International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING

ISSN:2147-6799 www.ijisae.org Original Research Paper

An Optimized Integer Representation through a Novel Numeric Encoding for Textual Data Compression

Kanak Pandit[1*], Harshali Patil[2], Poonam Joshi[3], Tarunima Mukherjee[4]

Submitted: 03/02/2024 **Revised**: 11/03/2024 **Accepted**: 17/03/2024

Abstract: The objective of this paper is to introduce a new variable sized integer encoding technique for file compression. The paper aims to compare the performance of the proposed method with established codes like Elias Gamma, Elias Delta, and Golomb. The study also seeks to examine the impact of varying log base values on compression ratio and runtime efficiency. The proposed method utilizes radix conversion and the Burrows Wheeler Transform for file compression. Performance comparison is conducted on the Calgary corpus, which includes both text and binary files. Existing codes like Elias Gamma, Elias Delta, and Golomb are executed on the files before evaluating the proposed code. Graphs are used to analyze the impact of log base values on compression ratio, while runtime efficiency is assessed. The proposed compression code achieves varied compression ratios (1.67 to 1.87) at radix r=4, highlighting its effectiveness over existing algorithms. A non-linear relationship between the log base and compression ratio is observed, plateauing as the log base increases. Runtime varies among files, with 'bib1' at the longest time (6.41 seconds) and 'obj1' the shortest (0.09 seconds). A positive correlation exists between the number of data points (n) and runtime, while a negative correlation is seen between 'n' and compression ratio, indicating lower ratios for larger 'n' files. Comparing its performance with established codes provides a benchmark for evaluation. Analyzing compression ratio trends and runtime efficiency offers insights into the effectiveness of the proposed method, adding to its novelty.

Keywords: *Burrows-Wheeler Transform, Elias Delta Code, Elias Gamma Code, Golomb Code, Numeric Encoding*

1. Introduction

As technology rapidly evolves, supported by increasingly advanced software and hardware, the dissemination of information worldwide via the internet has become swift and widespread[1]. While information technology experts can easily communicate through the internet, not all data can be transmitted effortlessly. Compression techniques alleviate challenges posed by large file sizes, facilitating quicker data transmission and conserving storage space on computers. By converting data sets into codes, compression reduces storage requirements and streamlines data transmission processes. Data compression can be considered a segment of information theory focused on reducing the quantity of data required for transmission[2]. The aim is to minimize the storage space required for storing data in devices and to facilitate data transmission through low-bandwidth channels of communication. One such technique is lossy compression and another is lossless[3]. In lossless compression, data can be decompressed to precisely replicate the original source data, whereas in latter, the restored data is not entirely indistinguishable from the source and may incur some content loss. It is typically applied to textual information such as financial records, software applications, written documents, and programming source code, while lossy compression is utilized for compressing multimedia items. Lossy compression methods often yield higher compression ratios compared to lossless techniques due to the removal of redundant or less essential data[4].

Some of the compression algorithms include:

1. Shannon Fano Algorithm

The Shannon Fano algorithm, employed in compression methods like zip and .rar formats, is a pivotal technique for data compression. While it offers compression benefits, its conventional implementation often results in relatively long codes. However, further advancements are needed to address inherent limitations in its length and to optimize their compression performance. Shannon Fano coding is more complex than Huffman coding, which is a similar algorithm that guarantees optimal codes[5].

2. Run Length Encoding (RLE):

It is a straightforward lossless methodology effective for sequences of identical data values. While highly useful for specific data types like icon files and line drawings, it may not be suitable for general datasets due to potential increases in file size. Despite its simplicity and applicability, RLE algorithms require enhancements to handle diverse data types and improve compression efficiency across various scenarios. For instance, RLE may not achieve optimal compression when data comprises short sequences of similar elements or consecutive non-identical elements. In such cases, alternative methods like Huffman coding may prove more effective[6].

