RESEARCH ARTICLE OPEN ACCESS

Loss less DNA Solidity Using Huffman and Arithmetic Coding

Lakshmi Mythri Dasari[#1], Ch Srinivasu[#2], Pathipati Srihari[#3]
[#1, #2] Department of Electronics and Communication
[#1,#2]Dadi Institute of Engineering and Technology,
[#3]National Institute of Technology Karnataka

Abstract
DNA Sequences making up any bacterium comprise the blue print of that bacterium so that understanding and analyzing different genes with in sequences has become an exceptionally significant mission. Naturalists are manufacturing huge volumes of DNA Sequences every day that makes genome sequence catalogue emergent exponentially. The data bases such as Gen-bank represents millions of DNA Sequences filling many thousands of gigabytes workstation storing capability. Solidity of Genomic sequences can decrease the storage requirements, and increase the broadcast speed. In this paper we compare two lossless solidity algorithms (Huffman and Arithmetic coding). In Huffman coding, individual bases are coded and assigned a specific binary number. But for Arithmetic coding entire DNA is coded in to a single fraction number and binary word is coded to it. Solidity ratio is compared for both the methods and finally we conclude that arithmetic coding is the best.
Key words: DNA, Solidity, Huffman coding, Arithmetic Coding, Solidity Ratio.

I. Introduction

The solidity of DNA sequences is one of the most challenging tasks in the field of data solidity. Since DNA sequences are the code of life. We expect them to be non-random and to present some regularity. It is natural to try taking advantage of such regularities in order compactly store the huge databases which are routinely handled by molecular biologists.

The amount of DNA being extracted from organisms and sequenced is increasing exponentially. This yields two problems a) storage b) comprehension. Despite the prevalence of broad band network connection, there still exists a need for compact representation of data to speed up transmission.

DNA is composed only from four chemicals bases: Adenine (A), Thymine (T), Guanine (G), and Cytosine (C). Human DNA consists of about 3 billion bases and more than 99% of those bases are the same in all people, the order of this base determines the information available for building an organism.

The solidity of this huge amount of produced DNA sequences is a very important and challenging task.

II. Proposed Work

In this paper we compress the DNA sequence by using Huffman and Arithmetic Coding.

a) ***Huffman Coding***: The Huffman procedure is based on two observations concerning optimum prefix codes.

1. In an optimum code, symbols that occur more frequently will have shorter code words than symbols that occur less frequently.
2. In an optimum code, the two symbols that occur least frequently will have same length.

The method stars by building a list of all alphabet symbols in descending order of their probabilities. It then constructs a tree.

The tree is built by going through the following steps**.**

1. Combine the two lowest frequencies (probabilities) and continue this procedure.
2. Assign "0" to higher frequency (prob.) and "1" to lower frequency (prob.) of each pair, or vice versa.
3. Trace the path for each character frequency from lower to upper point. Recording the ones and zeros along the path.
4. Assign each character (message) codes sequentially from right to left.

This is best illustrated by an example. Consider DNA sequences AATAAAATAAAACAAAATTAA AAGC whose length is 25.

.

Table I. Huffman coding Table

BASE	FREQUENCY	PROBABILITY	CODE WORD
A	**18**	**0.72**	**0**
T	**4**	**0.16**	**10**
G	**1**	**0.04**	**110**
C	**2**	**0.08**	**111**

The number of bits required to represent the sequence is 35 bits.

b) ***Arithmetic Coding***: In Huffman code, every symbol is assigned a code word. This problem is overcome in arithmetic coding by assigning one code to the entire input stream, instead of assigning codes to the individual symbols. The method reads the input stream symbol by symbol and appends more bits to the code each time a symbol is input and processed.

The algorithm for the arithmetic coding is given below

1. Define an interval [0,1) which is closed at 0 and open at 1.
2. Input the symbol "S" from the input stream.
3. Divide the interval into sub intervals proportional to symbol probability by using the below formulas

$$NewHigh = OldLow + (OldHigh - OldLow) * HighRange(x)$$

$$NewLow = OldLow + (OldHigh - OldLow) * LowRange(x)$$

4. Next subinterval is the main interval.
5. When all the symbols are coded, the final output should be any fractional number that uniquely identifies the interval which is known as tag.

This represents the whole symbol sequence. The tag is then represented in binary digits.

Minimum number of bits required to represent the tag is given by

$$N = \left\lceil \log_2 \left(\frac{1}{P(S)} \right) + 1 \right\rceil$$

Where $P(S)$ is the probability of all symbols.

