Transforming Big Data: A Novel Arithmetic Compression Method Based on Symbol Frequency

Andrii Yarmilko
Department of Automated Systems Software
Bohdan Khmelnytsky National University of Cherkasy
Cherkasy, Ukraine
a-ja@ukr.net

Inna Rozlomii
Department of Information Technologies
Bohdan Khmelnytsky National University of Cherkasy
Cherkasy, Ukraine
inna-roz@ukr.net

Yuliia Mysiura
Department of Automated Systems Software
Bohdan Khmelnytsky National University of Cherkasy
Cherkasy, Ukraine
julmisura@ukr.net

***Abstract*— The article proposes a new method for arithmetic compression of large volumes of data. The used compression approach is based on dividing the original data into slices depending on the frequency of occurrence of symbols. The resulting cuts are subjected to arithmetic compression, which ensures an effective reduction of the volume of data, while maintaining the quality and reliability of information. The new method studies the statistical information on the frequency of occurrence of characters in the data set, which allowed setting the optimal place for dividing the data into slices. The main advantage of the proposed method lies in its ability to adapt to the specific characteristics of the data set, providing optimal partitioning and compression. This makes it possible to achieve a higher degree of stability with traditional arithmetic embossing methods while maintaining data quality. An additional advantage is the possibility of applying the method to various information types of data, which makes it a universal tool for optimizing the storage and transmission of large volumes.**

Keywords*— *arithmetic compression, digital data processing, big data, frequency analysis, adaptive model, data encoding, compression efficiency

I. INTRODUCTION

As the data volumes in the modern information society grow, there is a need for effective methods of storing and transmitting information [1]. Big data is becoming one of the most important fields of scientific research and technological development. The constant increase of the amount of information creates challenges related to the processing, analysis and storage of this data [2-4]. Research on the transformation of big data is also important in the context of solving the issues of ensuring the security of information flows and rational use of system resources in dynamic cooperative production systems [5]. Among the numerous approaches to optimizing the use of big data, one of the most important is data compression, which allows reducing the volume of data while preserving the quality of information [6].

In this context, a new approach to arithmetic data compression based on the use of frequency analysis is proposed. This method involves dividing the source data into subsegments depending on the frequency of occurrence of characters. This allows you to efficiently compress sub-segments with a higher symbol frequency and achieve a higher degree of compression of the total volume of data. The aim of the new method lies in application in various areas where optimization of storage and transmission of large volumes of data is important.

The purpose of this study is to develop and analyze a new method of arithmetic data compression based on frequency analysis. In addition, the paper conducts a comparative analysis of the proposed method with existing approaches to arithmetic compression in order to determine its effectiveness and advantages.

II. RELATED WORKS

It can be noted during the analysis of the existing publications and sources that arithmetic compression methods have been the object of research for a long time [7-11]. It is worth noting the method of arithmetic coding with the addition of cryptographic functions presented in [12], which enables additional protection and confidentiality during data compression. Many of the existing methods are based on statistical approaches and the use of probabilistic models for data compression [13]. However, most of them do not take into account the frequency of occurrence of symbols when determining the optimal data partitioning points. Thus, there is a need to develop a new method that uses symbol frequency information for efficient arithmetic data compression.

III. PROBLEM DEFINITION

Nowadays, during the backdrop of rapid growth of the amount of information, prognosis and data management are becoming critical tasks. One important aspect of this task is the ability of efficient data compression for storage and transmission purposes. Big data requires innovative approaches to compression, as traditional methods efficiency become insufficient [14-16].

Arithmetic compression methods have long been the object of research in the field of data compression. These methods are based on the use of probabilistic models and statistical approaches for information compression [17]. However, most existing methods do not take into account the frequency distribution of characters in the source data, which limits their effectiveness. Existing approaches to arithmetic compression are not always able to adapt to the specifics of different data sets, which limits their versatility and ability to achieve the optimal degree of compression.

One of the key issues is the lack of an arithmetic compression method that uses symbol frequency information to efficiently slice data and compress it further. The use of such information would make it possible to achieve a greater degree of compression and preserve the quality of data.

