# Applying the Arithmetic Compression Method in Digital Speech Data Processing

Serhii Zabolotnii [1], Andrii Yarmilko [2], Inna Rozlomii [2] and Yuliia Mysiura [2]

[1] *Cherkasy State Business College, 243, Vyacheslava Chornovola st., Cherkasy, 18028, Ukraine*
[2] *Bohdan Khmelnytsky National University of Cherkasy, 81, Shevchenko Blvd., Cherkasy, 18031, Ukraine*

**Abstract**

The article investigates the application of the arithmetic compression method based on the division into subsections depending on the frequency of occurrence of symbols in the digital processing of speech data. The general problems of applying data compression in digital speech processing systems are revised. Authors propose a new method that uses statistical information about the frequency of occurrence of characters to partition the data into subsegments that can be more efficiently compressed using arithmetic compression. The main algorithms of the method are described. Since the method of arithmetic data coding used in the study requires high-precision calculations, long arithmetic is used in its software implementation. A way to optimize the calculation algorithm is proposed, given the general inefficiency of arithmetic coding using the methods of long arithmetic, in terms of both time spent and the size of the result. A comparison of the proposed method with other speech data compression algorithms was performed. The nature of the dependence of the required coding accuracy on the amount of input data and the dependence of the total encoding-decoding time on the size of the input data was discovered. The results of the study showed that the proposed method is more effective compared to other methods. However, in practical application it is necessary to take into account that the method has limitations. In particular, while increasing the number of subsections, the processing time increases, and while decreasing the number, the size of the compressed data increases. It is also important to take into account phonetic, syntactic and phraseological features of specific speech data, since different languages have different frequency of use of certain symbols. A new method of speech data compression can be used to improve transmission efficiency and improve audio file storage technologies.

**Keywords**

Arithmetic compression, long arithmetic, digital speech processing, audio coding, audio compression, compression efficiency, coding process speed.

## 1. Introduction

Nowadays digital speech processing systems are gaining more and more popularity, allowing to process speech data, in particular, sound and speech signals, interpret them and perform various actions based on them. A digital speech processing system includes such components as speech recognition and synthesis, speech translation, speech signal analysis and synthesis, acoustic speech modeling, information retrieval using speech queries, and much more [1].

The use of digital speech processing systems is widespread in the fields of automated telephone support, medicine, diagnostics, security, intelligence, the military and space industry, communications, the entertainment industry, and other areas where it is important to analyze and

*Information Technology and Implementation (IT&I-2023), November 20-21, 2023, Kyiv, Ukraine*
EMAIL: zabolotniua@gmail.com (Serhii Zabolotnii); a-ja@ukr.net (Andrii Yarmilko); inna-roz@ukr.net (Inna Rozlomii); julmisura@ukr.net (Yuliia Mysiura)
ORCID: 0000-0003-0242-2234 (Serhii Zabolotnii); 0000-0003-2062-2694 (Andrii Yarmilko); 0000-0001-5065-9004 (Inna Rozlomii); 0000-0002-3844-2563 (Yuliia Mysiura)

process speech data [2-4]. Since a significant amount of speech and text data is generated every day in the listed industries, their processing and storage becomes an extremely important task.

The need of text and speech content compression arises for many reasons [5]. One of the most important reasons is that resources, such as disk space, RAM, network bandwidth, and others, are limited. In particular, in medicine, speech and text data compression methods can help reduce the volume of medical documentation, in intelligence and security – to reduce the volume of transmitted data and store it in a more convenient format, and in the space industry – to reduce the time of data transmission and processing over long distances [6].

Compression allows to reduce the size of text and speech data, which ensures effective use of resources and increases the efficiency of their transmission and storage. Also, text compression can be used to reduce the reading time and improve productivity, and speech content compression can help improve the quality of voice information transmission [7]. In addition, compression is an important component of many text and speech processing systems, such as speech recognition systems, machine translation systems, speech synthesis systems, and others. In these systems, compression is used to increase performance and reduce the amount of data being processed.

Thus, the development of effective methods of compression of speech and text data is an important task in solving the problems of improving the technologies of their storage, transmission and usage.

