A System for Compression of Sequencing Data

Boryana Pulova-Mihaylova, Iliyan Mihaylov, Irena Avdjieva, and Dimitar Vassilev

FMI, University of Sofia St. Kliment Ohridski, 5 James Bourchier Blvd., Sofia 1164, Bulgaria

dimitar.vassilev@fmi.uni-sofia.bg

Abstract. The development of genomic sequencing technologies is directly related to the storage, analysis, visualization of huge amount of sequencing data. These data, in terms of quantity and quality, are a huge challenge for modern computer science and bioinformatics related to compression and decompression of huge data sets. The present study is devoted to the development of models and their implementation for compression of sequencing data.

The main aim of the work is to develop a web-based system for sequencing data compression that provides opportunities for faster and more accurate compression and decompression, noise protection and error correction.

For the purposes of this study, we developed models for compression and decompression based on the methods of literal coding. The developed Optimized algorithm is related to the well-known methods of Huffman and Shanon-Fano. Also are developed a web-based module (with user interface), a library of noise protection algorithms for the compression and decompression of omics data, and a server component to make the library accessible on the Internet.

The implemented web module with user interface provides easy access to server methods for compression and decompression of sequences. Used data formats allows the module to be integrated with open access systems such as NCBI, UniProt, Ensembl and other well-known external resources.

Keywords: Sequencing data, Compression, Noise protection, Web-based system.

1 Introduction

The introduction of high-throughput sequencing technologies is accompanied by generation of large amounts of biological data. The trend of increasing sets of biological sequence data is due to the following main reasons:

- The declining cost of genome sequencing provides conditions for even larger projects related to increased requirements for analysis of such data.
- Increasing knowledge about the relationship between genotype and phenotype creates enormous opportunities for more specific in-depth research of genome, especially in the context of personalized medicine.

Sequencing platforms are advancing both in length of sequencing reads and in speed of data generation [1].

Typically, storing and managing DNA data, or sequential data, is by encoded ASCII characters, which results in one byte for each base. Although this storing approach is wasteful in terms of space, since no compression is applied, many experts in the analysis of such data are accustomed to this data format. Its advantages are that it allows software applications and programming languages easily to access data, and also it is easy readable by humans. Of course, the big drawback of storing, analyzing and transferring data in this format is the need for huge and expensive resources of computer memory.

The situation becomes much worse when not only ready-made sequenced genomes are stored, but also raw data of a certain quality. Presently, storing 10,000 genomes is a work in progress, but the rapid pace of sequencing, as well as new third-generation sequencing technologies, pose major challenges to the data being generated. [2]

Most approaches for sequencing data compression are based on two of the most popular algorithms in recent years - Shannon-Fano and Huffman. These algorithms are compared to an optimized new version, developed in our study, which we will hereafter refer to as the "Optimized Algorithm", based on the theory of literal codes, but with an improvement in code uniformity and compression / decompression speed. Huffman and Shannon-Fano's algorithms result in compression as a result of non-uniform code, the main disadvantage of which is that it is prone to loss of information (in data transfer) and hence incorrect data recovery. With the Optimized Algorithm, a uniform code is obtained, which would easily identify the data loss and prevent the wrong data recovery. In addition, for large volumes of data (long DNA / RNA sequences) compression by a uniform code produces better results than a non-uniform compression.

2 Related work

There are several basic types of compression approaches according to the data processing method. They have some features in common but they are quite different both methodologically and in scope [3].

The main methods for compression of nucleotide sequencing data are based on statistics and entropy - by exploiting compression algorithms using repetition detection and consequence statistics [4]. The increasing number of re-sequenced genomes has led to many new suggestions for compression algorithms. In general, compression algorithms can be divided into bitwise encoding, vocabulary-based compression, statistical and referential approaches.

Bitwise encoding - a compression algorithm that can process input strings of any format. For the first common DNA characters (A, C, G, T as well as N (unknown)), seven bit encoding is used for three consecutive characters. Any other non-standard character is encoded with seven bits per character. Up to 128 additional characters can be encoded in this way. The free eight bit distinguishes whether a byte encodes three of the first main characters or some other, specific character [3].

