Burrows–Wheeler Transform Based Lossless Text Compression Using Keys and Huffman Coding.

Abstract: Text compression is one of the most significant research fields, and various algorithms for text compression have already been developed. This is a significant issue, as the use of internet bandwidth is considerably increasing. This article proposes a Burrows–Wheeler transform and pattern matching-based lossless text compression algorithm that uses Huffman coding in order to achieve an excellent compression ratio. In this article, we introduce an algorithm with two keys that are used in order to reduce more frequently repeated characters after the Burrows–Wheeler transform. We then find patterns of a certain length from the reduced text and apply Huffman encoding. We compare our proposed technique with state-of-the-art text compression algorithms.

Finally, we conclude that the proposed technique demonstrates a gain in compression ratio when

compared to other compression techniques. A small problem with our proposed method is that it

does not work very well for symmetric communications like Brotli.

1. Introduction

Managing the increasing amount of data that are produced by modern daily life activities is

not a simple task for symmetric communications. In articles [1,2], it is reported that, on average,

4.4 zettabytes and 2.5 exabytes of data were produced per day in 2013 and 2015, respectively. On the

other hand, the use of the internet is increasing. The total numbers of internet users were 2.4, 3.4,

and 4.4 billion in 2014, 2016, and 2019, respectively [3]. Though hardware manufacturing, companies

are producing plenty of hardware in an attempt to provide a better solution for working with huge amounts of data, it’s almost impossible to maintain this data without compression.

Compression is the representation of data in a reduced form, so that data can be saved while using a small amount of storage and sent with a limited bandwidth [4–7]. There are two types of compression techniques: lossless and lossy [8,9]. Lossless compression reproduces data perfectly from its encoded bit stream, and, in lossy compression, less significant information is removed [10,11]. There are various types of lossless text compression techniques, such as the Burrows–Wheeler transform (BWT), run-length coding, Huffman coding, arithmetic coding, LZ77, Deflate, LZW, Gzip, Bzip2, Brotli, etc. [12,13]. Some statistical methods assign a shorter binary code of variable length to the most frequently repeated characters. Huffman and arithmetic coding are two examples of this type of statistical method. However, Huffman coding is one of the best algorithms in this category [14].

Some dictionary-based methods, such as LZ77, LZW, etc., create a dictionary of substrings and assign them a particular pointer based on the substring's frequency. In [15], Robbi et al. propose a Blowfish encryption and LZW-based text compression procedure and show a better result than LZW coding.

Deflate provides a slightly poor compression, but its encoding and decoding speeds are fast.

2. PreviousWorks

Run-length coding is one of the text compression algorithms. It calculates symbols and their counts. When run-length coding is directly applied to data for compression, it sometimes takes more storage than the original data [17]. Shannon–Fano coding generates prefix codes of variable length based on probabilities and provides better results than run-length coding, but it is not optimal, as it cannot produce the same tree in encoding and decoding. David Huffman developed a data compression algorithm, reported in [18], which is normally used as a part of many compression

techniques. In this technique, a binary tree is generated by connecting the two lowest probabilities at a

time when the root of the tree contains the summation of the two probabilities. The tree is then used to

encode each symbol without ambiguity. However, Huffman coding cannot achieve an optimal code

length when it is applied directly. Arithmetic coding outperforms Huffman coding in terms of average

code length, but it takes a huge amount of time for encoding and decoding.

LZW is a dictionary-based lossless text compression algorithm and an updated version of

LZ78 [19]. In this technique, a dictionary is created and initialized with strings all of length one.

Subsequently, the longest string in the dictionary that matches the current input data is found.

Although LZW is a good text compression technique, it is more complicated due to its searching

complexity [20]. Deflate is also a lossless compression algorithm that compresses a text by using LZSS

and Huffman coding together, where LZSS is a derivative of LZ77. The Deflate procedure finds all of the

duplicate substrings from a text. Subsequently, all of the substrings are replaced by the pointer of the

substring that occurred first. The main limitation of Deflate is that the longer and duplicate substring

searching is a very lazy mechanism [21]. Gzip is another lossless, Deflate-based text compression

algorithm that compresses a text while using LZ77 and Huffman coding [22]. The pseudo-code of

LZ77 is reported in the reference [23].

