





Article

Comparison of Entropy and Dictionary Based Text Compression in English, German, French, Italian, Czech, Hungarian, Finnish, and Croatian

Matea Ignatoski ¹, Jonatan Lerga ^{1,2,*}, Ljubiša Stanković ³ and Miloš Daković ³

¹ Department of Computer Engineering, Faculty of Engineering, University of Rijeka, Vukovarska 58, HR-51000 Rijeka, Croatia; mignatoski@riteh.hr

² Center for Artificial Intelligence and Cybersecurity, University of Rijeka, R. Matejčić 2, HR-51000 Rijeka, Croatia

³ Faculty of Electrical Engineering, University of Montenegro, Džordža Vašingtona bb, 81000 Podgorica, Montenegro; ljubisa@ac.me (L.S.); milos@ac.me (M.D.)

* Correspondence: jlerga@riteh.hr; Tel.: +385-51-651-583

Received: 3 June 2020; Accepted: 17 June 2020; Published: 1 July 2020



Abstract: The rapid growth in the amount of data in the digital world leads to the need for data compression, and so forth, reducing the number of bits needed to represent a text file, an image, audio, or video content. Compressing data saves storage capacity and speeds up data transmission. In this paper, we focus on the text compression and provide a comparison of algorithms (in particular, entropy-based arithmetic and dictionary-based Lempel–Ziv–Welch (LZW) methods) for text compression in different languages (Croatian, Finnish, Hungarian, Czech, Italian, French, German, and English). The main goal is to answer a question: “How does the language of a text affect the compression ratio?” The results indicated that the compression ratio is affected by the size of the language alphabet, and size or type of the text. For example, The European Green Deal was compressed by 75.79%, 76.17%, 77.33%, 76.84%, 73.25%, 74.63%, 75.14%, and 74.51% using the LZW algorithm, and by 72.54%, 71.47%, 72.87%, 73.43%, 69.62%, 69.94%, 72.42% and 72% using the arithmetic algorithm for the English, German, French, Italian, Czech, Hungarian, Finnish, and Croatian versions, respectively.

Keywords: arithmetic; Lempel–Ziv–Welch (LZW); text compression; encoding; English; German; French; Italian; Czech; Hungarian; Finnish; Croatian

1. Introduction

We live in a digital age. One of the main characteristics of this age is the ability of individuals to exchange information freely. Daily, dozens of billions of digital messages are exchanged, photos are taken, articles are written, and so forth. These activities produce data that must be stored or transmitted. As data size increases, the cost of data storage and transmission time increases. To prevent these problems, we need to compress the data [1–4]. Data compression is the process of reducing the quantity of data used to represent a text file, an image, audio, or video content. There are two categories of data compression methods—lossy and lossless [5,6]. Lossy compression methods reduce the file size by removing some of the file’s original data [7,8]. The resulting file cannot be completely reconstructed. Lossy compression methods are generally used for compressing file types where data loss is not noticeable, such as video, audio, and image files. On the other hand, lossless data compression methods also reduce file size, but they also preserve file content, so there is no data loss when data is uncompressed. Of course, text files must be compressed using lossless data compression methods [9–12].

In this paper, we focus on text file compression, and therefore we study lossless compression methods. There are a few commonly used lossless compression methods such as Shannon-Fano Coding, Huffman Coding, Lempel–Ziv–Welch (LZW) Algorithm, and Arithmetic Coding [13–17]. The optimal compression method depends on many factors, including the text length and amount of repeating characters. Previous work has focused on the analysis of the compression of texts in languages whose scripts are less represented in computing. [18–21]. Kattan and Poli developed an algorithm that identifies best combination of compression algorithms for each text [22]. Grabowski and Swacha developed a dictionary based algorithm for language independent text compression [23]. Nunes et al. developed a grammar compression algorithm based on induced suffix sorting [24].

In this paper, the main question that will be answered is—“How does the language of a text affect the compression ratio?” We have collected and compared texts of various types such as stories, books, legal documents, business reports, short articles and user manuals. Some of the texts were collected only in English and Croatian, and others were collected in Croatian, Czech, Italian, French, German, English, Hungarian and Finnish. We limited research to languages based on Latin script due to the required number of bits to encode a single character. The algorithms we used for compression are Arithmetic Coding, as a representative of entropy encoding methods and LZW Algorithm, as a representative of dictionary-based encoding methods. LZW algorithm is used in Unix file compression utility *compress*.

The rest of the paper is organized as follows—we present a discussion of algorithms in Section 2 before presenting experimental results in Section 3, followed by a discussion Section 4. Finally, we draw our conclusions in Section 5.

2. Methods

2.1. Arithmetic Coding

Arithmetic coding is a lossless data compression method that encodes data composed of characters and converts it to a decimal number greater than or equal to zero and less than one. The compression performance depends on the probability distribution of each character in the text alphabet. The occurrence of infrequent characters significantly extends encoded data [4,25–27].

Entropy encoding is a type of lossless compression method which is based on coding frequently occurring characters with few bits and rarely occurring characters with more bits [28–30]. As in the most entropy encoding methods, the first step is creating a probability dictionary. This is done by counting the number of occurrences of each character and dividing it by the total number of characters in the text. The next step is assigning each character in the text alphabet a subinterval in the range $[0, 1)$ in proportion to its probability. When all characters have been assigned subintervals, the algorithm can start executing. In this step, a character that is not used in the text can be selected as the “end of text” character.

As can be seen from the given pseudocode in Algorithm 1 and in Figure 1, in the beginning, interval bounds are $[0, 1)$. The algorithm calculates new interval bounds for each character in the text. Once the algorithm reads the last character, which was determined previously, the algorithm stops. The encoded word can be any number from the resulting interval. It is recommended to take the final lower boundary of the resulting interval as the encoded word. Instead of using a distinctive character to determine the end of the text, the length of the text can be used to determine when the algorithm has to stop executing.

Algorithm 1: Arithmetic Coding Algorithm

```

l ← 0;
h ← 1;
xi ← first input character;
while xi not endOfMessage do
    l ← l + (h - l) * li;
    h ← l + (h - l) * hi;
    xi ← xi+1;
end
encodedMessage ← (l)2

```

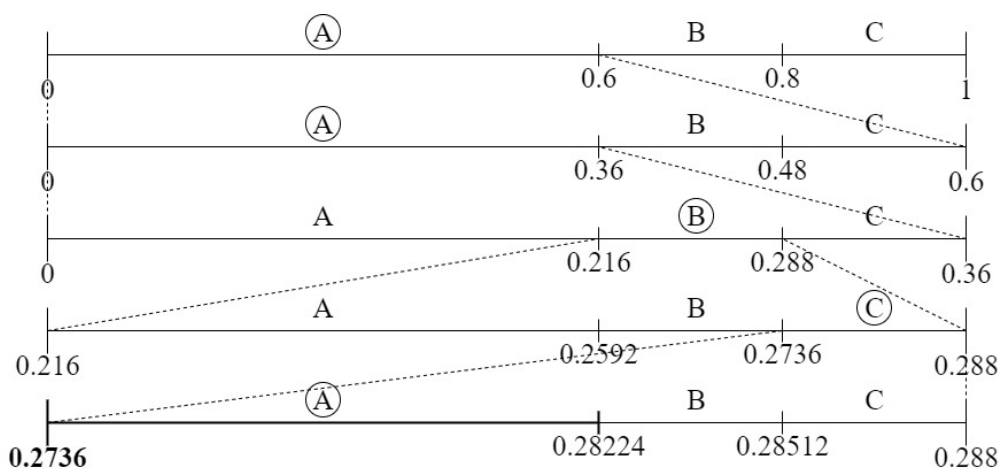


Figure 1. Arithmetic encoding the word AABCA.

