International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395 -0056
Volume: 03 Issue: 05 | May-2016 www.irjet.net p-ISSN: 2395-0072

Reverse Sequencing based Genome Sequence using Lossless Compression Algorithm

Rajesh Mukherjee [1], Subhrajyoti Mandal [2], Bijoy Mandal [1]

[1] *Dept. of CSE, NSHM Knowledge Campus, Durgapur, WB, India*

[2] *Datamatics Global Services Ltd, Bangalore, India*

Abstract - *Genome sequence based on reversed sequencing is a lossless compression algorithm. We will introduce a DNA compression algorithm, founded on exact reverse matching which gives the best compression results on standard DNA sequences benchmark. In an immensely long DNA sequence, searching of all exact reverses is non-trivial task. To find approximate reverses optimal for compression, this algorithm takes a long time (essentially a quadratic time search or even more). Also, obtaining high speed and best compression ratio is a challenging task. Proposed DNA sequences compression achieves a better compression ratio and runs significantly faster than any existing compression program for benchmark DNA sequences, simultaneously.*

Key Words: **lossless compression algorithm, encoding, decoding, palindrome, DNA sequences, ASCII character**

1. INTRODUCTION

Adenine, Cytosine, Guanine and Thymine are the four bases found in DNA. Those are abbreviated as A, C, G, T respectively. DNA sequencing is finding the order of DNA nucleotides or bases, i.e. in a genome, the order of As, Cs, Gs, Ts that make up an organism DNA. Sequencing the genome is an important step towards understanding it. The importance of genome sequence is that, a genome sequence does contain some clues about where genes are. These clues are useful for interpretation. The human genome is made up of over 3 billion of these nucleotides. The human genome is about 20-40 percent repetitive DNA, but bacterial and viral genomes contain almost no repetition [1].

With the completion of the human genome project, an enormous quantity of different genome sequences becoming available, whose size varies in the range of millions to billions of nucleotides. In both scientific and commercial communities there is an intensive activity targeted at sequencing the DNA of many species and studying the variability of DNA between individuals of the same species, which produces huge amounts of information that need to be stored and communicated to a large number of people. Therefore, there is a great need for fast and efficient compression of DNA sequences [2]. From the viewpoint of information science; we can use compression techniques to capture the properties of DNA sequences. It is known that DNA sequences have two characteristic structures. One is reverse complements and the other is approximate repeats. The reverse complement of a sequence is a reverse sequence whose each symbol is replaced with its complement one. The approximate repeats are repeats that contain errors. There have been developed several special-purpose compression algorithms for DNA sequences have been developed (Grumbach and Tahi [3], Chen, Kwong and Li [4], Lanctot, Li and Yang [5]). These algorithms use the structures and can achieve high compression ratio.

Now, it is known that DNA macromolecule comprises of two strands: Coding strand and Non-coding strand. The coding region contains the information (digital code) for synthesizing proteins. Only about ten percent of genetic material of Human beings contains coding region i.e. genes. The rest is considered to be non-coding. Non-coding strand of DNA does not carry any information necessary to make proteins [6]. Therefore, the compression ratio of coding and non-coding regions of DNA sequence must be different and the two regions should have different information theoretical entropy. This is supported by a biological hypothesis (Lanctot, Li and Yang [5]). From these scenarios, one fundamental question should be raised about the nature of the DNA sequence, i.e random or nonrandom. Unfortunately the compression of genetic sequences happens to be a very difficult task. They are at a glance, very similar to random strings and have only very hidden regularities. The classical algorithms like compact and compress from Unix and the text compression algorithm provided in [Nel 91] [6] namely static and adaptive Huffman's encodings, static and adaptive arithmetic encoding including higher order encodings and various substitution algorithms based on Ziv and Lempels methods for the text compression, fail to compress genetic sequences. Rather they extend the contents of the sequences, leading to negative compression rates [6].