## 2. Related Works

The existing data compression algorithms in digital speech processing systems include progressive compression, adaptive compression, vector quantization, and speech compression by polynomial approximation [8]. The authors of [9] propose an algorithm for speech data compression using the vector quantization technique using the Linde-Buzo-Gray, Kekre's Proportional Error, and Fast codebook Generation algorithms. Usually, transformations, which are lossy algorithms, are used to compress speech data. Such algorithms are acceptable for speech data compression, since the loss of quality is hardly noticeable to the human ear. However, vector quantization can provide even more data compression while maintaining the same quality.

The issue of using arithmetic coding for information compression in digital speech processing systems is poorly researched. Although this method is very effective for data compression, in particular for text messages, currently there is not enough research on its application to audio and video data in speech processing systems. One of the reasons for this limitation is the high computational complexity of arithmetic coding, since this method requires the exact calculation of probabilities for each symbol or group of symbols in the original message. This can become a problem in the field of processing large volumes of audio and video data in real time. In addition, gathering suitable data to build probabilistic models for arithmetic coding for speech data can be a challenging task. In further research, it is necessary to solve these questions and determine effective methods of applying arithmetic coding for information compression in digital speech processing systems.

The arithmetic coding method can be a promising direction of research in the field of lossless information compression in digital speech processing systems. This method allows to achieve a compression of high degree with close to no loss of quality. Research in the field of arithmetic coding for image compression has already shown promising results. For example, an arithmetic coding method has been developed for lossless image compression, which allows achieving a high degree of compression with little loss of quality. However, the application of arithmetic coding for speech data compression still leaves significant unexplored areas. Research in this direction can lead to the creation of new effective methods of lossless speech data compression [10-11].

## 3. Problem Definition

There are certain disadvantages associated with the use of speech content compression methods, which can affect the quality of compression and cause information loss. One of them is that compression methods can lead to the loss of important information about the speech content. For example, when using compression methods that remove useless (redundant) information from an

audio or video stream, certain sound elements or visual elements may be lost, which reduces the quality of playback. Another disadvantage is related to the requirements for computing resources for compression. Some methods of compressing speech content cause heavy loads on computing resources, which can lead to delays in the playback of an audio or video stream on specific hardware and software platforms. This is especially important for operations with real-time broadcast content, such as video broadcasts or telephone calls. There can also be some problems with the stability of data that is stored in a compressed form. For example, if an error occurred during data transmission, part of the compressed information might be lost. This may result in the need to re-compress and re-transmit the data.

Therefore, despite the existence of various methods of compressing speech content, each of which has its own advantages and disadvantages. The choice of compression method depends on the quality requirements and available computing resources for compression. According to this, the purpose of this study is to develop a method of arithmetic compression using frequency partitioning into subsegments. The method should provide effective compression of text or speech content with minimal information loss. Important criteria for the development of such a method are speed, compression quality, and requirements for computing resources.

## 4. Speech Data Compression Methods

The choice of speech compression method has to be based on the needs and characteristics of the specific application, such as processing time, audio quality requirements, and file size.

In general, speech data compression methods can be lossless or lossy. Lossless compression reduces data size without losing information. This is achieved through compression algorithms that are used to remove redundant information, duplicate data, and other methods that do not change the meaning of the data itself.

On the other hand, lossy compression may cause partly loss of some information. The lossy compression uses various methods, such as terminating the transmission of an audio signal that has a very low volume level, or removing low-frequency components from the signal that adversely affect speech intelligibility.

The following most common methods of speech data compression can be noted:

1. Arithmetic compression.
2. Fourier transform.
3. Wavelet transform.
4. Linear prediction.
5. Neural networks usage.

Each of these methods has its advantages and disadvantages, and their suitability for a task depends on the needs of the user, the nature of the data, and the other factors. However, arithmetic compression is one of the most efficient methods for lossless speech data compression, while the use of neural networks can be useful for lossy speech data compression with minimal information loss. Using arithmetic coding with frequency division into subsections can provide even greater compression efficiency, reducing the size of data without loss of quality.

