IEEE 14th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC-2021)
Singapore University of Technology and Design, Singapore

Text compression based on an alternative approach of run-length coding using Burrows-Wheeler transform and arithmetic coding

Md. Atiqur Rahman[1], Mohamed Hamada[1] & Md Asfaqur Rahman[2]
[1]The School of Computer Science and Engineering, Department of Computer and Information, The University of Aizu, Japan
[2]Department of Computer Science and Engineering, Islamic University, Kushtia, Bangladesh

Overview

- What is the importance of compression ?
- Background study
- Proposed coding
- Experimental results and analysis
- Conclusion

What is the importance of compression ?

If a video file is stored and the video contains the following properties.

- ✓Duration 30 minutes
- ✓Dimension 1000 x 1000
- ✓Color video
- ✓30 frames are moved in a second
- ✓Each pixel is coded in 8 bits

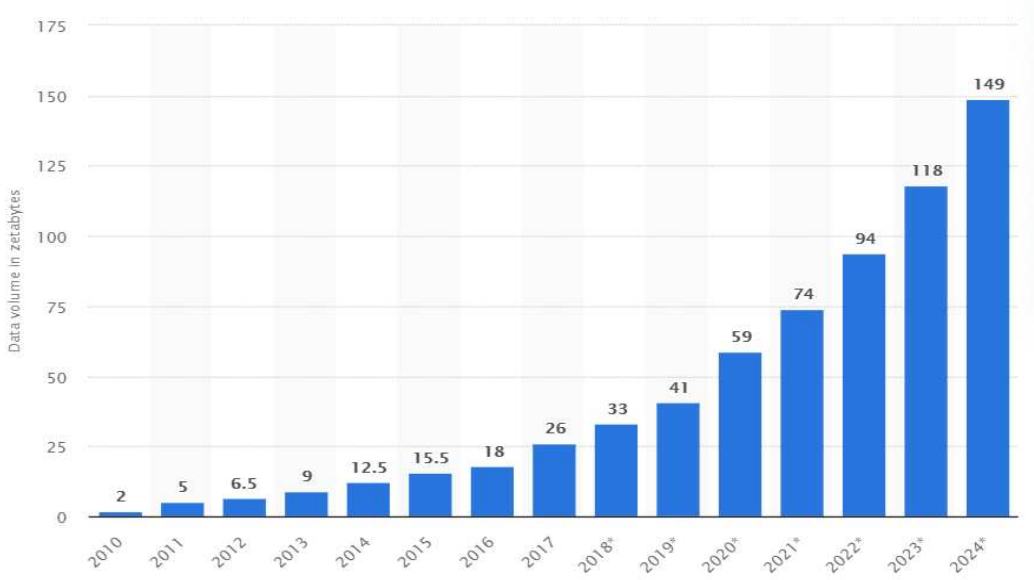
The video file will allocate approximately 150.87 GB memory.

According to DOMO's report, it was estimated that over 2.5 quintillion bytes of data are created every single day, and 1.7MB of data will be created every second for every person on earth by 2020.

3

What is the importance of compression ?

Statista, a leading provider of market and consumer data.



4

BACKGROUND STUDIES

Data compression is a way to convert an original file to another format so that the converted file uses less space than the original file with or without degrading its quality.

Data compression can be classified into two categories.

- Lossless
- Lossy

Lossy compression denotes significant data compression with the loss of relatively unimportant details; it can be utilized to compress text files, audio files, images, and videos. However, with regard to text compression, lossless compression is the only choice. Lossless compression is based on the removal of redundant data, so it focuses on the existence of redundant data

BACKGROUND STUDIES

The most popular text compression algorithms are LZW, Gzip, Bzip2, Lempel–Ziv–Markov chain algorithm (LZMA), Brotli etc. Many of them concentrate on the compression ratios, while others focus on speed.

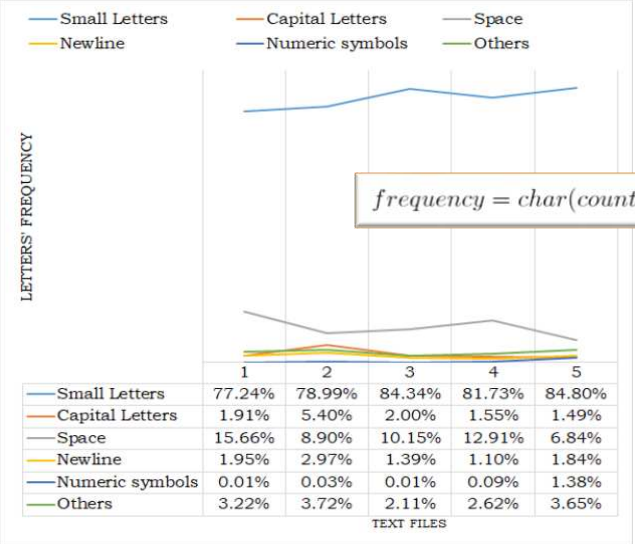
I. LZW is a dictionary-based lossless text compression algorithm and an updated version of LZ78. Although LZW is a good text compression technique, it is more complicated due to its searching complexity.