Consider the subsequence
AATAAAATAAAACAAAATTAAAAGC

Table II. Arithmetic Coding Table

BASE	FREQUENCY	PROBABILITY	RANGE
A	**18**	**0.72**	**[0.28,1)**
T	**4**	**0.16**	**[0.12,0.28)**
G	**1**	**0.04**	**[0.08,0.12)**
C	**2**	**0.08**	**[0,0.08)**

The coding values of the sequence are as follows:
A => [0, 0.08)
A=> [0.0224, 0.08)
T => [0.029312, 0.038528)
A=> [0.03189248, 0.038528)
A=> [0.033750425, 0.038528)
A=> [0.035088146, 0.038528)
A=> [0.036051305, 0.038528)
T=> [0.036348508, 0.036744779)
A=> [0.036459464, 0.036744779)
A=> [0.036539352, 0.036744779)
A=> [0.036516872, 0.036744779)
A=> [0.036638286, 0.036744779)
C=> [0.036638286, 0.036646806)
A=> [0.036640672, 0.036646806)
A=> [0.036642389, 0.036646806)
A=> [0.036643626, 0.036646806)
A=> [0.036644516, 0.036646806)
T=> [0.036644791, 0.036645157)
T=> [0.036644835, 0.036644893)
A=> [0.036644851, 0.036644893)
A=> [0.036644863, 0.036644893)
A=> [0.036644871, 0.036644893)
A=> [0.036644878, 0.036644893)
G=> [0.036644879, 0.036644879)
C=> [0.036644879, 0.036644879)

$$Tag = \frac{0.036644879 + 0.036644879}{2} = 0.036644879$$

Minimum number of bits required is 32.
Binary representation of 0.036644879 is 00000100101100001100011110000110.

From the above example, we can observe that the number of bits encoded for arithmetic coding is less.

III. Solidity Performance

Several Quantities are commonly used to express the performance of a solidity method.

1. Solidity Ratio:

$$CR = \frac{Sizeoftheoutputstream}{Sizeoftheinputstream}$$

A value of 0.6 means that the data occupies 60% of its original size after solidity. Values greater than 1 mean an output stream bigger than the input stream (negative solidity). The solidity ratio can also be called bpb (bit per bit), since it equals the number of bits in the compressed stream needed, on average, to compress on bit in the input stream.

IV. Outputs

Different DNA samples are taken solidity is done by two methods and are compared

Table III. Comparison Results

SL No	Accession Number	Original Size (bits)	After Solidity		Solidity Ratio	
			Huffman	Arithmetic	Huffman	Arithmetic
1	**AF348515.1**	**17160**	**4198**	**4108**	**0.75536**	**0.760606**
2	**AF007546**	**17024**	**4256**	**4202**	**0.75**	**0.753055**
3	**AF008216.1**	**5640**	**1410**	**1380**	**0.75**	**0.755319**
4	**AF186607.1**	**10408**	**2602**	**2591**	**0.75**	**0.751057**

Huffman and Arithmetic coding are applied on DNA sequences. The results shows that solidity is best achieved by Arithmetic coding rather than Huffman coding. Which are developed by MATLAB 8.1.0.604 (R2013a) on a core i5 processor with a 4GB of RAM.

V. Conclusion

The need for effective DNA solidity is evident in biological applications where storage and transmission of DNA are involved. Algorithm using the concept of Huffman and Arithmetic coding is proposed to compress DNA sequences. The Solidity ratio of Huffman coding is less, as the algorithm depends on the occurrence one of the elements with a high probability and the result is a short length of bits and this does not occur in DNA sequence. This is the major limitations of using Huffman code to compress DAN sequence.

Good Solidity ration can be achieved by using adaptive methods, Dictionary learning method, wavelet methods, soft evolutionary computing methods etc.

References

[1] G. Manzini and M. Rastero "*A Simple and Fast DNA Solidity*" February 17, 2004.

[2] H. Afify, M.Islam and M. Abdel wahedL. "*DNA LossLess Differential Solidity Algorithm Based on similarity of genomic sequence data base*. (IJCSIT) Vol 3, No4, August 2011.

[3] M. Burrows and D.J. Wheeler, "*A Block Sorting Lossless Data Solidity Algorithms*", Technical report124, Digital System Research center, 1994.

[4] **P.** G. Howard and J. **S.** Vitter, "*Analysis of arithmetic coding for data solidity*," ***Informat. Process. Manag.,*** vol. 28, no. 6, pp. 749-763, 1992.

[5] I. H. Witten, R. M. Neal, and J. G. Cleary, "*Arithmetic coding for data solidity*," ***Commun. ACM,*** vol. 30, no. 6, pp. 520-540, June 1987.