3. Lempel Ziv Welch (LZW):

It is a widely used technique employing dictionaries, forms the basis for many compression applications. While effective, its conventional implementation may not always achieve optimal compression results. This paper compares conventional LZW coding with proposed modifications, highlighting the

[1*] Computer Eng., Thakur College of Engineering and Technology, Mumbai:400066
ORCID ID : 0000-0001-7685-354
[2] Computer Eng., Thakur College of Engineering and Technology, Mumbai:400066
ORCID ID : 0000-0003-2052-9940
[3] Computer Eng., Thakur College of Engineering and Technology, Mumbai:400066
ORCID ID : 0009-0002-4671-6162
[4] Computer Eng., Thakur College of Engineering and Technology, Mumbai:400066
ORCID ID : 0009-0007-9146-5135
* Corresponding Author Email: kanakpandit17@gmail.com

importance of efficient dictionary management and compression output optimization[7]. LZW compression works best for files that have more repetitive data, which is only in case of monochrome images and text.

4. Tunstall Algorithm:
The Tunstall algorithm, introduced in 1967, revolutionized noiseless compression codes by mapping source symbols to fixed-length codewords. Despite its pioneering nature, the Tunstall algorithm may face challenges in effectively handling stochastic sources with variable-length codewords. Before initiating the parsing process, Tunstall coding necessitates the algorithm's awareness of the probability distribution associated with each letter of the alphabet.

5. Huffman Compression:
Huffman compression, a static method based on frequency analysis, offers efficient compression by allocating abbreviated codes to frequently appearing characters. While highly effective, it may not fully optimize compression for all datasets and scenarios. This algorithm relies on unique codes and character frequency distributions to achieve compression efficiency. However, this method does not endorse adaptive encoding.[8].

Diverse sized codes are created for the purpose of shrinking the content. Statistical coding methods, unlike fixed-length codes ,accomplish compression through the allocation of briefer symbols to more often occurring symbols and longer codes to less common symbols in the input document being compressed. These statistical methods necessitate knowledge of the probabilities associated with input symbols for generating variable-length codes. Examples of statistical methods include Huffman coding[9] and Shannon-Fano methods[10], which utilize symbol tables during the decoding process. However, the two-pass approach of statistical methods can be slow for certain systems like storage and sensory systems.
Alternatively, coding techniques like *Elias Gamma Code (EGC)*[11] , *Elias Delta Code (EDC)*, and *Golomb Code (GC)*[12] abstain from necessitating likelihood measures of source information for generating codes, making them known as varying sized whole number encoding techniques. As such numeric encodings do not depend on symbol tables or likelihood values, they are favored in sectors requiring swift encoding and storage.
This paper proposes a new variable-length integer code based on radix conversion, which serves as the ultimate stage encoder in the Burrows-Wheeler compressor[13]. The effectiveness of the newly suggested code is contrasted with current cutting-edge methods like EGC, EDC, and GC using the Calgary corpus dataset. Section II provides an overview of current codes. Section III introduces the innovative code, while Section IV investigates its effectiveness when utilized alongside the Burrows-Wheeler Transform (BWT) for textual content compaction, evaluated on the Calgary Corpus data. Finally, Section V concludes the study.

Some Variable-Length Integer Codes:

Such codes serve the purpose of efficiently representing non-negative integers in a compact format. Due to their ease of construction, these codes find extensive applications in compression of images, compression of videos, and compression of text.
We introduce such codes including Golomb Code (GC), Elias Gamma Code (EGC), and Elias Delta Code (EDC). These codes are specifically designed for the depiction of non-negative integers, offering efficient encoding and decoding mechanisms.

Unary Code
The Unary Code (UC), denoted as UC1, is versatile that adheres to the characteristic of having a prefix. It is characterized by its simplicity and effectiveness in encoding integers. The representation of an integer *"n"* consists of a sequence of *"(n-1)"* zeros or one, succeeded by a single one or zero. Consequently, the length of the Unary Code for an integer n is equivalent to n bits.