Therefore, a new arithmetic compression method is required, which would use frequency analysis to partition the data into subintervals and provide data compression that is more efficient. This study is aimed at the development and analysis of such a method, its advantages and potential applications in various areas of data processing and transmission.

IV. Compression Methods Based on Arithmetic Encoding

Let us consider existing compression methods that use arithmetic coding to reduce the amount of data. These techniques are key components of many modern compression algorithms, allowing information to be stored and transmitted with fewer bits.

The following common methods of data compression based on arithmetic coding can be noted:

- Adaptive arithmetic compression (AAC). This method uses adaptive probability models for data compression. It is able to adapt the probabilities of symbols based on previous information, which allows effectively compression of data from various types of sources [8-10].
- Range arithmetic compression. This method breaks the range of possible character values into subranges, with each character displayed in a corresponding subrange. The probabilities of selecting sub-ranges are set according to the frequency of occurrence of symbols [18].
- Context-tree Weighting (CTW). This method uses tree structures to store the character context. It supports the modeling of long dependencies between symbols, providing efficient compression of sequential data [11].
- Block arithmetic compression. This method divides the raw data into blocks of a certain size and compresses each block separately. It is particularly efficient for data where sub-segments can be compressed individually, but may lose efficiency on smaller blocks [19].
- Method of frequency analysis. This method is based on dividing the data into subsegments depending on the frequency of occurrence of characters. Its feature is the use of frequency analysis to determine the optimal places of breakdown and sub-segments for compression [20].

Each of these methods uses its own arithmetic coding strategy, adapted to a certain type of data or specific task requirements, and has its own advantages and disadvantages, and the choice of a particular method will depend on the type of data to be compressed and the requirements for compression and decompression.

The variety of existing methods includes the use of context, probabilistic models, tree structures and other approaches to achieve effective compression [14]. The main features of the compression methods are presented in Table I. This table provides an overview of some specific arithmetic compression methods and their main characteristics. While examining and comparing these methods, it is important to consider how they meet the specific requirements and types of data to be compressed. Each method has its advantages and limitations, and the choice of a particular approach may depend on the characteristics of the data, as well as performance and compression requirements.

TABLE I. Properties Of The Methods Of Data Arithmetic Encoding

Method	Main characteristics	Advantages	Disadvantages
Adaptive Arithmetic Compression (AAC)	Uses context models to adapt probabilities	Effective for a variety of data types	Requires more hardware resources
Range arithmetic compression	Splits a range of characters for each character	Simple interface	May show limited efficiency
Context-tree Weighting	Uses tree structures to adapt probabilities	Efficiently compresses sequential data	Requires more memory to store trees
Block Arithmetic Compression	Divides data into blocks and compresses them individually	Good for compressing large data, can have multi-threaded implementation	May lose compression efficiency on smaller blocks
Frequency analysis method	Uses frequency of occurrence of symbols for data partitioning	Effective for data with uneven symbol frequency	Requires additional calculations to calculate frequencies

V. Optimized Method of Data Compression

A. *The Algorithm of Arithmetic Data Encoding and Decoding*

Compressing large volumes of data requires taking into account the software and hardware capabilities of the platform that performs data compression. In the arithmetic compression methods described above, the following trends are revealed when the volume of input data increases:

- Methods that use trees in the process (Context-tree Weighting) face the fact that the size and complexity (number of levels) of the tree increases with the increase in the size of the input data. This means that for a certain amount of input data, the size of the tree may exceed the size of the input data. In addition, processing such a tree takes a lot of time.
- Methods that use statistical calculations during operation (frequency analysis method, adaptive arithmetic compression) face the fact that the statistical data required for their operation can be large.

Methods that use real numbers during calculations (range compression method) face the problem of achieving the necessary accuracy of calculations to ensure the possibility of restoring the initial data.

Therefore, in order to be able to compress data of arbitrary volume effectively, it is necessary to develop a method that combines the advantages of existing arithmetic compression methods and at the same time minimizes their disadvantages.