Dictionary-based compression - performs with multiple iterations over the input data. With each pass, it detects longer sequences and gradually enriches the dictionary. The algorithm runs until the dictionary is changed or the frequency threshold is reached. The created dictionary is used to encode DNA sequences and is partially stored together with compressed data. This encoding allows random access to the compressed DNA sequences [5].

Another approach is so-called tree-based compression [6]. Splay trees are self-tuning binary search trees in which items that are last accessed are accessed faster. This property can improve the performance when you encounter local frequency changes. The methodology refers to the process of rearranging the trees as a specific element is placed at the root of the tree. Like evolutionary trees, symbols close to the root have shorter codes than symbols in other nodes. Another problem is the unequal distribution of DNA across the genome. One possible solution is to split the input of blocks and encode each block individually using different Markov models [7].

In [8], a "gene-wise compressor" is proposed as a mean for encoding non-repetitive portions of DNA sequences. In the initial implementation, the probabilities of the characters are evaluated and the corresponding Huffman coding is selected. The result is divided into blocks. Finally, each block is restructured in a way that allows efficient coding of the execution length to be used in the final step. Another approach is the input sequence is fragmented into blocks that do not overlap. For each block, a set of agents competes for coding: a Markov model, a two-bit base representation, and an approximate repetition that identifies repetitions interrupted by only a few SNPs (single nucleotide polymorphisms). The model that broadcasts the shortest code length is selected and the compressed output is subjected to additional arithmetic compression. Unfortunately, this approach can handle only four main characters in the input sequences. A method is proposed where non-repetitive regions are encoded as well as discrepancies in repetitive regions or mutations like SNPs with arithmetical coder, based on a Markov model. The resulting bit stream is split into blocks to allow random access.

Mixed approach for compression: there are many variations and different views and discussions in the literature. It is suggested that to store the differences between the input compression sequence and the reference sequence [9]. They look at three types of single based features: insert, delete, and replace. The main contribution of their work is to analyze the process of encoding integers for absolute and relative reference positions in the sequence. However, the authors emphasize that the choice of reference sequence has a greater impact on the compression ratio than the actual overall coding scheme. Similarly, other sources present a compression reference algorithm that only accepts SNPs and many basic INDELs between inputs and reference sequences. Each compression record consists of a reference position and additional data such as coincidence length or raw sequence.

Table 1. Compression algorithms comparison

Compression method	Compression
Bitwise encoding	2:1 – 6:1
Dictionary-based compression	4:1 – 6:1
Statistical model	4:1 – 8:1
Mixed model	1:1 – 400:1

Once the sequence is compressed, it must be sent over the network. The algorithms used for compression, namely Huffman, Shannon-Fano, and the optimized algorithm, which belongs to the class of literal coding, are very susceptible to errors. If even one bit is confused when transmitting network information or reading to and from the recorder, restoring the correct sequence is not possible. For this purpose, it is necessary to use error correction algorithms. Literature sources cite noise protection algorithms as one of the most widely used for this purpose [10]. They are applicable in all places where it is necessary to correct errors at the lowest possible level (bitwise). In damping waves, artificial intelligence models are also widely used in pattern recognition and error correction in generated images. One of the most common algorithms is that of Reed Mahler, suitable for data transmission in places where the connection is poor or very weak [10].

The main goals of the work are: to present a new method for compression of nucleotide sequencing data with improved performance, accuracy and noise protection based on the known methods of Huffman, Shannon-Fano and Reed-Miller, as well as to develop an interoperable and reusable software implementation of the elaborated method.

3 Suggested methodology

The suggested compression approach in this study is based on coding and more specifically on the Shannon-Fano and Huffman algorithms. The main idea behind the Shannon-Fano and Huffman algorithms is to set a binary code to match each of the characters in the input message (in our case bases in the input sequence). The algorithm has the following characteristic properties:

1. The length of the codes is variable (the code is uneven). Here is our first modification of the algorithm, where the length of the codes is always the same i.e. the compressed message has a uniform code. The sequences being

considered for compression are of the same length, and this comes from the most used formats such as FASTA and FASTQ, where the length of the sequence is always specified, which could be used in advance to build a classification tree. Another feature of the code length is that when the individual characters in the message are replaced with their corresponding binary values, a compressed message of different sizes may be received. This is because for each symbol, a different binary value is given, which in some cases can be 1 and in others, it can be a combination of 0 and 1. For example, for "A" a value of 11 can be calculated, for "T" - a value equal to 101, which causes also change in the string size. This can be improved when using DNA sequence data because they have a small number of characters that can be encoded with equal length.

