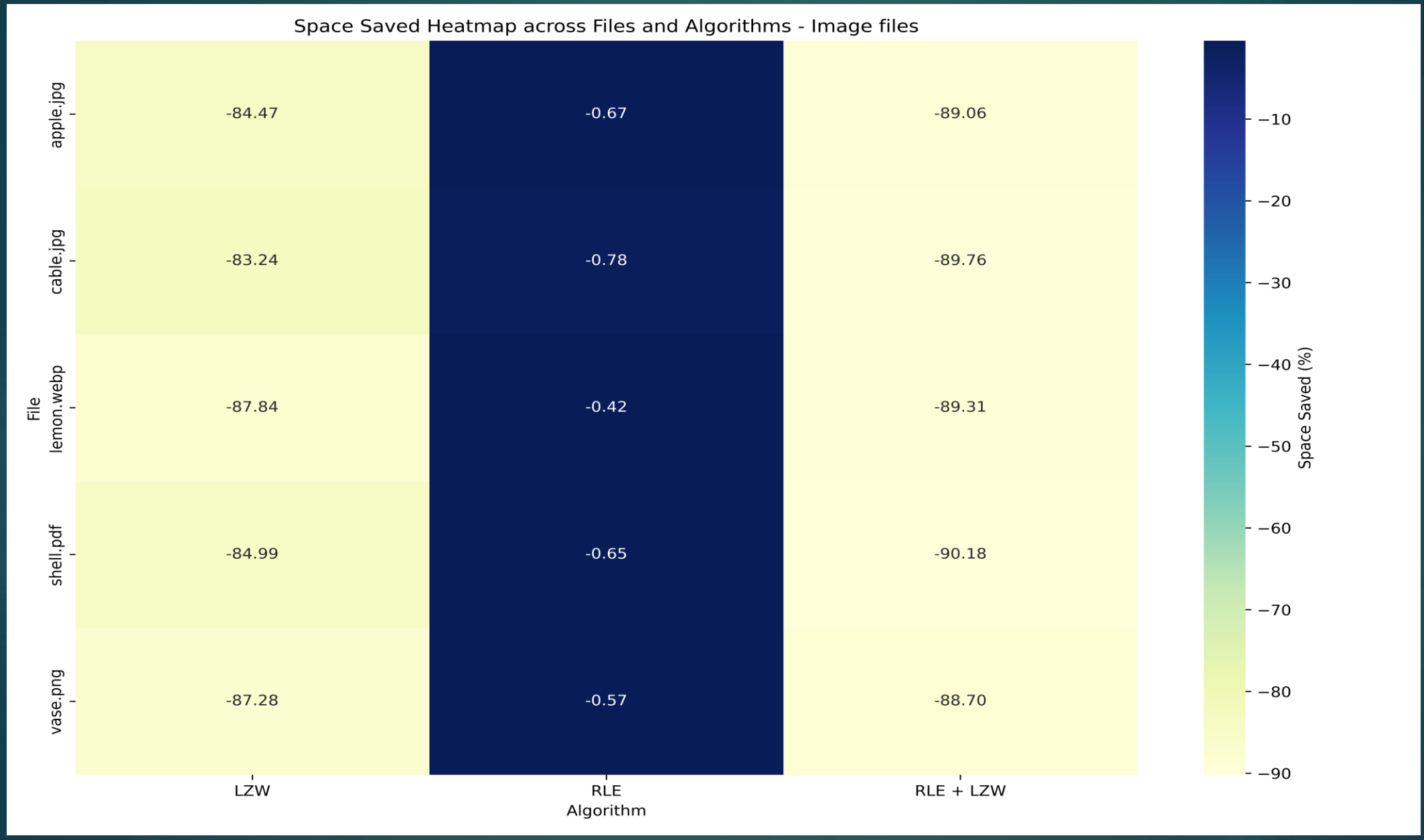
Results: Image Files – Original vs Compressed size (KB)



