DOI: 10.15514/ISPRAS-2025-37(5)-14

Modification of the Smith-Waterman Algorithm for Local Alignment of Genetic Sequences Based on the Window Method

E.S. Bezuglova, ORCID: 0000-0002-7608-0452 <bezuglova.ketrin@yandex.ru>
E.M. Shiriaev, ORCID: 0000-0002-2359-1291 <egor.shiriaev@reaneling.ru>
N.N. Kucherov, ORCID: 0000-0003-0337-0093 <nik.bekesh@yandex.ru>
M.G. Babenko, ORCID: 0000-0001-7066-0061 <whbear@yandex.ru>

North-Caucasus Federal University, Faculty of Mathematics and Computer Science named after Professor N.I. Chervyakov, 1, Pushkin st., Stavropol, 355017, Russia.

Abstract. The paper presents a modified algorithm for local alignment of genetic sequences based on the Smith-Waterman algorithm, using window method and run-length encoding. an experimental comparison of the performance of the proposed approach with the classical algorithm by such metrics as execution time, average and peak memory usage is carried out. The results demonstrate the effectiveness of the modification while preserving the quality of alignment, especially in resource-constrained environments. The work has practical implications for bioinformatics tasks involving genome analysis, gene annotation and homologous site search.

Keywords: Smith-Waterman algorithm; run-length encoding; window method; genetic sequences; local alignment; bioinformatics.

For citation: Bezuglova E.S., Shiriaev E.M., Kucherov N.N., Babenko M.G. Modification of the Smith-Waterman algorithm for local alignment of genetic sequences based on the window method. Trudy ISP RAN/Proc. ISP RAS, vol. 37, issue 5, 2025, pp. 183-194. DOI: 10.15514/ISPRAS-2025-37(5)-14.

Acknowledgements. The research was supported by the Russian Science Foundation Grant No 25-71-30007, https://rscf.ru/en/project/25-71-30007/.

Модификация алгоритма Смита-Ватермана для локального выравнивания генетических последовательностей на основе метода окна

Е.С. Безуглова, ORCID: 0000-0002-7608-0452 <bezuglova.ketrin@yandex.ru>
Е.М. Ширяев, ORCID: 0000-0002-2359-1291 <egor.shiriaev@reaneling.ru>
Н.Н. Кучеров, ORCID: 0000-0003-0337-0093 <nik.bekesh@yandex.ru>
М.Г. Бабенко, ORCID: 0000-0001-7066-0061 <whbear@yandex.ru>

Северо-Кавказский федеральный университет,
факультет математики и компьютерных наук имени профессора Н.И. Червякова,
Россия, 355017, Ставрополь, Пушкина, д. 1.

Аннотация. В статье представлен модифицированный алгоритм локального выравнивания генетических последовательностей, основанный на алгоритме Смита-Ватермана, с использованием метода окон и кодирования длин серий. Проведено экспериментальное сравнение производительности предлагаемого подхода с классическим алгоритмом по таким метрикам, как время выполнения, среднее и пиковое потребление памяти. Результаты демонстрируют эффективность модификации при сохранении качества выравнивания, особенно в условиях ограниченных вычислительных ресурсов. Работа имеет практическое значение для задач биоинформатики, связанных с анализом геномов, аннотированием генов и поиском гомологичных участков.

Ключевые слова: алгоритм Смита-Ватермана; кодирование длин серий; метод окон; генетические последовательности; локальное выравнивание; биоинформатика.

Для цитирования: Безуглова Е.С., Ширяев Е.М., Кучеров Н.Н., Бабенко М.Г. Модификация алгоритма Смита-Ватермана для локального выравнивания генетических последовательностей на основе метода окна. Труды ИСП РАН, том 37, вып. 5, 2025 г., стр. 183–194 (на английском языке). DOI: 10.15514/ISPRAS–2025–37(5)–14.

Благодарности. Исследование выполнено за счет гранта Российского научного фонда № 25-71-30007, https://rscf.ru/project/25-71-30007/.