The task of the arithmetic compression method is representing text or speech content as a single number in a certain numerical interval. This method can be applied to different types of data such as text, image, video and audio. Frequency segmentation involves dividing text or speech content into segments with different frequencies of use. This allows storing more frequent information with fewer bits. Arithmetic compression is an effective way to reduce the volume of text data, but its application can be useful in other fields, in particular, in digital speech processing. Audio files and video files with speech content can be compressed this way, therefore saving disk space and increasing the speed of data transferring.

Application of this method of audio and video files with speech content compression involves converting a sound wave into a sequence of symbols that can be encoded using arithmetic coding. At the same time, each character from the text or speech content is encoded into a real number interval based on its frequency of occurrence in the text. For this, frequency partitioning is used, that is the partitioning of a set of symbols into subsets, each of which contains symbols with approximately the

same frequency in the text. The basic idea of arithmetic compression is that each sub-segment formed by splitting the text is coded using arithmetic coding. Once the text or speech content has been segmented, the interval for each segment can be determined based on its frequency in the text. Each character is then encoded into the real number interval corresponding to its subsegment. The intervals for all symbols are combined into one large interval that can be encoded and transmitted more efficiently.

One of the main challenges of applying the arithmetic compression method to processing audio and video files is the preservation of quality of sound and video. In particular, among the approaches used to solve this problem, the method of reducing the number of bits for encoding each symbol and the methods of pseudo-random symbol generation are known.

In general, the method of arithmetic compression based on dividing content into subsections is a powerful tool for optimizing content's parameters in information processes. For example, when compressing audio files, the audio data is divided into subsegments corresponding to the frequency range, and then arithmetic compression methods are applied to reduce the file size. At the same time, sound quality is preserved, as the most important and frequent frequencies are preserved with high accuracy, while less important frequencies can be removed or replaced with less accurate values.

Similar methods can be applied to compress video files with speech content. In this case, the images are divided into subsections corresponding to certain colors and areas of the image, and then arithmetic compression methods are applied to reduce the file size. This processing preserves picture and sound quality, as the most important and frequent information elements are stored with high precision.

# 5. Proposed Method

## 5.1. Algorithm of Arithmetic Data Compression and Decompression

The arithmetic data compression algorithm is using as the example the processing of text messages. This example is general in relation to other types of messages, since any information can be presented to a program as a sequence of characters. The algorithm takes the segment of real numbers [0; 1). Then it matches each symbol of the input text with a real number segment, length of which is equal to the frequency of this symbol's occurrence in the text. After calculation of the frequency of symbols' occurrence in the text, the algorithm compiles the complete working segment, which consists of sequentially located sub-segments. Thus, each character of the input text will be matched with a segment of real numbers.

The data coding process consists of the following steps:

- Considering 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 algorithm replaces the working segment with a sub-segment that corresponds to the first character, and continue encoding process.
- Division of the new working segment into sub-segments, the length of which is proportional to the frequency of occurrence of symbols in the text. The parameters of the subsections are determined by formulas:

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

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

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 – the current symbol, $H_{Range}(X)$ – the upper limit of the current symbol, and $L_{Range}(X)$ – is the lower limit of the current symbol.

- Repeat the previous step for each subsequent character until the processing of the source text is finished.

After the algorithm has finished its work, we get an interval (lower and upper limits) from which any number should be taken, as well as the working segment, divided according to the frequency of occurrence of characters in the source text. It is also necessary to preserve the length of the source text so that the decoding algorithm can have a sign of completion.

The decoding algorithm is the inverse of the encoding algorithm. The input data for it is a real number – the encoded message, the length of the message, as well as the working segment, broken down according to the frequency of occurrence of characters in the source text (frequency table). The algorithm has the following steps:

- Determining of the interval of the working segment where the current code lies. This makes a possibility of determining the first character of the source text. If the length of the original message is greater than 1, then the algorithm continues its work;
- To decode the second symbol, it is necessary to normalize the current subinterval – to bring it to the segment [0; 1) according to the formula:

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

where C is the current code, X is the current character, $H_{\text{Range}}(X)$ – the upper limit of the current character, $L_{\text{Range}}(X)$ – the lower limit of the current character. Next, the current subinterval is replaced by the one corresponding to the decoded character;

- Repeat the previous step until the message is fully decoded.