2. The letters with higher probability are more likely to be encoded in fewer bits than those with less likely ones. In addition, this is the next modification of the algorithm we suggest: an approach how to calculate the probability of occurrence of individual characters in the message using frequency analysis, which is the most time consuming process in this algorithm. In the case of sequence compression, we make an improvement where there is no need to perform frequency analysis because the characters included in the sequence are countable small number. For DNA and RNA sequences, the symbols are 4 in number, for which mapping (aligning) has been prepared preliminary for their respective binary sequences.

3. The message shall be decoded unambiguously.

Advantages: When we encode the more common characters with shorter bit sequences and the less common ones with longer bit sequences, we get better results with respect to the length of the message being transmitted.

Disadvantages: A major disadvantage of uneven codes is their sensitivity to wrong bits. For them, a wrong bit can cause the entire message to be decoded incorrectly or impossible until the end. Whereas, with uniform codes, one wrong bit causes only one character to be corrupted.

3.1 Building the optimal code

An optimized algorithm developed in the study, according to which sequences based on the theory of optimal letter coding will be compressed has following operational features:

1. Sequence type recognition (RNA or DNA).
2. Depending on the sequence, predefined binary values for each of the symbols used in the sequence are selected - using the predefined values saves the

following procedures, which are necessary to construct the optimal code for each arbitrary message in the Shannon-Fano algorithms and Huffman:

a. The set of symbols of source A is arranged in order to reduce the probability of message occurrence – p_j

b. The set of probabilities P_i is divided into two groups ($p_1, p_2, \ldots, p_j$) and ($p_{j+1}, p_{j+2}, \ldots, p_m$), so that the difference shown below is minimal:

$$\sum_{i=1}^{j} p_i - \sum_{i=j+1}^{j} p_i \qquad (1)$$

c. The characters whose probabilities are in the first group are assigned with the r-th code letter 0 and those in the second group are assigned with the r-th code letter 1.

δ. For a single-element probability group, the procedure is completed, and for each of the other groups in the multi-step procedure, the elements are numbered from 1 to m and go recursively (r = r + 1) to step 2.

3. Each of the characters in the sequence is replaced by the corresponding binary entry.

Following the changes made to the Shannon-Fano algorithm and the Huffman algorithm, all properties of the letter algorithms coming from the definitions and theorems related to those algorithms retain, ignoring the steps for frequency analysis and computation of the binary comparison tree.

Once the sequence is compressed, it can be stored or sent in a much smaller. Reading the compressed sequence is another major problem with the Shannon-Fano and Huffman algorithms. In the Optimized Algorithm suggested in this study, this problem is solved by using a predefined coding tree. In Shannon-Fano and Huffman's algorithms, in order to decode a properly transmitted message, it is necessary to have the same encoding tree generated when compressing the message. The transfer of message for both algorithms consists of two parts - the encoded message, and the coding tree so that the message can be read. These two parts do not always need to be transferred together. Because algorithms can also be used in the context of information security, where not everyone can decode the message, i.e. not everyone should have a coding tree. In bioinformatics, this is an important feature for working with sequences.

In the proposed method, only one has a coding tree. Each compressed / encoded sequence can only be read by the same programming apparatus with which it was compressed / encoded. This is because a unique hashing algorithm is used to generate the coding tree in order to ensure the uniqueness of each coding tree. Each sequence has its own encoding tree and can only be decoded / decompressed by this encoding tree. This approach eliminates the shortcomings of both Shannon-Fano and Huffman's algorithms for decoding and protecting information. In the proposed Optimized Approach, each of the characters will be encoded with exactly 2 bits. As before the compression, each character occupied 8 bits, here is the first place where we have a 1:4 compression. After applying the algorithms, a compression of up to 1:400 can be achieved.

3.2 Compressing algorithm comparative analysis

The three algorithms' coding/decoding abilities were tested with a 20-letter, 160-bit sample DNA sequence: AACTTTGACGGTATACGCAA.