Finally, the encoded text needs to be converted into binary code. The length of the binary code depends on the Shannon information value, which quantifies the amount of information in a message [31]. One can calculate the Shannon information using the following formula:

$$I(x_1x_2x_3...x_n) = -\log_2(p(x_1) * p(x_2) * p(x_3) * ... * p(x_n)) = -\sum_{i=1}^n \log_2(p(x_i)). \tag{1}$$

First, the probabilities of each character in the text need to be multiplied, and then the binary logarithm of a product is calculated. The length of the binary code is one integer larger than the result of the previous calculation. Once the text is converted to binary, it is ready to be transmitted or stored.

For decoding text (pseudocode of the arithmetic decoding algorithm is given in Algorithm 2), a subinterval dictionary made in the first step of the algorithm is used. Decoding begins with converting binary code to a decimal number. In the next step, the algorithm finds a subinterval where encoded text fits, and then it concatenates a subinterval key to the decoded message. The next step is to calculate a new value of the encoded text and repeat the subinterval search. There are two conditions on which the algorithm exits: either when it decodes an "end of text" distinctive character or after a preset number of repetitions. It depends which information the decoder has, "end of text" character or length of text.

Algorithm 2: Arithmetic Decoding Algorithm

```

 $l \leftarrow (\text{encodedMessage})_{10};$ 
 $\text{decodedMessage} \leftarrow "";$ 
while  $x_i$  not endOfMessage do
  |  $\text{find } x_i, l \in I(x_i);$ 
  |  $\text{decodedMessage} \leftarrow \text{decodedMessage} + x_i;$ 
  |  $l \leftarrow \frac{l-l_i}{h_i-l_i};$ 
end

```

2.2. Lempel–Ziv–Welch Algorithm

The Lempel–Ziv–Welch (LZW) Algorithm (pseudocode of which is given in Algorithm 3) is a dictionary-based lossless data compression method. Unlike Arithmetic Coding, for LZW compression, there is no need to know the probability distribution of each character in the text alphabet. This allows compression to be done while the message is being received. The main idea behind compression is to encode character strings that frequently appear in the text with their index in the dictionary. The first 256 words of the dictionary are assigned to extended ASCII table characters [13,32,33].

Algorithm 3: LZW Coding Algorithm

```

 $w \leftarrow \text{first input character};$ 
while  $x$  not endOfMessage do
  |  $x \leftarrow \text{next input character};$ 
  | if  $wx$  not in dictionary : then
  | |  $\text{dictionary}[w]$  append to encoded word ;
  | | add  $wx$  to dictionary ;
  | |  $w \leftarrow x;$ 
  | else
  | |  $w \leftarrow wx$ 
  | end
end

```

The pseudocode and Figure 2 present the steps of the algorithm. Two main values are stored in the algorithm, the word w , and current character x . In the beginning, the word w is the first text character. In each iteration, the algorithm reads a text character x and checks if there is a wx key in the dictionary. If wx is in the dictionary, w takes a value of wx , and the algorithm continues with the execution. In the other case, the corresponding value of w in the dictionary is added to the encoded word, whereafter the dictionary is upgraded with wx key and w takes a value of x . The algorithm stops when the end of file character is read.

s	a	m	o	u	p	r	a	v	n	o	p	r	a	v	n	i	w	x	Encoded message	Dictionary (index)
	.																s	a	115	sa (257)
		.															a	m	97	am (258)
			.														m	o	109	mo (259)
				.													o	u	111	ou (260)
					.												u	p	117	up (261)
						.											p	r	112	pr (262)
							.										r	a	114	ra (263)
								.									a	v	97	av (264)
									.								v	n	118	vn (265)
										.							n	o	110	no (266)
											.						o	p	111	op (267)
												.					p	r		
													.				pr	a	262	pra (268)
														.			a	v		
															.		av	n	264	avn (264)
																.	n	i	110	
																	i	end	105	ni (265)
																	end		256	

Figure 2. Lempel–Ziv–Welch (LZW) encoding of the word SAMOUPRAVNOPRAVNI

The LZW algorithm achieves the excellent compression ratio when compressing long text files that contain repeated strings [32].

The LZW Decoding Algorithm (pseudocode of which is given in Algorithm 4) creates the dictionary the same way as it is created for encoding. The first 256 words of the dictionary are assigned to extended ASCII table characters as well. The algorithm reads each code in the encoded word, writes its value from the dictionary to a decoded word, and upgrades a dictionary.

Algorithm 4: LZW Decoding Algorithm

```

decodedMessage ← "";
foreach code in encodedWord do
    w ← dictionary[code];
    decodedMessage ← decodedMessage + w;
    x ← first character in dictionary[nextCode];
    add to dictionary wx;
end
    
```

3. Results

The representative test data are prose texts, two legal texts and two user manuals. Because some of the test alphabets consist of non-ASCII characters, each character is stored using 16 binary bits. The output of the LZW Coding Algorithm is a sequence of integers; each of them is stored using 16 binary bits as well. The size of data compressed using the Arithmetic Algorithm in binary bits is equal to Shannon’s information Equation (1) of the original data. The compression results are shown in the Figures 3–5. Data compressed using the LZW Coding Algorithm varies from 20% to 45% of its original size. Text data compressed using Arithmetic Coding is ~30% of its original size.

3.1. Literary Text Compression

We present results for three prose texts of different lengths—a short story The Little Match Girl by Hans Christian Andersen, novella The Decameron, Tenth Day, Fifth Tale by Giovanni Boccaccio and novella The Metamorphosis by Franz Kafka (shown in Figures 3–5, respectively).