## 5.2. The Minimum Accuracy Hypothesis

The arithmetic data encoding method used in the paper requires calculations of a precision that conventional floating-point calculations cannot provide. One way to ensure the required precision is to use long arithmetic. Its feature is that the maximum accuracy of calculations with is limited only by the hardware resources of the computer. Also, while using long arithmetic, it is possible to set the accuracy of calculations corresponding to the needs of the task.

Let us assume that there is some dependence of the minimum calculation accuracy on the amount of input data, which is ensured when using arithmetic compression to minimize the amount of information. Then we explore how the required minimum accuracy of calculations changes depending on the amount of input data. Figure 1 shows the example of dependency by one of the series of input data. Results for the other series of data streams with other properties were similar.

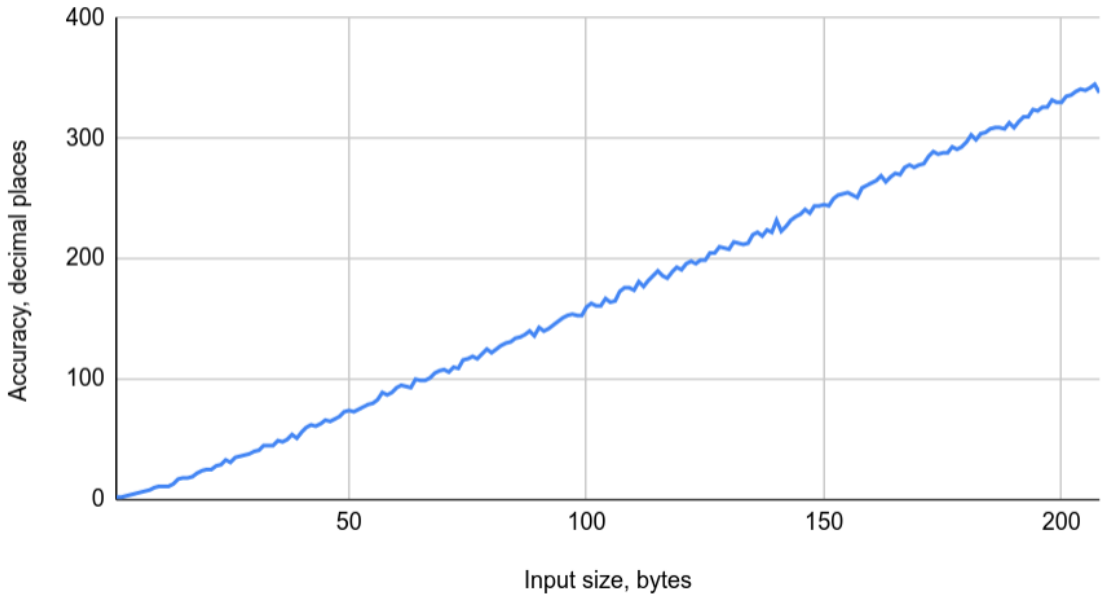


**Figure 1:** Dependency of the amount of input data (in bytes) on accuracy (in decimal places).

The research was conducted in two stages. On the first, the generation of input data – random strings with length varying from one character to a certain specified number of characters was performed. The second step is to try arithmetic encoding, and then decode the string with the specified precision: ε=0.1 (one decimal place), then ε=0.01 (two decimal places), and so on. This process

continues until an attempt at arithmetic encoding and then decoding results in output which is equal to the input string.

The results of the experiments were obtained in the form of dependencies of the amount of input data in bytes (X axis) on accuracy in decimal places (Y axis). The input data for each experiment were obtained using random generator. Therefore, some statistical error, which doesn't affect results of experiments, occurs.

It can be concluded from these results that the dependency of the desired coding accuracy on the amount of input data is linear (the linear coefficient is approximately equal to 1.75), taking into account the statistical error associated with the input data. Since the data is random, the statistical error does not affect the result of the study.

## 5.3. Dependence of the Time Spent on Encoding on the Amount of Input Data

Let's investigate the dependency of the time spent on arithmetic coding on the amount of input data. To do this, we measure the time spent on successive operations of arithmetic coding and decoding of a line (the necessary accuracy of calculations is determined by the formula:

$$Acc = 1.75 \times Num_{char} \quad (4)$$

A graphical presentation of the results is shown on Figure 2.