Life represents order. It is not chaotic or random [7]. Thus, we expect the DNA sequences that encode Life as nonrandom. Naturally they should be very compressible. There are also strong biological evidences in supporting this claim: It is well-known that DNA sequences, especially in higher eukaryotes, contain many repeats. It is also established that many essential genes (like rRNAs) have many copies. It is believed that there are only about a thousand basic protein folding patterns. Further it has been conjectured that genes duplicate themselves sometimes for evolutionary or simply for "selfish" purposes. These all concretely support that the DNA sequences should be reasonably compressible. It is well recognized that the compression of DNA sequences is a very difficult task [7, 8, 9, 10]. DNA sequences only consist of 4 nucleotide bases (a, c, g, t). It has to noted that t is replaced with u in the case of the RNA. 2 bits are enough to store each base. However, if one applies standard compression software such as the Unix "compress" and "compact" or the MS-DOS archive programs "pkzip" and "arj", they all expand the file with more than 2 bits per base, although all these compression software are universal compression algorithms. These software are designed for text compression [11], while the regularities in DNA sequences are much subtler due to the characteristic structures of DNA such as palindromes, approximate repetition, reverse substring etc. It is our purpose to study such subtleties in DNA sequences. Most of the DNA compression methods fall into two categories. First is statistical method, which compresses data by replacing a more popular symbol to a shorter code. Second is dictionary-based scheme, which compresses data by replacing long sequences by short pointer information to the same sequences in a dictionary [12].

In statistical methods, arithmetic coding and CTW are known to compress the DNA data well [13]. However, they have disadvantages such as low decompression speed. Also, Huffman coding cannot compress efficiently [14]. For dictionary-based methods, LZ77 scheme is known to be the best method for compressing DNA data so far. Several DNA-oriented algorithms have been tried to make the best of the characteristics of DNA, such as reverse complement and point mutation in order to apply LZ77 scheme more efficiently [15]. Proposed algorithm consists of two phases:

One is find all exact reverses and other is encoding exact reverse regions and non-reverse regions. We have developed for fast and sensitive homology search [16], as our exact reverse search engine. Compression of DNA sequences is a very challenging task. This can be seen by the fact that no commercial file-compression program achieves any compression on benchmark DNA sequences we use in this paper. Several compression algorithms specialized for DNA sequences have been developed in earlier studies elsewhere.

We will present a DNA compression algorithm, based on reverse substring and corresponding reverse original substring is placed in Library file. This reverse original substring creates a dynamic Look Up Table and place ASCII character in appropriate places on source file that gives the best compression results on standard benchmark DNA sequences. We will 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 find the compression ratio, compression rate result in other orientation such as the reverse (means the substring is reverse and find the exact reverse in normal sequences), the complement and the reverse complement the input sequences. Also we can find the compression rate, compression ratio of randomly generated equivalent length of artificial DNA sequence. Compare all the results to each other.

In this paper, if not otherwise mentioned, we will use lower case letters u, v, to denote finite strings over the alphabet {a, c, g, t},|u| denotes the length of u, the number of characters in u. Uj is the i-th character of u. Uj:j is the original substring of u from position i to position j. The first character of u is ul .Thus u = u| : |u|-i. and |v| denotes the length of v, the number of characters in v. vi is the i-th character of v. vi:j is the reverse substring of v from position i to position j. ui:j match with vi:j. The first character of v is v1. Thus v = v1 : |v|-i. The minimum different between u-v is of substring length. The reverse substring found if ui:j = vi:j and count exact maximum reverse of ui:j We use ε to denote empty string and ε=0. Also we can say that transpose of u is v. In reversible substring found if u=v all cases. Place u in the library file. It is quite pertinent to mention that, if original substring are repeated in this the sequence, in that situation we cannot provide any corresponding ASCII code for repeating the substrings, just we can place the substring base pair in output file. So, the output file size relatively increases.

2. METHODS

2.1 File format

We will begin discussing file type as text file (file extension is dot txt) contain a series of successive four base pairs (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 compression and decompression. The output file also the text file, contains the information of both unmatch four base pairs 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 for example total number of base pair in a file is n, so the file size is n bytes. ASCII character is also required one byte for storing. On the basis of ASCII code availability, we can take input as a lower case letter of a, t, g and c.