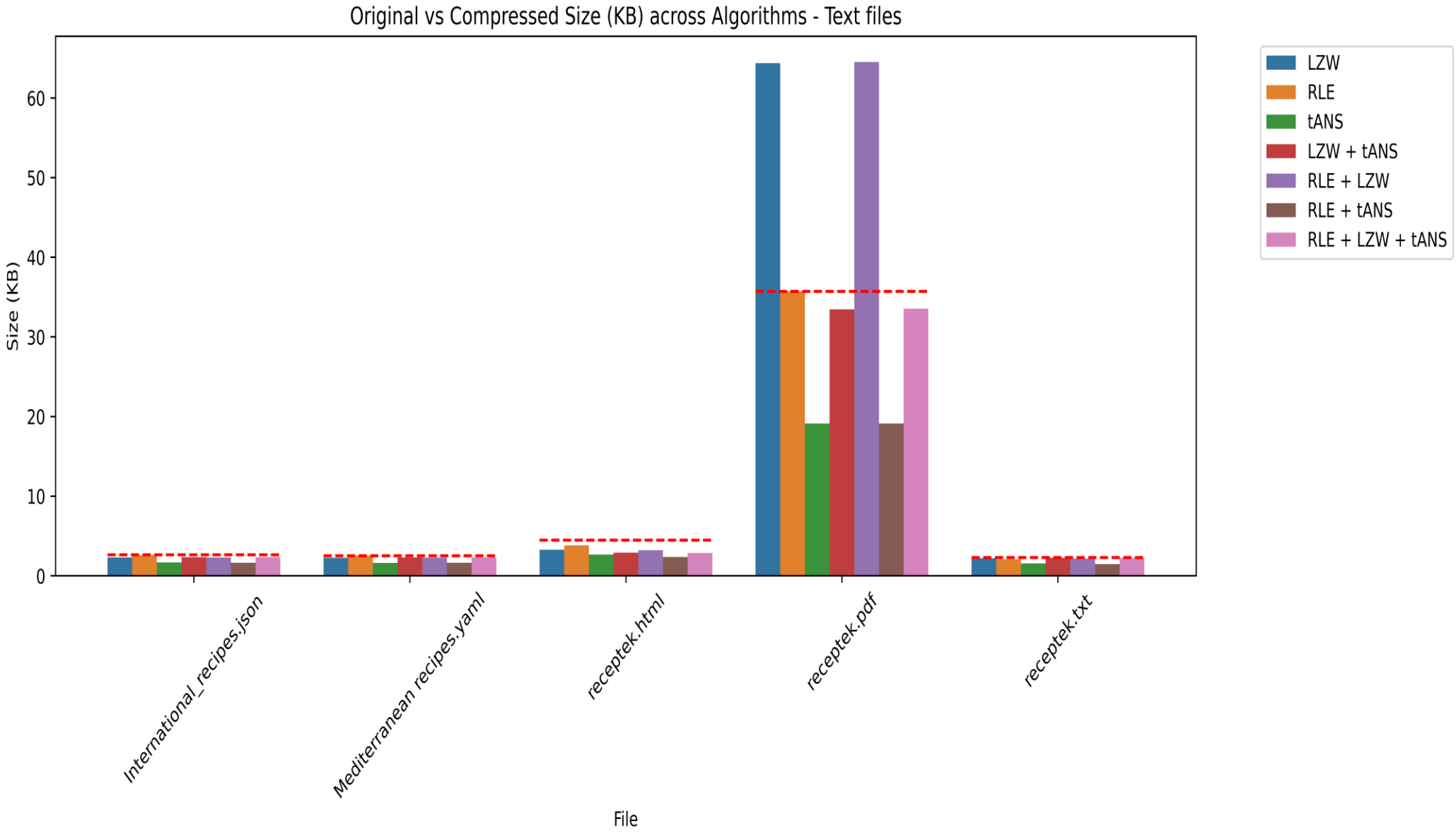
Results : Image files - Compression Ratio (%)