II. Gzip is another lossless, Deflate-based text compression algorithm that compresses a text while using LZ77 and Huffman coding. The main limitation of Deflate is that the longer and duplicate substring searching is a very lazy mechanism.

III. The Lempel–Ziv–Markov chain algorithm (LZMA) is a dictionary-based text compression technique that is similar to LZ77. LZMA uses a comparatively small amount of memory for decoding and it is very good for embedded applications.

IV. Bzip2, on the other hand, compresses only a single file using the Burrows-Wheeler transform, the move-to-front (MTF) transform and Huffman entropy coding techniques. Although bzip2 compresses more effectively than LZW, it is slower than Deflate but faster than LZMA.

Proposed method



	1	2	3	4	5
Small Letters	77.24%	78.99%	84.34%	81.73%	84.80%
Capital Letters	1.91%	5.40%	2.00%	1.55%	1.49%
Space	15.66%	8.90%	10.15%	12.91%	6.84%
Newline	1.95%	2.97%	1.39%	1.10%	1.84%
Numeric symbols	0.01%	0.03%	0.01%	0.09%	1.38%
Others	3.22%	3.72%	2.11%	2.62%	3.65%

Fig. 1: Comparison of the frequencies of letters in text files

$$frequency = char(count + 96) \quad (1)$$

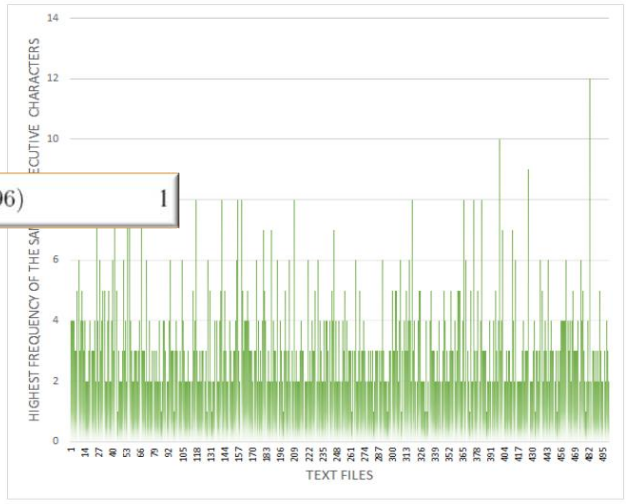
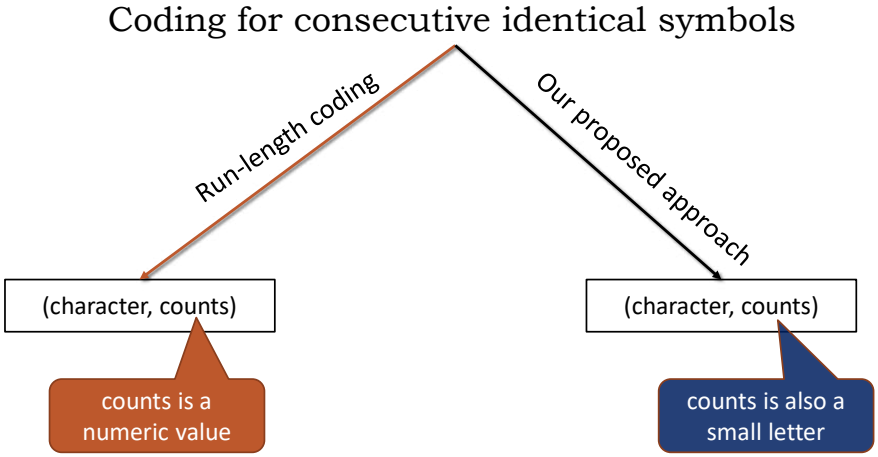
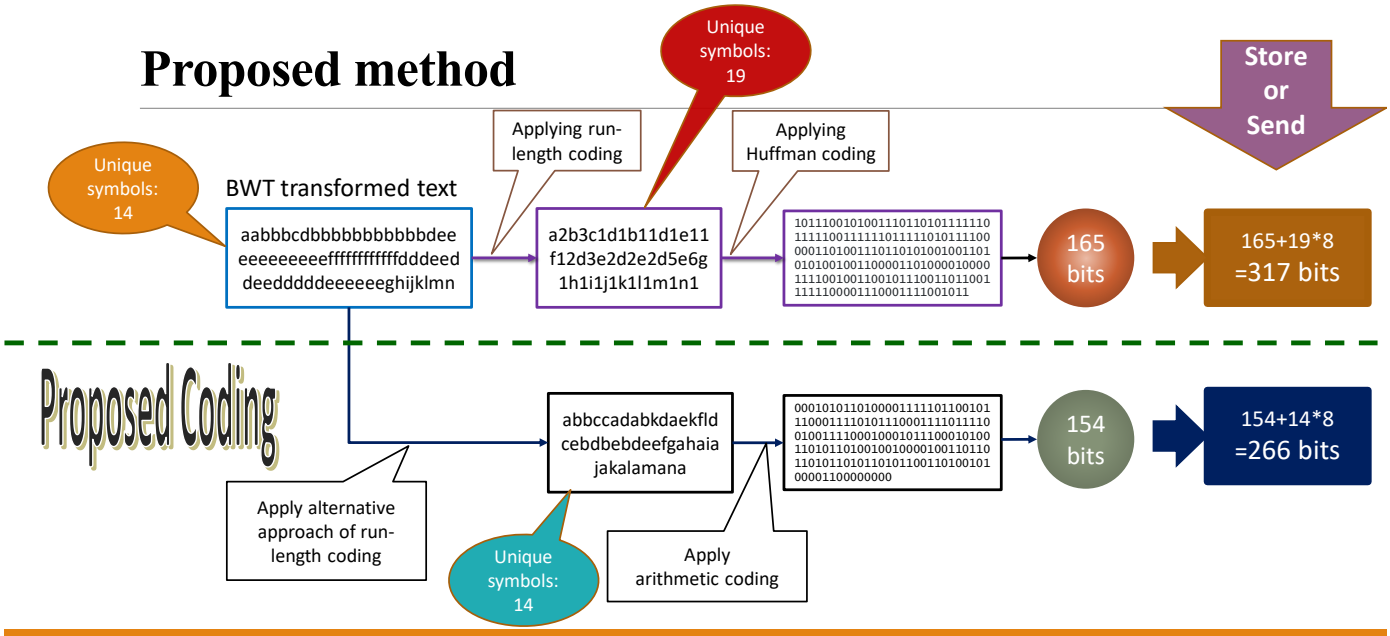