1. Introduction

Genetic sequence alignment is a bioinformatics technique that is based on placing two or more sequences of deoxyribonucleic acid (DNA), ribonucleic acid (RNA) or protein monomers under each other in such a way that similar sequence regions can be observed to understand functional, structural and evolutionary relationships [1].

The application of biological sequence alignment methods played a key role in one of the most significant projects of modern science - the Human Genome Project [2], in which the results of alignment formed the basis for subsequent functional annotation of genes, identification of genetic variants associated with diseases, and development of personalized medicine.

Among the local alignment algorithms, the most popular is the Smith-Waterman algorithm [3], which provides accurate matching of sequence fragments through dynamic programming. Despite the high accuracy of this algorithm, its classical representation has significant computational costs, namely, its time and space complexity is $O(n \times m)$, where n and m are the lengths of the compared sequences. This limits its applicability when analysing long sequences or in conditions of limited computational resources (e.g., on lined systems or in fog computing environments).

Current approaches in algorithm optimization focus on making use of high-performance computing. These include vectorization using single instruction, multiple data (SIMD) principles [4-5], parallel implementations on GPUs [6-7], and the use of specialized architectures [8]. Despite the efficiency of the above approaches, the solutions require additional hardware and are not always applicable in distributed computing environments.

The actual task is to develop compact and efficient algorithms that do not depend on the architectural features of the computing platform.

In the framework of the proposed study, a modification of the classical Smith-Waterman local alignment algorithm is implemented using several known, but previously unused, joint techniques: binarization of global sequences, data compression using the run-length encoding (RLE) method [9], and partitioning data into fixed windows. Although the use of this method separately is widely used in computational biology and information technology, the presented combination and its practical application to local competition problems are constantly new.

The main novelty and originality of the proposed solution are as follows:

- initials and names of the authors;
- an original scheme for binarization of nucleotides sequentially is proposed, which allows for a significant reduction in the volume of original data. In combination with RLE encoding of binary data, high compression efficiency is achieved, especially in areas with repeating symbols typical of genomic sequences;
- integration of binary representation and RLE encoding allows for speeding up the process of sequence comparison, since the competition is reduced to bit operations instead of symbol-by-symbol comparisons. This, in turn, provides additional performance gains, especially in conditions of limited computing resources;
- the use of fixed windows for data processing provides not only a further reduction in computational costs, but also the possibility of efficient implementation of the algorithm in parallel environments and the distribution of distributed environments, such as systems with limited computing power or fog computing.

The paper proposes a modification of the Smith-Waterman algorithm based on the preliminary binary representation of sequences followed by bitwise compression based on RLE. This approach allows to significantly reduce the amount of processed data and speed up computations by simplifying the matching operations. In addition, an adapted penalty system for the binary format is introduced to preserve the algorithm's sensitivity to biologically significant changes.

2. Method

2.1 Data presentation

To reduce the data volume of the processed information and increase the efficiency of local alignment operations, the proposed algorithm first performs the conversion of nucleotide sequences into a binary data representation [10]. This conversion is based on the binary encoding of each nucleotide using a two-bit scheme, which provides a compact and convenient representation of the input data.

In the proposed approach, a unique two-bit combination is assigned to each symbol of a nucleotide sequence from the set $\{A, C, T, G\}$:

$$A \to 00, C \to 01, T \to 10, G \to 11.$$

This scheme provides efficient memory usage because it allows any sequence of length n to be represented as a bit string of length $2n$ bits, which halves the character representation compared to a character representation using 8 bits per character.

The conversion of the character string to a binary string is implemented by the function «dna_to_binary(dna_str)», which replaces each nucleotide with the advised 2-bit string. The reverse conversion is implemented by the function «binary_to_dna(binary_str)» and is necessary to restore the interpreted result after calculations in binary representation.

