Hoffman Codation for DNA Sequences Compression

Bacem Saada, *Member, IAENG*, Jing Zhang

***Abstract*— In this paper, we will introduce a two phases compression algorithm based on the binary representation of DNA sequences. In the first phase, we will use an implemented version of the Hoffman codation to efficiently compress and convert the DNA sequence into binary representation. Thereafter, we will compress the resulting DNA using the Extended-ASCII encoding through which one character can represent 16 bytes. The remarkable compression ratio and the easy way to implement our algorithm makes its use interesting.**

***Index Terms*—Extended-ASCII representation, Hoffman codation, DNA compression, horizontal compression;**

I. INTRODUCTION

Nowadays, a huge quantity of digital content is used, shared, analyzed and stored. Powerful computers should be used to properly analyze and store this content. Consequently, two problems have arisen: the first is the encoding of the data and the second is the time required to process them. To reduce data sizes, many data compression methods have been implemented. Compressors such as JPEG and MPEG are lossy compressors that try to remove some information that human being are unable to notice in images. Lossless compressors, on the other hand, compress data without any loss of its information. Therefore, they are used for text compression methods and thus for DNA sequences.

A DNA sequence is a biomolecule present in all cells. This biomolecule contains the genetic information required for the functioning and development of all living beings. Each nucleotide is composed of a nitrogenous base; adenine (A), cytosine (C), guanine (G) or thymine (T). Many public databases (GENBANK, EMBL, etc.) store gigabytes of DNA sequences. This quantity continues to grow to reach the double in only 18 months. The DNA sequences are stored in raw format and this may consequently lead to redundant data. For this reason, we aim at proposing DNA sequences compression algorithms that reduce the size of the DNA sequences and properly analyze and choose which data to be stored.

In this article, we will start with a review of existing DNA sequences compression algorithms (Section II). In section III, we will present our DNA sequences compression approach and explain how it significantly reduces their sizes. Finally, in section IV, we will illustrate the experimental results and we will draw a comparison of ratio between our algorithm and existing algorithms.

Manuscript received June 08, 2016; revised July 14, 2016.

Bacem Saada, Ph.D. Student with Harbin Engineering University, College of Computer Science and Technology, Harbin, China, (email:basssoum@gmail.com).

Jing Zhang, Professor with Harbin Engineering University, College of Computer Science and Technology, Harbin, China, (email: zhangjing@hrbeu.edu.cn).

II. EXISTING COMPRESSION ALGORITHMS

The compression of DNA sequences is based on text compression algorithms. However, researchers proved that conventional text compression algorithms are insufficient for DNA sequences compression. Therefore, they proposed specific compression algorithms. There are two classes of DNA sequences compression algorithms. The algorithms for DNA compression in horizontal mode and the algorithms for DNA Compression in vertical mode. The first class compresses a single sequence based on its genetic information. For example, Biocompress [1] seeks repetitions in a sequence. Biocompress-2[2] uses a Markov model to compress non-repetitive regions of a sequence. By applying these algorithms to the standard benchmark data [3], the compression ratio is 1.85 BpB for Biocompress and 1.78 BPB for biocompress-2.

Some DNA sequences compression algorithms are based on the binary representation of the nucleotides (e.g. A = 00, C = 01, G = 10, T = 11). For example, GENBIT [4] divides sequences into 8-bit blocks and makes a 9^{th} bit. If the block is identical to that above it, the 9^{th} bit is equal to 1, otherwise to 0. DNABIT [5] divides the sequence into small blocks and compresses them while taking into consideration whether they existed previously or not. Saada, B. and Zhang, J compress the DNA sequence to less than 25% of its initial size by using the extended-ASCII representation and applying the RLE technique to compress the similar blocks and keep only one instance [6].

The second class analyzes the genetic information of a set of sequences to identify only one sequence that would represent the whole set. For example, DNAZIP package [7] introduces a series of techniques dividing a genome into small blocks and compressing them. LZ77 [8] proposes a compression technique for a set of genomes belonging to the same genus. Saada, B. and Zhang, J. introduced some techniques to convert the DNA sequence to hexadecimal representation and detect regions of similarities between a set of sequences [9]. They also introduced an algorithm that detects the longest common chain for a set of sequences belonging to the same genus and uses it as representative of the whole set [10].