The software and hardware capabilities of the platform that performs data compression usually do not allow processing a large amount of information in one block, so the input data should be divided into blocks of a size that ensures the maximum speed of data compression and optimal memory consumption. In addition, the division into blocks allows you to develop a multi-threaded implementation of the compression algorithm, which will allow more efficient use of the capabilities of the hardware and software platform.

To achieve optimal calculation accuracy, the size of a single block of data must be a multiple of the size of a valid data type (for most platforms it is equal to 8 bytes, or 64 bits).

After dividing the input data into blocks, the actual data encoding process begins. The input data is treated as a sequence of text characters in order to abstract from the data type. The classic algorithm for arithmetic data compression goes as follows:

1) The first character of the input data stream is processed. It corresponds to any number from the interval [0; 1), which corresponds to this symbol. If the text consisted of exactly one character, then the work of the algorithm is completed. If there are more than one characters in the source text, then the working segment must be replaced with a sub-segment that corresponds to the first character, and the encoding process should continue.
2) The new working segment obtained in the previous step is divided into sub-segments, the length of which is proportional to the frequency of occurrence of characters in the text. This can be done using (1-2):

$$H = L_{\text{old}} + (H_{\text{old}} - L_{\text{old}}) \cdot H_{\text{Range}}(X) \quad (1)$$

$$L = L_{\text{old}} + (H_{\text{old}} - L_{\text{old}}) \cdot L_{\text{Range}}(X), \quad (2)$$

where L is the new lower limit of the working segment, H is the new upper limit of the working segment, L_{Old} is the current lower limit of the working segment, H_{Old} is the current upper limit of the working segment, X is the current symbol, $H_{Range}(X)$ is the upper limit of the current symbol, $L_{Range}(X)$ is the lower limit of the current symbol.

3) Step 2 is repeated for each subsequent input data symbol until the end of the input data stream.

The output data of the algorithm is any number of the interval obtained during the last iteration of the algorithm, the working segment, divided according to the frequencies with which symbols occur in the initial data stream (file), as well as the length (size, or the number of symbols) of the input data.

The decoding algorithm performs inverse operations to the encoding algorithm. The input data is an encoded message - a real number, the length of the message, as well as a working segment, divided according to the frequency of occurrence of characters in the source text (equivalent to a frequency table). Data decoding algorithm goes as follows:

1) The first symbol of the decoded data stream is determined by finding the symbol from the frequency table, which corresponds to the number from the current interval. If the length of the original message is greater than 1, then the algorithm continues its work.
2) To decode the next symbol, the current subinterval is normalized by bringing it to the segment [0; 1) according to (3):

$$C = \frac{C - L_{\text{Range}}(X)}{H_{\text{Range}}(X) - L_{\text{Range}}(X)} \quad (3)$$

where L is the new lower limit of the working segment, H is the new upper limit of the working segment, L_{Old} is the current lower limit of the working segment, H_{Old} is the current upper limit of the working segment, X is the current symbol, $H_{Range}(X)$ is the upper limit of the current symbol, $L_{Range}(X)$ is the lower limit of the current symbol.

3) Step 2 is repeated for each subsequent input data symbol until the end of the input data stream.

When applying this algorithm to a large amount of data, the speed of operations with real numbers may be insufficient (this is especially evident during the compression of data streams in real time). Therefore, it is necessary to define a combination of bit operations equivalent to mathematical operations on real numbers.

Since the considered algorithm uses numbers belonging to the interval [0; 1), then it is sufficient to represent only the fractional part of the number in memory. For this, it is necessary to use an integer type, the size of which is equal to the size of the real number type – unsigned long integer type (unsigned int64). Then the number 0 will have the form of the hexadecimal number 0x00000000, and the number 1 will have the form of the hexadecimal number 0x11111111. Also, to preserve the accuracy of calculations, it is necessary to represent the boundaries of subsegments in the form of structures with two fields that store the numerator and denominator of the fraction, respectively.