This conversion allows bitwise comparisons instead of character comparisons, which significantly speeds up the work of the modified algorithm. Instead of string operations comparing two nucleotides (for example, $A == G$), an exclusive-or operation (XOR) [11] is performed between 2-bit codes, and the result is 0 only if there is a complete match. This approach is easily scalable for vector computations and hardware optimisations including the SIMD principle. The described contributions are presented in Algorithm 1.

Algorithm №1 Function for representing a sequence in binary form

Input: dna_str – string of characters $\{A, C, T, G\}$
Output: $binary\_str$ – string of characters $\{0, 1\}$

1. $bin\_map \leftarrow \{'A': '00', 'C': '01', 'T': '10', 'G': '11'\}$
2. $binary\_str \leftarrow empty\ string$
3. For i from 0 to length $(dna\_str) - 1$ is executed
3.1. $nucleotide \leftarrow dna\_str[i]$
3.2. $bin\_str \leftarrow binary\_str + bin\_map[nucleotide]$
3.3. End of cycle
4. Return bin_str
5. Define $dna\_map \leftarrow \{'00': 'A', '01': 'C', '10': 'T', '11': 'G'\}$
6. If $length(bin_{str}) mod\ 2 \neq 0$ then
7. $bin\_str \leftarrow bin\_str + '0'$
8. For i from 0 to $length(bin\_str) - 1\ step\ 2$ do
8.1. $bits \leftarrow bin\_str[i] + bin\_str[i+1]$
8.2. $nucleotide \leftarrow dna\_map[bits]$
8.3. $dna\_str \leftarrow dna\_str + nucleotide$
8.4. End of cycle
9. Return dna_string

2.2 Application of compression

The After converting the nucleotide sequence into a binary representation, data compression is performed. It is needed in order to reduce redundancy and reduce the amount of information that needs to be equalized. The repetitive sequence encoding algorithm, RLE, has been used for this purpose.

The RLE method is one of the simplest and least resource-intensive methods of data compression, in which sequences consisting of identical characters (in this case – bits) are replaced by a pair of values: character and number of repetitions. For example, the substring 000001111 will be encoded as 05 14, which will reduce the total amount of representation in the presence of long single-type data blocks.

When representing genetic data in binary form, the proposed method shows high efficiency, because binary strings, binary strings derived from nucleotide sequences, often contain repetitive fragments, for example, homopolymer regions – sequences like AAAA, CCCC, etc. The binary format of genetic data shows high efficiency. In addition, the binary format itself has low entropy compared to the character format, which further enhances the efficiency of RLE coding [12].

The following factors explain the choice of compression technique:

- the RLE algorithm has a low computational complexity of $O(n)$, so it is suitable for processing large data sets without significant computational cost [13];
- window method compression, where a string in binary representation is split into fixed-length fragments, each of which is compressed separately. These actions make the algorithm robust to local changes and decoding with minimal resource consumption;

- unlike more complex methods (e.g., Huffman's algorithm [14] or arithmetic coding [15]), recovering data from an RLE representation does not require the construction of external tables and can be performed in a single linear pass, which is especially important in resource-constrained environments such as embedded systems or fog computing nodes [16];
- after applying RLE, additional data compression is possible, providing multi-layer compression without data loss.

The Huffman algorithm [14, 17] and arithmetic coding [15, 18] are used to efficiently encode character sequences, especially in the presence of statistical differences in character frequencies, but they require the construction and storage of auxiliary structures, which increases the amount of metadata and complicates decoding.

The compression methods LZ7 and LZ78 [19-20] and their derived algorithms, including the Lempel-Ziv-Welch algorithm (LZW) and the Lempel-Ziv-Markov chain algorithm (LZMA) [21], show a high compression rate when processing large amounts of text data [19], but they also require large buffers and time for preliminary analysis, which makes them less suitable for low-level optimization in a limited computational environment.

RLE, in turn, has high speed and minimal memory requirements, which makes it particularly efficient in the presence of patterned, repetitive patterns - exactly the kind of structures characteristic of genetic sequences. In addition, compared to Huffman coding and LZW, RLE shows better performance on short fragments and in applications with «real-time» data processing [22].