The Shannon-Fano and Huffman algorithms code the sample in three steps: 1) symbol frequency analysis, 2) grouping, 3) assigning each symbol a binary code. Once the binary codes for each of the four symbols are assigned, a coding binary tree is generated, resulting in coding the sequence as 41-bit string:

00110101010111011011111110010011011111000

Decoding requires the generated binary code and the coding tree, which is traced from root to leaves depending on the input symbol – left path for 0 and right – for 1.

The Optimized Algorithm omits the frequency calculation step an assigns a two-digit bit code to each of the four symbols: A=00, C=01, G=11, T=10. This coding table is static and does not need to be generated each time. Coding is done by simple substitution, resulting in the following 40-bit string:

0000011010101100011111100010000111010000

Decoding the sequence requires the coded sequence and the static table, thus omitting the need of a coding tree.

As shown in Table 2, the Optimised Method produces 5% less bit length than Shanon-Fano and Huffman methods. The result code is also uniform, which may prevent possible bit loss, while in uneven code such errors cannot be detected due to different length of bits in coding sequence.

Table 2. Comparative analysis of compression algorithms

Coding method	Input length	Compressed length
Shannon-Fano	20 bases, 160 bits	41 bits
Huffman	20 bases, 160 bits	41 bits
Optimized method	**20 bases, 160 bits**	**40 bits**

3.3 Software realization and user interface

The application utilizes several platforms combined under Ubuntu Linux OS. There are three main software components:

- User interface
- C++ library
- Programming adaptor to connect the library to an intermediate language

The user interface is based on Vue.js, Vuetifie, node.js, and npm version manager. Vue.js uses JavaScript templates to generate HTML in real time, which allows the interface to work on different devises and different screen resolutions, because each HTML is generated for the specific devise. Vuetifiee is a programming module that upgrades Vue.js by adding predefined templates. Node.js allows JavaScript to be used as server provisioning method by dynamic building in executable code. It provides the possibility to call each individual HTML element in the user interface separately, and visualize the result when ready – an approach called „reactive programming". JavaScript was chosen because of its compatibility with RESTful services and JSON data transfer format.

Version management of the used Vue.js, Vuetifile and Node.js modules is done by npm, which allows multiple storage for the modules to be registered, and automatically detects and downloads them.

3.4 Choosing programming medium for the compression algorithms

C++ was chosen for the algorithms for literal and noise protection coding, for its execution speed and lower memory usage, compared to other languages. It also allows direct memory access, easy and intuitive realisation of bitwise operations and lower-level resource management. These advantages make algorithms written in C++ more optimal than C# and Java. This is one of the main reasons to include connections between platforms in this application, because it utilises both C++ and C#.

4 Results and Discussion

Our web-based application for sequence data compression is adapted for various device screen resolutions – from mobile phone (375x812) and tablet (768z1024) to 8k (7680x4320). The user interface is shown on Figure 1 (mobile devise). Filed (B) allows the selection of file(s) to be compressed. The application accepts FASTA and MULTIFASTA file formats, and one or multiple files can be selected, thus allowing parallel compression. The selected files are listed as labels (A) in field (B), and can be deleted by pressing the "clear" symbol (C).

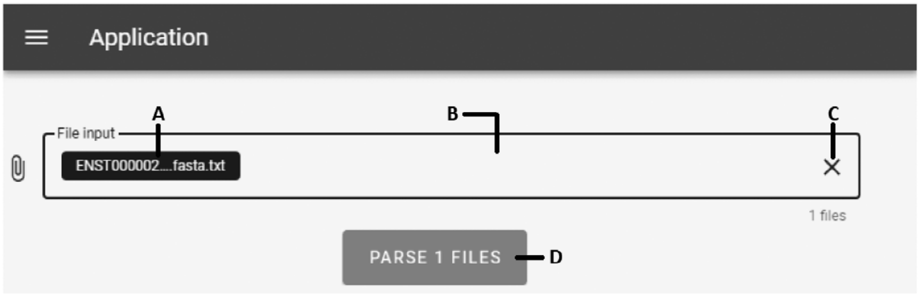


Fig. 1. User interface – File input field (B) with one file uploaded (A) and confirmation button (D)

Button (D) confirms the query and shows how many files are selected for compression. Then, the files are parsed to extract metadata such as sequence IDs, file size (in bytes), and sequence length (in number of nucleotides).