Fig. 2: Maximum frequencies of consecutive identical symbols in text files after the Burrows-Wheeler transform is applied

Proposed method



Proposed method



Proposed method

Algorithm 3: Proposed encoding approach

1 Read a text file, calculate the length (textlen), and set J =1;
2 Split the text into small files of the same length (SL), apply BWT to each file individually, and then merge all the small files;
3 **while** $J \leq textlen$ **do**
4 *Count the identical consecutive characters and store the two symbols (the character and char(count+96)) to an array (A);*
5 *J=J+count;*
6 **end**
7 *Apply the pattern matching technique to further reduce the array (A);*
8 *Apply arithmetic encoding to the array (A) to get an encoded bit-stream;*
9

Burrows–Wheeler transform (BWT), a reversible technique that converts a set of characters into runs of similar characters.

Proposed method

Algorithm 4: Proposed decoding approach

```
Apply arithmetic decoding approach to the encoded
  bit-stream ;
Set J=1 and calculate the length (DSlen) of the
  decoded string (DString);
while J ≤ DSlen do
    Store DString[J] to an array (Reconstructedtext[J])
      Set I = 1;
    while I < (DString[J+1]-96) do
        Store DString[J] to an array
          (Reconstructedtext[J+I])
    end
    J=J+(DString[J+1]-96)+1;
end
Replace the patterns in the Reconstructedtext;
Split the array (Reconstructedtext) into a set of small
  files of size (SL);
Apply the inverse BWT to each file separately;
Merge the files into one file;
```


Experimental Results and Analysis

TABLE I: Comparison of results in terms of bits per symbol

Text Files	PAQ8n	Bzip2	Deflate	Gzip	LZW	LZMA	Brotli	Proposed
1	0.615	0.718	0.629	0.66	0.748	0.77	0.612	0.588
2	0.664	0.795	0.668	0.68	0.771	0.799	0.619	0.537
3	0.57	0.682	0.586	0.612	0.707	0.727	0.578	0.582
4	0.643	0.767	0.673	0.69	0.772	0.823	0.643	0.573
5	0.643	0.783	0.688	0.683	0.818	0.792	0.611	0.594
6	0.785	0.912	0.785	0.829	0.926	0.932	0.771	0.802
7	0.834	0.955	0.936	0.904	0.871	0.987	0.72	0.723
8	0.632	0.743	0.682	0.669	0.764	0.747	0.544	0.559
9	0.719	0.862	0.754	0.76	0.754	0.934	0.695	0.753
10	0.648	0.751	0.712	0.698	0.834	0.801	0.655	0.642
Average	**0.675**	**0.797**	**0.711**	**0.719**	**0.797**	**0.831**	**0.645**	**0.635**

More bits per symbol

Table I shows that on average, the Brotli text compression method gave the lowest BPS of 0.645, a result that was 4.65%, 23.57%, 10.23%, 11.47%, 23.57%, and 28.84% better than PAQ8n, Bzip2, Deflate, Gzip, LZW, and LZMA, respectively. However, our technique gives a BPS of 0.635, which is 1.57% better than Brotli.

Conclusion

- In this paper, we have proposed a text compression strategy using the BWT, an alternative approach to run-length coding, and arithmetic coding.
- This technique can be used for any cases of text compression.