In the proposed approach, local alignment is not performed at the nucleotide level, but in the space of binary strings that were obtained by performing binary coding.

The splitting into windows is performed using the function «split_into_windows», after which separate processing is performed. Each window is compressed using the RLE method. The data recovery process allows to exactly restore the original window content and, if necessary, to perform an accurate restoration of the binary sequence (Algorithm 2).

An important aspect of the proposed approach is that binarization and subsequent RLE coding of genetic sequences do not impose restrictions on the ability to identify alignments of arbitrary length or location. Despite the fact that the local alignment algorithm is performed in binary representation, this is an exclusively intermediate form of data that ensures efficient processing. After the alignment procedure is completed, an exact reverse transformation from the binary representation to the original nucleotide sequences is implemented. This ensures complete compliance of the alignment results with the original sequences without loss of information accuracy. Thus, any alignments found at the binary level are unambiguously and accurately translated back into the original nucleotide sequences, preserving the biological significance and accuracy of the final result.

2.3 Window method

Window method is a commonly used technique in sequence, signal, and text processing algorithms, where the input data is partitioned into fixed-size non-overlapping segments (windows) [23]. Unlike sliding window techniques, each window in this method is processed independently without overlap. In the context of biological data processing, particularly sequence alignment, the window method improves computational efficiency by reducing the working data set size. This approach also facilitates parallel execution, as each window can be processed independently, which enhances performance and optimizes memory usage.

In the proposed window method, S is a binary string of length n and the window size is W. The string S is partitioned into windows, each consisting of a sequence of length W, except for the outermost window which can be shorter if the length of string n is not divisible by W.

Algorithm №2 Data compression and recovery

Input: $binary\_str, window\_size$

Output: $compressed\_windows, decompressed\_windows$

1. $windows \leftarrow empty\ list$
2. For i from 0 to length ($binary\_str$) step $window\_size$ do
2.1. $window \leftarrow substring$ of $binary\_str$ for index i to $i + window\_size$
2.2. Append $window$ to $windows$
3. End of cycle
4. $compressed\_windows \leftarrow empty\ list$
5. For each $window$ in $windows$ do
5.1. If $window$ is empty, then continue
5.2. $compressed \leftarrow empty\ string$
5.3. $current\_bit \leftarrow first\ bit\ of\ window$
5.4 $count \leftarrow 1$
5.6 For bit in $window$ starting from second position do
5.6.1. If $bit = current\_bit$, then
5.6.1.1. $count \leftarrow count + 1$
5.6.2. Else
5.6.2.1. Append $current\_bit + count$ to $compressed$
5.6.2.2. $current\_bit \leftarrow bit$
5.6.2.3. $count \leftarrow 1$
5.7 Append final $current\_bit + count$ to $compressed$
5.8. Append $compressed$ to $compressed\_windows$
6. End of cycle
7. $decompressed\_windows \leftarrow empty\ list$
8. For each $compressed\_win$ in $compressed\_windows$ do
8.1. $window \leftarrow empty\ string$
8.2. i $\leftarrow 0$
8.3. While $i < length(compressed\_win)$ do
8.3.1. $bit \leftarrow compressed\_win[i]$
8.3.2. $count \leftarrow integer\ value\ of\ compressed\_win[i+1]$
8.3.3. Append $bit * count$ to $window$
8.3.4. $i \leftarrow i + 2$
8.4. Append $window$ to $decompressed\_windows$
9. End of cycle
10. Return $compressed\_windows, decompressed\_windows$

The process of partitioning into windows:

$$windows(S, W) = [S[i: i + W] | i = \{0, W, 2W, \dots, \left\lfloor (\frac{n-W}{W} \right\rfloor \times W\}],$$

where i is the start index of each window, W is the window size, and $S[i: i + W]$ is a slice of the string S starting at index i and of length W. The expression $\left\lfloor (\frac{n-W}{W} \right\rfloor \times W$ ensures that no window exceeds the bounds of the sequence.