Studying the DNA sequences belonging to the same genus revealed that they contain similar substrings [11]. Besides, the same DNA sequence contains redundant substrings. For these reasons, we will present, in this paper, a new DNA sequences compression algorithm that compress the redundant substrings of a sequence.

ISBN: 978-988-14048-2-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

III. Our Proposed Algorithm

A. Description of the algorithm

Our algorithm is a statistical algorithm which works in two phases. During the first phase it implements the Hoffman coding to code the frequencies of the nucleotides into a binary representation. On the second phase, to reduce the size of the output data, it converts the bits into Extended ASCII character.

B. Presentation of the algorithm

The main objective of the first phase of our algorithm is to assign variable-length that codes the input blocks of the DNA sequences. The lengths of the assigned codes are based on the frequencies of the blocks. The most frequent block is assigned the smallest code and the least frequent block is assigned the largest code.

1. Extraction Phase

In this phase, our algorithm reads a DNA sequence and divides it into blocks of four nucleotides. The blocks would be stored in a vocabulary table associated with their frequency in the whole DNA sequence. As the length of each block is four characters, the vocabulary size is 256 words (4^4).

Here is an example of this phase of the algorithm is executed how :

ACGT ACGT GATC TAAC ACGT TAAC GAGA AAAC

the vocabulary table will be illustrated as follows:

Table. 1. Vocabulary table content

Index	Word	Frequency
0	ACGT	3
1	GATC	1
2	TAAC	2
3	GAGA	1
4	AAAC	1

To prepare the data to be encoded in the second phase, the content of the vocabulary table is sorted by the frequency of each block. The content of the vocabulary table is shown in table 2.

Table. 2. Vocabulary table content after sorting

Index	Word	Frequency
0	ACGT	3
1	TAAC	2
2	GATC	1
3	GAGA	1
4	AAAC	1

2. Encoding Phase

The second phase of our algorithm encodes the blocks into binary representation following the procedure below:

- Level 0: the first 2^0 words of the vocabulary table that have the most frequency will be coded respectively by 00 and 0.
- Level 1: the next 2^1+0 to 2^2+2^0 words will be coded with four bits by adding 00 for the two higher frequency words and 11 as a prefix for the two others.
- Level l: In general, for the l level, the next 2^l words in the positions from $2^{l-1}+(2^{l-2}+\ldots+0)$ to $2^l+(2^{l-1}+\ldots+0)$ will be coded by adding 00 and 11 as a prefix to the codes generated in the l-1 step.

This procedure is repeated until the N words in the vocabulary table are encoded.

For our example, the codes generated are as shown in table 3.

Table. 2. Vocabulary table codes

Index	Word	Code
0	ACGT	00
1	TAAC	10
2	GATC	0001
3	GAGA	0010
4	AAAC	1101

The output sequence will be as follows:

00 00 0001 11 00 11 0001 1101

3. Additional Structure for storing the level of codes

The number of bits assigned to each block varies from one block to another. For this reason, we need to add an extra table to store the level of coding of each block. This structure will help us to decode the final output chain and obtain the original DNA sequence.

Our example will be as follows:

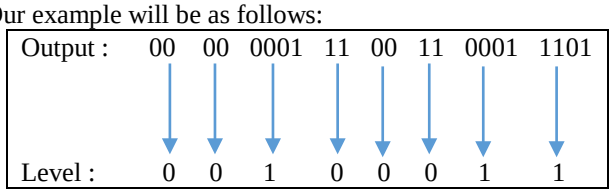

Fig. 1. Additional data structure

4. Compression of the binary output to extended-ASCII representation

To better reduce the size of the output result, we will convert the binary representation to an extended-ASCII representation. The benefit from the use of this technique is that one extended-ASCII character encodes 8 binary digits. This means that the output result will be reduced to 12.5% of its initial representation (fig.2).

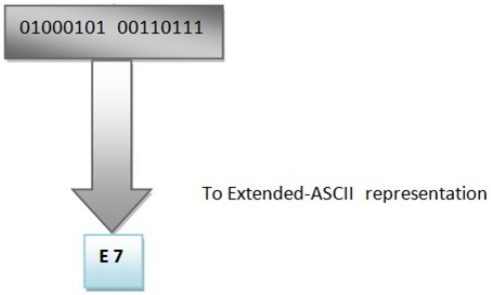

Fig. 2. Conversion to extended-ASCII representation

ISBN: 978-988-14048-2-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

The output result of our example will be as follows:

Output = **1100110001110100**
ASCII = **Ìt**
In this case, we added the suffix "00" to the end of the output file to get 8 binary digits and be able to apply the extended-ASCII coding.
While applying the decoding phase, by referring to the additional data structure, we will delete those bits.