2.2 Generating the substring from input sequence

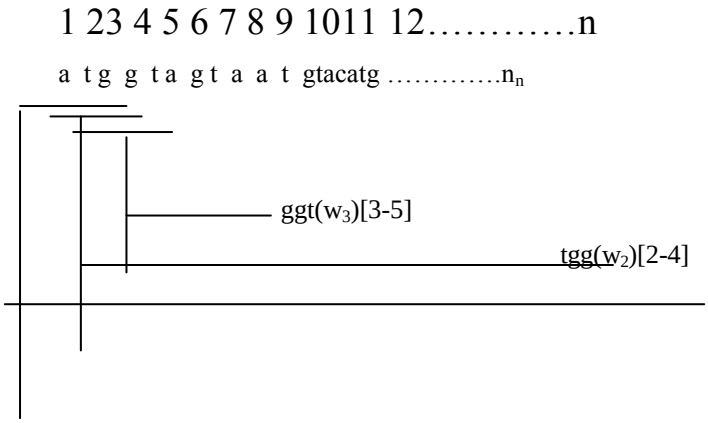

Fig.-I: Substring creation both in normal substring, reverse substring.

From the pictorial representation of fig-1 it is clear that for ith substring W;. i, is the starting position of the substring and j= (i-1) + 1, is end position of the substring; where 1 is the substring length.

2.3 Example

As for example if substring length is 3 then:

W1 starting position (i)=l and (end position)j= (1-1) + 3=3,

W2 starting position (i)=2 and (end position)j= (2-1) + 3=4 and

W3 starting position (i)=3 and (end position)j= (3-1) + 3=5 and so on.

The substring length is less than 3 (three) has no importance in matching context therefore we consider the substring size in the range: 3 < 1 < n.

Therefore, the range for i and j are as 1 < i < n-1+1 and 1 < j <n respectively.

2.4 Searching for exact reverse substring

Consider a finite sequence s over the DNA alphabet {a, c, g, t}. An exact reverse is a substring in s that can be transformed from another substring in s with edit operations (reverse, insertion). We only encode those exact reverses that provide profits on overall compression. This method of compression is as shown below:

1. Run the program and output all exact reverses into a list s in the order of descending scores;

2. Extract a reversible r with highest score from list s, then replace all r by corresponding ASCII code into another reverse list o and place original substring corresponding ASCII code in library file.

3. Process each reversible in s so that there is no overlap with the extracted reversible r;

4. Go to step 2 if the highest score of reversible in s is still higher than a pre-defined threshold; otherwise exit.

2.5 Example

Let s=atggtagtagtagtaggttgg.....n

{atg substring match with four places of reversible substring (gta), tgg substring match with one place

on reverse substring (ggt), ggt substring match with one place on reverse substring (tgg) and so on.

First replaced highest match score of original substring relating all reversible atg substring by ASCII

Character and insert ASCII equivalent symbol in i^{th} position and original substring encoded by corresponding ASCII code.

B-atg!!!!ggttgg { B is intermediate encoding step}

o=atg!!!!ggt#[where o is the compress output file]

All those extracted repeats in list B then parse a DNA sequence into a mixture of regions with little structure and reverse regions each of which can be replaced by a substring previously located.

2.6 Encoding procedure

An exact reverse can be presented as two kinds of triples, first is (I, m, p), where I means the original substring length and 1 reversible substring length, m and p show the starting positions of two substrings, second replace : this operation is expressed as (r; p; char) which means replacing the exact reversible r substring at position p by ASCII character char.

In order to recover an exact reverse correctly the following information must be encoded in the output data stream.

2.7 Encoding Analysis

We can write s= atggtagtagtagtaggttgg.....n n>0 and l<i<n-l+l.

Consider the sequence is defined by s, consider substring store in S[m] and all reversible substring are store in S[p].

After breaking the sequence(s) into substring of three bases long we can get the result as below. So, we can get an original substring S[m]=S[l].....S[n-2*H-l] l<m<n-2*l+l and Reversible substrings are S[p]=S[l].....S[n-l+l] l<p<n-I+l.

If the number of substring in S[m], total number of substring are generated by (n-2*l+l) and number of mach reversible substring in S[p], total match reversible substrings are (n-1+1)

As per above example s[m]— *s[l]=atg and so on and s[p] — »s[l]=gta and so on.