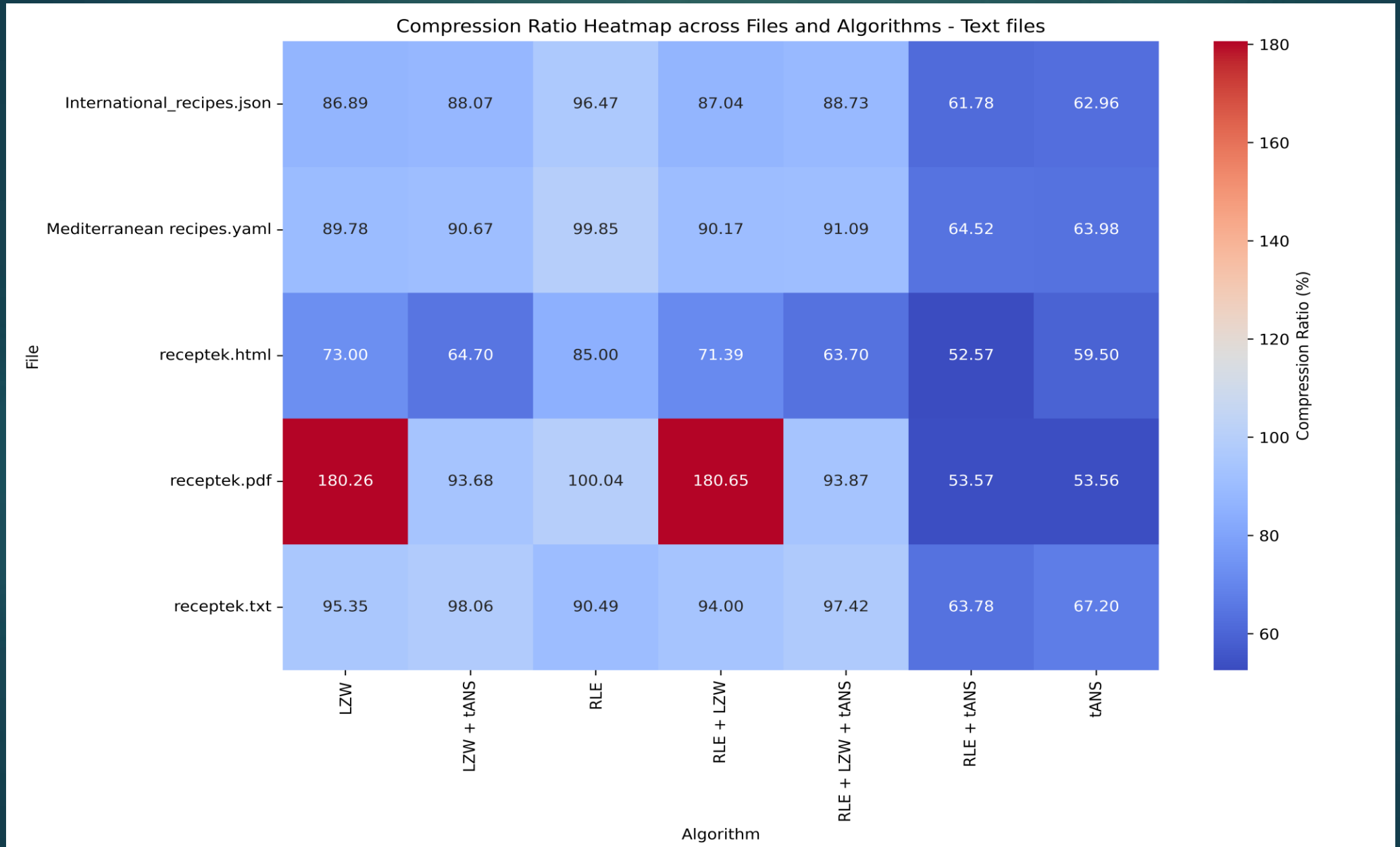
Results: Image files - Space saved (%)