The algorithm with appropriate changes looks like this:

1) Generate the frequency table for the alphabet that forms the text.
2) Consider the first character of the text. It corresponds to any number from the working subsegment that corresponds to this symbol. If the text consisted of exactly one character, then the algorithm ends its work. If there are more than one characters in the source text, then the working segment is replaced with a sub-segment that corresponds to the first character, and continue encoding;
3) Divide the new working segment into sub-segments, the length of which is proportional to the frequency of occurrence of characters in the text. This can be done using (1-2);
4) If the value of the upper limit is less than the hexadecimal number 0x80000000 (which is equivalent to the decimal fraction 0.5), then bit 0 is output. Otherwise, bit 1 is output;
5) The values of the upper and lower limits are shifted by one bit to the left, and then the bitwise OR value of the upper limit and the number 1 is performed.
6) Step 2 is repeated for each subsequent input data symbol until the end of the input data stream is reached [21].

Mathematical operations in the decoding algorithm are similar to those in the encoding algorithm.

B. Comparison of the performance of the arithmetic coding algorithm on different sets of input data

In the process of working with a large amount of data, there is often a problem of the impossibility of processing the entire amount of data with one stream or file due to the large

amount of information and hardware limitations. Therefore, it is advisable to investigate how the size of a single block of data affects the speed of the algorithm.

The structure of the study was the following:

1) An initial block of big data was generated;
2) From this block of data, data sets for research were generated by slicing them into smaller blocks of a certain size;
3) The time spent on the complete coding of each of the data sets for the experimental study was measured. Since encoding and decoding take approximately the same time, it is necessary to perform only the data encoding.

The research parameters were as follows: the total size of the data block is 1 gigabyte, the data blocks were generated with the tools of the Linux operating system. The study was conducted without the use of multithreading tools.

The results of the experiments are given in Table II, as well as in Fig. 1 and Fig. 2. The abnormally high encoding time of a large number of small blocks is due to the fact that some time is spent on data input and output. With a small block size to be encoded, the system spends more time reading and writing data than compressing data.

TABLE II. ENCODING TIME OF LARGE BLOCKED INPUT DATA

Number of blocks	Size of a single data block, bytes	Encoding time of a single block, s	Total data encoding time, s
1073741824	1	0,003	3221225,472
536870912	2	0,004	2147483,648
268435456	4	0,003	805306,368
134217728	8	0,003	402653,184
67108864	16	0,003	201326,592
33554432	32	0,003	100663,296
16777216	64	0,003	50331,648
8388608	128	0,003	25165,824
4194304	256	0,003	12582,912
2097152	512	0,003	6291,456
1048576	1024	0,004	4194,304
524288	2048	0,004	2097,152
262144	4096	0,005	1310,72
131072	8192	0,007	917,504
65536	16384	0,011	720,896
32768	32768	0,019	622,592
16384	65536	0,034	557,056
8192	131072	0,067	548,864
4096	262144	0,124	507,904
2048	524288	0,318	651,264
1024	1048576	0,502	514,048
512	2097152	0,954	488,448
256	4194304	2,476	633,856
128	8388608	4,248	543,744
64	16777216	7,564	484,096
32	33554432	15,713	502,816
16	67108864	30,33	485,28
8	134217728	60,822	486,576
4	268435456	121,271	485,084
2	536870912	243,962	487,924
1	1073741824	485,593	485,593

Presented in Fig. 1-2 dependencies allow us to conclude that too large a number of single data blocks, and, therefore, too small a single block size, leads to a decrease in data compression speed.

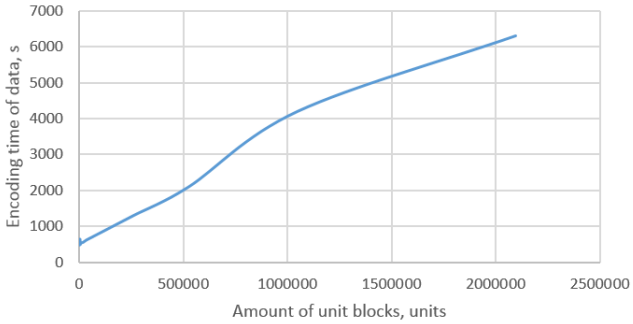

Fig. 1. Dependence of the time spent on data compression on the amount of unit blocks