If the string length n is not a multiple of W, the last window will have a shorter length equal to $n\ mod\ W$, where $n\ mod\ W$ is the remainder of dividing n by W. The partitioning algorithm ensures that the entire string is covered by non-overlapping windows, with the last one possibly shorter than the others.

Partitioning a string into smaller, fixed-size windows allows for the distribution of computational tasks, as each window can be processed independently. This feature is particularly beneficial in

multitasking or multiprocessor systems, where each window can be assigned to a separate thread or process. The technique also contributes to optimized memory usage, enabling processing of smaller data portions without requiring the entire sequence to be loaded into memory.

Thus, the described algorithms can be implemented as a unified program or used independently in unrelated tasks. Based on these modules, a working program has been developed, and its results are presented in the next section.

3. Conducting the experiment

To evaluate the efficiency of the proposed algorithm for local alignment of genetic sequences, a series of computational experiments were conducted.

The study was carried out on the basis of programmes described in the Python programming language [24], on a dataset of genetic sequences ranging from 100 to 3000 nucleotides, with a step of 100, on equipment with the following characteristics:

- CPU: Apple M2 16 cores, 3.4GHz;
- RAM: 8GB LPDDR4X;
- Storage: 512GB NVMe SSD PCIe 3.0;
- Operating System: macOS Sonoma 15.4.

The experiment was conducted in three phases:

1) measurement of speed, average memory and peak memory when performing local alignment using the classical Smith-Waterman algorithm;
2) measurement of speed, average memory and peak memory when performing local alignment using the modified algorithm based on the Smith-Waterman algorithm;
3) comparisons of the obtained results.

Average memory usage was computed as the mean value of allocated memory during the entire execution, while peak memory corresponds to the maximum memory usage observed at any point. These metrics were chosen to reflect both sustained and worst-case memory behavior.

The results showed that as sequence length increased from 100 to 3000 nucleotides, the classical Smith-Waterman algorithm exhibited a quadratic increase in execution time (up to 520 seconds). In contrast, the modified version maintained a nearly constant execution time between 10 and 20 seconds. This demonstrates a significant reduction in computational load – approximately 11.7-fold (Fig. 1).

Fig. 2 shows a comparison of the average memory consumption between the classical and modified algorithms as a function of sequence length. The results demonstrate that the classical algorithm exhibits linear growth in average memory usage as the input sequence length increases, exceeding 20,000 KB for sequences of 3,000 nucleotides.

In contrast, the modified algorithm displays a more stable behavior: after a brief initial rise, average memory usage stabilizes at approximately 3,800 KB and remains nearly constant regardless of further increases in input size.

These findings confirm that the proposed method is particularly effective for use in systems with limited computational resources, where predictable and low memory usage is critical.

Fig. 3 presents a comparison of peak memory consumption between the classical and modified algorithms. The classical Smith-Waterman algorithm is characterized by an exponential increase in peak memory usage as sequence length grows, reaching approximately 72,088 KB for input sequences of 3,000 nucleotides.

In contrast, the modified algorithm shows significantly lower peak memory usage: after a moderate initial increase, the values stabilize around 7.8 MB, regardless of further input growth.

