DNA SEQUENCE COMPRESSION USING STATISTICAL BASED COMPRESSION ALGORITHM

Rexline S J
Associate Professor/Department of Computer Science, Loyola College, Chennai, India
Jasmine Jintha A
Associate Professor/Department of Computer Science, Loyola College, Chennai, India
Trujilla Lobo F
9803, Creek front Road, Jacksonville, Florida-32256, USA.

Abstract: Large amount of space is required to store biological sequences in DNA database like Gen Bank sequence database. The data storage for biological sequences has become very essential in today's current situation. Standard compression algorithms like WinRAR, WinZip are not competent enough to compress biological sequences. In recent times, special algorithms have been introduced specifically for the purpose of compressing the biological sequences like DNA and protein sequences. In this paper, the Statistical compression algorithm is explored to compress the biological sequences. In comparison with the existing general purpose compression algorithms, the proposed Statistical compression algorithm based method compresses these types of sequences better and at the same time the cost of Statistical compression algorithms like Huffman coding, Arithmetic Coding are almost insignificant.

Keywords: Arithmetic Coding, DNA sequences, Huffman Coding, lossless data Compression.

Introduction: In general, compression is a process which reduces the size of the original file without any loss of information in which it saves storage space and reduces the communication costs and also it reduces the time taken to search the pattern or portion of a file through the compressed file. Therefore, compression is considered to be an important research area to improve its algorithms and compressing technologies. Lossless data compression techniques are often partitioned into statistical based compression techniques and dictionary based compression techniques. Statistical compression algorithm is based on the probability that certain character will occur. Huffman Coding and Arithmetic Coding are the kind of statistical coders. Dictionary based compression method exploits repetitions in the data. This coding scheme makes use of the fact that certain groups of consecutive characters occur more than once and assign a codeword to that certain occurrences. Most of the dictionary coders are based on LZ77 and LZ78 and are widely employed to compress the data. The LZW is also one of the dictionary based compression algorithm in which the dictionary is created dynamically and index values are used to represent the repeated dictionary words. The advantage of LZW over the LZ77-based algorithms is its speed because of the limited numbers of string comparisons is enough to perform.

Modern Bioinformatics science produces enormous amounts of genomic sequences, such as nucleotide and amino acid sequences [12]. Rapid growths of molecular research technologies and developments in information technologies have produced a significant amount of data associated with molecular biology. The human genome contains billions of deoxyribonucleic acid (DNA) base pairs. Downloading and maintaining the DNA sequences are much cost consuming factors due to the increasing amount of genome sequences. Hence, decreasing the space required to store the DNA sequences has become a very important new challenge faced by researchers. The genetic activity of every living organism is organized by billions of individual cells. The control-center of each cell is the deoxyribonucleic acid (DNA) that contains a complete set of instructions needed to direct the functioning of each and every one of the cells. The substance of the DNA is the same for all living organisms. The DNA of all organisms has four components in common. They are the four nucleotide bases; namely Adenine, Cytosine, Guanine, and Thymine [10]. They are represented using the first character of their names; namely A, C, G and T respectively. There is another unknown base element represented by the letter N. Therefore the DNA

sequence is represented as a set of {A, C, G, T, and N}. The first four elements are represented as a double helix with A & T in one helix and C & G in another helix. The element N is still remains unknown and is yet to have pictorial representations but participates in the functionalities of a DNA. The use of DNA in genetic engineering, forensics, bioinformatics, and DNA nanotechnology and anthropology applications has been extensive. The DNA Database called GenBank is created and maintained by the National Center for Biotechnology Information (NCBI) .The other two repositories maintain similar data are European Molecular Biology Laboratory (EMBL) and DNA Database of Japan (DDJB). GenBank keeps on growing at an exponential rate, it occupies large space and so it is necessary to compress and store the DNA sequence data. The volume of data creates severe storage and data communications problems. Thus, reduction of the DNA sequence storage costs has become a necessity. Standard compression algorithms like WinRAR, WinZip are not competent enough to compress biological sequences. The Statistical compression algorithm is explored to compress the biological sequences to improve the compression ratio of the biological sequences.