Table 1. UC foer the whole numbers 1 to 5

Numbers	UC	Symbol	Quantity
1	0	Φ	magnetic flux
2	10	*B*	magnetic flux density, magnetic induction
3	110	*H*	magnetic field strength
4	1110	χ, κ	susceptibility
5	11110	χ_ρ	mass susceptibility

As seen in Table 1[14], UC is depicted for 1 to 5.

2. Elias Gamma Code
It was introduced by Peter Elias in 1975. It is designed to encode integers efficiently. The representation of an integer n comprises two main components: the unary part denoted as UC(L) and the binary part represented as ~B(n). Here, UC(L) refers to the unary code representing the length (L) of the binary encoding of n, while ~B(n) denotes the binary form of n excluding its leading bit. The EGC is constructed by concatenating UC(L) and ~B(n), represented as UC(L) | ~B(n).

Table 2. EGC for the whole numbers 1 to 5

Integer (n)	EGC	Symbol	Quantity
1	1	Φ	magnetic flux
2	10	*B*	magnetic flux density, magnetic induction
3	11	*H*	magnetic field strength
4	100	χ, κ	susceptibility
5	101	χ_ρ	mass susceptibility

Table 2[14] illustrates EGC for the whole numbers 1 to 5.

3. Elias Delta Code
Peter Elias developed the Elias Delta Code (EDC)[15]. It consists of primary components: the Gamma Part and the binary part denoted as ~B(n). The Gamma Part represents the Elias Gamma Code representing the bit length (L) of ~B(n), while the binary part signifies the binary form of the integer n excluding its highest bit. Therefore, EDC is constructed as EGC(L) | ~B(n), where EGC(L) represents the Elias Gamma Code of the bit length (L).

Table 3. EDC for the whole numbers 1 to 5

Integer (n)	EDC	Symbol	Quantity
1	1	Φ	magnetic flux
2	100 | 0	*B*	magnetic flux density, magnetic induction
3	100 | 1	*H*	magnetic field strength
4	101 | 00	χ, κ	susceptibility
5	101 | 01	χ_ρ	mass susceptibility

Table 3[14] illustrates EDC for the whole numbers 1 to 5.

4. Golomb Code

Golomb codes partition all index values i into uniform groups of equal size[16]. A number *a*>0 is initially divided by a divisor *b*. The remainder *r* and quotient *q* are then utilized to encode the given number a (>0). These values are obtained using the following equations[16]:

$x=(a-1)/b$ Eq. (1)

$r=a-qb-1$ Eq. (2)

The Golomb Code consists of two parts. The initial segment of GC encodes the value of *q+1* in unary, while the second part encodes the binary value of *r*. For instance, when the divisor, *b*=5, it results in five possible remainders. These remainders, 0, 1, 2, 3 and 4 are encoded as 00, 01, 10, 11 and 100 respectively. These codes are illustrated in Table 4, which presents the Golomb Codes for divisors *b*=3 and *b*=4.

Table 4. GC for the whole numbers 1 to 5

Integer n	Goulomb Code	Symbol	Quantity
	d=3	Φ	magnetic flux
1	0 | 0	*B*	magnetic flux density, magnetic induction
2	0 | 10	*H*	magnetic field strength
3	0 | 11	χ, κ	susceptibility
4	10 | 0	χ_ρ	mass susceptibility

Table 4[14] illustrates GC for the whole numbers 1 to 5.

2. Methodology

In this section, we introduce a fresh approach to encoding integers, grounded in the concept of radix conversion. Representing integers efficiently holds paramount importance, reducing storage overheads and facilitating swift integer retrieval. Our proposed encoding method involves transforming the binary representation of integers into a radix-r format. The crux of this approach lies in recognizing that higher radix-r systems often require fewer digits for representation compared to lower radix-r systems. Consequently, encoding integers in a higher radix-r system demands fewer bits for the length part of the code. Thus, our encoding scheme comprises two main components: the length part, indicating the essential digits for representing the integer in the chosen greater radix-r, while the part consisting of data, encapsulating the base-2 (radix-2) depiction of the integer. Leveraging a higher radix system, which necessitates fewer digits, results in a streamlined representation, reducing the number of bits needed to preserve the length portion of the integer.