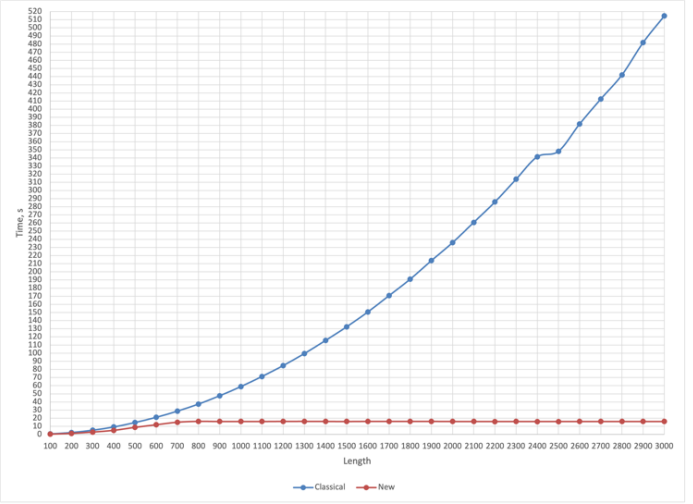

Fig. 1. Comparison of average time used

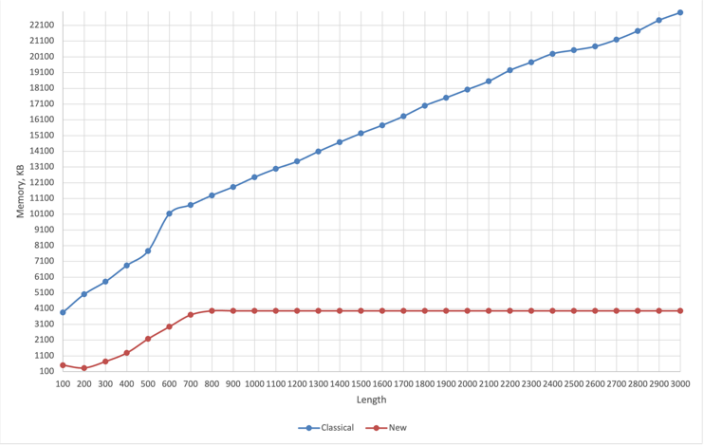

Fig. 2. Comparison of average memory consumption

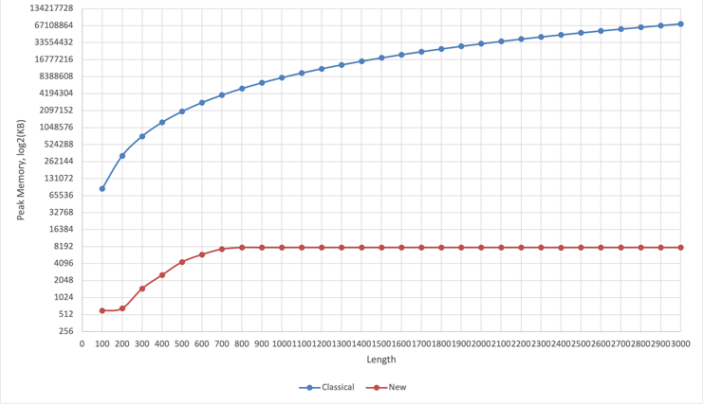

Fig. 3. Comparison of peak memory consumption

These results confirm the effectiveness of the proposed window-based compression and processing technique, which substantially reduces memory load while preserving the accuracy of the alignment.

4. Conclusion

The paper proposes an algorithm for local alignment of genetic sequences based on window method and RLE compression. The results of computational experiments show a significant reduction of time and resource costs. In particular, the modified algorithm demonstrates a 11.7-fold reduction in execution time compared to the original algorithm. Also, a 4.8-fold reduction in average and peak memory consumption is obtained, which makes the proposed method promising for big data analysis and implementation in environments with limited computational capabilities.

In the future, it is planned to extend the proposed approach by including adaptive compression strategies that will automatically select the most efficient coding scheme depending on the structure of the input sequence. In addition, a promising direction is the implementation of the modified algorithm in distributed computing systems, including fog computing environments and embedded devices, in order to assess the scalability and stability of the algorithm under resource constraints. The integration of the developed method into existing bioinformatics pipelines remains an urgent task, which will allow us to assess its practical significance in solving applied problems, such as homology search, gene annotation and analysis of changes in genomes.

