JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue
JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)
An International Scholarly Open Access, Peer-reviewed, Refereed Journal

# A Compression Algorithm for DNA Sequences based on two LOOK UP TABLEs

**MD. SYED MAHAMUD HOSSEIN[1] , Subhajit Das[2]**
[1]Department of Computer Science & Informatics, Haldia Institute of Technology, Haldia
[mahamud123@gmail.com]
[2]Nayagram Bani Bidyapith,Nayagram

**Abstract**

Storing and transmitting of DNA and RNA sequences is a well known research challenge. The problem has got magnified with increasing discovery and availability of DNA and RNA sequences. The paper represent a new DNA sequence compression algorithm that is based on two Look Up Table, first LUT tables consists of factorial of 4 bases that is 24 sub-sting, each sub-sting is of 4 bases long and another table consists of $4^3$(64) bases that is 64 sub-sting , each sub-string is of 3 bases long. The sub-strings are combined into a two Look up Table based pre-coding routine. The technique can approach a compression rate of 2.6 bits /base and even lower. Information security is the most challenging question to protect the data from unauthorized user. The proposed method may protect the data from hackers. Compression of the genome sequences will help to increase the efficiency of their use. The biggest advantage of this algorithm is fast execution, small memory occupation and easy implementation. Since the program to implement the technique have been written originally in the C language, (Windows platform, and TC compiler) it is possible to run in other microcomputers with small changes (depending on platform and Compiler used). The program runs on the IBM personal computer, requires 512K main memory, without additional hardware except for disk drives and printer. The execution is quite fas, all the operations are carried out in fraction of seconds, depending on the required task and on the sequence length.

**Keywords** :Biology and genetics, Data Compaction and Compression

Abbreviations: LUT, Look Up Table

## 1. INTRODUCTION

With more and more complete genomes of prokaryotes and eukaryotes becoming available and the completion of Human Genome Project on the horizon, fundamental questions regarding the characteristics of these sequences arise. Life represents order. It is not chaotic or random [1]. Thus, we expect the DNA sequences that encode life to be nonrandom. In other words, they should be very compressible. There is also strong biological evidence that supports this claim, it is well known that DNA sequences only consist of four nucleotide bases {*a*, *c*,*g*, *t*}, and one byte are enough to store each base. All this evidence gives more concrete support that the DNA sequences should be reasonably compressible. It is well recognized that the compression of DNA sequences is a very difficult task [2-6]. However, if one applies standard compression tools such as the Unix “compress” and “compact” or the MS-DOS archive programs “pkzip” and “arj”, they all expand the file. These tools are designed for text compression [2], while the regularities in DNA sequences are much subtler. It means that DNA sequences do not have the same properties for the traditional compression algorithms to be counted on. This requires a better model for computing the DNA content such that better data compression results can be achieved. In fact, it is our purpose to reveal such subtleties, such as Look Up Table of factorial of 4 base pair consist 24 sub-string and 3 letter codons 64 sub-string Look Up Table , match with source DNA sequences by using a more appropriate compression algorithm. In this article, we will present a DNA compression algorithm, LUT-3 and LUT-4, based on exact matching between Look Up Table and source file and that gives the best compression results on standard

benchmark DNA sequences. We will present the design rationale of LUT-3 and LUT-4 based on exact matching, discuss details of the algorithm, provide experimental results and compare the results with the one most effective compression algorithm for DNA sequence (gzip-9).
We devised a new DNA sequence compression algorithm combining two Look Up Table pre-coding routine which maps the factorial of ATGC ( form 24 sub-string ) into 24 ASCII characters and another 3 letter codon(form 64 sub-sting )into 64 ASCII characters. Since the essence of compression is a mapping between source file and destination file, the compression algorithm dedicates to find the relationship. We migrate this idea to our research on DNA sequence compression. We are trying to build a finite LUT which implements the mapping relationship of our coding process. Some experiments indicate that the compression ratio is 2.6 bits/base.