5.The Use of the Run–Length Encoding algorithm

The extended-ASCII representation may contain some repeated sequences of nucleotides. To better compress the sequence, we apply the technique of Run-Length Encoding that detects similar adjacent characters and keeps only one instance of this character. An additional data structure is needed to store the occurrence of these characters and the frequency of its repetition (fig.3).

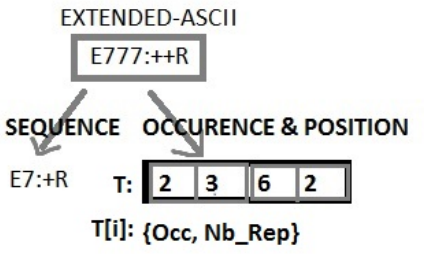

Fig. 3. RLE data structure

6.Decoding Phase

The first step of the decoding phase consists of loading the extended-ASCII representation. If an RLE data structure is used, we would retrace the original representation.
From this Extended-ASCII encoding, we will get an integer number that will be converted into a binary representation. This bit stream will allow the building of the original DNA sequence (Fig. 4).

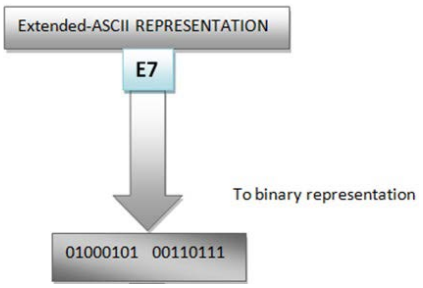

Fig. 4. Conversion to nucleotide representation

The third step consists of loading the additional structure of the blocks' level and retrospecting the original Output file. Our example will be as follows:

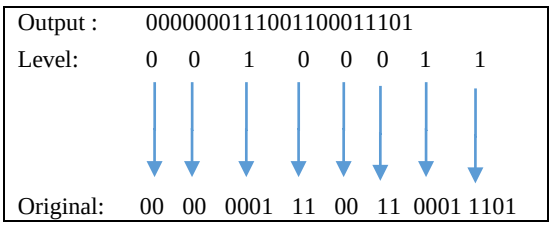

Fig. 5. Additional data structure

The last step is to load the vocabulary table and build the original DNA sequence.

IV. Experimental Results

A. Evaluation Metrics

In order to measure the performance of our algorithm, we use entire genomes. These genomes, having a large number of nucleotides, allow us to calculate the contribution of our approach in terms of compression ratio.

B. Performance in terms of data compression

To achieve our experimental study, we used the Human Globin Gene (HUMHBB), the Human Sequence of Contig (HUMHDABCD) the Mitochondrial genome (MPOMTCG) and the Vaccinia Virus genome (VACCG) whose sizes exceed 180000 nucleotides.
As indicated in table III, applying our algorithm increases the compression ratio of those genomes. The experimental results demonstrate that most of the existing DNA compression algorithms have a compression ratio higher than 1.7 BpB. Our algorithm provides better results and its compression ratio is less than 1 BpB for the compression of all genomes used for the experiments.

TABLE III. Comparison with other algorithms

Sequence	Base Pair	DNA Pack	CTW+LZ	Hoff
HUMHBB	73308	1.77	1.810	0.749
HUMHDABCD	58864	1.74	1.822	0.774
MPOMTCG	186609	1.89	1.90	0.72
VACCG	191737	1.76	1.76	0.671

By applying the conversion to extended-ASCII representation and using the RLE technique, the size of the sequence is reduced to less than 25% of its initial size as shown in table IV.

TABLE IV. PERFORMANCE OF OUR ALGORITHM ON DIFFERENT DNA SEQUENCES AND GENOMES

Sequence Name	Initial size in bits	Size after applying the compression techniques in bits	Size after the Extended-ASCII Compression
HUMHBB	146616	54906	6864
HUMHDABCD	117728	45562	5692
MPOMTCG	373218	134358	16794
VACCG	191737	128656	16082

C. Experiments in Time execution

To measure the execution time of our approach, we used a computer with an Intel i3-2375M processor cadenced at 1.5 Ghz and a 4GB Ram memory.