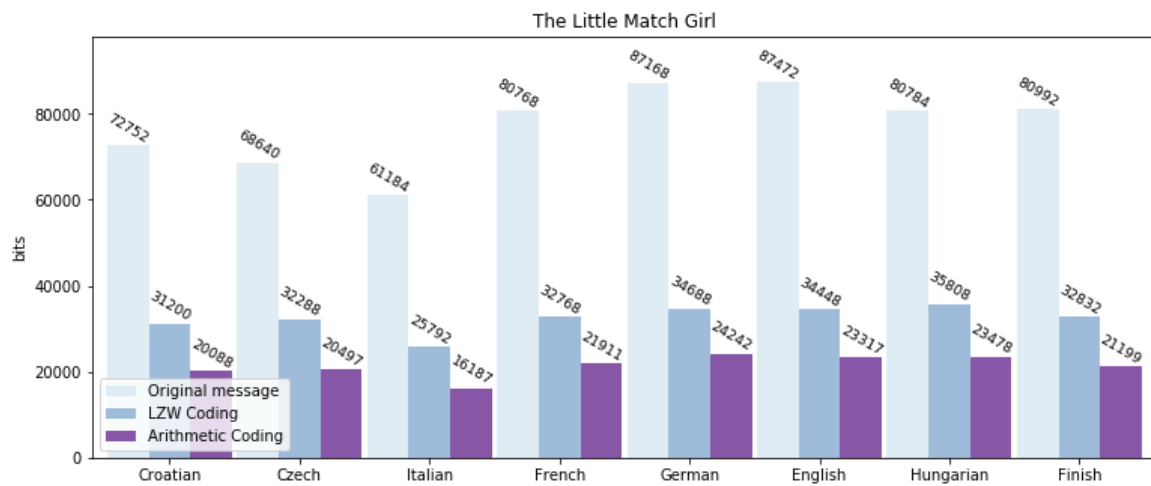


Figure 3. Encoding The Little Match Girl.

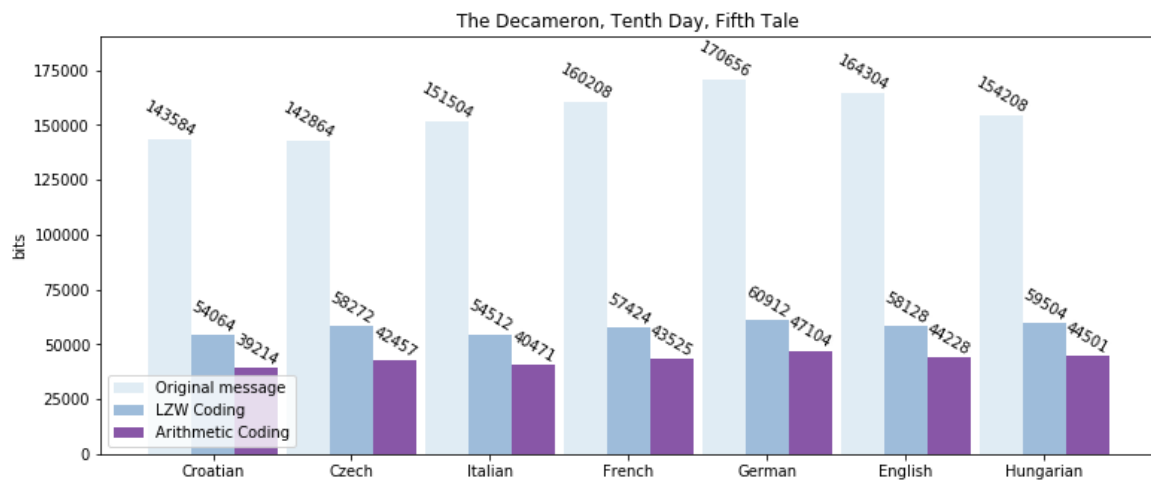


Figure 4. Encoding The Decameron Tale.

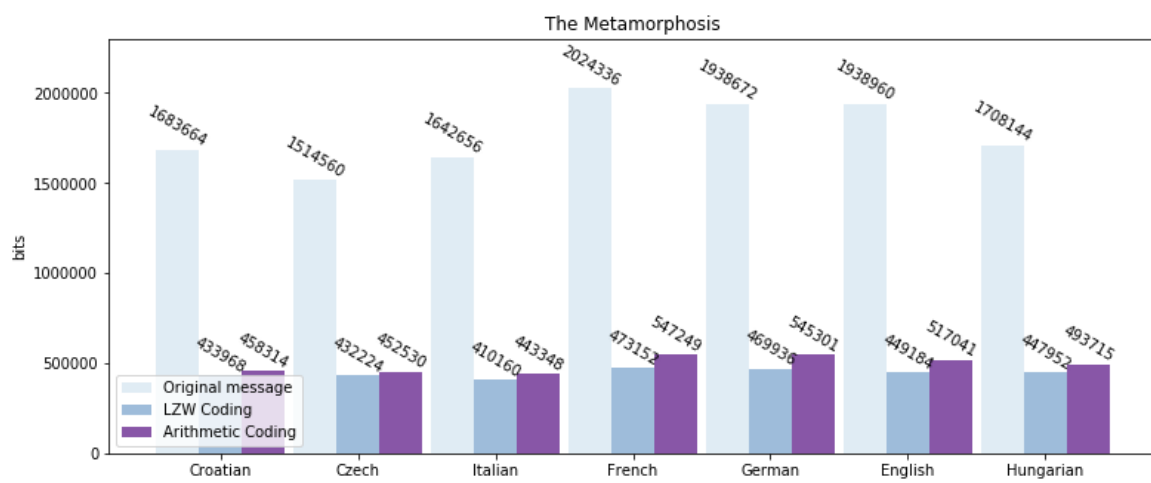


Figure 5. Encoding The Metamorphosis.

3.2. Legal Text Compression

As legal text compression, we present compression results for The European Green Deal and Charter of Fundamental Rights of the European Union given in Figures 6 and 7, respectively.

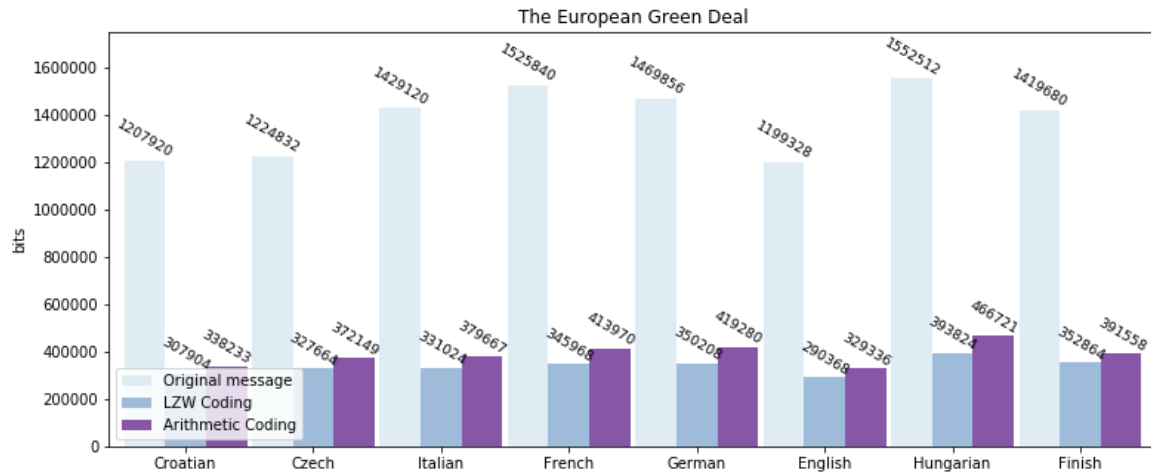


Figure 6. Encoding The European Green Deal.