## 2. METHODS

2.1: File format :
We will begin discussing file structure is of text file (file extension is dot txt) contain a series of successive of four base pair (a,t,g and c ) and end with blank space ahead the end of file. Text file is the basic element to which we consider in compression and decompression. The output file also text file, contains the information of both unmatch four base pair and a coded value of ASCII character.
The coded valued are located in the encoded section. The coded information is written into destination file byte by byte. The file size depends on number of base pair present in the input file and output file measured by byte, i.e. File size (in byte)= number of base pair in a file(in byte).As per example total no. of base pair in a file is n, so the file size is n byte. ASCII character also required one byte for storing.

**2.2 : Condition for Input file :** The source file/input file must be case sensitive. The input file must be lower case letter of a, u, g and c, incase the input file contain upper case letter algorithm automatically convert into lower case letter. If input file contain RNA data, no problem for processing the algorithm. In case of input file contains DNA, convert into RNA.
Example: AATGCCCCTTTTGGGA……..n where n is the total length of the DNA sequence.
This sequence convert into lower case letter as aatgccccttttggga……..n.
This sequence is valid DNA sequence but not process the algorithm.
Convert into RNA by replacing u in place of t as aaugccccuuuuggga……n, is a valid operational sequence. DNA sequence is act as a input as well as output file.

**2.3 : Encoding and Decoding steps :** First input file encoded by Look Up Table-I, encoded data store on another output file.
**2.3.1: Table –I**
Look up table that we used first time for encoding

| Bases | Char | Bases | Char | Bases | Char |
|---|---|---|---|---|---|
| augc | !(33) | ucag | )(41) | gcau | 1(49) |
| aucg | “(34) | ucga | *(42) | gcua | 2(50) |
| aguc | #(35) | ugac | +(43) | caug | 3(51) |
| agcu | $(36) | ugca | ,(44) | cagu | 4(52) |
| acgu | %(37) | gacu | -(45) | cgau | 5(53) |
| acug | &(38) | gauc | .(46) | cgua | 6(54) |
| uagc | ‘(39) | guac | /(47) | cuga | 7(55) |
| uacg | ((40) | guca | 0(48) | cuag | 8(56) |

Genome sequence consist of four base pair A,T,G and C for DNA and A,U,G and C in case of RNA. That means four character possible orientation is factorial of 4 that is 24 sub string(Factorial(4)=24). All sub-string has 4 bases long. The look-up table describes a mapping relationship between RNA segment and its corresponding characters. We assume that every four characters in source RNA sequence(without $N^2$) will be mapped into a character chosen from the character set which consists of 24 ASCII characters. You can choose other character sets for coding. The one that we used is given in table-I. The braces behind each

character contains the corresponding ASCII codes of these characters. For easy implementation, characters a,u,g,c will no longer appear in pre-coded file and A,T,G,C will appear in pre-coded file. For instance, if a segment "aaugccccuuuuggga……..n " has been read, in the destination file, we represent them as "aAcccdggga…….$n_1$". Obviously, the destination file is case-sensitive.

**Procedure :** Algorith are not perform if non base characters are present in input file. Thus they are A,U,G and C. In such all cases. we read bases 4 by 4. But in some condition, we can't read four successive bases. For example, the segment aatg…. has been read but pre-coding table are not provide ASCII character in that situation we can progress one character forward and left one character in left hand phases and take another four base segment is atgc, again match with pre-coding table if match found place ASCII character in appropriate places for that segment phases. Again raed another four bases segment and match with pre coding table, repeat this methods until at end of file reach. But at the end of file three base segment are remaining We cannot find any arrangement in table-I. In this circumstances, we just write the original segment into destination file.

We know that each character require 1 byte ( 8 bit) for storing. Using lookup table compaction ratio per word is 4:1 when match is found.

In case of above example string length = 16 that means 16 byte require for storing. After encoding on the basis of lookup table , reduce string is aAcccdggga…….. of length 10, require 10 byte for storing this string.