For test compression, four sequences with different length from the Ensembl database were saved as a single, multifasta file:

- ENST00000288602.11 - BRAF-201, B-Raf proto-oncogene serin/threonine kinase
- ENSE00001025715.1 chromosome:GRCh38:X:38317316:38317465:-1
- ENSE00001025717.1 chromosome:GRCh38:X:38321027:38321089:-1
- ENSG00000157764.13 chromosome: GRCh38:7:140719327:140924928:-1

Each sequence is compressed with the three algorithms, and the time needed for compression is calculated. Figure 2 shows the result for each sequence in the file as a separate, auto-expandable field (Б). Each field shows information about the sequence:

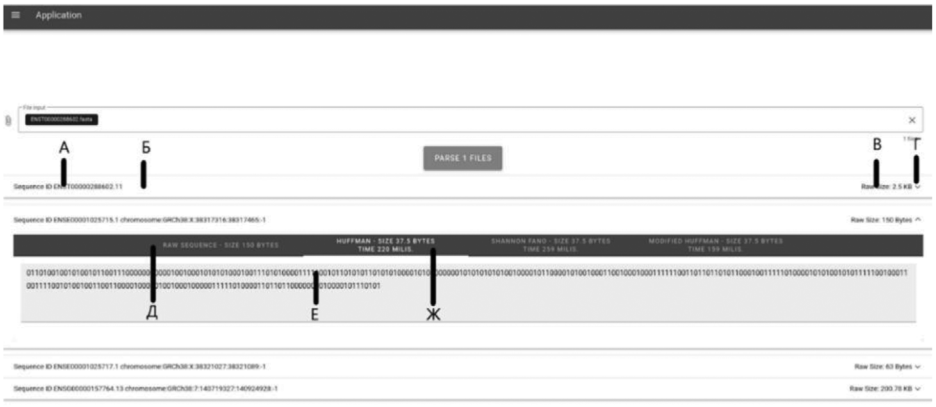

Application
PARSE 1 FILES
А
Б
В
Г
Д
Е
Ж
RAW SEQUENCE - SIZE 150 BYTES
HUFFMAN - SIZE 37.5 BYTES TIME 220 MILIS.
SHANNON FANO - SIZE 37.5 BYTES TIME 259 MILIS.
MODIFIED HUFFMAN - SIZE 37.5 BYTES TIME 159 MILIS.
Sequence ID ENST00000288602.11
Raw Size: 2.5 KB
Sequence ID ENSE00001025715.1 chromosome:GRCh38:X:38317316:38317465:-1
Raw Size: 150 Bytes
Sequence ID ENSE00001025717.1 chromosome:GRCh38:X:38321027:38321089:-1
Raw Size: 63 Bytes
Sequence ID ENSG00000157764.13 chromosome:GRCh38:7:140719327:140924928:-1
Raw Size: 200.78 KB

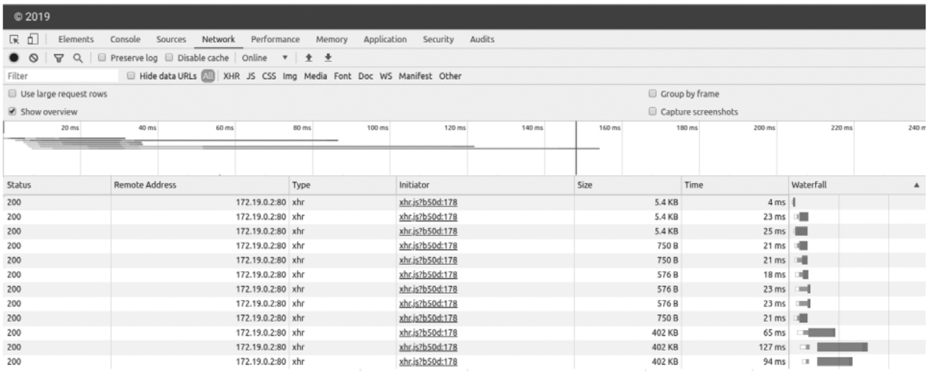
Fig. 2. Visualization of compressed sequences

- Sequence name (A) – this is obtained from the FASTA format, and if it matches a naming pattern from an entry in one of the public sequence databases, a hyperlink to that entry is created

- Current sequence size [in bytes] (В) – this is the sum of the byte size of the ASCII characters corresponding to the letters coding each nucleotide. Then, depending on the sequence length, it is converted to the closest B, KB, MB, GB of TB value by following the International Electrotechnical Commission (IEC) standard where 1 KB = 1024 B.
- Drop-down arrow (Г) shows/hides detailed information about the compressed sequence
- In the detailed field, (Д) is the sequence name and size label, and by clicking on it the original, uncompressed sequence is shown. (Ж) shows the sequence after compression (Е) by each of the three algorithms. Additionally, for each algorithm, a summary of the size [bytes] and time [seconds] of compression is shown.