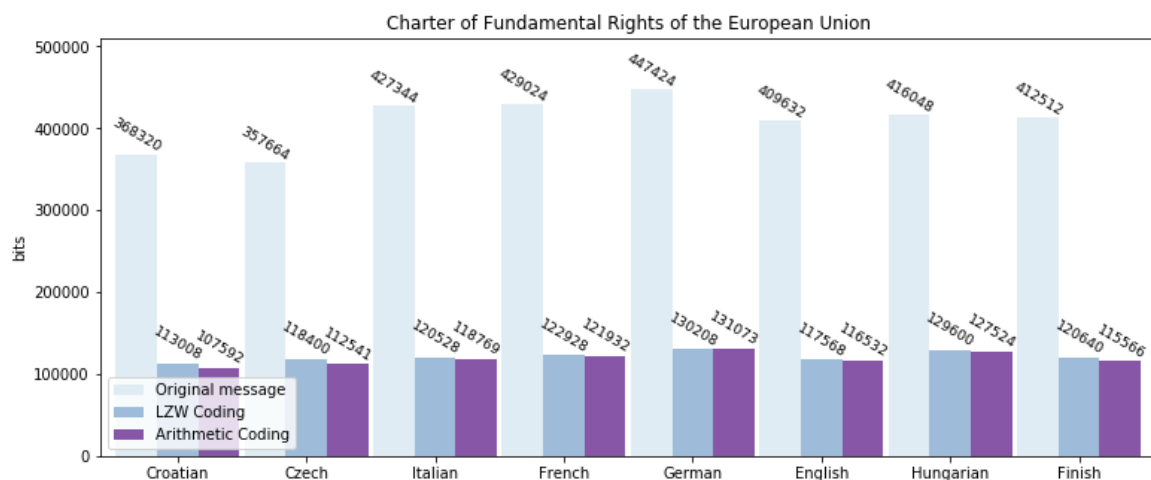


Figure 7. Encoding Charter of Fundamental Rights of the European Union.

3.3. User Manual Compression

We present compression results for Samsung Q6F Smart TV user manual and Candy CMG 2071M Microwave Oven user manual shown in Figures 8 and 9, respectively.

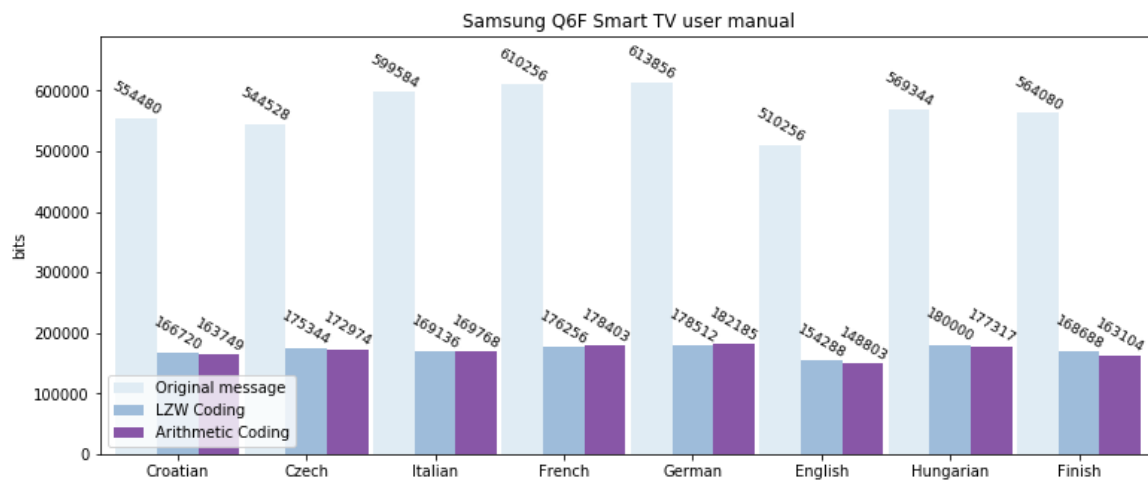


Figure 8. Encoding Samsung Q6F Smart TV user manual.

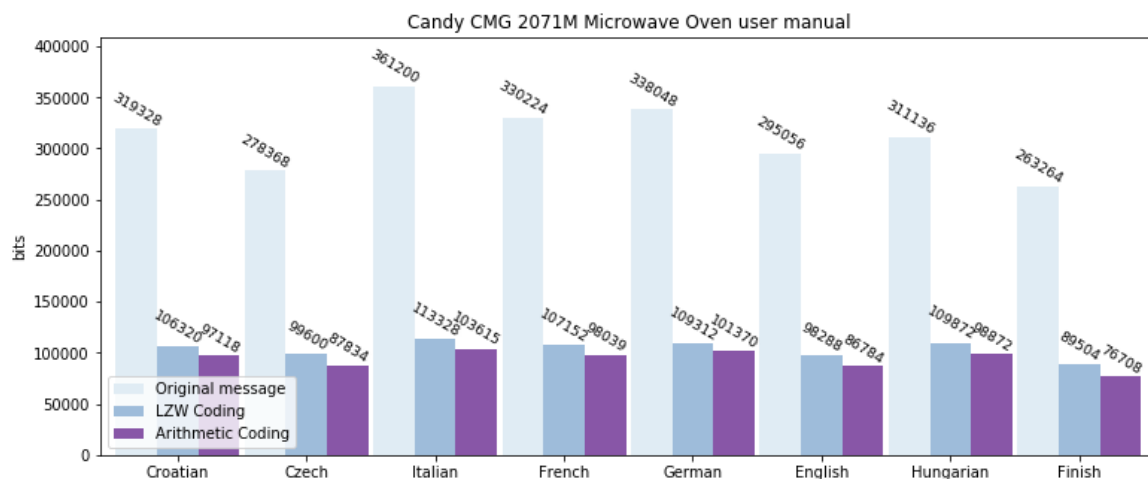


Figure 9. Encoding Candy CMG 2071M Microwave Oven user manual.