### 2.3.2: Table –II

Look up table that we used second time for encoding

| Bases | Char | Bases | Char | Bases | Char |
|---|---|---|---|---|---|
| uuu | 9 (57) | cca | O (79) | gau | h (104) |
| uuc | : (58) | ccg | P (80) | gac | i (105) |
| uua | ; (59) | acu | Q (81) | gaa | j (106) |
| uug | < (60) | acc | R (82) | gag | k (107) |
| cuu | = (61) | aca | S (83) | ugu | l (108) |
| cuc | > (62) | acg | T (84) | ugc | m(109) |
| cua | ? (63) | gcu | U (85) | uga | n (110) |
| cug | @ (64) | gcc | V (86) | ugg | o (111) |
| auu | A (65) | gca | W (87) | cgu | p (112) |
| auc | B (66) | gcg | X (88) | cgc | q (113) |
| aua | C (67) | uau | Y (89) | cga | r (114) |
| aug | D (68) | uac | Z (90) | cgg | s (115) |
| guu | E (69) | uaa | [ (91) | agu | t (116) |
| guc | F (70) | uag | \ (92) | agc | v (118) |
| gua | G (71) | cau | ] (93) | aga | w(119) |
| gug | H (72) | cac | ^ (94) | ggg | x (120) |
| ucu | I (73) | caa | _ (95) | ggu | y (121) |
| ucc | J (74) | cag | ` (96) | ggc | z (122) |
| uca | K (75) | aau | b (98) | gga | { (123) |
| ucg | L (76) | aac | d (100) | agg | \|(124) |
| ccu | M (77) | aaa | e (101) | | |
| ccc | N (78) | aag | f (102) | | |

Since there are 4 possible bases (A, C, G, U) and 3 bases in the codon, there are 4 * 4 * 4 = 64 possible codon sequences. However, the codon AUG can also be used as a signal to initiate translation, while the codons UAA, UAG, and UGA are terminal codons signaling the end of translation. That leaves a 61 codon sequences that can code for amino acids (AUG can also code for an amino acid). However, there are only 20 amino acids. Therefore the genetic code is redundant, meaning that a single amino acid could be coded for by several different codons. That means four character possible orientation is $4^3$=64. All sub-string has 3 bases long. The look-up table describes a mapping relationship between RNA segment and its corresponding characters. We assume that every three characters in source RNA sequence(without $N^2$) will

be mapped into a character chosen from the character set which consists of 64 ASCII characters. You can choose other character sets for coding. The one that we used is given in Table-II. The braces behind each character contains the corresponding ASCII codes of these characters. For easy implementation, characters a,u,g,c will no longer appear in pre-coded file and A,T,G,C will appear in pre-coded file. For instance, if a segment "aaugccccuuuuggga……..n " has been read, in the destination file, we represent them as "aDNc9txa…….$n_1$".Obviously, the destination file is case-sensitive

**Procedure :** If non base characters are present in input file, algorithm are not performed. Thus they are A,U,G and C. In such all cases. we read bases 3 by 3. But in some condition, we can't read three successive base. For example, the segment aat…. has been read but pre-coding table are not provide ASCII character in that situation we can progress one character forward and left one character in left hand phases and take another three base pair forward that is atg, again match with pre-coding table if match found place single ASCII character in appropriate places for that segment phases. Again raed another three bases segment and match with pre coding table-II, repeat same methods until at end of file reach. But at the end of file two base segment are remaining We cannot find any arrangement in table-II. In this circumstances, we just write the original segment into destination file.

We know that each character require 1 byte ( 8 bit) for storing. Using lookup table compaction ration per word is 3:1 when match is found.

In case of above example string length = 16 that means 16 byte require for storing this string. After encoding on the basis of lookup table of 3 sub string length size , reduce string length 8, require 8 byte for storing this string.

[2]N Refers to those not available base in RNA sequence

$n_1$ = n-2*No. of base pair match.

**2.4 :** For that purpose we have develop two encoding(compression) algorithm as well as two 'C' programmed, first programmed is LUT-3, Second programmed is LUT-4. Also developed two decoding(decompression) algorithms as well as two programmed for decoding the encode sequences.

**2.4:1: Compression algorithm**

**2.4:1:1: Algorithm for LUT-3**

Input:-a file containing ATGC sequence

Output:-a file that compress ATGC sequence

In this algorithm we have to first define the look up table that containing different combinations of atgc and its substitute character. Then we follow the following steps for our DNA compression technique

Step1← Take the input file contain ATGC sequence.