References

[1]. Baxevanis A.D., Bader G.D., Wishart D.S. Bioinformatics. Hoboken, NJ: John Wiley & Sons, 2020, 656 p.
[2]. Olson M. V. The human genome project. Proceedings of the National Academy of Sciences, 1993. Vol. 90, no. 10, pp. 4338-4344.
[3]. Smith T. F., Waterman M. S. Identification of common molecular subsequences. Journal of Molecular Biology, 1981, vol. 147, no. 1, pp. 195-197.
[4]. Flynn M. J. Some Computer Organizations and Their Effectiveness. IEEE Transactions on Computers, 1972, vol. C-21, no. 9, pp. 948–960, DOI: 10.1109/TC.1972.5009071.
[5]. Farrar M. Striped Smith-Waterman speeds database searches six times over other SIMD implementations. Bioinformatics, 2007, vol. 23, no. 2, pp. 156-161, DOI: 10.1093/bioinformatics/btl582.
[6]. Barron E. T., Glorioso R. M. A micro controlled peripheral processor. Conference record of the 6th annual workshop on Microprogramming, in MICRO 6. New York, NY, USA: Association for Computing Machinery, 1973, pp. 122-128, DOI: 10.1145/800203.806247.
[7]. Liu Y., Wirawan A., Schmidt B. CUDASW++ 3.0: accelerating Smith-Waterman protein database search by coupling CPU and GPU SIMD instructions. BMC Bioinformatics, 2013, vol. 14, no. 1, p. 117, DOI: 10.1186/1471-2105-14-117.
[8]. Rognes T. Faster Smith-Waterman database searches with inter-sequence SIMD parallelization. BMC Bioinformatics, 2011, vol. 12, no. 1, p. 221, DOI: 10.1186/1471-2105-12-221.
[9]. Robinson A. H., Cherry C. Results of a prototype television bandwidth compression scheme. Proceedings of the IEEE, 1967, vol. 55, no. 3, pp. 356–364, DOI: 10.1109/PROC.1967.5493.
[10]. Collett D. Modelling Binary Data. 2nd ed. New York: Chapman and Hall/CRC, 2002, p. 367, DOI: 10.1201/b16654
[11]. Lavrov I., Maksimova L. Problems in Set Theory, Mathematical Logic and the Theory of Algorithms. Springer Science & Business Media, 2003, p. 275.
[12]. Sayood K. Introduction to data compression. Morgan Kaufmann, 2017, p. 735.
[13]. Salomon D. A concise introduction to data compression. Springer Science & Business Media, 2007, p. 305.
[14]. Cormen T. H., Leiserson C. E., Rivest R. L. Introduction to algorithms. MIT press, 2022, p. 1251.
[15]. Rissanen J., Langdon G. G. Arithmetic Coding. IBM Journal of Research and Development, 1979, vol. 23, no. 2, pp. 149-162, DOI: 10.1147/rd.232.0149.
[16]. Chervyakov N., Babenko M., Tchenykh A., Dvoryaninova I., Kucherov N. Towards reliable low cost distributed storage in multi-clouds. 2017 International Siberian Conference on Control and Communications (SIBCON), 2017, pp. 1-6. DOI: 10.1109/SIBCON.2017.7998476.
[17]. Huffman D. A. A Method for the Construction of Minimum-Redundancy Codes. Proceedings of the IRE, 1952, vol. 40, no. 9, pp. 1098-1101, DOI: 10.1109/JRPROC.1952.273898.
[18]. Witten I. H., Neal R. M., Cleary J. G. Arithmetic coding for data compression. Commun. ACM, 1987, vol. 30, no. 6, pp. 520-540, DOI: 10.1145/214762.214771.

[19]. Ziv J., Lempel A. A universal algorithm for sequential data compression. IEEE Transactions on Information Theory, 1977, vol. 23, no. 3, pp. 337-343, DOI: 10.1109/TIT.1977.1055714.
[20]. Winters K. D., Owsley P. A., French C. A., Bode R. M., Feeley P. S. Adaptive data compression system with systolic string-matching logic, US5532693A, 1996.
[21]. Welch T. A. A Technique for High-Performance Data Compression. Computer, 1984, vol. 17, no. 06, pp. 8-19, DOI: 10.1109/MC.1984.1659158.
[22]. Nelson M., Gailly J. L. The data compression book 2nd edition. M & T Books, New York, NY., 1995, p. 576.
[23]. Gusfield D. Algorithms on Stings, Trees, and Sequences: Computer Science and Computational Biology. SIGACT News, 1997, vol. 28, no. 4, pp. 41-60, DOI: 10.1145/270563.571472.
[24]. Rana Y. Python: simple though an important programming language. International Research Journal of Engineering and Technology (IRJET), 2019, vol. 6. no. 2, pp. 1856-1858.