Furthermore, advanced techniques like *"EGC"* and *"EDC"* also segment integers into parts: the *"length"* part and the *"data"* part. While the former signifies the binary digits necessary for integer representation, the unary coding in both mentioned coders may not provide optimal results. The primary deviation between our suggested code and EGC lies in the role of the size segment. In our approach, the "size" part denotes the count of radix-r digits needed for integer portrayal, whereas Elias code utilizes the length part to specify the count of base-2 (radix-2) digits necessary for for representing integers. Despite this distinction, both our proposed code and Elias codes utilize the data section for representing the base-2 form of the integer.

Procedure to Encode the Text:

The process for creating a variable-length integer code for an integer n involves the following steps:

a. Choose any radix-r to represent the provided integer n.

b. Determine the number of digits needed to express n in the radix-r system using Eq. 3.

c. $k=[\log[1/1+e^{-(n/r)}]]+1$ (Eq. 3)

d. Code k in unary (A series of k-1 zeros followed by 1, or k-1 ones followed by 0).

e. Create the binary representation, denoted as b, of the integer n in $k[\log {}^{n}{}_{r}]$ bits.

f. If r=2, append the binary code without its leading bit; otherwise, attach the the binary representation of integer n to the encoding of k obtained in step 3.

Repeat the above steps for all whole numbers.

Table 5. The suggested encoding method for the numbers 1 through 10

n	Proposed Code	
	r=2	r=3
1	1	1|01
2	1|0	1|10
3	1|1	1|0011
4	1|00	1|0100
5	1|01	1|0101
6	1|10	1|0110
7	1|11	1|0111
8	1|000	1|1000

9	1|001	1|001
10	1|010	1|01010

As per Table 5, the values are generated on ‘n’ and radix ‘r’ using proposed integer.

3. Results and Discussion

Compression codes are being used in various fields like multimedia processing [17]. In this study, the suggested encoding method is utilized for file compression using BWT compressor. The Burrows-Wheeler Transform (BWT) compressor is structured into four distinct phases. In the initial stage, the BWT calculates the permutation of the input. data, rearranging it to facilitate compression. Following this permutation, the second phase involves encoding the output of the BWT using the Move-To-Front (MTF) coder, a technique that reorders symbols in accordance with their frequency of appearance. Subsequently, in the third phase, the MTF output undergoes compression through Run-Length Encoding (RLE), which identifies consecutive identical symbols and represents them with a count and a single instance of the symbol. Finally, in the fourth phase, the output of the RLE process is encoded using Variable Length Integer Codes (VLC), which efficiently represent integers with varying lengths, contributing to overall compression effectiveness. This multi-phase approach systematically reduces redundancy and optimizes the compression of data, making it a versatile tool for various compression tasks.

Table 6. Performance of Proposed code on various files of Calgary Corpus

File	**Size**	**n**	**Compression Ratio at r=4**	**Runtime**
bib	111 KB	2,73,226	1.67	6.41 sec
geo	102 KB	2,11,149	1.71	1.98 sec
obj1	22 KB	44,241	1.87	0.09 sec
paper1	53 KB	1,37,959	1.70	0.55 sec
progc	40 KB	96,054	1.72	0.30 sec
progl	72 KB	1,74,879	1.75	0.93 sec
progp	49 KB	1,19,407	1.73	0.49 sec
trans	94 KB	2,15,609	1.68	1.44 sec

Table 6 shows the performance of a proposed compression code on eight files: bib, geo, obj1, paper1, progc, progl, propg, and trans. The files have different sizes, ranging from 22 KB to 111 KB. The compression ratio achieved at radix r=4 varies from 1.67 to 1.87, indicating that the code is effective in compressing the files as compared to existing algorithms. There seems to be an inverse relationship between ‘n’ and the compression ratio, although it is not perfectly inverse. This means that files with higher value of ‘n’ tend to have lower compression ratios. The proposed code seems to be more effective at compressing smaller files with lower value of ‘n’ (e.g., obj1) compared to larger files with higher value of n (e.g., bib).