4. Discussion

Both compression algorithms (arithmetic and LZW) have proven effective. The compression ratio varies depending on the text language. The Italian, French, and English alphabets consist of 26 letters. The Croatian alphabet consists of 30 letters, 3 of which are composed of two characters from the rest of the alphabet. Therefore, the Croatian alphabet may be considered to consist of 27 different characters. The Finnish alphabet consists of 29 letters. The German alphabet consists of 30 letters. The Czech alphabet consists of 42 letters or 41 different characters. The Hungarian alphabet consists of 44 letters or 35 different characters.

Figures 3–9 present the compression results. In terms of the percentage of text size reduction, the compression results are as follows. Arithmetic compression results for Croatian, Czech, Italian, French, German, English, Hungarian, and Finnish translations of The Little Match Girl are as follows: 72.39%, 70.14%, 73.54%, 72.87%, 72.19%, 73.34%, 70.94%, and 73.83% respectively. The compression results for Croatian, Czech, Italian, French, German, English, and Hungarian versions of The Decameron Tale translations are as follows: 72.69%, 70.28%, 73.29%, 72.83%, 72.4%, 73.08%, and 71.15%. The results for arithmetic compression of The Metamorphosis are 72.78%, 63.87%, 73.01%, 72.97%, 71.87%, 73.33%, and 71.1% for the listed languages. Arithmetic compression results for The European Green Deal translations are as follows: 72%, 69.62%, 73.43%, 72.87%, 71.47%, 72.54%, 69.94%, and 72.42% for the Croatian, Czech, Italian, French, German, English, Hungarian, and Finnish

respectively. Finally, compression results for Charter of Fundamental Rights of the European Union are 70.79%, 68.53%, 72.21%, 71.58%, 70.7%, 71.55%, 69.35%, and 71.99%. Arithmetic compression results for Samsung Q6F Smart TV user manual are as follows: 70.47%, 68.23%, 71.69%, 70.77%, 70.32%, 70.84%, 68.86%, and 71.09% respectively. Finally, the compression results for Candy CMG 2071M Microwave Oven user manual translations are as follows: 69.59%, 68.45%, 71.31%, 70.31%, 70.01%, 70.59%, 68.22%, and 70.86% for the Croatian, Czech, Italian, French, German English, Hungarian, and Finnish respectively.

The compression of texts in Italian, French, English and Finnish achieved the best compression ratio. The compression ratio of Croatian and German texts is close to the compression ratio of texts in languages with smaller alphabets. Compressing texts in Czech and Hungarian stands out the most. Czech versions of The Little Match Girl, The Decameron Tale, The Metamorphosis and Charter of Fundamental Rights of the European Union compression is >2% lower than compression of the same texts in different languages with fewer letters in the alphabet. Figure 10 shows arithmetic compression results. Compression ratio change compared to English is shown in Figure 11.

The number of different characters impacts compression performance. More different characters in alphabet increase the number of subintervals in Arithmetic Coding and extend the encoded message accordingly.

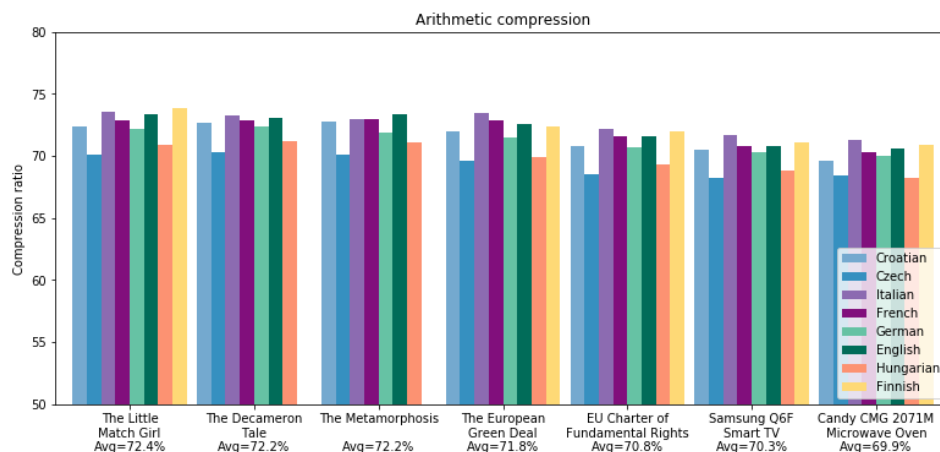


Figure 10. Arithmetic compression results.

The Little Match Girl is 4–6 thousand characters long text, depending on the text language, which makes it the shortest of three prose texts that are shown in this paper. The LZW compression results for compressing this text are 57.11%, 52.96%, 57.85%, 59.43%, 60.21%, 60.62%, 55.67%, and 59.46% for the Croatian, Czech, Italian, French, German, English, Hungarian, and Finnish respectively. These results show that LZW compression is not ideal for shorter texts. The Decameron Tale is approximately twice as long as The Little Match Girl, and the results of compressed Tale are as follows 62.35%, 59.21%, 64.02%, 64.16%, 64.31%, 64.62%, and 61.41% for the Croatian, Czech, Italian, French, German, English, and Hungarian, respectively. The Metamorphosis is up to 130 thousand characters long. Results of LZW compression of The Metamorphosis are 74.22%, 71.46%, 75.03%, 76.63%, 75.76%, 76.83%, and 73.78% for the Croatian, Czech, Italian, French, German English, and Hungarian, respectively. We can conclude that LZW compression is affected more by the text length than the arithmetic coding. The LZW compression results for Croatian, Czech, Italian, French, German, English, Hungarian, and Finnish translations of The European Green Deal are as follows: 74.51%, 73.25%, 76.84%, 77.33%, 76.17%, 75.79%, 74.63%, and 75.14%, respectively. The compression results for Charter of Fundamental Rights of the European Union translations are as follows: 69.32%, 66.9%, 71.8%, 71.35%, 70.9%, 71.3%, 68.85%, and 70.75% for the listed languages. The LZW compression results for Croatian, Czech, Italian, French, German, English, Hungarian, and Finnish translations of Samsung Q6F Smart TV user manual are as follows: 69.93%, 67.8%, 71.8%, 71.12%, 70.92%, 69.76%, 68.38%, and 70.1%, respectively. Finally,

the compression results for Candy CMG 2071M Microwave Oven user manual translations are as follows: 66.71%, 64.22%, 68.62%, 67.55%, 67.66%, 66.69%, 64.69%, and 66%, respectively for the for Croatian, Czech, Italian, French, German, English, Hungarian, and Finnish.