Информация об авторах / Information about authors

Екатерина Сергеевна БЕЗУГЛОВА окончила магистратуру по специальности «Математика и компьютерные науки» в 2023 году в Северо-Кавказском федеральном университете. В настоящее время она является аспирантом, младшим научным сотрудником научно-исследовательской лаборатории биологической и медицинской информатики медико-биологического факультета Северо-Кавказского федерального университета. Ее научные интересы: биоинформатика, анализ данных, машинное обучение, нейронные сети, облачные вычисления.

Ekaterina Sergeevna BEZUGLOVA graduated with a master's degree in Mathematics and Computer Science in 2023 from the North Caucasus Federal University. She is currently a postgraduate student and a senior mid-level employee at the Research Laboratory of Biological and Medical Informatics at the Faculty of Medicine and Biology at the North Caucasus Federal University. Her research interests include bioinformatics, data analysis, machine learning, neural networks, and cloud computing.

Егор Михайлович ШИРЯЕВ окончил магистратуру по специальности «Прикладная математика и информатика» в 2022 году в Северо-Кавказском федеральном университете. В настоящее время он является аспирантом, инженером-исследователем Департамента науки и ассистентом кафедры вычислительной математики и кибернетики Северо-Кавказского федерального университета. Его научные интересы лежат в области гомоморфных методов шифрования, нейронных сетей, сохраняющих конфиденциальность, облачных вычислений и кибербезопасности.

Egor Mikhailovich SHIRIAEV graduated from the master's program "Applied Mathematics and Computer Science" in 2022, at the North Caucasus Federal University. Currently, he is a postgraduate student, a research engineer at the Department of Science, and an assistant at the Department of Computational Mathematics and Cybernetics at the North Caucasus Federal University. His research interests are in homomorphic encryption methods, privacy-preserving neural networks, cloud computing and cybersecurity.

Николай Николаевич КУЧЕРОВ получил степень бакалавра по специальности «Компьютерные науки» и кандидата технических наук в Северо-Кавказском федеральном университете, Ставрополь, Россия, в 2012 и 2018 годах соответственно. С 2020 года он работает доцентом кафедры математического анализа, алгебры и геометрии Северо-Кавказского федерального университета, Ставрополь, Россия. Его научные интересы включают облачные вычисления, высокопроизводительные вычисления, системы остаточных чисел, нейронные сети и кибербезопасность.

Nikolay Nikolaevich KUCHEROV – Cand. Sci. (Tech.) since 2018. He graduated from North-Caucasus Federal University, Stavropol, Russia, in 2012, and has been working as an Assistant

Professor with the Department of Mathematical Analysis, Algebra and Geometry, North-Caucasus Federal University, Stavropol, Russia, since 2020. His research interests include cloud computing, high-performance computing, residue number systems, neural networks, and cybersecurity.

Михаил Григорьевич БАБЕНКО – доктор физико-математических наук, заведующий кафедры вычислительной математики и кибернетики факультета математики и компьютерных наук имени профессора Н.И. Червякова ФГАОУ ВПО «Северо-Кавказский федеральный университет». Сфера научных интересов: облачные вычисления, высокопроизводительные вычисления, система остаточных классов, нейронные сети, криптография.

Mikhail Grigoryevich BABENKO – Dr. Sci. (Phys.-Math.), Head of the Department of Computational Mathematics and Cybernetics, Faculty of Mathematics and Computer Science named after Professor N.I. Chervyakov, North Caucasus Federal University. His research interests include cloud computing, high-performance computing, residue number systems, neural networks, cryptography.