In the current example, there is practically no difference in the compressed sequence size between the three algorithms; considerable difference is seen when compressing sequences of several GB in size.

Different access points on the server are used for each of the three sequence compression algorithms. The calling for each algorithm is also parallelized for each sequence, as seen in Figure 3:



Status	Remote Address	Type	Initiator	Size	Time	Waterfall
200	172.19.0.2:80	xhr	xhr.js?b50d:178	5.4 KB	4 ms	
200	172.19.0.2:80	xhr	xhr.js?b50d:178	5.4 KB	23 ms	
200	172.19.0.2:80	xhr	xhr.js?b50d:178	5.4 KB	25 ms	
200	172.19.0.2:80	xhr	xhr.js?b50d:178	750 B	21 ms	
200	172.19.0.2:80	xhr	xhr.js?b50d:178	750 B	21 ms	
200	172.19.0.2:80	xhr	xhr.js?b50d:178	576 B	18 ms	
200	172.19.0.2:80	xhr	xhr.js?b50d:178	576 B	23 ms	
200	172.19.0.2:80	xhr	xhr.js?b50d:178	576 B	23 ms	
200	172.19.0.2:80	xhr	xhr.js?b50d:178	750 B	21 ms	
200	172.19.0.2:80	xhr	xhr.js?b50d:178	402 KB	65 ms	
200	172.19.0.2:80	xhr	xhr.js?b50d:178	402 KB	127 ms	
200	172.19.0.2:80	xhr	xhr.js?b50d:178	402 KB	94 ms	

Fig. 3. Parallel execution of queries for each compression algorithm

Another part of the server provides remote access to compression algorithms, developed as libraries. The library is accessed either through the server or by downloading and compiling it, thus making it usable by applications without Internet access. The library can be used as a DLL from several high-level programming languages, such as C# and Java.

During development, several compression algorithms were tested, and the performance of each is summarized on Figure 4:

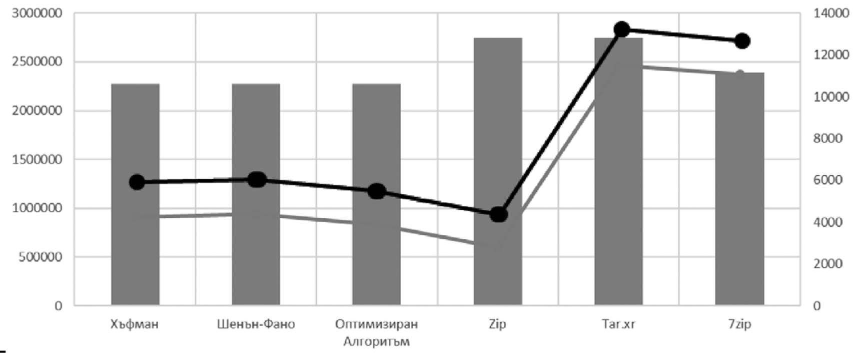


Fig. 4. Compression methods comparison: left vertical scale shows size in bits, right scale shows time in milliseconds. Bars indicate the sequence size after compression, and line shows the time.

The illustrated above is based on the compression of a 9,219,726 byte sequence, obtained from the public database Genome, part of NCBI. This is the entire genome sequence of the bacteria *Bradyrhizobium japonicum*, which can be accessed from Genome's own FTP servers, or through the RefSeq database under the ID NC_017249.1.

Six compression methods were applied to the sequence, including the already discussed in this paper Shannon-Fano, Huffman, Optimized Algorithm. They were compared to: 1) Zip algorithm, developed for compressing and archiving one or multiple files; 2) Tar.xz – UNIX and Windows-based compression software that allows no-loss compression; and 3) 7zip – open-source file archiving software.