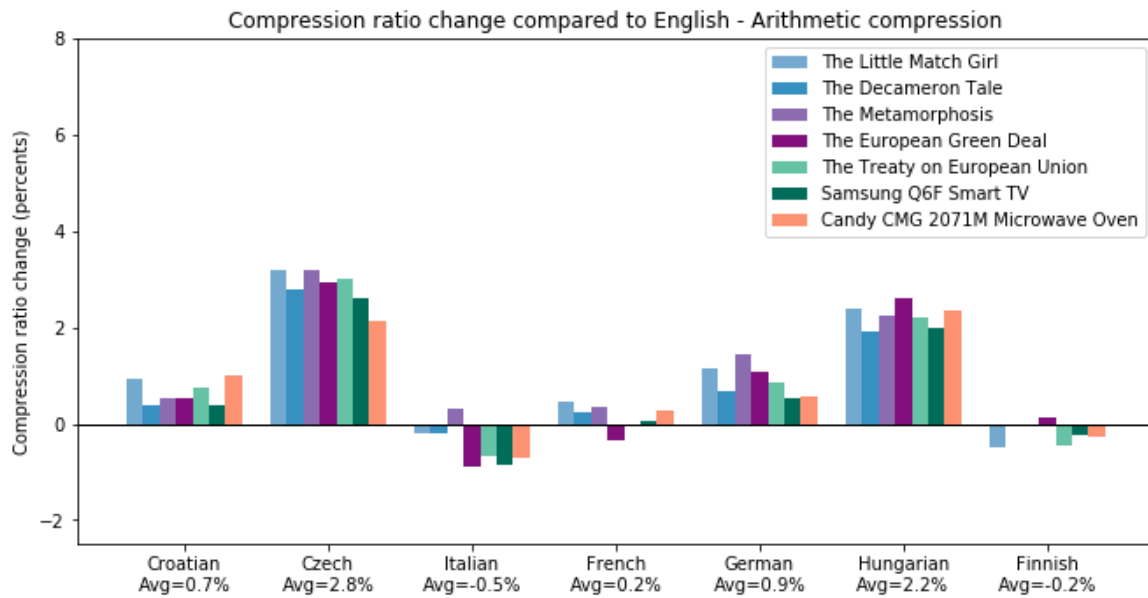


Figure 11. Compression ratio change compared to English—Arithmetic compression (positive percentages signify larger compressed file size when compared to English).

Figure 12 shows LZW compression results. Compression ratio change compared to English is shown in Figure 13.

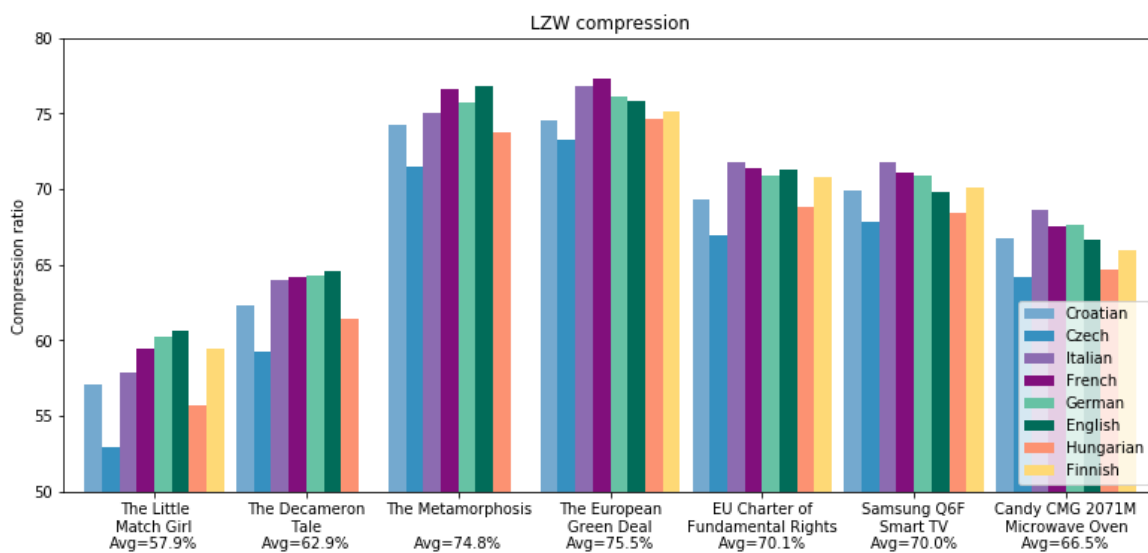


Figure 12. LZW compression results.

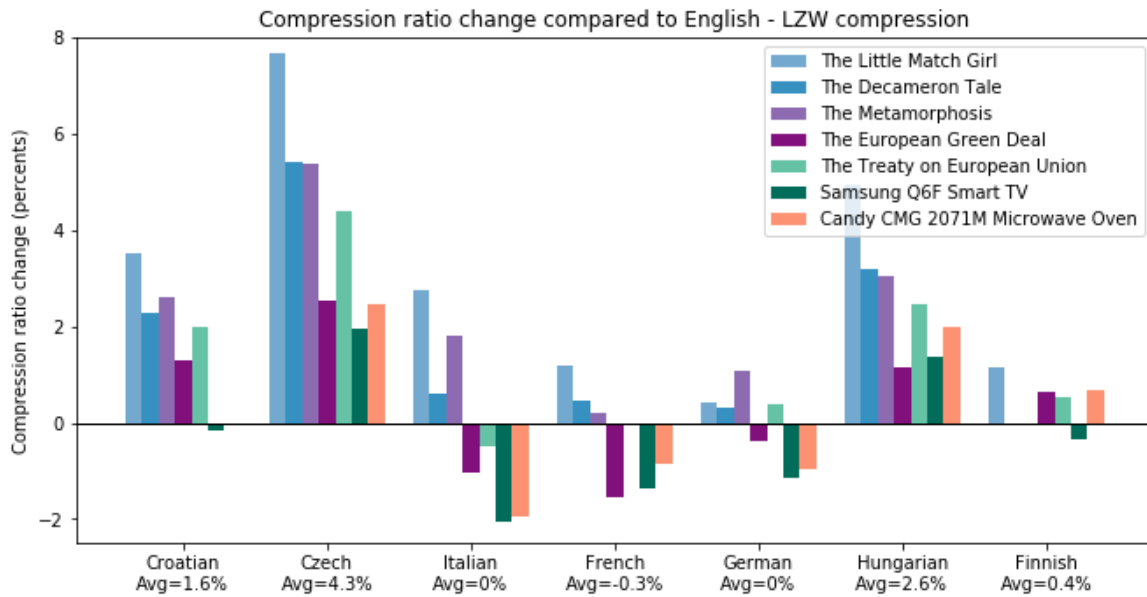


Figure 13. Compression ratio change compared to English—LZW compression (positive percentages signify larger compressed file size when compared to English).

Figure 14 shows the compression rate for different lengths of text. Generally, as text length increases, the compression ratio of the LZW algorithm increases. In our test texts, the exception is the compression ratio of Charter of Fundamental Rights of the European Union which compression achieves better results than compression of, longer text, Smart TV user manual. The reason for this irregularity is the repetition of the word ‘Article’. As stated in Section 2, LZW compression is based on encoding character strings that frequently appear in the text. Arithmetic encoding achieves significantly better compression ratio for compressing texts up to 20,000 characters, the LZW algorithm achieves better compression ratio for compressing texts longer than 100,000 characters.