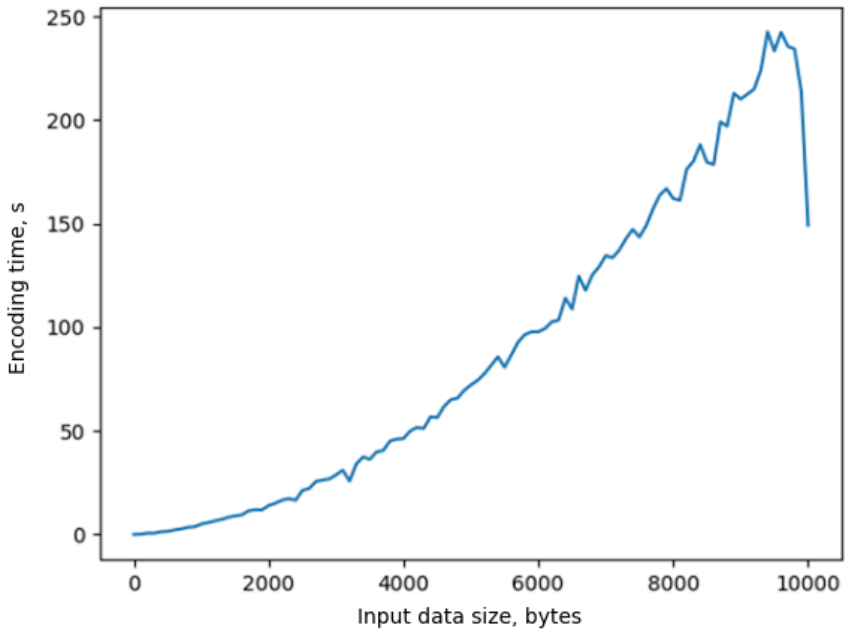


**Figure 2:** Dependency of the time spent on encoding on the amount of input data.

Taking into account the statistical error associated with the input data, we see that the dependency of the total encoding-decoding time (in seconds) on the size of the input data (in bytes) is polynomial. This means that in order to transmit a large amount of data, it should be divided into smaller parts and already coded to minimize the consumption of time and hardware resources.

In Figure 2, it can be seen that the time spent on conducting the last experiment in the sequence (compression of the last set of data) is significantly less than the time spent on conducting previous experiments. This is because the experiments (arithmetic encoding of files of different sizes) were performed sequentially, one after the other, using a multi-threaded toolkit that used a large number of threads. The obtained results illustrate the importance of the correct number of streams for the optimal performance of the data encoding and decoding system. They also illustrate the amount of time needed

to support the lifecycle of the large number of threads. An excessive number of threads can lead to serious delays in the work of the software implementation.

## 5.4. Optimization of the Algorithm

The results of the previous study show that arithmetic coding using long arithmetic methods is not efficient either in terms of time spending (a large file on several MB can be coded for tens of minutes) or in terms of the size of the result, since the required accuracy of calculations – the number of decimal places – exceeds the size input data in bytes. With the need to store a frequency table of the characters that occur in the input data, this results in the size of the file that results from the arithmetic encoding being much larger than the size of the original file. Moreover, this does not correspond to the original goal – the maximum possible compression of the source file.

Let's analyze the most critical part of the algorithm in terms of execution time and, thus, its software implementation – mathematical calculations.

In the long arithmetic used to implement the algorithm explored in the previous paragraph, numbers are represented by objects rather than numeric primitives. This way of representing numbers has its advantages and disadvantages. The advantage is that the long arithmetic classes that represent integers and real numbers have virtually no restrictions on the absolute value of the number (integer types) and the number of decimal places (real numbers). This gives wide possibilities for mathematical calculations. So, for example, the Decimal class, which is in modern programming languages and represents a real number of arbitrary precision, is used in financial calculations.

The main disadvantage of long arithmetic is that it represents mathematical operations (addition, multiplication, etc.) as operations on objects that take longer to perform than mathematical operations on numeric primitives (int, float, double, and other numeric data types). This disadvantage is most significant when the algorithm performs a large number of calculations (such as the algorithm for arithmetic data coding, which is considered in this paper).