The Lempel–Ziv–Markov chain algorithm (LZMA) that was developed by Igor Pavlov 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 [24]. 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 [25]. PAQ8n is a lossless text compression method that incorporates the JPEG model into paq81. The main limitation of PAQ8n is that it is very slow [26,27]. Brotli, which was developed by Google, is a lossless text compression method that performs compression using the lossless mode of LZ77, a Huffman entropy coding technique and second order context modeling [28]. Brotli utilizes a predefined static dictionary holding 13,504 common words [29,30]. It cannot compress large files well because of its limited sliding window [31].

Burrows–Wheeler transform (BWT) in [32] transforms a set of characters into runs of identical

characters. It is completely reversible, and no extra information is stored without the position of

the last character. The transformed character set can be easily compressed by run-length coding.

Proposed Method

There are many algorithms used to compress a text. Some examples are Huffman, run-length,

LZW, Bzip2, Deflate, Gzip, etc. coding-based algorithms [34–38]. Many algorithms focus on the

encoding or decoding speed during text compression, while others concentrate on the average

code length. Brotli provides better compression ratios than other state-of-the-art techniques for

text compression. However, it uses a large static dictionary [30]. What makes our proposal special?

We can apply our proposed method to a large file as well as a small file. The main limitation of BWT

is that it takes huge amounts of storage and a lot of time to transform a large file [7,39,40]. Our first

interesting innovation is that we split a large file into a set of smaller files, where each file contains the

same number of characters, and then apply the Burrows–Wheeler transform (BWT) to the smaller files

individually to speed up the transformation. We do not use any static dictionary because searching

for a word or phrase in a dictionary is very complicated and time consuming [14]. We change the use

of run-length coding a bit after the Burrows–Wheeler transform (BWT), because run-length coding

takes the symbol and its count, and it only works well when characters are repeated more in a text.

When a character is alone in a text, which normally happens, two values (the character and its count)

are stored after encoding, which increases the use of storage. Our second interesting change is that we

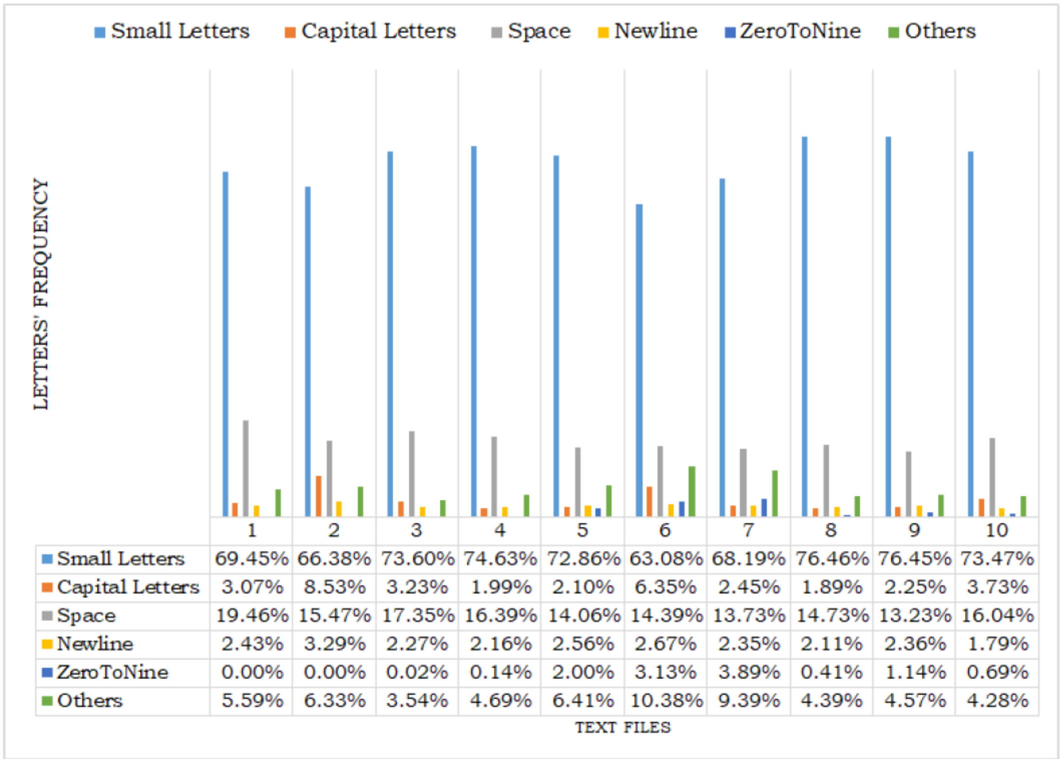
will only replace the characters repeated more than three times in a row by a key, the character, and its