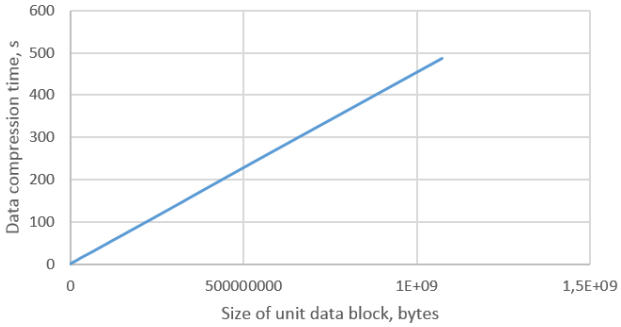

Fig. 2. Dependence of the time spent on data compression on the size of the unit block

From the results of the experimental study, it can be concluded that for more effective compression of large volumes of data, attention should be paid to the following criteria:

- The number of single blocks into which the input data stream is divided must not be too large, because at the same time the time that the computer system spends on information input and output becomes a significant amount.
- A properly implemented multi-threaded version of the arithmetic compression algorithm can significantly increase overall performance.

Compression efficiency also depends on the type of data, as different data types have different statistical distributions of byte (character) frequencies in a file. In other words, if the input file already contains compressed data (an example of such a format is JPEG), then the efficiency of the subsequent compression of information will be lower. This statement is true for all data compression methods.

DISCUSSION

This article discusses the variety of existing methods of arithmetic data compression and their main characteristics. It is noted that each method has its advantages and limitations, which makes them suitable for various tasks of compression of large volumes of information. It is important to note that the efficiency of arithmetic compression is determined not only

by the choice of a specific method, but also by the specifications of the data to be compressed. Some methods may be more suitable for textual data with complex relationships between characters, while others may be more efficient for numeric sequences or other types of information.

The arithmetic compression frequency analysis method proposed in our study uses an innovative approach based on using the frequency of occurrence of symbols to divide data into subsegments. The segmentation process is performed using frequency analysis, which allows identification of the characters and their combinations that occur most often. This allows for an adaptive compression model that can effectively account for a variety of structures and patterns in the data. This approach allows achieving a greater degree of compression of the total amount of data, as it is aimed at optimal use of resources for storing and transmitting information.

One of the main advantages of this method is its variability and adaptability to different types of data. It can be applied effectively to a variety of formats, for example, text data, images or videos. In addition, due to frequency analysis, this approach is able to adapt to changes in the data structure, ensuring constant compression efficiency.

The application of a new approach to arithmetic data compression can have important consequences for many areas that work with large amounts of information. Reducing the amount of data contributes to a more efficient use of computing resources, reduces data storage requirements and speeds up the transfer of information. This can have a positive impact on a variety of industries, from scientific research and medical diagnostics to the entertainment industry and information technology.

Evaluating the properties of the proposed approach, we performed compression of large volume audio and video data. The achieved levels of compression are significant in terms of overall positive consequences for the dynamics of communication systems. A positive property of the method is its acceptability for joint use with cryptographic methods. This creates prerequisites for greater complexity of information infrastructure systems and increasing their reliability (dependability) both by sending a minimized amount of data without losing their variety and veracity, and by implementing information protection measures.

Therefore, a new approach to arithmetic data compression based on frequency analysis is a promising direction of research in the field of optimizing the use of big data. Its ability of efficient compression of data with high information quality can open new opportunities for the development of information technology and various industries based on data processing. Continued research in this area should focus on developing methods that are more versatile, efficient, and adaptable to a wide range of data. Understanding the variety of existing arithmetic compression methods and their characteristics will help researchers and engineers choose the most suitable method for specific data compression tasks, ensuring efficient and high-quality optimization of information processing.

REFERENCES

[1] S. Boubiche, D. E. Boubiche, A. Bilami and H. Toral-Cruz, "Big data challenges and data aggregation strategies in wireless sensor networks", IEEE access, 6, pp. 20558-20571, 2018.

[2] F. Restuccia and T. Melodia, "Big data goes small: Real-time spectrum-driven embedded wireless networking through deep learning in the RF loop", in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, IEEE, pp. 2152-2160, April, 2019.