But there is a way to achieve the necessary accuracy of calculations without using the means of long arithmetic. If some knowledge of computer arithmetic is used, then we will find that we need only three arithmetic operations – addition, subtraction, bit shift to the right or left, as well as bitwise logical operations - conjunction (bitwise AND) and disjunction (bitwise OR).

Let's start with the representation of numbers in memory. 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. Let us use an unsigned long integer type (unsigned int64). Then the number 0 will be presented in the form of a sixteen number 0x00000000 (binary fraction 0.0000000000000000000000000000000000000000000000000000000000000000 – 64 binary places). And the number 1 will have the form of a binary fraction 0.1111111111111111111111111111111111111111111111111111111111111111 (also 64 binary places). This representation is correct because the algorithm uses numbers in the interval [0; 1). Therefore, the number 1, which is binary 1.0, must be represented with the closest number from the interval, being 0.1111111111111111111111111111111111111111111111111111111111111111 (also 64 binary places). It is also possible to represent the boundaries of subsegments in the form of structures with two fields that store the numerator and denominator of the fraction, respectively.

Thus, the modified algorithm has such steps:

- A frequency table is created for the alphabet that forms the text.
- The first character of the text is processed. It fits to any number from the working subsegment that corresponds to this symbol. If the text consists 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 by a subsegment that corresponds to the first character, and the encoding continues;
- The new working segment must be 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 formulas 1-2.

- If the value of H is less than the hexadecimal number 0x80000000 (decimal fraction 0.5), then bit 0 is put to the output stream. Otherwise, if the value of L is greater than the hexadecimal number 0x80000000, then bit 1 is output.
- Shift the values of L and H by one bit to the left, and then perform a bitwise OR of the value of H and the number 1.
- These steps are repeated for each subsequent character until the source text is fully processed [12].

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

## 5.5. Comparison of the Speed of Arithmetic Coding Algorithm Implementations in Different Programming Languages

The performance of the proposed arithmetic coding algorithm strongly depends on the selected implementation tools (programming language). This study compared the implementation of this algorithm in two common languages: Python and C++ (Table 1).

**Table 1**

Dependence of the speed of the arithmetic compression method on the method of software implementation

| № of experiment | Size of the input file, KB | Python encoding time, s | C++ encoding time, s |
|---|---|---|---|
| 1 | 1.5 | 0.125 | 0.042 |
| 2 | 14.6 | 0.6 | 0.012 |
| 3 | 687.6 | 20.483 | 0.314 |
| 4 | 933.7 | 34.697 | 0.450 |
| 5 | 1206.2 | 31.353 | 0.456 |
| 6 | 3039.88 | 107.978 | 1.494 |

The time spent on decoding a file is approximately the same as the encoding time. Therefore, the total processing time (encoding and decoding of the file) will be approximately twice as large as the given results. From Table 1, we can see that the implementation of the arithmetic coding algorithm in the Python language takes longer than the implementation in the C++ language, and this difference increases when the amount of input data increases. A large file of about 3 megabytes is processed by the Python implementation 100 times slower than the C++ implementation. Therefore, it is advisable to implement arithmetic coding and decoding algorithms in the C++ language, and then compile it in the form of a library.

In general, the significant advantage of C++ implementation is expected. However, this advantage has to be considered in combination with other properties of the software implementation of the algorithm, such as the specifics or limitations of development process of the software product, computer infrastructure or the possibility of integrating the modules of the software product with each other. It should be noted that the choice of tools for the software implementation of the compression algorithm depends on the context of its implementation in the application system, taking into account its hardware and software platform, as well as the available means of working with data structures necessary for the correct implementation of data compression algorithms and their unpacking (decoding).

## 5.6. Comparison of the efficiency of modified arithmetic compression method with the classic variant of the algorithm

Let us compare the efficiency and speed of the classic algorithm of arithmetic compression, described in section 3, in comparison with the modified algorithm described in subsection 5.4. Fig. 4-5 show the ratio of performance, as well as the efficiency of data compression.

To compare the effectiveness of arithmetic data compression methods, the approach to input data generation was changed. The method of generating a text file from a dictionary was used to create a large input file. In Figure 3 and Figure 4 a comparison of the time spent and the efficiency of compression of data blocks by the classic method of arithmetic coding and the modified method presented in section 5 is given.