This substring method requires reducing the complexity of the programmed execution.

2.8 Each original substring match with all other reversible substring for finding the exact maximum reversible substring

Match condition occur if S[m]=S[p] p=l+l

Step→ I

S[l] match with S[p] to S[n-l+l] and count S[l]

{As for example S[l]=atg where substring size=3

and S[4]=gta, S[5]=gta....S[19]=tgg

So, S[l] substring match at 4 places

Then m and p incremented by one

Step→2

Match S[2] match with S[p] to S[n-l+l] and count S[2]

[As for example S[2]=tgg

and S[5]=tag, S[6]=agt

Sc . S[2] substring match at one places

Then m and p incremented by one

Step→3

This method will continue to S[n-l+l]

So S[n-2*l+l] match with S[p] to S[n-2*l+l] and count S[n-2*l+l]

So, S[n-2*L+l] reverse only one place if mach occur.

Step→4

Store all reverse count in descending order and find all exact maximum reverse count

Step→5

Replace exact maximum reversible substring by corresponding ASCII code and place original

substring in library file by his corresponding ASCII code, and create a on-line Look Up Table.

Step→6

Repeat Step-1 to step-5 excluding ASCII code

Step→7

If the highest score of reverses in s is still higher than a pre-defined threshold; otherwise exit.

As per above example: Now we find maximum reverse probability. This substring is replaced first.

Here, we can get S[l] = (atg) original substring reversible (gta) substring match 3 times in this

sequence.

This original substring is placed in Look Up Table, encoding corresponding ASCII character [32(

space)] and replace all reversible substrings by ASCII character (54).

So, n=Length of the string = Total number of base pairs in s = File size in byte

The Encoding procedure follow this rule and produce compression output file.

S[m] match with S[p] to S[n-l+l], place ASCII character in the output file in the ith position. Each match cases the value of p is incremented by;

p=number of unmatch character+ (number of reverse

substring match * substring length +1)

Otherwise S[m]^S[p] to S[n-l+l] place base pairs in output file in the ith position. If unmatch occurs, the values of m and p are incremented by one.

At the end, we can get the compressed output file o which contains the unmatch a, t, g and c and ASCII character set.

At the end we can get the compressed file, corresponding input sequence.

So, o= atg !!!!ggt#.....ni where m is the length of output file. Output file size is ni byte. And library file :atg ! ggt #

2.9 Decoding procedure

Decoding time first requires on-line Library file, which was created at the time of encoding the input

file. On this particular value , the encoded input string is decoded and produce the output original file.

2.10 Look Up Table

O= atg !!!! ggt#.....ni where ni is the length of output string (n>n]).

At the time of decoding each ASCII character is replaced by corresponding base pair i,e O[M]=L[k] where O[M] is defined by output sequence and L[k] is defined by library file substring. If match occurs in between L[k]; L[32] to L[53] with O[M]; O[54] to O[256], place ASCII equivalent substring in the ith places in output file. The value of M is incremented by one. If no match found in between L[54] to L[256] with k[32], place base pair in the i^{th} position in output file. The value of k is incremented by one.

This process will continue until M—m position will appear.

The Decoding process mentioned this rule and produce original output string. Match found if o[m]=L[32] to L[54] place ASCII character equivalent substring in the i^{th} position. If match found, the value of m is incremented by one.

Otherwise o[M]^L[32] to L[53] place base pair in the ith position in output file. If unmatched occurs, the value of k is incremented by one.

For easy implementation, characters a, t, 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 "atggtagtagtagtaggttgg.....n " has been read, in the destination file, we represent them as "atg !!!!ggt#.....ni". Obviously, the destination file is case-sensitive

It is known that each character requires 1 byte (8 bit) for storing. In the above example string length = 21 that means 21 bytes are required for storing this string. After encoding on the basis of reverse techniques of 3 substring length, reduce string length is 11, require 11 byte for storing this string.

2.11 Random string generation method

We have generate a string of four symbols (a, t, g and c) of any arbitrary length, it is user requirement. This method simply use random function C++.