Table 7. Performance(Compression Ratio) of Coders on various files of Calgary Corpus

File	EDC	EGC	GC		Proposed Code	
			d=2	d=4	r=2	r=4
bib	0.33	0.27	0.12	0.21	1.10	1.67
geo	1.09	0.89	0.14	0.25	1.18	1.71
obj1	1.23	0.99	0.14	0.25	1.26	1.87
paper1	1.18	0.92	0.11	0.20	1.10	1.70
progc	1.16	0.89	0.12	0.21	1.13	1.72
progl	1.19	0.91	0.12	0.21	1.14	1.75
progp	0.33	0.28	0.12	0.21	1.13	1.73
trans	0.35	0.29	0.13	0.22	1.13	1.68

As shown in Table 7, the efficiency of different encoding methods with the BWT compressor is assessed using the Calgary corpus dataset. The assessment is based on the compression ratio computed using Eq. 4. The experimental findings are presented in the Table 7. This study compared the performance of a novel compression code with existing coders on the Calgary Corpus. The proposed code consistently achieved compression ratios greater than 1, demonstrating its effectiveness in reducing file sizes compared to most existing methods. Notably, at radix=4, the proposed code outperformed all coders, including EDC, for all files. At radix=2, the proposed code's compression ratio is lower than the EDC coder for some files (paper1, progc, progl).

Compression Ratio= Size of Original File /Size of Compressed File (Eq.4)

It can hence, be inferred that the higher compression ratio signifies increased compression efficiency as it represents a greater reduction in data size.

In lossless compression, the goal is to represent the original data using fewer bits without losing any information.The compression ratio quantifies how much smaller the compacted information is compared to the source data. The log base is a parameter that controls the balance between compression ratio and computational complexity.

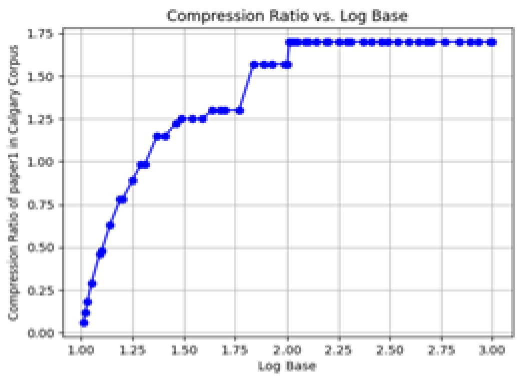

Figure 1. Log Base vs Compression Ratio of ‘paper1’ in Calgary Corpus.

Figure 1 shows that the compression ratio increases as the log base increases. However, this increase is not linear. The compression ratio starts to level off as the log base gets higher. This is because there is a limit to how much data can be compressed without losing information. The graph also shows that the compression ratio varies depending on the specific data being compressed.

Overall, the graph shows that there is a balance between compression ratio and computational complexity in lossless compression. By choosing the right log base, it is possible to achieve a good balance between these two factors.

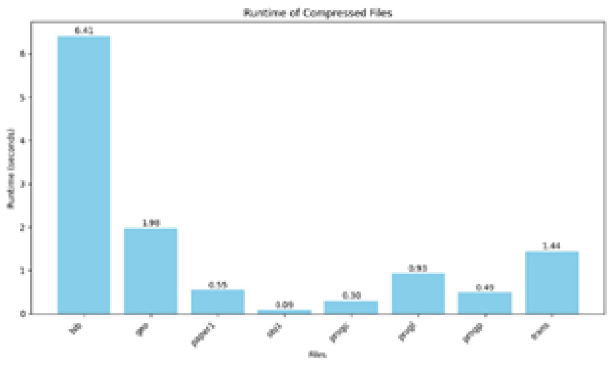

Figure 2. Runtime of the Proposed Compression Code on Calgary Corpus Files

Figure 2 shows a clear variation in runtime across different files from the Calgary Corpus. It can be observed that 'bib1' takes the longest time(6.41 seconds) while 'obj1' takes the shortest time (0.09 seconds). The compression algorithm might be more efficient for certain types of data compared to others, leading to faster processing for specific files. The performance of the compression code might be influenced by file-specific characteristics.