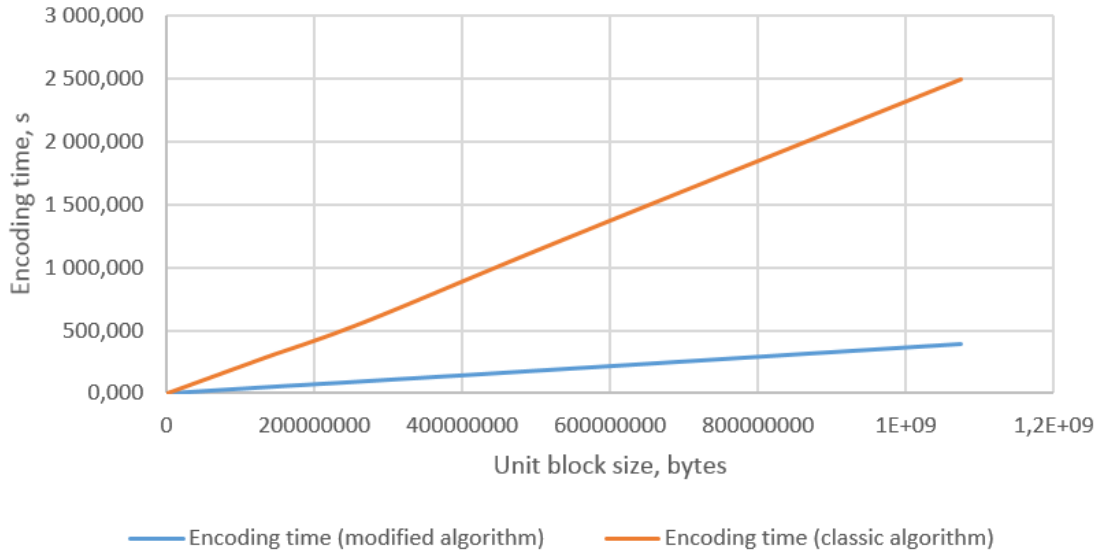


**Figure 3:** Dependency of compression (encoding) time on unit block size for the two algorithms

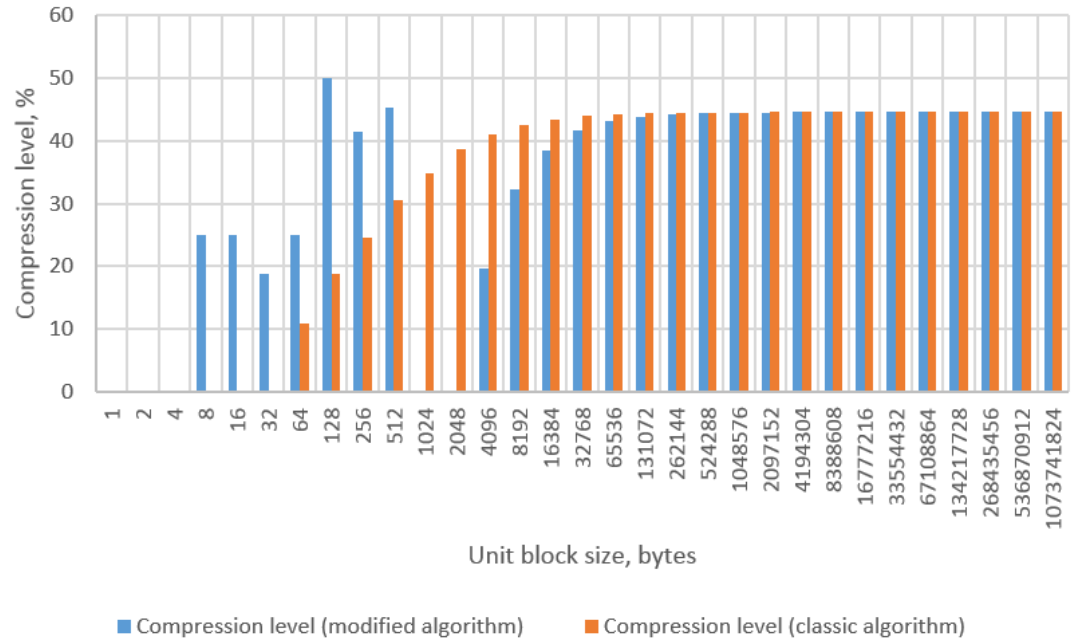


**Figure 4:** Dependency of data compression level on the unit block size for the two algorithms

The obtained results allow us to conclude that the modified data compression algorithm presented in this article is faster than the classical one. The level of data compression when using both algorithms on large blocks of data is approximately the same. On some data sets, due to their structural features, the result of information compression may become larger than the original data, but such an anomaly is rare and indicates that the statistical distribution of symbols in the data block is close to uniform.