3. ALGORITHM EVALUATION

3.1 Accuracy:

As to the DNA sequence storage, accuracy must be taken firstly in that even a single base mutation, insertion, deletion or SNP would result in huge change of phenotype as we see in the sicklemia.It is not tolerable that any mistake exists either in compression or in decompression. Although not yet proved mathematically, it could be inferred from reverse techniques that our algorithm is accurate, since every base arrangement uniquely corresponds to an ASCII character.

3.2 Efficiency

We can see that the internal reverse algorithm can compress original file from substring length (I) into I characters for any DNA segment, and destination file uses less ASCII character to represent successive DNA bases than source file.

3.3 Space Occupation

Our algorithm reads characters from source file and writes them immediately into destination file, it costs very small memory space to store only a few characters. The space occupation is in constant level. In our experiments, the OS has no swap partition. All performance can be done in main memory which is only 512 MB on our PC.

4. EXPERIMENTAL RESULTS AND DISCUSSION

We tested reverse techniques on standard benchmark data used in [17]. For testing purpose we use eight types of data. These tests are performed on a computer whose CPU is Intel P-IV 3.0 GHz core 2 duo(I024FSB), Intel 946 original mother board, IGB DDR2 Hynix, 160GB SATA HDD Seagate. Since the program to implement the technique have been written originally in the C++ language, (Windows XP platform, and TC compiler) it is possible to run in other microcomputers with small changes (depending on platform and compiler used). The program, requires 512K, without additional hardware except for disk drives and printer.

The definition of the compression ratio [11]; 1-(|O|/2| /)), where |7| is number of bases in the input DNA sequence and |O| is the length (number of bits) of the output sequence, The compression rate, which is defined as (|O|/| -/]), where |7] is number of bases in the input DNA sequence and |O| is the length (number of bits) of the output sequence. Total reduce file size is equal to Compress file size plus Library file size in byte, i.e (T-C+L byte). The improvement [9] over gzip-9, which is defined as (Ratio_of_gzip-9 - Ratio_of_RCR)/Ratio_of_gzip-9*100. The compression ratio and compression rate are presented in Table-I to Table-III. Our result compared with gzip-9 [l1] in the same table. The compression ratio and compression rate are shown in different table in column. The normal sequence and artificial sequences result shown in Table-I and the reverse, the complement and the reverse complement sequences result shown in table-II to table-III. The artificial results are shown in same Table.

Sequence Size	Sequence Type	Normal Sequence								Artificial Normal Sequence							
	Sequence Name	atatsgs	ateflu23	atrdnaf	atrdnai	celk07e12	hsg6pdgen	mmzp1g	rlxlxfg512	XX1	XX2	XX3	XX4	XX5	XX6	XX7	XX8
	Base pair/ File size in byte	9647	6022	10014	5287	58949	52173	10833	19338	9647	6022	10014	5287	58949	52173	10833	19338
Substring Size-4	Reduce file Size byte (C)	5181	3282	5236	2703	30029	27973	5203	9110	5211	3232	5320	2843	31559	28023	5777	10318
	Lib file Size (L)	162	162	156	144	174	168	280	280	150	162	158	150	156	156	162	162
	Total Compress File size(T)	5343	3444	5392	2847	30203	28141	5483	9390	5361	3394	5488	2993	31715	28179	5939	10480
	Compression Rate (bits/base)	4.430808	4.575224	4.307569	4.307925	4.098865	4.315029	4.049109	3.88458	-1.22287	-1.2544	-1.19213	-1.06442	-1.15203	-1.16043	-1.19293	1.16775
	Compare With gzip-9	4.62605	4.62605	4.62605	4.62605	4.62605	4.62605	4.62605	4.62605	4.62605	4.62605	4.62605	4.62605	4.62605	4.62605	4.6260	4.62605
	Improvement	0.042205	0.010987	0.068845	0.068768	0.11396	0.067233	0.124716	0.160282	038978	025345	.052267	.021013	.069664	.065973	.051922	.062806
	Reduce File Size Byte(C)	4279	2694	4450	2457	25589	22241	4723	8258	4347	2712	4502	2418	25785	22774	4861	8647
	Lib. File Size (L)	760	664	880	528	1088	1072	800	864	848	720	880	690	1072	1088	920	1016
	Total comp-ress file (T)	5039	3358	5330	2985	26677	23313	5523	9122	5195	3432	5382	3108	26857	23862	5781	9663
	Compression ratio	-1.08935	-1.23049	-1.12901	-1.25837	-0.81017	-0.78736	-1.03932	-0.88685	-1.15405	-1.27964	-1.1498	-1.3514	-0.8228	-0.829	-1.13458	-0.99875
	Compression Rate(bits/base)	4.17871	4.469764	4.258039	4.516739	3.62035	3.574723	4.078649	3.77371	4.30807	4.55928	4.29958	4.70285	3.64477	3.65890	4.26917	3.99751
	Compare With gzip-9	4.62605	4.62605	4.62605	4.62605	4.62605	4.62605	4.62605	4.62605	4.62605	4.62605	4.62605	4.62605	4.62605	4.62605	4.62605	4.62605
	Improvement	0.0967	0.035683	0.079552	0.023609	0.217399	0.227262	0.11833	0.184248	.06875	.01443	0.07087	-0.01660	.21211	8 .20896	.07714	.13586

