A Compression and Encryption Algorithms on DNA Sequences using R^2CP and Modified Huffman Technique

Md. Syed Mahamud Hossein
D.O.,R.O.,Kolaghat, D.V.E.T., West Bengal

ABSTRACT

A lossless compression algorithm, for genetic sequences, based on two phase, 1st phase- searching for exact R^2CP is reported. The compression results obtained in the algorithm show that the exact R^2CP are one of the main hidden regularities in DNA sequences. The proposed DNA sequence compression algorithm is based on R^2CP substring and creates online Library file acting as a Look Up Table. The R^2CP substring is replaced by corresponding ASCII character. Information security is the most challenging question to protect the data from piracy. This proposed method may protect the data from hackers. For better security purpose we have introduced a new security technique in 2nd phase that is selection encryption method. In this technique the data are encrypted either in the Look Up table or in compressed file or in both. It can also provide the data security, by using ASCII code and online library file acting as a signature. The size of library file is too small with respect to compressed file. Compressing the genome sequence will help to increase the effect of their uses. Speed of encryption and security levels are two important measurements for evaluating any encryption system. Selective encryption, where a part of message is encrypted keeping the remaining part unencrypted, can be a viable proposition for running encryption system in resource constraint. This algorithm is tested on benchmark DNA sequences. The running time of this algorithm is very few second and the complexity is $O(n^2)$. The algorithm can approach a compression rate of 3.447387 bit/base using 1st phase compression technique, again the output of the 1st phase compression are used in 2nd phase compression techniques, at the end ultimate the resultant compression rate of 2.01 bit/bases.

When a user searches for any sequence for an organism, a encrypted compressed sequence file can be sent from the data source to the user. The encrypted compressed file then can be decrypted & decompressed at the client end resulting in reduced transmission time over the Internet. A encrypted compression algorithm that provides a moderately high compression with encryption rate with minimal decryption with decompression time.

Keywords: Repeat, Reverse, Complement, Palindrome, Compression, Security

Abbreviation of R^2CP : Repeat, Reverse, Complement and Palindrome

1. INTRODUCTION

With more and more complete genomes of prokaryotes and eukaryotes becoming available and the completion of human genome project in the horizon, fundamental questions regarding the characteristics of these sequences arise. The compressibility of DNA sequences. Life represents order. It is not chaotic or random [1]. Thus, expect the DNA sequences that encode life as nonrandom. Naturally they should be very compressible. These are also strong biological evidences in supporting this claim: It is well-known that DNA sequences, especially in higher eukaryotes, contain many sub string of R^2CP . 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 [2, 3, 4, and 5]. The DNA sequences only consist of 4 nucleotide bases {a,t,g,c}, 8 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 8 bits per base, although all these compression software are universal compression algorithms. These software's are designed for text compression [6], while the regularities in DNA sequences are much subtler. It is our purpose to study such subtleties in DNA sequences. We will present a DNA compression algorithm, based on exact matching that gives the best compression results on standard benchmark DNA sequences. However searching for all exact R^2CP in a very long DNA sequences is not a trivial task. These algorithms achieves high speed, best compression ratio and runs significantly faster than any existing compression program for benchmark DNA sequences. Proposed algorithm consists three phases: i) finding all exact R^2CP and ii) encode exact R^2CP regions and non-match regions and iii) Encryption, compress file or in both. We have developed for fast and sensitive homology search [7], as our exact R^2CP search engine. Compression of DNA sequences. We will present a DNA compression algorithm, based on R^2CP substring and corresponding R^2CP substring is placed in the a library file, this R^2CP substring create a dynamic Look Up Table and place ASCII character in appropriate places on source file and that gives the best compression results on standard benchmark DNA sequences with fast and efficient manner. The compression takes fraction of second for execution better than other techniques. We will discuss details of the algorithm, provide exponential results and compare the result with the one most effective compression algorithm for DNA sequence.

Now a days information security is a most challenging question, the hackers hack the data and process, manipulate the information, as a result to break the genome phenotype. This compression method provides two tier security i) the data are compressed, generates two separate files individually and each file contains ASCII code of 256 different characters ii) Apply selective encryption on Library file or compress file or both.

Selective encryption is the process of selecting a part of a whole message, to be gone through the process of encryption, keeping the remaining portion of the message in the clear in such a way that the security is not compromised. In the selective encryption, only a fraction(r) of whole message or plaintext is selected for encryption and the remaining part is kept in the clear as shown in the figure 1.1. Selection of the r part is vital from the security point of view in case of selective

encryption. The criteria for selection of r vary according to the type of media. Intuitively, as r increases, the security level also increases at the cost of increased time of encryption.