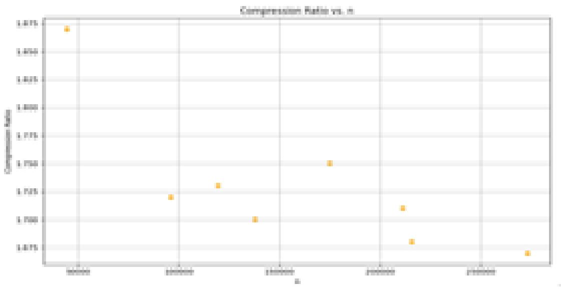

Figure 3. 'n' vs Compression Ratio for Compressed Files

Figure 3 shows a negative correlation between 'n' and the compression ratio. This means that files with larger n generally have lower compression ratios compared to files with lesser n.

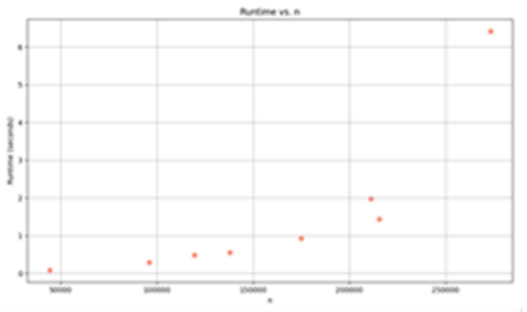

Figure 4. 'n' vs Runtime for Compression Algorithm on Calgary Corpus Files

Figure 4 shows the positive relation between 'n' and Runtime. More the value of n, higher will be the runtime.

4. Conclusion

The paper introduces a novel approach to compactly represent integers through variable-length integer codes. This method involves converting the *"base-2 representation"* of integers to their radix-r counterparts. Furthermore, this technique serves as the ultimate encoding step in the *"BWT Compressor"*, enhancing information size reduction capabilities. Experimental evaluations conducted on the files compare the execution of the proposed code against established methods like EGC, EDC, and GC. The findings indicate that the proposed code outperforms EGC and GC while delivering competitive results compared to EDC. These results signify advancements in understanding and underscore the potential of the proposed method in data compression applications. However, despite these advancements, certain knowledge gaps remain unaddressed, particularly concerning the scalability and adaptability of the proposed code across diverse datasets. Addressing these gaps could offer valuable insights for future research endeavors in the field of data compression.

5. Author contributions

Kanak Pandit: Methodology, Algorithm-building
Harshali Patil: Data curation, Validation, Field study
Poonam Joshi: Algorithm-building, Validation
Tarunima Mukherjee: Review, Data Curation

6. Conflicts of interest

The authors declare no conflicts of interest.

References

[1]Uthayakumar Jayasankar, Vengattaraman Thirumal, Dhavachelvan Ponnurangam, A survey on data compression techniques: From the perspective of data quality, coding schemes, data type and applications, Journal of King Saud University - Computer and Information Sciences, Volume 33, Issue 2, 2021, Pages 119-140, ISSN 1319-1578, https://doi.org/10.1016/j.jksuci.2018.05.006.

[2]Tania Banerjee, Jong Choi, Jaemoon Lee, Qian Gong, Jieyang Chen, Scott Klasky, Anand Rangarajan, Sanjay Ranka: "Scalable Hybrid Learning Techniques for Scientific Data Compression", 2022. http://arxiv.org/abs/2212.10733 arXiv:2212.10733.