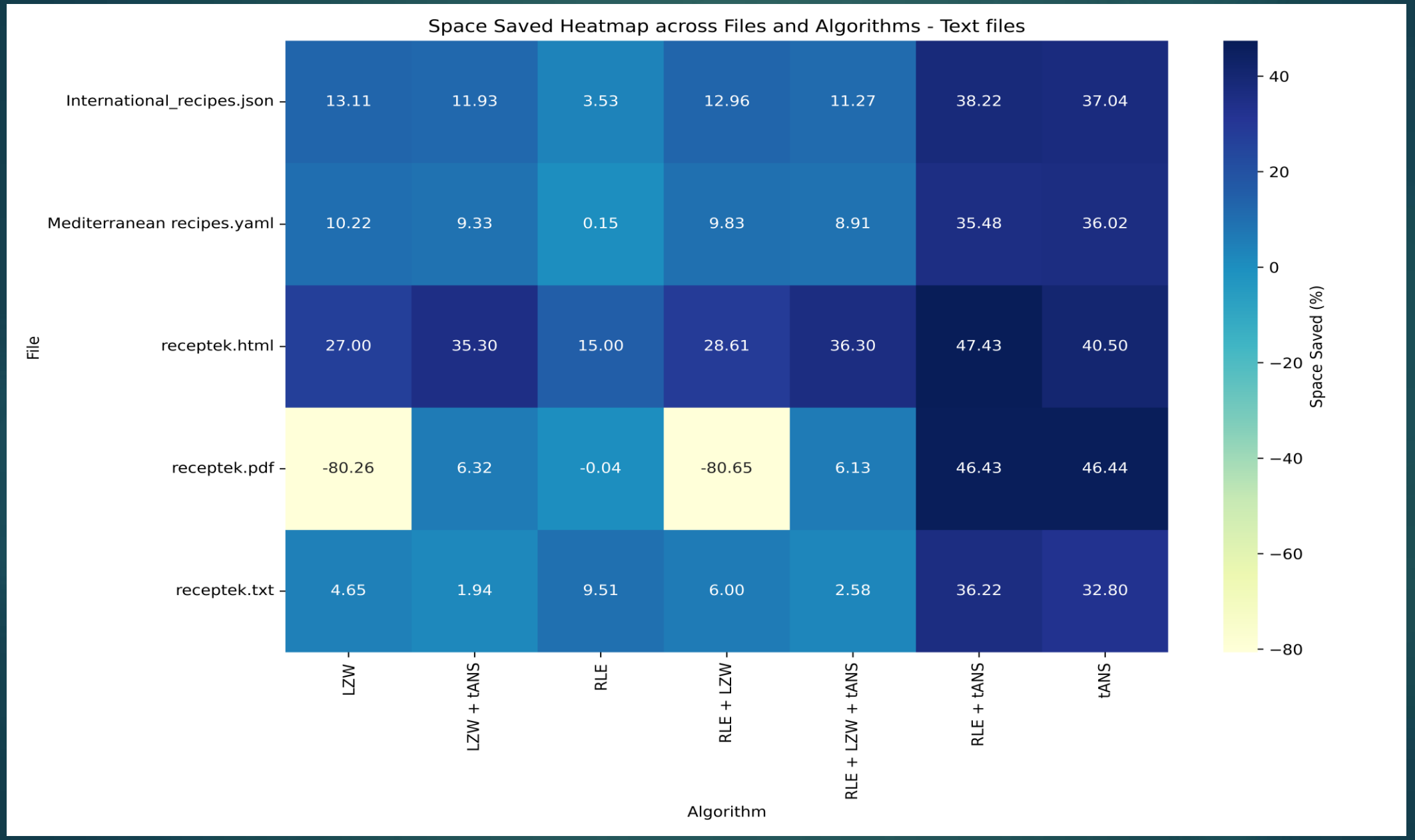
Results: Text files – Original vs Compressed size (KB)