The paper is organized as follows: Section II presents the existing compression algorithm; Section III presents the Statistical compression algorithm to calculate the compression performance. Section IV substantiates the achievability and competence of the proposed method and finally Section V contains the conclusions.

Existing compression algorithms: Researchers have proved that the redundancy in data representation is considerably reduced by compression algorithms as they help decrease the storage required for that data. WinRAR is a file archiver utility for Windows, developed by Eugene Roshal. It can create and view archives in RAR or ZIP file formats, and unpack numerous archive file formats. To enable the user to test the integrity of archives, WinRAR embeds CRC32 or BLAKE2 checksums for each file in each archive. WinRAR supports creating encrypted, multi-part and self-extracting archives. WinRAR is a Windows-only app. The algorithm is based on Lempel-Ziv (LZSS) and prediction by partial matching (PPM) compression.

WinZip mostly uses the DEFLATE algorithm, which is based on using Huffman Codes and LZ77. This is a good, general-purpose compression algorithm known as "deflate". It is the same basic algorithm as is used for *Legacy compression*, but is optimized for speed rather than compressed size. Therefore it will generally compress the files somewhat faster, but the compressed files will be somewhat larger. LZ77 is an open-source data compression method that uses a dictionary compression scheme that uses a larger dictionary size than the deflate method. It can produce a higher compression ratio than older methods.
Bzip2 is an open-source data compression algorithm [13] that compresses most files more effectively than the traditional deflate methods, but it can be somewhat slower. Bzip2 extension designates a pure data compression format not providing file archival feature and BZip2 compression algorithm is based on Burrows–Wheeler transform [3]. Bzip2 algorithm can be easily made parallel and benefit of recent multi-core CPU, but faster than more powerful compression schemes as in WinRAR.

DNA Compression Algorithm using Statistical compressions: Huffman coding, introduced by David Huffman in 1952[8], compresses texts by assigning shorter codes to more frequently used symbols and longer codes to less frequently used ones. This coding is an entropy encoding algorithm used for lossless data compression. A specific method in this coding is used to choose the representation for each symbol, resulting in a prefix-free code [1] that expresses the most common characters using shorter strings of bits. High level languages can be compressed well with Huffman coding algorithm [4] with less memory requirement [5] and high speed [6]. Data structures used in implementation of Huffman coding is also easy to understand and speed up the decoding and encoding process too [14, 15].

Algorithm for building a Huffman code tree is as follows:
a) Create a list of free nodes, where each node corresponds to each symbol present in the source file.
b) Select two free nodes with the lowest frequency from the list.
c) Create a parent node for two nodes found in b) with a frequency equal to the sum of the two child nodes.

d) Remove the two nodes found in b) from the list of free nodes, and add the parent node created in c) to the list.
e) Repeat the process starting from b) until only a single tree remains.
f) After building the Huffman tree, Assign 0 for a left branch and 1 for a right branch.
g) Creates a prefix code for each symbol from the alphabet by traversing the binary tree from the root to the node, which corresponds to the symbol.
The approach of Huffman's algorithm is illustrated in the following DNA sequences with 27 symbols.
GCGAGATCAGACATATAGAGAAAGAGC
Figure 1 illustrates an example of building the Huffman tree. The algorithm starts with a list of nodes 'A', 'G', 'C', and 'T', with frequencies 12, 8, 4, and 3 respectively. The frequencies of each symbol are calculated and a prefix-free tree labeled with 0 (left child) or 1 (right child) shown in figure1 is created for these bases. The generated Prefix code table is given below in Table 1:

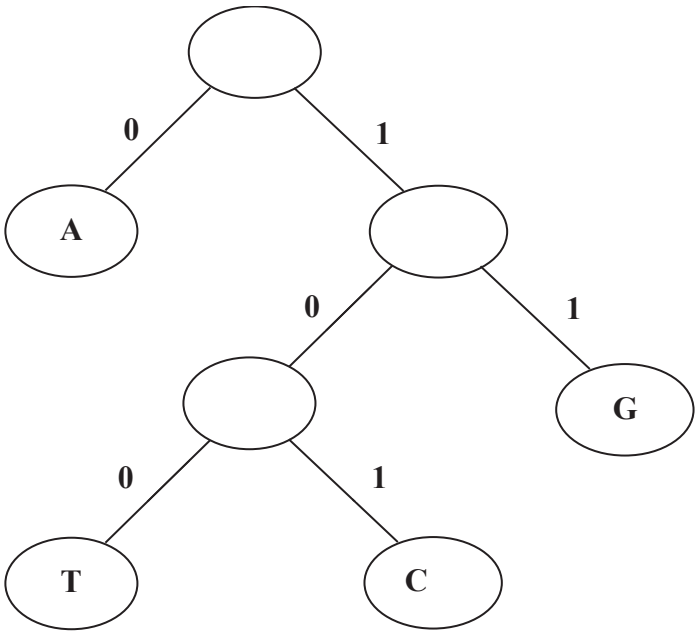

Fig 1: Prefix-Free Tree

Table 1: Prefix Code Table

Symbol	Frequency	Codeword
A	12	0
G	8	11
C	4	101
T	3	100

GCGAGATCAGACATATAGAGAAAGAGC
Single symbol needs 8 bits of storage space
Space required for the above text: 24 x 8 = 192 bits
Here the plaintext is encoded with 49 bits as follows:
11 101 11 0 11 0 100 101 0 11 0 101 0 100 0 100 0 11 0 11 0 0 0 11 0 11 101
Storage space for required for the bases 4 x 8 = 32 Codeword = 09
Total space required for the encoded text = 49 +32 +09 = 90 bits.
Though the usage of Huffman code is wide and frequent, these codes have three major disadvantages. Firstly, two passes are required over the document. Secondly, the coding table is stored along the document in order to reconstruct it.
Procedure for decode the compressed DNA sequence is as follows:
Extract the bits one by one from the compressed file.

If the bit is present in the Prefix Code Table as a codeword, then replace it with the DNA base Else repeat step a.
The approach of Huffman's algorithm is illustrated in the following sample bits.
11 101 11 0 11 0 100
Extract the bit 1, since the single bit 1 is not present in the Prefix Code Table, Extract next bit 1, 11 is present in the Prefix Code Table, replace it with the base B. Extract next bit 1, since the single bit 1 is not present in the Prefix Code Table, Extract next bit 0, 10 is not present in the Prefix Code Table, 101 is extracted and replaced with the base C and so on to get the source file.

Arithmetic coding has been efficiently developed in the place of Huffman coding [2]. In Huffman coding, every input symbol has been replaced by a specific code whereas in arithmetic coding, a stream of input symbols has been replaced by a single floating-point output number. Depending on the length of the message, more bits are needed in the output number [16]. Arithmetic coding is particularly constructive when dealing with symbols with high probabilities. It is also useful that output from an arithmetic coding process is a single number varying from 0 to 1 , not including 1 [11]. This single number can be uniquely decoded to create the exact stream of symbols that has gone into its construction. In order to construct the output number, the symbols being encoded should have a set of probabilities assigned to them. Once the character probabilities are known, the individual symbols need to be assigned a range along a "probability line", which is nominally 0 to 1[9]. It does not matter what characters are assigned to which segment of the range, as long as it is done in the same manner by both the encoder and the decoder. Each symbol is allocated the value in the range of 0-1 that matches the probability appearance of the symbol. It is understood that the symbol holds the values of all except the higher number. Consequently, the last symbol has the range of values between 0.9-0.9999 and not 1. In order to properly decode the first character, the final coded message has to be a number greater than or equal to the range of the first character of the actual stream. The range, that this could fall in, should be maintained to encode this number. So, after the first character is encoded, the algorithm must continue with the next character in actual stream. After the first character is encoded, the low and the high of the first character now bound to the range for the output number. The remaining of the encoding procedure is that each new symbol to be encoded will further control the possible range of the values. If it was the first number in the text, then low and high ranges of values are set directly to those values.