count. The position of the character’s sequence in a reduced text is identified by the key. Huffman

coding provides more compression if a text has a higher frequency of characters. We have analyzed ten

large text files from [41], and the outcomes are shown in Figure 1. The figure shows that the frequency

of lowercase letters in any text is always much higher than other types of letters.



	1	2	3	4	5	6	7	8	9	10
Small Letters	69.45%	66.38%	73.60%	74.63%	72.86%	63.08%	68.19%	76.46%	76.45%	73.47%
Capital Letters	3.07%	8.53%	3.23%	1.99%	2.10%	6.35%	2.45%	1.89%	2.25%	3.73%
Space	19.46%	15.47%	17.35%	16.39%	14.06%	14.39%	13.73%	14.73%	13.23%	16.04%
Newline	2.43%	3.29%	2.27%	2.16%	2.56%	2.67%	2.35%	2.11%	2.36%	1.79%
ZeroToNine	0.00%	0.00%	0.02%	0.14%	2.00%	3.13%	3.89%	0.41%	1.14%	0.69%
Others	5.59%	6.33%	3.54%	4.69%	6.41%	10.38%	9.39%	4.39%	4.57%	4.28%

Figure 1 shows that, on average, all of the files contain 71.46%, 3.56%, 15.48%, 2.40%, 1.14%,

and 5.96% small letters, capital letters, space, newline, zero to nine, and others, respectively. We have

calculated that, averagely, the frequency of lowercase letters is 94.95%, 78.11%, 96.67%, 98.31%, and 91.72% higher than the frequency of capital letters, spaces, newlines, zero-to-nine (0–9) characters,

and other characters, respectively.

Additionally, we have analyzed 5575 small text files of lengths less than or equal to 900 characters.

We see that a maximum of twenty of the same characters are repeated consecutively at a time after

the Burrows–Wheeler transform is applied to the small texts that are shown in Figure 2. There are

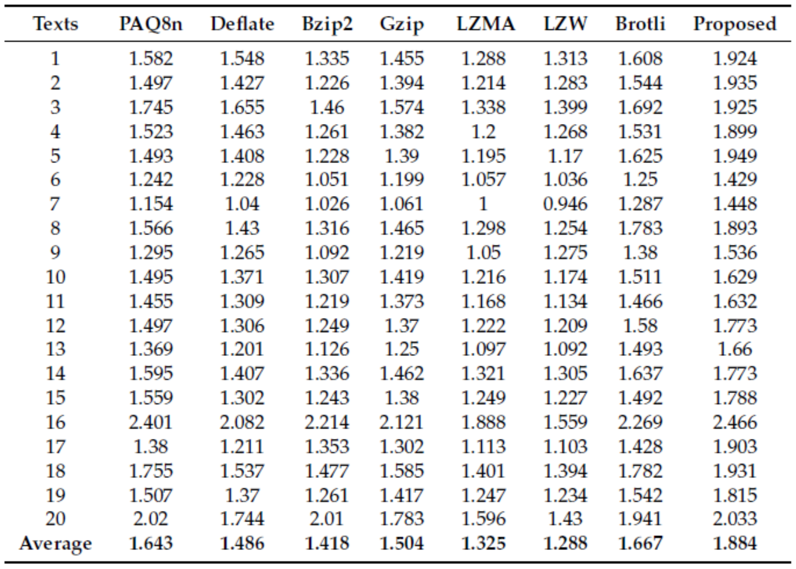
twenty-six lowercase characters in the English alphabet. Accordingly, we have replaced the character