Since the speed of the system has a higher priority while working with large volumes of data, the modified data compression algorithm with the method of arithmetic coding, proposed in this article, will improve the performance of systems for working with large data in general.

## 6. Conclusion

In this paper, the application of the method of arithmetic compression based on the division into subsections depending on the frequency of occurrence of symbols was explored in the field of digital processing of speech data. The research results show that the proposed method gives good compression results with fairly low processing time.

However, it must be taken into account that the method of arithmetic compression based on the division into subsegments depending on the frequency of occurrence of symbols has its limitations. For example, if the number of subsections is increased, the processing time also increases, and if you decrease the number of subsections, the size of the compressed data output increases. Also, it is important to take into account the peculiarities of specific speech data, since different languages may have different frequency of use of certain symbols.

Further research can be aimed at improving the method of arithmetic compression based on partitioning into subsegments depending on the frequency of occurrence of symbols, in particular at the development of adaptive methods that would be able to dynamically adapt to the peculiarities of the input data. Also worthy of attention are other methods of speech data compression, in particular, based on prediction and summary of emissions. Applying more sophisticated compression methods has the potential for even higher compression efficiency and reduced processing time.

## 7. References


[1] S. Furui, Digital speech processing: synthesis, and recognition, CRC Press, 2018.

[2] R. Haeb-Umbach, S. Watanabe, T. Nakatani, M. Bacchiani, B. Hoffmeister, M. L. Seltzer, M. Souden, Speech processing for digital home assistants: Combining signal processing with deep-learning techniques, IEEE Signal processing magazine, 36(6) (2019) 111-124.

[3] S. K. Jagtap, M. S. Mulye, M. D. Uplane, Speech coding techniques, Procedia Computer Science, 49 (2015) 253-263.

[4] L. Tan, J. Jiang, Digital signal processing: fundamentals and applications, Academic Press, 2018.

[5] R. Giurda, E. Georganti, H. G. Hassager, T. Dau, Effects of wide dynamic range compression on speech signals with respect to reverberation, The Journal of the Acoustical Society of America, 141(5) (2017) 3821-3821.

[6] Y. Zhang, D. Xiao, Q. Ren, S. Guo, and F. Mo, An effective speech compression based on syllable division, in: Proceedings of Meetings on Acoustics 172ASA, Acoustical Society of America, 29(1) (2016) p. 055002)..

[7] M. Cernak, A. Asaei, A. Hyafil, Cognitive speech coding: examining the impact of cognitive speech processing on speech compression, IEEE Signal Processing Magazine, 35(3) 2018 97-109.

[8] M. Arif, and R. S. Anand, Application of polynomial method for speech signal transmission, in: 2015 IEEE 11th International Colloquium on Signal Processing & Its Applications, CSPA, IEEE, 2015, pp. 135-139.

[9] B. van Niekerk, L. Nortje, H. Kamper, Vector-quantized neural networks for acoustic unit discovery in the zerospeech 2020 challenge, arXiv preprint (2020) arXiv:2005.09409.

[10] A. Masmoudi, W. Puech, A. Masmoudi, An improved lossless image compression based arithmetic coding using mixture of non-parametric distributions, Multimedia Tools and Applications, 74 (2015) 10605-10619.

[11] A. Masmoudi, A. Masmoudi, A new arithmetic coding model for a block-based lossless image compression based on exploiting inter-block correlation, Signal, Image and Video Processing, 9 (2015) 1021-1027.

[12] M. Nelson, Data Compression With Arithmetic Coding, 2014. URL: https://marknelson.us/posts/2014/10/19/data-compression-with-arithmetic-coding.html.