Table I

Sequence Size	Sequence Type	Normal Sequence								Artificial Normal Sequence							
	Sequence Name	atatsgs	atefla23	atrdnaf	atrdnai	celk07e12	hsg6pdgen	mmzp3g	xlxfg512	XX1	XX2	XX3	XX4	XX5	XX6	XX7	XX8
	Base pair/ File size in byte	9647	6022	10014	5287	58949	52173	10833	19338	9647	6022	10014	5287	58949	52173	10833	19338
Substring Size-4	Reduce file Size byte (C)	5179	3226	5236	2701	30027	27971	5280	9152	5200	3222	5318	2841	31537	28021	5761	10216
	Lib file Size (L)	162	165	156	144	174	168	280	280	150	162	168	150	156	156	162	162
	Total Compress File size(T)	5341	3394	5392	2845	30201	28139	5469	9432	5359	3384	5486	2991	31713	28177	5923	10478
	Compression Ratio	-1.21457	-1.2544	-1.15378	-1.152449	-1.0493	-1.15736	-1.01939	-0.95098	-1.22204	-1.24776	-1.19133	-1.26293	-1.15189	-1.16027	-1.18702	-1.1673
	Compression Rate (bits/base)	4.42915	4.50880	4.30756	4.30489	4.098594	4.314722	4.03877	3.90195	4.44408	4.49552	4.38266	4.52582	4.30379	4.32055	4.37408	4.33468
	Compare With gzip-9	4.62605	4.62605	4.62605	4.62605	4.62605	4.62605	4.62605	4.62605	4.62605	4.62605	4.62605	4.62605	4.62605	4.62605	4.6260	4.62605
	Improvement	0.042205	0.010987	0.068845	0.068768	0.11396	0.067233	0.124716	0.160282	0.038978	0.025345	0.052267	0.021013	0.069604	0.065973	0.051922	0.062806
	Reduce File Size Byte(C)	4279	2694	4450	2457	25589	22241	4723	8258	4347	2712	4502	2418	25785	22774	4861	8647
	Lib. File Size (L)	760	664	880	528	1088	1072	800	864	848	720	880	690	1072	1088	920	1016
	Total compress file (T)	5039	3358	5330	2985	26677	23313	5523	9122	5195	3432	5382	3108	26857	23862	5781	9663
	Compression ratio	-1.08935	-1.23049	-1.12901	-1.25837	-0.81017	-0.78736	-1.03932	-0.88685	-1.15403	-1.27964	-1.1498	-1.3514	-0.8238	-0.829	-1.13458	-0.99875
	Compression Rate(bits/base)	4.17871	4.460976	4.258039	4.516739	3.62035	3.574723	4.078649	3.77371	4.30807	4.55928	4.29958	4.70285	3.64477	3.65890	4.26917	3.99751
	Compare With gzip-9	4.62605	4.62605	4.62605	4.62605	4.62605	4.62605	4.62605	4.62605	4.62605	4.62605	4.62605	4.62605	4.62605	4.62605	4.62605	4.62605
	Improvement	0.0967	0.035683	0.079552	0.023629	0.217399	0.227262	0.11833	0.184248	.06873	.01443	0.07057	-0.01660	.212118	.20906	.07714	.13586