count in lowercase letters using the formula (character's count + 96), so that the lowercase letters keep

the frequency higher and we can obtain a higher compression ratio. The proposed above-mentioned

idea can only reduce the characters repeated four times or more at a time. However, a file can contain many other characters that can be repeated two or three times. Our third interest is to reduce the characters that are repeated exactly three times using the formula (second key, the character). As a result, we can only store two characters instead of three and further reduce the length of the file. Changing characters that appear twice does not help to reduce the file length, so we keep these characters the same.

Table 1. Comparison among compression ratios.



Texts	PAQ8n	Deflate	Bzip2	Gzip	LZMA	LZW	Brotli	Proposed
1	1.582	1.548	1.335	1.455	1.288	1.313	1.608	1.924
2	1.497	1.427	1.226	1.394	1.214	1.283	1.544	1.935
3	1.745	1.655	1.46	1.574	1.338	1.399	1.692	1.925
4	1.523	1.463	1.261	1.382	1.2	1.268	1.531	1.899
5	1.493	1.408	1.228	1.39	1.195	1.17	1.625	1.949
6	1.242	1.228	1.051	1.199	1.057	1.036	1.25	1.429
7	1.154	1.04	1.026	1.061	1	0.946	1.287	1.448
8	1.566	1.43	1.316	1.465	1.298	1.254	1.783	1.893
9	1.295	1.265	1.092	1.219	1.05	1.275	1.38	1.536
10	1.495	1.371	1.307	1.419	1.216	1.174	1.511	1.629
11	1.455	1.309	1.219	1.373	1.168	1.134	1.466	1.632
12	1.497	1.306	1.249	1.37	1.222	1.209	1.58	1.773
13	1.369	1.201	1.126	1.25	1.097	1.092	1.493	1.66
14	1.595	1.407	1.336	1.462	1.321	1.305	1.637	1.773
15	1.559	1.302	1.243	1.38	1.249	1.227	1.492	1.788
16	2.401	2.082	2.214	2.121	1.888	1.559	2.269	2.466
17	1.38	1.211	1.353	1.302	1.113	1.103	1.428	1.903
18	1.755	1.537	1.477	1.585	1.401	1.394	1.782	1.931
19	1.507	1.37	1.261	1.417	1.247	1.234	1.542	1.815
20	2.02	1.744	2.01	1.783	1.596	1.43	1.941	2.033
Average	**1.643**	**1.486**	**1.418**	**1.504**	**1.325**	**1.288**	**1.667**	**1.884**

Table 1 shows that averagely LZW provides the lowest (1.288) compression ratio and Brotli the highest (1.667) among state-of-the-art techniques. Although PAQ8n provides 3.04%, 4.3%, 5.5%,

and 3.91% better results than Brotli for the texts 3, 15, 16, and 20, respectively, Brotli shows 1.44%,

10.86%, 14.94%, 9.78%, 20.52%, and 22.74% more compression than PAQ8n, Deflate, Bzip2, Gzip, LZMA,

and LZW, on average. It can be seen that the proposed technique provides better results, having

a higher (1.884) compression ratio on average. Specifically, the proposed technique demonstrates,

on average, a compression ratio 12.79% higher than PAQ8n, 21.13% higher than Deflate, 24.73% higher

than Bzip2, 20.17% higher than Gzip, 31.63% higher than LZMA, 31.7% higher than LZW, and 11.52%

higher than Brotli. We can see from Figure 6 that the compression ratio for the proposed technique is

higher for every sample.

Lossless text compression is a more significant matter when there is a highly narrow-band

communication channel and less storage available. We have proposed a completely lossless

text compression while using the Burrows–Wheeler transform, two keys, and Huffman coding.