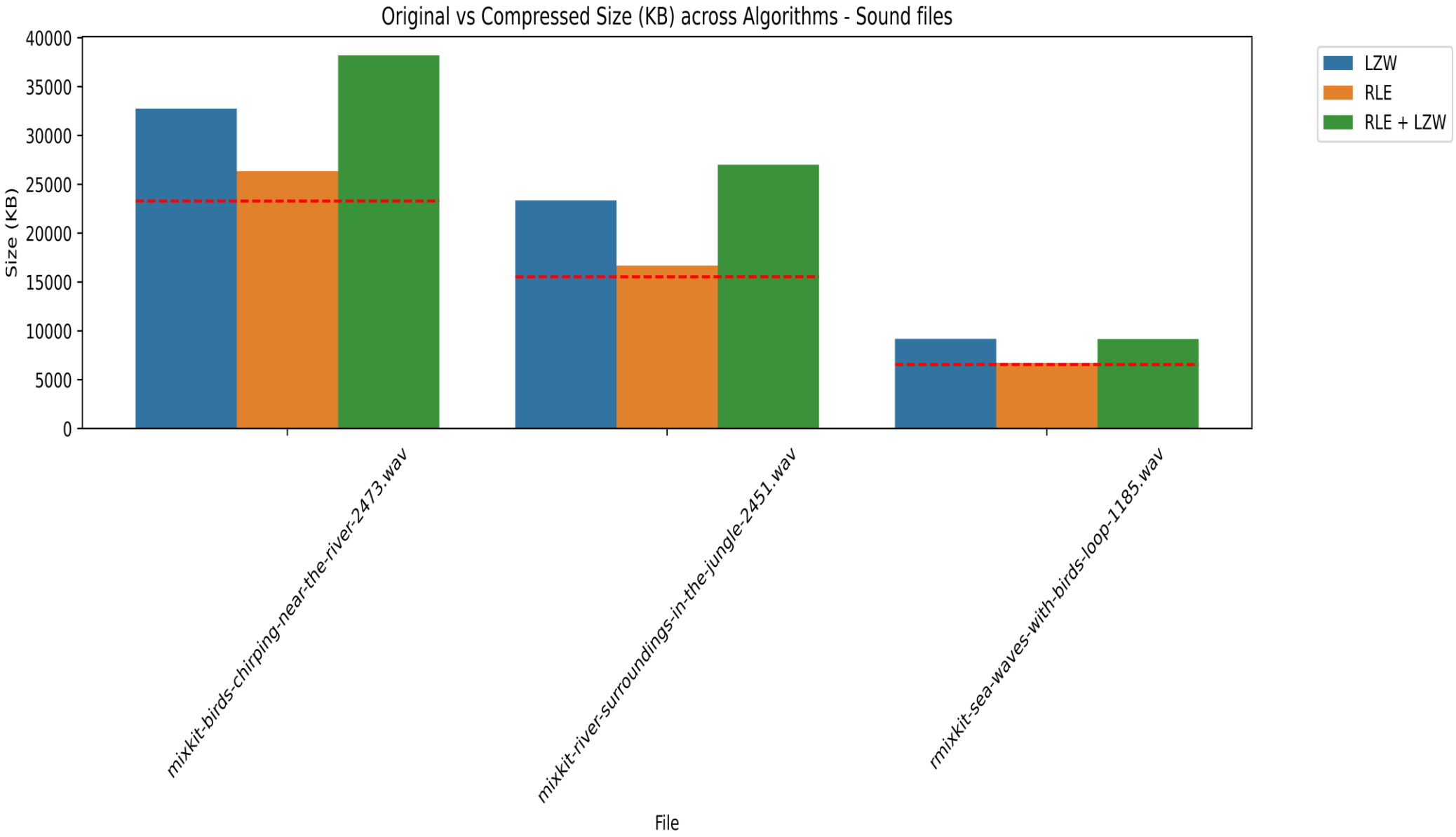
Results: Text files - Compression Ratio (%)