Table II

Sequence Size	Sequence Type	Normal Sequence								Artificial Normal Sequence							
	Sequence Name	statsgs	atefla23	atrdnaf	atrdnai	celk07e12	hsg6pdgen	mmzp3g	xlxfg512	XX1	XX2	XX3	XX4	XX5	XX6	XX7	XX8
	Base pair/ File size in byte	9647	6022	10014	5287	58949	52173	10833	19338	9647	6022	10014	5287	58949	52173	10833	19338
Substring Size-4	Reduce file Size byte (C)	5165	3254	5248	2659	30029	27973	5203	9110	5211	3232	5320	2843	31559	28023	5777	10318
	Lib file Size (L)	162	162	168	144	180	174	280	280	150	162	168	150	156	156	162	162
	Total Compress File size(T)	5327	3416	5416	2803	30067	28061	5481	9388	5359	3384	5488	2993	31715	28179	5939	10480
	Compression Rate (bits/base)	4.430808	4.575224	4.307569	4.307925	4.098865	4.315029	4.049109	3.88458	-1.22287	-1.2544	-1.19213	-1.26442	-1.15203	-1.16043	-1.19293	1.16775
	Compare With gzip-9	4.62605	4.62605	4.62605	4.62605	4.62605	4.62605	4.62605	4.62605	4.62605	4.62605	4.62605	4.62605	4.62605	4.62605	4.6260	4.62605
	Improvement	0.042205	0.010987	0.068845	0.068768	0.11396	0.067233	0.124716	0.160282	0.038978	0.025345	0.052267	0.021013	0.069604	0.065973	0.051922	0.062806
	Reduce File Size Byte(C)	4279	2694	4450	2457	25589	22241	4723	8258	4347	2712	4502	2418	25785	22774	4861	8647
	Lib. File Size (L)	760	664	880	528	1088	1072	800	864	848	720	880	690	1072	1088	920	1016
	Total compress file (T)	5039	3358	5330	2985	26677	23313	5523	9122	5195	3432	5382	3108	26857	23862	5781	9663
	Compression ratio	-1.08935	-1.23049	-1.12901	-1.25837	-0.81017	-0.78736	-1.03932	-0.88685	-1.15403	-1.27964	-1.1498	-1.3514	-0.8238	-0.829	-1.13458	-0.99875
	Compression Rate(bits/base)	4.17871	4.460976	4.258039	4.516739	3.62035	3.574723	4.078649	3.77371	4.30807	4.55928	4.29958	4.70285	3.64477	3.65890	4.26917	3.99751
	Compare With gzip-9	4.62605	4.62605	4.62605	4.62605	4.62605	4.62605	4.62605	4.62605	4.62605	4.62605	4.62605	4.62605	4.62605	4.62605	4.62605	4.62605
	Improvement	0.0967	0.035683	0.079552	0.023629	0.217399	0.227262	0.11833	0.184248	.06873	.01443	0.07057	-0.01660	.212118	.20906	.07714	.13586

Table III

Experimental results show that, the normal sequence is highly compressible than their other orientation. 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 exact reverses, users can obtain original sequences in a time that cannot be felt. Additionally, our algorithm can be easily implemented.

From these experiments, we conclude that internal reverse matching pattern are the same in all type of sources and Look Up Table plays a key role in finding similarities or regularities in DNA sequences. Output file contains 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. This technique provides the high security to protect nucleotide sequence in a particular source. Here we can get better security than static Look Up Table. But experimental result showing no meaningful changes are found using other orientation taking as an input sequence. In case of artificial sequences the compression rate are almost same in all sequences. The

program was tested with Normal and Artificial DNA sequence and Compression ratio and Compression rate is tabulated in Table-I.