Step2← if( input file is not open)
Print Unable to open the file
Exit from the program.
Else
Convert every character of input file in
lower case.
Go to step 3
End of if structure.
Step 3← set loc variable to the beginning of the input file.
Step4← while (end of input file)
Set count=loc
for i=0 to 2
A[i]=(value(loc))th character
count=count+1
end of for structure

step5← if(A[ ]array match with
look up table)
put the corresponding character in output file
loc=loc+3
else
put the A[0]th character in output file
loc=loc+1;
end of if structure.
step 6← end of while structure

**2.4:1:2: Algorithm for LUT-4**
Input:-a file containing ATGC sequence
Output:-a file that compress ATGC sequence

In this algorithm we have to first define the look up table that containing different combinations of atgc and its substitute character. Then we follow the following steps for our DNA compression technique
Step1← Take the input file contain ATGC sequence.
Step2← if( input file is not open)
Print Unable to open the file
Exit from the program.
Else
Convert every character of input file in lower case.
Go to step 3
End of if structure.
Step 3← set loc variable to the beginning of the input file.
Step4← while (end of input file)
Set count=loc
for i=0 to 3
A[i]=(value(loc))th character
count=count+1
end of for structure
step5← if(A[ ]array match with look up table)
put the corresponding character in output file
loc=loc+4
else
put the A[0]th character in output file
loc=loc+1;
end of if structure.
step 6← end of while structure

**2.4:2: Decompression algorithm**
**2.4:2:1: Algorithm for LUT-3**
Input :-a file in compress form
Output:-original ATGC sequence
This algorithm take the compressed DNA sequence as a input file and scan each character
If a match is found then put the matched sequence in output file otherwise put the unmatched character in output file directly. This method develop by algorithm define below.
Step 1← while(end of input file)
Step2← scan each character of input file
Step 3←if (each character match with LOOK UP)
put the corresponding sequence in output
else
put unmatched character to the output file

step 4← end of while structure

**2.4:2:2: Algorithm for LUT-4**

Input :-a file in compress form

Output:-original ATGC sequence

This algorithm take the compressed DNA sequence as a input file and scan each character
If a match is found then put the matched sequence in output file otherwise put the unmatched character in output file directly. This method develop by algorithm define below.

Step 1← while(end of input file)
Step2← scan each character of input file
Step 3←if (each character match with LOOK UP)
put the corresponding sequence in output
else
put unmatched character to the output file
step 4← end of while structure

## 3. EXPERIMENTAL RESULTS

We tested LUT(3 bases sub-string or 4 bases sub-string) on standard benchmark data used in [3]. These standard sequences come from a variety of sources and include the complete genomes of two mitochondria: MPOMTCG, PANMTPACGA (also called MIPACGA); two chloroplasts: CHNTXX and CHMPXX (also called MPOCPCG); five sequences from humans: HUMGHCSA, HUMHBB,HUMHDABCD,HUMDYSTROP,HUMHPRTB; and finally the complete genome from two viruses: VACCG and HEHCMVCG (also called HS5HCMVCG).
These tests are performed on a computer whose CPU is Intel P-IV 2.4GHz core 2 duo(1024FSB), Intel 946 original mother board, IGB DDR2 Hynix, 160GB SATA HDD Segate.
The definition of the compression ratio[1] is the same as in [3]; i.e.,$1-(|O|/2|I|)$, where $|I|$ is number of bases in the input DNA sequence and |O| is the length (number of bits) of the output sequence, other compression ratio[2] which is defined as $1-(|O|/|I|)$, where $|I|$ is the length(number of byte) number of bases in the input DNA sequence and |O| is the length (number of byte) of the output sequence. The compression rate, which is defined as $(|O|/|I|)$, where $|I|$ is number of bases in the input DNA sequence and |O| is the length (number of bits) of the output sequence. The improvement[9] of LUT-3 and LUT-4 over gzip-9, which is defined as ( Ratio_of_gzip-9 – Ratio_of_LUT-3)/Ratio_of_gzip-9 *100 with respect to LUT-3, in case of LUT-4, calculation are done by same process. The improvement are verified on average value in each case. The compression ratio and compression rate of LUT-3, as well as those of LUT-4 are presented in Table-III to Table-VI. We have also compared LUT-3 and LUT-4 with gzip-9[7] in same table through Table-III to Table-VI.. In figure-I show the compression result in all data set at glance where x axis show the input file size(in byte) and y axis show the output file size(in byte).