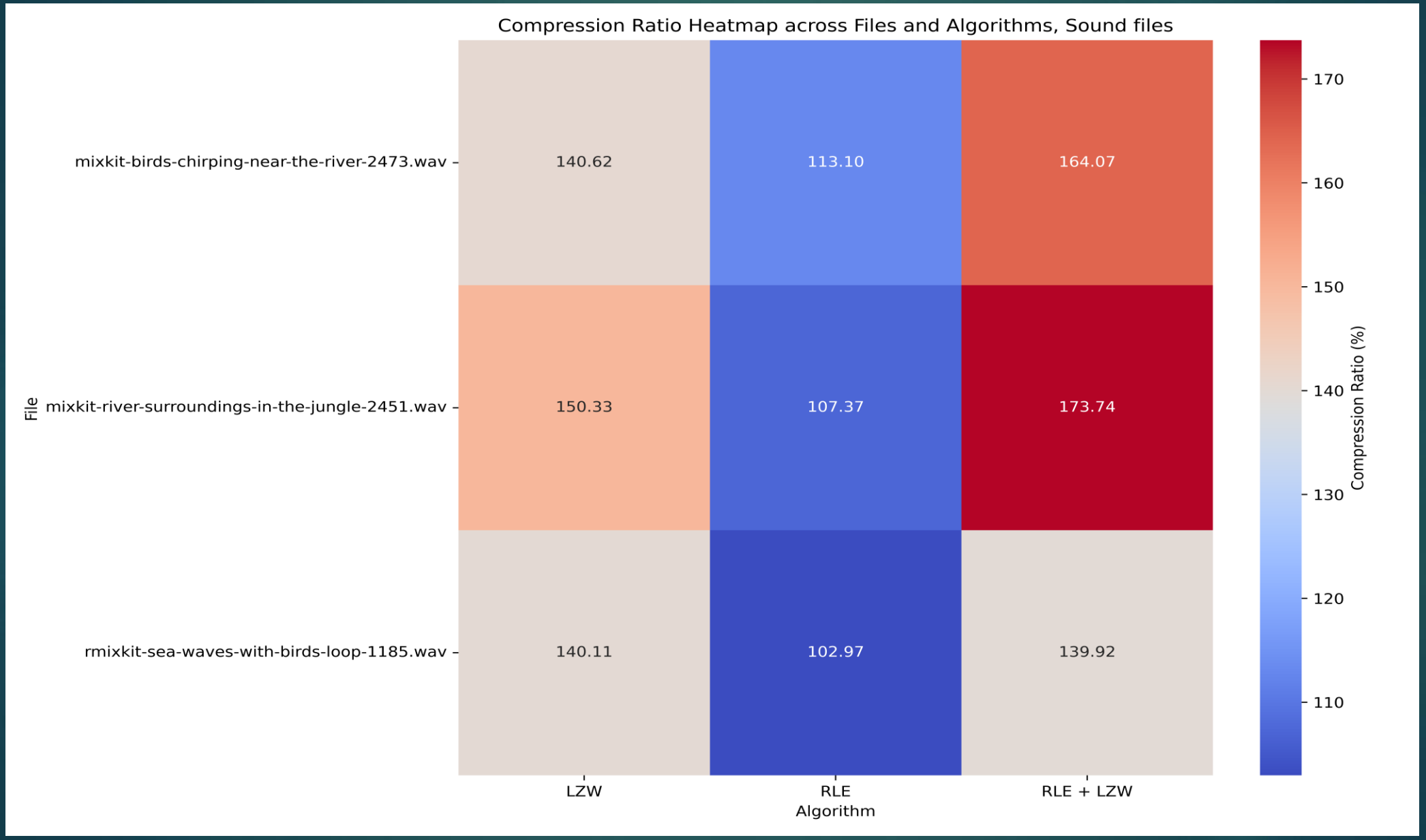
Results: Text files - Space saved (%)



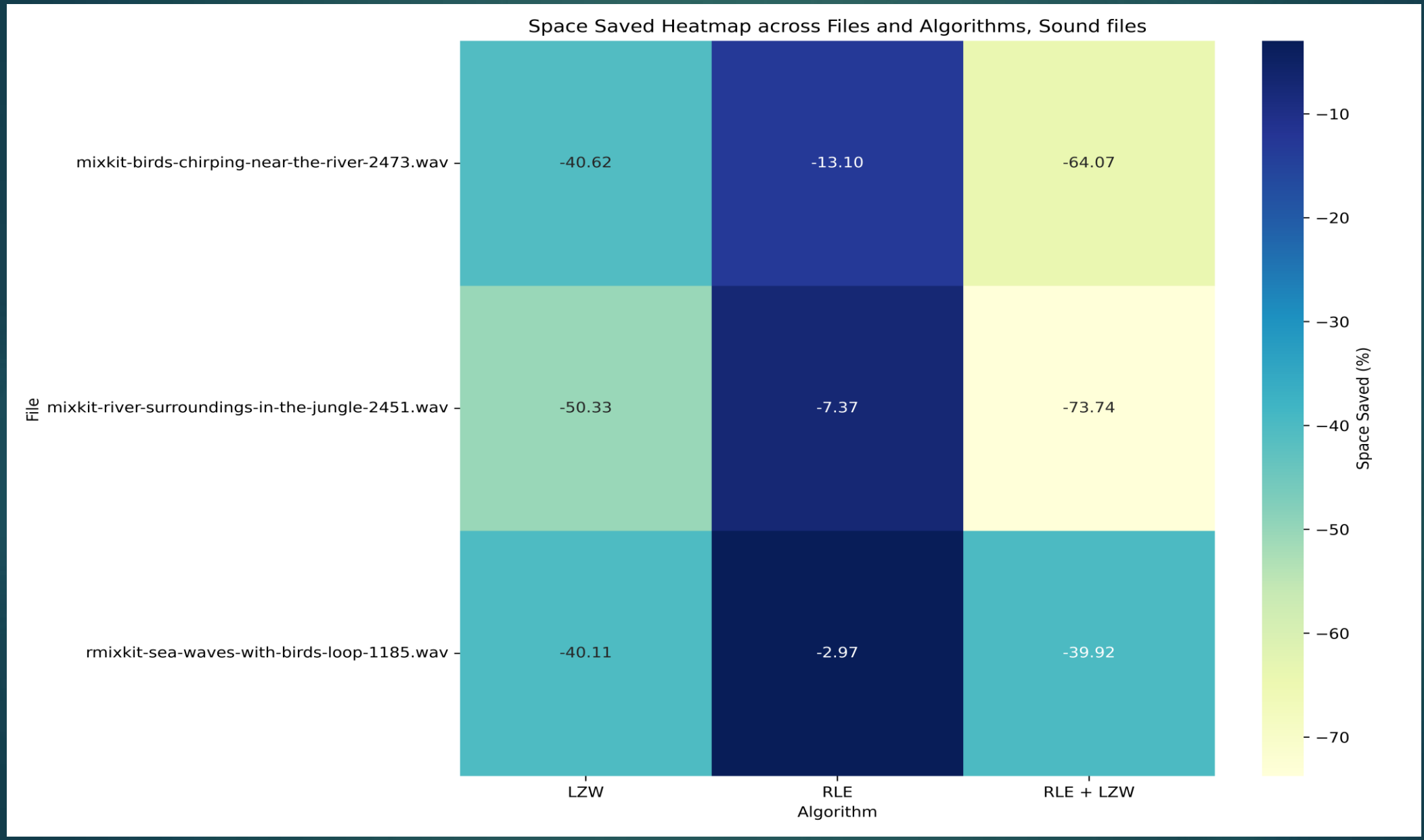
Results: Sound files – Original vs Compressed size