Algorithm for the Arithmetic Coding is as follows:
begin
count source symbols
interval *Ivalue*: = **new** interval 0..1
divide *Ivalue* according to rate of source symbols.
readSymbol(*X*)
while (*X*!=EOF) **do**
begin
new *Ivalue* := subinterval *Ivalue* matching *X*
divide *Ivalue* according to rate of source symbols.
readSymbol(*X*)
end
output(best number from *Ivalue*)

Performance Analysis: To experiment the statistical based compression techniques, a standard set of DNA sequences are compressed with these algorithms and results are compared with the other standard general purpose compressors. There is not any fixed compression ratio, because: it can use different compression algorithms and each of them has a different ratio for different data and the compression ratios of almost all algorithms depends on the data they compress, to be more exact they depends on the entropy of the data (the amount of randomness) and also by adjusting some parameters it is possible to gain higher or lower compression ratio at the cost of the processing time. Efficiency of the proposed method is measured in terms of bits per base (BPB). BPB = (size of compressed file/ size of uncompressed file) x 8. The main concern of the compression algorithm for these sequences is with the compaction ability but not consider with the time taken for the compression.

The following 7 benchmark standard data set of DNA sequences are used in this paper for the purpose of analysis: One chloroplast genomes (CHMPXX), three human genes (HUMHDABCD, HUMHBB and HUMHPRTB) and two virus genomes (HEHCMVCG and VACCG). The DNA sequences are available in FASTA file format in DNA databases which can also retrieved by any text processor. The DNA sequence file is that a single word having no white spaces, soft return or an end of line marker, with a constraint that a nucleotide may appear only nine consecutive times.

Statistical properties of DNA sequences have been extensively studied. Compression algorithms run on ASCII files. A DNA sequence stored in an ASCII file can be stored in a file 25% of its initial size, by a simple encoding of the four characters of the DNA base (A, C, G, and T) on two bits against one byte. The five bases of DNA sequences {A, C, G, T, N} cannot be represented using 2 binary bits. General purpose compression algorithms do not perform well with DNA sequences, resulting quite often in expansion rather than compression. Table 2 shows the Compression ratio of the general purpose compression algorithm for DNA Sequences. Compression ratio of the general purpose compression algorithm for DNA Sequences is compared against with Huffman coding algorithm and Arithmetic coding Algorithm. These Algorithms are implemented and tested with the standard DNA Sequence Corpus. Table 3 shows the

Table 2: Compression ratio of the General Purpose Compression Algorithm for DNA Sequences

Sequence Name	File Size (Bytes)	WinRAR (Bytes)	WinRAR (BPB)	WinZip (Bytes)	WinZip (BPB)	Bzip2 (Bytes)	Bzip2 (BPB)
Chmpxx	121024	34022	2.248	34775	2.298	32099	2.122
Humhbb	73308	20306	2.215	20780	2.267	19684	2.148
Humhdabcd	58864	16110	2.189	16624	2.259	15215	2.067
Humhprtb	56737	15804	2.228	16228	2.288	14854	2.094
Hehcmvcg	229354	66233	2.310	67757	2.362	62169	2.168
Vaccg	191737	53188	2.219	54561	2.276	50209	2.095
Average(BPB)			2.238		2.251		2.060

Table 3: Compression Ratio of the Statistical Compression Algorithm for DNA Sequences

Sequence Name	File Size (Bytes)	Arithmetic Coding (Bytes)	Arithmetic Coding (BPB)	Huffman Coding (Bytes)	Huffman Coding (BPB)
Chmpxx	121024	28776	1.902	29234	1.932
Humhbb	73308	18463	2.015	18354	2.003
Humhdabcd	58864	15077	2.049	14743	2.004
Humhprtb	56737	14362	2.025	14212	2.004
Hehcmvcg	229354	57858	2.018	57366	2.000
Vaccg	191737	46889	1.956	47962	2.001
Average(BPB)			2.002		1.992

Compression ratio of the Statistical compression algorithm for DNA Sequences. According to the results given in Table 3, it is proved that the Statistical compression algorithms give better compression performance compared with existing general purpose compression algorithms.