## 4. RESULT DISCUSSION:

In Chen's paper[8],you can feel the time of program running since they are in second level. Some ones even cost minutes or hours of time to run. But our algorithm runs almost $10^3$ time faster than them. ,our algorithm performances better than it in both compression ratio and elapsed time. Our algorithm is very useful in database storing. You can keep sequences as records in database instead of maintaining them as files. By just using the pre-coding routine, users can obtain original sequences in a time that can't be felt. Additionally, our algorithm can be easily implemented while some of them will take you more time to program.
From these experiments, we conclude that pre coding matching patter are same in all type of sources and pre coding Look up Table plays a key role in finding similarities or regularities in DNA sequences. Straight

line graph declared that compression rate are same in all type of sources. Output file contain ASCII character with unmatch a,u,g and c so, it can provide information security which is very important for data protection over transmission point of view. The compression ratio and compression rate prove that "Life represent order, it is not chabtic or randam". Output result show that LUT-3 algorithm is better than LUT-4 algorithm with respect to Compression ratio, compression rate, speed for both encoding and decoding. We have successfully compressed the nucleotide sequences without dataloss any data in encoding as well as decoding time.

## 5.CONCLUSION:

In this article, we discussed a new DNA compression algorithm whose key idea is LUT. This compression algorithm gives a good model for compressing DNA sequences that reveals the true characteristics of DNA sequences. The compression results of LUT-3 for DNA sequences also indicate that our method is more effective than many others. LUT-3 is able to detect more regularities in DNA sequences, such as mutation and crossover, and achieve the best compression results by using this observation.LUT-3 fails to achive higher compression ratio than others standard method.

## 6. FUTURE WORK :

We are trying to do more, such as combining our LUT pre-coding routine with other compression algorithms, to revise our algorithm in order to improve its performance.
Encoding time "Sub-sequence size-1" base segment are remaining, (if at the end of file segment are not match exactly with pre-coded table)We cannot find any arrangement in table-I or table-II. In this circumstances, we just write the original segment into destination file. To increase the probability of compaction we match the sequence in other orientation such as reverse , complement and reverse complement the input file.
Availability: The executable files will be available upon request for academics.

References

[1] M. Li and P. Vitányi, An Introduction to Kolmogorov Complexity and Its Applications, 2nd ed. New York: Springer-Verlag, 1997.
[2] R. Curnow and T. Kirkwood, "Statistical analysis of deoxyribonucleic acid sequence data-a review," J. Royal Statistical Soc., vol. 152, pp. 199-220, 1989.
[3] S. Grumbach and F. Tahi, "A new challenge for compression algorithms: Genetic sequences," J. Inform. Process. Manage., vol. 30, no. 6, pp. 875-866, 1994.
[4] É. Rivals, O. Delgrange, J.P. Delahaye, M.Dauchet, M.O. Delorme et al., "Detection of significant patterns by compression algorithms: the case of Approximate Tandem Repeats inDNAsequences," CABIOS, vol. 13, no. 2, pp. 131-136,1997.
[5] K. Lanctot, M. Li, and E.H. Yang, "Estimating DNA sequence entropy,"in Proc. SODA 2000, to be published.
[6] D. Loewenstern and P. Yianilos, "Significantly lower entropy estimates for natural DNA sequences," J. Comput. Biol., to be published (Preliminary version appeared in a DIMACS workshop, 1996.)
[7] T.Matsumoto,K.Sadakame and H.Imani, "Biological sequence compression algorithm", Genome Informatics 11:43-52 (2000).
[8] X. Chen, M. Li, B. Ma, and J. Tromp, "Dnacompress:fast and effective dna sequence compression," Bioinformatics, vol. 18,2002.
[9] Xin Chen, San Kwong and Mine Li, "A Compression Algorithm for DNA Sequences Using Approximate Matching for Better Compression Ratio to Reveal the True Characteristics of DNA", IEEE Engineering in Medicine and Biology,pp 61-66,July/August 2001.
[11] ASCII code. [Online]. Available: http://www.asciitable.com
[12] National Center for Biotechnology Information,http://www.ncbi.nlm.nih.gov