Results: Sound files - Compression Ratio(%)



Results: Sound files - Space saved (%)



Conclusion

- Many sample files were unsuitable for **tANS** compression due to:
  - Overly skewed data frequency, or
  - A high degree of "uniqueness" in the data (the data patterns were not as repetitive)
- **tANS** compression effectiveness is highly dependent on data type.
- **Binary files:**
  - Generally tolerable, most compressed well, but some actually increased in size after compression.
- **Text files:**
  - All compressed successfully with **tANS**, but results varied widely depending on file format.
- **Image and sound files:**
  - Compression with **tANS** failed in all cases because of the high variability in frequencies.

References

- [1] API Video, "What is run-length encoding (RLE)?", 2025. [Online]. Available: https://api.video/what-is/run-length-encoding/. [Accessed: Jun. 5, 2025].
- [2]Dremio, "Run-Length Encoding," 2025. [Online]. Available: https://www.dremio.com/wiki/run-length-encoding/. [Accessed: Jun. 5, 2025].
- [3] FastPix, "Understanding Run-Length Encoding: Data Compression Simplified," Feb. 19, 2025. [Online]. Available: https://www.fastpix.io/blog/what-is-run-length-encoding
- [4]R. Awati, "What is LZW compression and how does it work? – TechTarget Definition," TechTarget, July 2023. [Online]. Available: https://www.techtarget.com/whatis/definition/LZW-compression. [Accessed: Jun. 5, 2025].
- [5]GeeksforGeeks, "LZW (Lempel-Ziv-Welch) Compression technique - GeeksforGeeks," Apr. 2017. [Online]. Available: https://www.geeksforgeeks.org/lzw-lempel-ziv-welch-compression-technique/. [Accessed: Jun. 5, 2025].
- [6] T. A. Welch, "A technique for high-performance data compression," *Computer*, vol. 17, no. 6, pp. 8–19, 1984, doi: 10.1109/MC.1984.1659158.
- [7]J. Duda, "Asymmetric numeral systems: entropy coding combining speed of Huffman coding with compression rate of arithmetic coding," 2014, arXiv:1311.2540. [Online]. Available: https://arxiv.org/abs/1311.2540.
- [8] H. Yamamoto and K. Iwata, "Encoding and decoding algorithms of ANS variants and evaluation of their average code lengths," *arXiv preprint*, 2024, doi: 10.14923/transfunj.2024jai0001. [Online]. Available: https://arxiv.org/abs/2408.07322.

Project link: https://github.com/adamburgert/LZW-tANS-RLE_Compression