Conclusion: The concept of Extended Prefix-Free Binary Tree is used to compress DNA sequences which are repetitive as well as non repetitive in nature. Since Huffman code satisfies Prefix property, it's an efficient way to encode and decode DNA sequences. It is very clear that this Prefix-Free Binary Tree technique could provide a better compression performance of up to 75 % reduction in size of DNA sequence file and also it maintains an appealing compression and decompression speed.

References:

1. Alistair Moffat and Andrew Turpin, " On the Implementation of Minimum Redundancy Prefix Codes", IEEE Transactions On Communications, October 1997, Vol. 45, No. 10.
2. Amir Said, "Introduction to Arithmetic Coding -Theory and Practice", Imaging Systems Laboratory, HP Laboratories Palo Alto, April 21, 2004, , HPL-2004-76.
3. M. Burrows and D.J. Wheeler, "A Block-Sorting Lossless Data Compression Algorithm", SRC Research Report 124, Digital Systems Research Center, Palo Alto, CA, 1994.
4. Y. Choueka, S. T. Klein, and Y. Perl, "Efficient variants of Huffman codes in high level languages," in Proc. 8th ACM–SIGIR Conf. Inform. Retrieval, Montreal, Canada, June 1985, pp. 122–130.
5. M. Hankamer, "A modified Huffman procedure with reduced memory requirements," IEEE Trans. Commun., vol. 27, June 1979, pp. 930–932.
6. R. Hashemian, "High speed search and memory efficient Huffman coding," IEEE Trans. Commun., vol. 43, Oct. 1995, pp. 2576–2581.
7. D. R. McIntyre and F. G. Wolff, "An efficient implementation of Huffman decode tables," Congressus Numerantium, vol. 91, 1992, pp. 79–92.
8. D. A. Huffman, "A method for the construction of minimum-redundancy codes", Proceedings of the Institute of Radio Engineers, 40 (9), September 1952, pp. 1098–1101.
9. A. Moffat, R. M. Neal, and I. H. Witten, "Arithmetic coding revisited", ACM Transactions on Information Systems, 16 (3), 1998, pp. 256–294.
10. Rahul Vishwakarma1 and Newsha Amiri, "High Density Data Storage in DNA Using an Efficient Message Encoding Scheme", International Journal of Information Technology Convergence and Services (IJITCS) , April 2012, pp. 256–294.
11. J.Rissanen and G. G. Langdon, "Arithmetic coding", IBM Journal of Research and Development, 23 (2), 1979, pp. 149–162.
12. Sebastian Wandelt, Marc Bux, and Ulf Leser, " Trends in Genome Compression", Knowledge Management in Bioinformatics,Institute for Computer Science, Humboldt-Universit□at zu Berlin, Germany ,June 4, 2013.
13. Seward J. The bzip2 1.0.2 program. 2002.http://sources.redhat.com/bzip2/.
14. A.Sieminski, "Fast decoding of the Huffman codes," Inform. Processing Lett., vol. 26, May 1988, pp. 237–241.
15. H.Tanaka, "Data structure of the Huffman codes and its application to efficient encoding and decoding," IEEE Trans. Inform. Theory, vol. IT-33, Jan. 1987, pp. 154–156.
16. 16. Witten, I.H., R.M. Neal and J.G. Cleary, "Arithmetic coding for data compression", Commun.ACM:30, 1987, pp: 520-540.

***