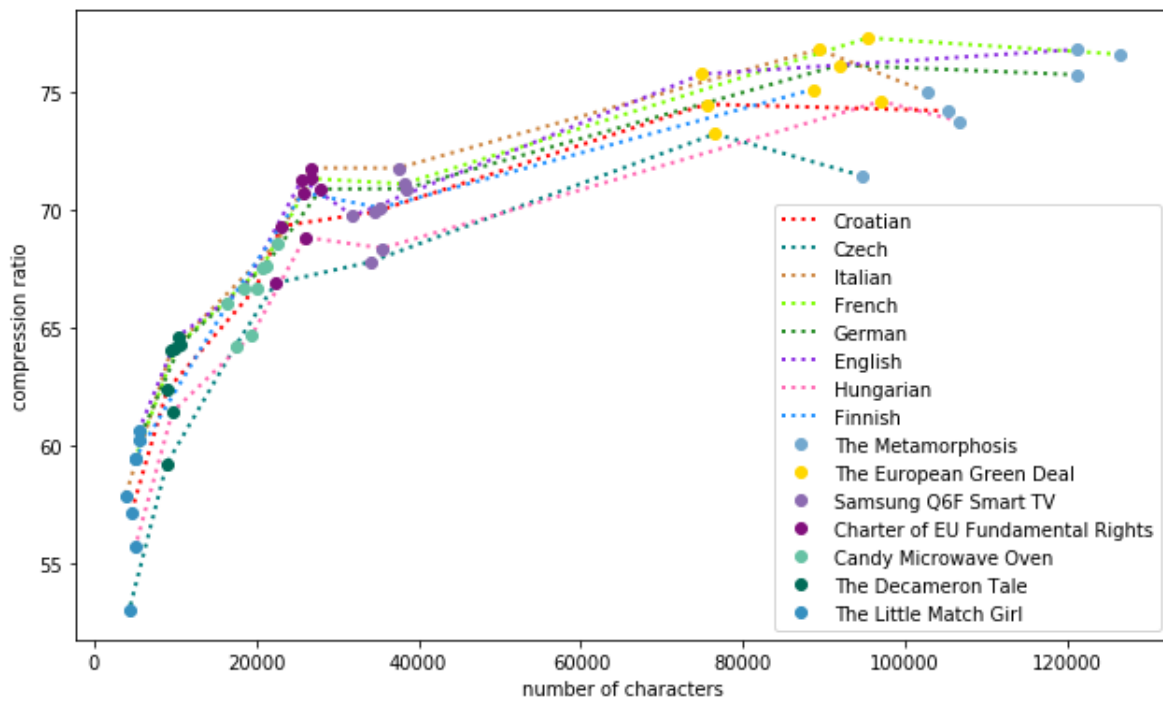


Figure 14. LZW compression results—text length dependence.

In addition to the size of the alphabet and text length, the LZW compression also affects the form of the word. We corroborate this by comparing Croatian and English grammar. Croatian grammar is more complex than English grammar. In Croatian grammar, the form of the word depends on the tense, case, and position in the sentence; English grammar also changes the form of the word but in far fewer cases. The LZW compression is based on encoding repeating strings. Because of these differences in grammar, English texts achieve a better compression ratio than their Croatian equivalents. In Figure 15 there are several values from the end of the LZW dictionary. It is shown that encoded strings in English are longer and contain more complete words.

KEY	VALUE	KEY	VALUE
. Gro	28386	u. Po	27437
owi	28387	ostavš	27438
ing mo	28388	ši sve	27439
ore si	28389	e šu	27440
ilent	28390	utl	27441
and al	28391	ljivij	27442
lmost un	28392	ji i	27443
ncon	28393	i spo	27444
nsc	28394	oraz	27445
ciousl	28395	zumi	27446
ly und	28396	ijevaj	27447
ders	28397	jući se	27448
standing e	28398	pogledi	27449
each other	28399	ima, na	27450
r in t	28400	amata	27451
their g	28401	ali su	27452
glances	28402	u mis	27453

Figure 15. Example from LZW dictionaries in English and Croatian.

5. Conclusions

Data compression is the process of reducing the number of bits needed to represent data. Compressing data both reduces the need for storage hardware and speeds up file transfer.

Choosing the right compression algorithm is not a simple task because the performance of each algorithm depends on the text type, length of data, and other text characteristics. Arithmetic Coding achieves a significant compression ratio regardless of the length of the text, but algorithm performance decreases as text length increases. Time and space complexity are crucial parts of any algorithm, and that makes the algorithm not suitable for universal use.

The LZW Algorithm achieves excellent compression ratio when compressing long text files that contain repetitive strings. The algorithm takes a short time to execute and uses minimal resources.

The main question posed in this paper is—“How does the language of a text affect the compression ratio?” and, as it can be seen from results, the answer is positive—there are some differences in compression ratios between texts in different languages and different types of texts. When choosing a compression algorithm, it is important to determine which algorithm achieves the best compression ratio for each language and/or type of text.

Author Contributions: Conceptualization, M.I. and J.L.; methodology, M.I. and J.L.; software, M.I.; validation, L.S. and M.D.; formal analysis, M.I., J.L., L.S., and M.D.; investigation, L.S. and M.D.; resources, M.I.; data curation, M.I.; writing—Original draft preparation, M.I.; writing—Review and editing, J.L., L.S., and M.D.; visualization, M.I.; supervision, J.L.; project administration, J.L., L.S., and M.D.; funding acquisition, J.L., L.S., and M.D. All authors have read and agreed to the published version of the manuscript.