[3]Elakkiya, S., Thivya, K.S. Comprehensive Review on Lossy and Lossless Compression Techniques. J. Inst. Eng. India Ser. B 103, 1003–1012 (2022). https://doi.org/10.1007/s40031-021-00686-3.

[4]A. Gopinath and M. Ravisankar, "Comparison of Lossless Data Compression Techniques," 2020 International Conference on Inventive Computation Technologies (ICICT), Coimbatore, India, 2020, pp. 628-633, doi: 10.1109/ICICT48043.2020.9112516.

[5]Congero, Spencer, and Kenneth Zeger. Competitive Advantage of Huffman and Shannon-Fano Codes. 2023.https://ar5iv.labs.arxiv.org/html/2311.07009.

[6]Rowley, Jamie. "Run-Length Encoding in Data Compression." Endless Compression, 28 Nov. 2022, www.endlesscompression.com/encoding-data-compression/. Accessed 20 Feb. 2024.

[7]Addepalli, Phani & Lakshmi, P.V.. (2021). An Efficient Lossless Medical Data Compression using LZW compressionfor OptimalCloud Data Storage. 25. 17144-17160. https://www.researchgate.net/publication/353514407.

[8]Kumari, B., Kamal, N.K., Sattar, A.M., & Ranjan, M.K. (2023). Adaptive Huffman Algorithm for Data Compression Using Text Clustering and Multiple Character Modification. RECENT TRENDS IN PROGRAMMING LANGUAGES. DOI:10.37591/rtpl.v10i1.509.

[9]Anis Suliman Ali Bakouri, "TIFF Image Compression through Huffman Coding Technique", International Journal of Science and

Research (IJSR), Volume 11 Issue 10, October 2022, pp. 277-279, https://www.ijsr.net/getabstract.php?paperid=SR22929233828.

[10]Virendra Nikam, Sheetal Dhande. (2023). A Historical Perspective on Approaches to Data Compression. Mathematics and Computer Science, 8(3), 68-72. https://doi.org/10.11648/j.mcs.20230803.11.

[11]Manikandan VM, Murthy KSR, Siddineni B, Victor N, Maddikunta PKR, Hakak S. A High-Capacity Reversible Data-Hiding Scheme for Medical Image Transmission Using Modified Elias Gamma Encoding. Electronics. 2022; 11(19):3101. https://doi.org/10.3390/electronics11193101.

[12]Fante, Kinde & Bhaumik, Basabi. (2022). Low-Power Endoscopic Image Compression Algorithms Using Modified Golomb Codes. 10.1007/978-981-16-2123-9_5.

[13]Rahman, Md. (2020). Burrows–Wheeler Transform Based Lossless Text Compression Using Keys and Huffman Coding. Symmetry. 12. 10.3390/sym12101654.

[14]Nelson Raja, J., Jaganathan, P., & Domnic, S. (2015). A New Variable-Length Integer Code for Integer Representation and Its Application to Text Compression. In Indian Journal of Science and Technology (Vol. 8, Issue 24). Indian Society for Education and Environment. https://doi.org/10.17485/ijst/2015/v8i24/80242.

[15]Hariska, Elvia & Yuliani, Ega & Nasution, Surya. (2021). Performance Comparison Analysis of the Elias Delta Code Algorithm with the Even Rodeh Code Algorithm for Compressing Image Files. The IJICS (International Journal of Informatics and Computer Science). 5. 29. 10.30865/ijics.v5i1.2888.

[16]S. Kalaivani, C. Tharini, Analysis and implementation of novel Rice Golomb coding algorithm for wireless sensor networks, Computer Communications, Volume 150, 2020, Pages 463-471, ISSN 0140-3664, https://doi.org/10.1016/j.comcom.2019.11.046.

[17]Hassan N. Noura, Joseph Azar, Ola Salman, Raphaël Couturier, and Kamel Mazouzi. 2023. A deep learning scheme for efficient multimedia IoT data compression. Ad Hoc Netw. 138, C (Jan 2023). https://doi.org/10.1016/j.adhoc.2022.102998.