[3] H. U. Amin, M. Z. Yusoff and R. F. Ahmad, "A novel approach based on wavelet analysis and arithmetic coding for automated detection and diagnosis of epileptic seizure in EEG signals using machine learning techniques", Biomedical Signal Processing and Control, 56, p. 101707, 2020.

[4] S. M. Darwish, "A modified image selective encryption-compression technique based on 3D chaotic maps and arithmetic coding", Multimedia Tools and Applications, 78(14), pp. 19229-19252, 2019.

[5] A. Yarmilko, I. Rozlomii and H. Kosenyuk, "Hash Method for Information Stream's Safety in Dynamic Cooperative Production System", in *Mathematical Modeling and Simulation of Systems: Selected Papers of 16th International Scientific-practical Conference MODS 2021*, Springer International Publishing, Cham, Switzerland, pp. 173-183, 2022.

[6] L. Wen, K. Zhou, S. Yang and L. Li, "Compression of smart meter big data: A survey", Renewable and Sustainable Energy Reviews, 91, pp. 59-69, 2018.

[7] S. Pandit, P. K. Shukla, A. Tiwari, P. K. Shukla, M. Maheshwari and R. Dubey, "Review of video compression techniques based on fractal transform function and swarm intelligence", International Journal of Modern Physics B, 34(08), pp. 2050061, 2020.

[8] Z. Guo, J. Fu, R. Feng and Z. Chen, "Accelerate neural image compression with channel-adaptive arithmetic coding", in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, IEEE, pp. 1-5, May, 2021.

[9] J. J. Ding, I. H. Wang and H. Y. Chen, "Improved efficiency on adaptive arithmetic coding for data compression using range-adjusting scheme, increasingly adjusting step, and mutual-learning scheme", IEEE Transactions on Circuits and Systems for Video Technology, 28(12), pp. 3412-3423, 2017.

[10] O. Rippel and L. Bourdev, "Real-time adaptive image compression", in *International Conference on Machine Learning*, PMLR, pp. 2922-2930, July, 2017.

[11] V. Lungu, I. Papageorgiou and I. Kontoyiannis, "Bayesian change-point detection via context-tree weighting", in *2022 IEEE Information Theory Workshop (ITW)*, IEEE, pp. 125-130, November, 2022.

[12] S. T. Klein and D. Shapira, "Integrated encryption in dynamic arithmetic compression", Information and Computation, 279, p. 104617, 2021.

[13] H. Devi Kotha, M. Tummanapally and V. K. Upadhyay, "Review on lossless compression techniques", Journal of physics: conference series, Vol. 1228, No. 1, p. 012007, IOP Publishing, 2019.

[14] U. Jayasankar, V. Thirumal and D. "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, 33(2), pp. 119-140, 2021.

[15] K. Sayood, Introduction to data compression. Morgan Kaufmann, 2017.

[16] Y. Xing, G. Li, Z. Wang, B. Feng, Z. Song and C. Wu, "GTZ: a fast compression and cloud transmission tool optimized for FASTQ files", BMC bioinformatics, 18(16), pp. 233-242, 2017.

[17] S. Boopathiraja, P. Kalavathi and S. Chokkalingam, "A hybrid lossless encoding method for compressing multispectral images using LZW and arithmetic coding", Int J Comput Sci Eng, 6, pp. 313-318, 2018.

[18] S. J. Sarkar, P. K. Kundu and G. Sarkar, "Development of lossless compression algorithms for power system operational data", IET Generation, Transmission & Distribution, 12(17), pp. 4045-4052, 2018.

[19] U. Sharma, M. Sood and E. Puthooran, "A block adaptive near-lossless compression algorithm for medical image sequences and diagnostic quality assessment", Journal of digital imaging, 33(2), pp. 516-530, 2020.

[20] Q. Liu, Y. Xu, and Z. Li, "DecMac: A Deep Context Model for High Efficiency Arithmetic Coding", in *2019 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, IEEE, pp. 438-443, February, 2019.

[21] "Data Compression With Arithmetic Coding," M. Nelson [Online] Available: https://marknelson.us/posts/2014/10/19/data-compression-with-arithmetic-coding.html (Accessed on August 20, 2023).