ISBN: 978-988-14048-2-4
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

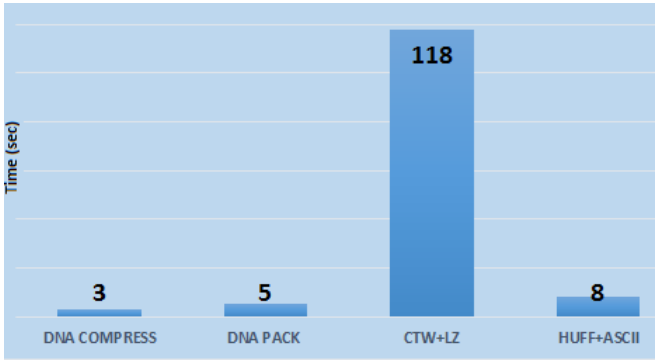

Fig. 6. Execution time comparison between our approach and other algorithms

Figure 6 presents the execution time by applying the approach on the VACCG genome. It proves that its execution time is less than CTW+LZ algorithm and slightly higher than DNA Pack and DNA Compress. To further reduce our approach's execution time, it is possible to parallelize its execution.

V. Conclusion and Future Work

The main advantage of our approach is that it allows to have a compression ratio per base lower than 0.8 BpB thus better than all existing compression algorithms. The algorithm is also easy to implement and interesting to use as the five techniques compress the initial nucleotide representation to less than 25%.

In our future work, we will try to associate our techniques to other compression algorithms based on statistical approaches to compress the DNA sequences with a rate higher than the rate of current existing algorithms.

ACKNOWLEDGMENT

This paper is funded by the International Exchange Program of Harbin Engineering University for Innovation-oriented Talents Cultivation.

REFERENCES

[1] Matsumoto, T., Sadakane, K., Imai, H., et al., 2000, Can General-Purpose Compression Schemes Really Compress DNA Sequences?, Computational Molecular Biology, Universal Academy Press, 76–77.

[2] Grumbach S. and Tahi F.: Compression of DNA Sequences. In Data compression conference, pp 340-350. IEEE Computer Society Press, 1993.

[3] Korodi, G., Tabus, I., Rissanen, J., et al., 2007, DNA Sequence Compression Based on the normalized maximum likelihood model, Signal Processing Magazine, IEEE, 24(1), 47-53.

[4] Grumbach, S., Tahi, F.: A new Challenge for compression algorithms: genetic sequences. Journal of Information Processing and Management 30, 866–875 (1994).

[5] A.AppaRao, "DNABIT compress-compression of DNA sequences," in Proc. the Bio medical Informatics, 2011.

[6] Saada, B., & Zhang, J. (2015). DNA Sequences Compression Algorithm Based on Extended-ASCII Representation. In Proceedings of the World Congress on Engineering and Computer Science (Vol. 2).

[7] Ahmed, S., Brickner, D. G., Light, W. H., Cajigas, I., McDonough, M., Froyshteter, A. B., ... & Brickner, J. H. (2010). DNA zip codes control an ancient mechanism for gene targeting to the nuclear periphery. Nature cell biology, 12(2), 111-118.

[8] Ahmed, S., Brickner, D. G., Light, W. H., Cajigas, I., McDonough, M., Froyshteter, A. B., ... & Brickner, J. H. (2010). DNA zip codes control an ancient mechanism for gene targeting to the nuclear periphery. Nature cell biology, 12(2), 111-118.

[9] Saada, B., & Zhang, J. (2015, November). DNA sequences compression algorithms based on the two bits codation method. In Bioinformatics and Biomedicine (BIBM), 2015 IEEE International Conference on (pp. 1684-1686). IEEE.

[10] Saada, B., & Zhang, J. (2015). Vertical DNA Sequences Compression Algorithm Based on Hexadecimal Representation. In Proceedings of the World Congress on Engineering and Computer Science (Vol. 2).

[11] Saada, B., & Zhang, J. (2015). Representation of a DNA Sequence by a Subchain of its Genetic Information. In Proceedings of the World Congress on Engineering and Computer Science (Vol. 2).