What distinguishes our proposed method? First, to speed up the transformation, we split a large text

file into sets of smaller files that ultimately increase the speed of compression. Second, we do not count

all characters, which is done in run-length coding. We count only the characters that are repeated more

than two times consecutively and replace the value of the letter count by a lowercase letter to increase

the frequency of characters in the text, as each text contains the maximum number of lowercase letters.

Third, we look for patterns of a certain length that have the highest frequency, so that we can get better results after applying Huffman coding.

The experimental outcomes show that the proposed method performs better than the seven

algorithms that we compared it to: PAQ8n, Deflate, Bzip2, Gzip, LZMA, LZW, and Brotli. When used

on the twenty sample texts, the proposed method gives an average of 21.66% higher compression than

the methods that were described in this article.

References

1. Northeastern University Graduate Programs. HowMuch Data Is Produced Every Day? 2020. Available online:

https://www.northeastern.edu/graduate/blog/how-much-data-produced-every-day/ (accessed on

17 September 2020).

2. Walker, B. Every day big data statistics—2.5 quintillion bytes of data created daily. VCloudNews 2015,

Available online:
https://www.dihuni.com/2020/04/10/every-day-big-data-statistics-2-5-quintillionbytes-

of-data-created-daily/(accessed on 10 September 2020).

3. Blog.microfocus.com. How Much Data Is Created on The Internet Each Day? Micro Focus Blog.

2020. Available online:
https://blog.microfocus.com/how-much-data-is-created-on-the-internet-each-day/

(accessed on 18 May 2020).

4. Pu, I.M. Fundamental Data Compression; Butterworth-Heinemann: Oxford, UK, 2005.

5. Salomon, D.; Motta, G. Handbook of Data Compression; Springer Science & Business Media: Berlin/Heidelberg,

Germany, 2010.

6. Porwal, S.; Chaudhary, Y.; Joshi, J.; Jain, M. Data compression methodologies for lossless data and comparison

between algorithms. Int. J. Eng. Sci. Innov. Technol. (IJESIT) 2013, 2, 142–147.

7. Sayood, K. Introduction to Data Compression; Morgan Kaufmann: Burlington, MA, USA, 2017.

8. Rahman, M.A.; Rabbi, M.F.; Rahman, M.M.; Islam, M.M.; Islam, M.R. Histogram modification based lossy

image compression scheme using Huffman coding. In Proceedings of the 2018 4th International Conference

on Electrical Engineering and Information & Communication Technology (iCEEiCT), Dhaka, Bangladesh,

13–15 September 2018; pp. 279–284.

9. Rahman, M.A.; Islam, S.M.S.; Shin, J.; Islam, M.R. Histogram Alternation Based Digital Image Compression

using Base-2 Coding. In Proceedings of the 2018 Digital Image Computing: Techniques and Applications

(DICTA), Canberra, Australia, 10–13 December 2018; pp. 1–8.

10. Sadchenko, A.; Kushnirenko, O.; Plachinda, O. Fast lossy compression algorithm for medical images.

In Proceedings of the 2016 International Conference on Electronics and Information Technology (EIT),

Odessa, Ukraine, 23–27 May 2016; pp. 1–4.

11. Pandey, M.; Shrivastava, S.; Pandey, S.; Shridevi, S. An Enhanced Data Compression Algorithm.

In Proceedings of the 2020 International Conference on Emerging Trends in Information Technology and

Engineering (ic-ETITE), Tamil Nadu, India, 24–25 February 2020; pp. 1–4.

12. Oswald, C.; Sivaselvan, B. An optimal text compression algorithm based on frequent pattern mining.

J. Ambient. Intell. Humaniz. Comput. 2018, 9, 803–822. [CrossRef]

13. Portell, J.; Iudica, R.; García-Berro, E.; Villafranca, A.G.; Artigues, G. FAPEC, a versatile and efficient data

compressor for space missions. Int. J. Remote Sens. 2018, 39, 2022–2042. [CrossRef]

14. Rahman, M.; Hamada, M. Lossless image compression techniques: A state-of-the-art survey. Symmetry 2019,

11, 1274. [CrossRef]