The results clearly show the advantage of the first three algorithms that utilize bitwise coding, which result in a very similar performance, producing smaller compressed file and are generally faster than the others (with the exception of Zip).

Huffman and Shannon-Fano give similar results when working with DNA sequences. This is due to the fact that DNA uses only 4-letter alphabet, thus creating similar frequency analyses and overlapping trees with each of those algorithms. The other three methods perform poorly due to them being versatile for various data types and not having been optimized to work with sequences.

The main advantage of our Optimized Algorithm is the speed, because it omits the multiple frequency calculations and tree generation of the other two. It also performs better with sequence decompression, not needing to send or store the compressed sequence tree. This results in faster and safer data transfer, because a sequence-coding tree is also sensitive information in terms of software security. All datasets, samples and the software realization itself, are available:
https://github.com/BroyanaPulova/DNA_compression_web
https://github.com/BroyanaPulova/DNA_compression

5 Conclusion

The Optimised Algorithm proposed here has been developed for sequence data compression. It performs better in terms of speed, noise-reduction, and compression size than similar methods such as Shannon-Fano and Huffman. A library with these bitwise coding algorithms was created, with compression/decompression methods for each. The library utilises low-level primitives so that it can be re-used on various platforms.

The Optimised sequence compression algorithm was implemented in a number of software products with web-based user interface that allows easy, multi-device and multi-resolution access to the server. All components work in a container-based environment, which allows fast and easy regulation.

Future work on this project is going to expand its abilities by adding more compression algorithms, the ability to choose which one(s) to use, and updating the library to work with more programming

Acknowledgements

This work is supported by the project "GloBIG: A Model of Integration of Cloud Framework for Hybrid Massive Parallelism and its Application for Analysis and Automated Semantic Enhancement of Big Heterogeneous Data Collections"; contract DN02/9/2016.

References

[1] Kahn, S.D. (2011) On the future of genomic data. Science (80-.). https://doi.org/10.1126/science.1197891

[2] Via, M., Gignoux, C. and Burchard, E.G. (2010) The 1000 Genomes Project: New opportunities for research and social challenges. Genome Med. https://doi.org/10.1186/gm124

[3] Hudson, T.J., Anderson, W., Aretz, A., Barker, A.D., Bell, C., Bernabé, R.R. et al. (2010) International network of cancer genome projects. Nature. https://doi.org/10.1038/nature08987

[4] Trelles, O., Prins, P., Snir, M. and Jansen, R.C. (2011) Big data, but are we ready? Nat. Rev. Genet. https://doi.org/10.1038/nrg2857-c1

[5] Schadt, E.E., Turner, S. and Kasarskis, A. (2010) A window into third-generation sequencing. *Human Molecular Genetics*,. https://doi.org/10.1093/hmg/ddq416

[6] Kuruppu, S., Beresford-Smith, B., Conway, T. and Zobel, J. (2012) Iterative dictionary construction for compression of large DNA data sets. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*,. https://doi.org/10.1109/TCBB.2011.82

[7] Cao, M.D., Dix, T.I., Allison, L. and Mears, C. (2007) A simple statistical algorithm for biological sequence compression. *Data Compression Conference Proceedings*,. https://doi.org/10.1109/DCC.2007.7

[8] Antoniou, D., Theodoridis, E. and Tsakalidis, A. (2010) Compressing biological sequences using self adjusting data structures. *Proceedings of the IEEE/EMBS Region 8 International Conference on Information Technology Applications in Biomedicine, ITAB*,. https://doi.org/10.1109/ITAB.2010.5687689

[9] Kaipa, K.K., Bopardikar, A.S., Abhilash, S., Venkataraman, P., Lee, K., Ahn, T. et al. (2010) Algorithm for DNA sequence compression based on prediction of mismatch bases and repeat location. *2010 IEEE International Conference on Bioinformatics and Biomedicine Workshops, BIBMW 2010*,. https://doi.org/10.1109/BIBMW.2010.5703941

[10] Pratas, D. and Pinho, A.J. (2011) Compressing the human genome using exclusively Markov models. *Advances in Intelligent and Soft Computing*,. https://doi.org/10.1007/978-3-642-19914-1_29