Funding: This work was fully supported by the Croatian Science Foundation under the project IP-2018-01-3739 and IP-2020-02-4358, Center for Artificial Intelligence and Cybersecurity—University of Rijeka, University of Rijeka under the projects uniri-tehnic-18-17 and uniri-tehnic-18-15, and European Cooperation in Science and Technology (COST) under the project CA17137.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Celikel, E.; Dalkilic, M.E. A New Encoding Decoding Scheme for Text Compression with Embedded Security. *Math. Comput. Appl.* **2004**, *9*, 475–484.
2. Rozenberg, L.; Lotan, S.; Feldman, D. Finding Patterns in Signals Using Lossy Text Compression. *Algorithms* **2019**, *12*, 267.
3. Shahbahrami, A.; Bahrampour, R.; Rostami, M.; Mobarhan, M. Evaluation of Huffman and Arithmetic Algorithms for Multimedia Compression Standards. *arXiv* **2011**, arXiv:1109.0216.
4. Mbewe, P.; Asare, S.D. Analysis and comparison of adaptive Huffman coding and arithmetic coding algorithms. In Proceedings of the 13th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD), Guilin, China, 29–31 July 2017.
5. Robert, L.; Nadarajan, R. Simple lossless preprocessing algorithms for text compression. *IET Softw.* **2009**, *3*, 37–45.
6. Katugampola, U.N. A New Technique for Text Data Compression. In Proceedings of the 2012 International Symposium on Computer, Consumer and Control, Taichung, Taiwan, 4–6 June 2012; pp. 405–409.
7. Howard, P.G. Lossless and lossy compression of text images by soft pattern matching. In Proceedings of the DCC '96: Proceedings of the Conference on Data Compression, Snowbird, UT, USA, 31 March–3 April 1996; pp. 210–219.
8. Al-Dubae, S.A.; Ahmad, N. New Strategy of Lossy Text Compression. In Proceedings of the 2010 First International Conference on Integrated Intelligent Computing, Bangalore, India, 5–7 August 2010; pp. 22–26.
9. Quddus, A.; Fahmy, M.M. A new compression technique for binary text images. In Proceedings of the Second IEEE Symposium on Computer and Communications, Alexandria, Egypt, 1–3 July 1997; pp. 194–198.
10. Xu, J.; Zhang, W.; Xie, X.; Yang, Z. SSE Lossless Compression Method for the Text of the Insignificance of the Lines Order. *arXiv*, **2017**, arXiv:1709.04035.
11. Sayood, K. *Introduction to Data Compression*; 5th ed.; Elsevier: Amsterdam, The Netherlands, 2018; Chapter 6, pp. 165–185, ISBN 978-0-12-809474-7.
12. Kavitha, P. A Survey on Lossless and Lossy Data Compression Methods. *Int. J. Comp. Sci. Eng. Technol.* **2016**, *7*, 1277–1280.
13. Shanmugasundaram, S.; Lourdasamy, R. A Comparative Study Of Text Compression Algorithms. *Int. J. Wisdom Based Comput.* **2011**, *1*, 68–76.
14. Bhattacharjee, A.K.; Bej, T.; Agarwal, S. Comparison Study of Lossless Data Compression Algorithms for Text Data. *IOSR-JCE J. Comp. Eng.* **2013**, *11*, 15–19.
15. Abliz, W.; Wu, H.; Maimaiti, M.; Wushouer, J.; Abiderexiti, K.; Yibulayin, T.; Wumaier, A. A Syllable-Based Technique for Uyghur Text Compression. *Information* **2020**, *11*, 172.
16. Zhang, N.; Tao, T.; Satya, R.V.; Mukherjee, A. A flexible compressed text retrieval system using a modified LZW algorithm. In Proceedings of the Data Compression Conference, Snowbird, UT, USA, 29–31 March 2005; p. 493.
17. Garain, U.; Chakraborty, M.P.; Chanda, B. Lossless Compression of Textual Images: A Study on Indic Script Documents. In Proceedings of the 18th International Conference on Pattern Recognition (ICPR'06), Hong Kong, China, 20–24 August 2006; pp. 806–809.
18. Mohamed, A.S.; El-Sawy, A.H.; Ahmad, S.M. Data compression for Arabic text. In Proceedings of the Fifteenth National Radio Science Conference, Cairo, Egypt, 24–26 February 1998.
19. Kuruvila, M.; Gopinath, D.P. Entropy of Malayalam language and text compression using Huffman coding. In Proceedings of the First International Conference on Computational Systems and Communications (ICCS), Trivandrum, India, 17–18 December 2014.
20. Morihara, T.; Satoh, N.; Yahagi, H.; Yoshida, S. Japanese text compression using word-based coding. In Proceedings of the DCC '98 Data Compression Conference, Snowbird, UT, USA, 30 March–1 April 1998.

21. Farhad Mokter, M.; Akter, S.; Palash Uddin, M.; Ibn Afjal, M.; Al Mamun, M.; Abu Marjan, M. An Efficient Technique for Representation and Compression of Bengali Text. In Proceedings of the 2018 International Conference on Bangla Speech and Language Processing (ICBSLP), Sylhet, Bangladesh, 21–22 September 2018; pp. 1–6.
22. Kattan, A.; Poli, R. Evolutionary lossless compression with GP-ZIP. In Proceedings of the IEEE World Congress on Computational Intelligence, Hong Kong, China, 1–6 June 2008.
23. Grabowski S.; Swacha, J. Language-independent word-based text compression with fast decompression. In Proceedings of the VIth International Conference on Perspective Technologies and Methods in MEMS Design, Lviv, Ukraine, 20–23 April 2010; pp. 158–162.
24. Saad Nogueira Nunes, D.; Louza, F.; Gog, S.; Ayala-Rincón, M.; Navarro, G. A Grammar Compression Algorithm Based on Induced Suffix Sorting. In Proceedings of the 2018 Data Compression Conference, Snowbird, UT, USA, 27–30 March 2018; pp. 42–51.
25. Langdon, G. An Introduction to Arithmetic Coding. *IBM J. Res. Dev.* **1984**, *28*, 135–149.
26. Sarkar, S.J.; Kar, K.; Das, I. Basic arithmetic coding based approach for compressing generation scheduling data array. In Proceedings of the 2017 IEEE Calcutta Conference (CALCON), Kolkata, India, 2–3 December 2017; pp. 21–25.
27. Husodo, A.Y.; Munir, R. Arithmetic coding modification to compress SMS. In Proceedings of the 2011 International Conference on Electrical Engineering and Informatics, Bandung, Indonesia, 17–19 July 2011; pp. 1–6.
28. Vijayvargiya, G.; Silakari, S.; Pandey, R. A Survey: Various Techniques of Image Compression *arXiv* **2013**, arXiv:1311.6877.
29. Behr, F.; Fossum, V.; Mitzenmacher, M.; Xiao, D. Estimating and comparing entropies across written natural languages using PPM compression. In Proceedings of the Data Compression Conference, DCC, Snowbird, UT, USA, 25–27 March 2003; p. 416.
30. Ezhilarasan, M.; Thambidurai, P.; Praveena, K.; Srinivasan, S.; Sumathi N. A New Entropy Encoding Technique for Multimedia Data Compression. In Proceedings of the International Conference on Computational Intelligence and Multimedia Applications (ICCIMA 2007) Sivakasi, Tamil Nadu, India, 13–15 December 2007; pp. 157–161.
31. Shannon, C.E. A Mathematical Theory of Communication *Bell Syst. Tech. J.* **1948**, *27*, 379–423.
32. Dheemanth, H.N. LZW Data Compression. *AJER* **2014**, *3*, 22–26.
33. Hasan, M.R.; Ibrahimy, M.I.; Motakabber, S.M.A.; Ferdaus, M.M.; Khan M.N.H. Comparative data compression techniques and multicompression results *IOP Conf. Ser. Mater. Sci. Eng.* **2013**, *53*, 012081.



© 2020 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).