5 CONCLUSIONS

In this article, we discuss a new DNA compression algorithm whose key idea is internal reverse. This compression algorithm gives a good model for compressing DNA sequences that reveals the true characteristics of DNA sequences. The compression results of reverse DNA sequences also indicate that our method is more effective than many others. This method is able to detect more regularities in DNA sequences such as, mutation and crossover and achieve the best compression results by using this observation. This method fails to achieve higher compression ratio than other standard methods, but it has provided very high information security. Important observations are: Reverse substring length vary from 2 to 5 and no distinguishable match found in case the substring length becoming six or more.

The substring length three is highly reversible than substring length four or five. That is why substring length of three is highly compressible over substring length of four or five. Normal sequence is highly compressible than reverses, complement and reverse complement sequences. Cellular DNA sequences compression rate and compression ratio are different because each sequence come from different sources, where as artificial DNA sequences compression rate and compression ratio are same in all time in all data sets.

REFERENCES

[1] Chen, X., Kwong, S., and Li, M. (1999) "A compression algorithm for DNA sequences and its applications in genome comparison", Genome Informatics , 10, 52–61.

[2] Matsumoto, Toshiko., Sadakane , Kunihiko., Imai , Hiroshi. (2000) , "Biological Sequence Compression Algorithms", Genome Informatics, 11, 43–52.

[3] Grumbach, S. and Tahi, F. (1994) "A new challenge for compression algorithms: genetic sequences", Information Processing & Management, 30, 875–886.

[4] Chen, X., Kwong, S., and Li. M., "A compression algorithm for DNA sequences and its applications in genome comparison", Genome Informatics, 10:52–61, December 1999.

[5] Lanctot, J. K., Li, M., and Yang, E., (2000) "Estimating DNA sequence entropy". Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms, 409–418.

[6] M. Li., and P. Vitanyi., (1997)"An Introduction to Kolmogorov Complexity and Its Applications, 2nd ed. New York", Springer-Verlag.

[7] Curnow, R. and Kirkwood, T., "Statistical analysis of deoxyribonucleic acid sequence data :a review", J. Royal Statistical Society, 152:199-220,(1989).

[8] Grumbach, S. and Tahi, F., "A new challenge for compression algorithms", genetic sequences,J. Information Processing and Management, 30(6):875-866, 1994.

[9] Lanctot, K., Li, M., and Yang , E.H., "Estimating DNA sequence entropy" , to appear in SODA', 2000.

[10] Rivals, E., Delahaye, J.-P., Dauchet, M., and Delgrange, O., "A Guaranteed Compression Scheme for Repetitive DNA Sequences", LIFL Lille I University, technical report IT-285, 1995.

[11] Bell, T.C., Cleary, J.G., and Witten, I.H., "Text Compression", Prentice Hall, 1990.

[12] Hisahiko Sato, Takashi Yoshioka, Akihiko Konagaya, Tetsuro Toyoda, "DNA Data Compression in the Post Genome Era" Genome Informatics 12: 512–514 (2001).

[13] Matsumoto, T., Sadakane, K., and Imai, H., "Biological sequence compression algorithms", Genome Informatics, 11:43–52, 2000.

[14] Matsumoto, T., Sadakane, K., Imai, H., and Okazaki, T., "Can general-purpose compression schemes really compress DNA sequences?, Currents in Computational Molecular Biology", Universal Academy Press, 76–77, 2000.

[15] Chen, X., Kwong, S., and Li, M., "A compression algorithm for DNA sequences and its applicationsin genome comparison", Genome Informatics , 10:52–61, 1999.

[16] Ma, B., Tromp, J and Li, M. (2002) "PatternHunter – faster and more sensitive homology search", .Bioinformatics, 18, 440 – 445.1698.

[17] 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.

[18] Xin Chen, San Kwong and Mine Li, "A Compression Algorithms 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.

[19] T.Matsumoto, K.Sadakame and H.Imani, "Biological sequence compression algorithm", Genome Informatics 11:43-52 (2000).

[20] ASCII code. [Online]. Available: http://www.asciitable.com

[21] National Center for Biotechnology Information, http://www.ncbi.nlm.nih.gov