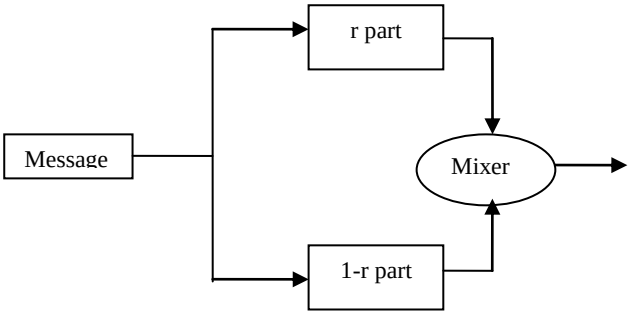


Figure 1

In case of selective encryption of compressed text, as compression follows the encryption process, as shown in Figure 1, the attacks based on statistical properties of the plain text will not be possible because of the reduction of redundancy due to process of compression. Further, in case of selective encryption of compressed text, the property of certain compression algorithms that the reconstruction information is concentrated in a small fraction of compressed text is utilized [8]. This selective encryption approach not only reduces the time complexity for encryption and decryption due to encryption of only the part of the compressed data where reconstruction information are mostly concentrated and but also it reduces the storage and communication cost.

This approach of selective encryption has got some advantage due to constraint of the network bandwidth before communicating and also it needs to be encrypted so to maintain confidentiality or to protect the digital rights. For ubiquitous access of digital contents, selective encryption is a viable proposition in the face of constraints of storage and computational capacity.

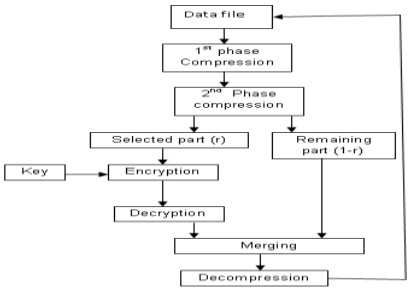


Figure 2

Though a lot of works have been done on selective encryption of images, videos, speech etc, not much work has been done on selective encryption of compressed text specially in Bioinformatics field. In this work, we wish to review the existing selective encryption algorithms and also study the effectiveness of selective encryption on compressed text generated through static Huffman encoding. The effectiveness of the algorithm will be measured by different parameters like Lavenstein Distance and relative frequency. The Lavenstein distance is a measure to find the dissimilarity between two strings proposed by. It is defined as the no number of substitution, insertion and deletion operations that are needed to transform the source string to the target string. The relative frequency parameter will indicate the status of redundancy of the encrypted text as defined by

$$\frac{\sum f_i * n_i}{\text{Total number of character}}$$

Where f_i is the frequency and n_i is the number of character having same frequency.

If relative frequency is low then redundancy become also low.

There is a similarity between the process of data compression and process of encryption. The goal for both the processes is to reduce the redundancy in the source message. According to Shannon [9], for perfect lossless compression algorithm, the average bit rate is equal to the source entropy.

Due to the combination of the process of compression and the process of encryption, two benefits are realized:

1. Conservation of storage space and communication bandwidth
2. Encryption cost is reduced.
3. The attacks on the basis of statistical property of the source bit stream are thwarted.

2.1 HUFFMAN CODING

Statistical codes represent data blocks of fixed length with variable-length code words. Huffman coding is one type of statistical code. This coding is also one type of entropy coding. entropy encoding is a lossless data compression scheme that is independent of the media's specific characteristics. entropy coding assigns codes to symbols so as to match code lengths with the probabilities of the symbols. Typically, these entropy encoders are used to compress data by replacing symbols represented by equal-length codes with symbols represented by codes where the length of each codeword is proportional to the negative logarithm (is $-\log_b P$, where b is the number of symbols used to make output codes and P is the probability of the input symbol) of the probability. Therefore, the most common symbols use the shortest codes.

The efficiency of a Huffman code depends on the frequency of occurrence of all distinct fixed length blocks in a set of data. The most frequently occurring blocks are encoded with short code words, whereas the less frequently occurring ones are encoded with large code words. In this way, the average codeword length is minimized. It is obvious however that, if all distinct blocks in a data set appear with the same (or nearly the same) frequency, then no compression can be achieved. Among all statistical codes, Huffman offer the best compression since they provably provide the shortest average codeword length. Another advantageous property of a Huffman code is that it is prefix free; i.e., no codeword is the prefix of another one. This makes the decoding process simple and easy to implement.

Let T be the fully specified test set. Let us also assume that if we partition the test vectors of T into blocks of length l, we get k distinct blocks $b1, b2, \ldots, bk$ with frequencies (probabilities) of occurrence $p1, p2, \ldots, pk$, respectively. The entropy of the test set is defined as

$$H(T) = -\sum_{i=1}^{k} P_i (\log_2 p_i)$$

and corresponds to the minimum average number of bits required for each codeword. The average codeword length of a Huffman code is closer to the aforementioned theoretical

entropy bound compared to any other statistical code. In practice, test sets have many don't care (*x*) bits. In a good encoding strategy, the don't cares must be assigned such that the entropy value *H*(*T*) is minimized. In other words, the assignment of the test set's *x* values should skew the occurrence frequencies of the distinct blocks as much as possible. We note that the inherent correlation of the test cubes of *T* (test vectors with *x* values) favors the targeted occurrence frequency skewing and, consequently, the use of statistical coding. To generate a Huffman code, we create a binary tree. A leaf node is generated for each distinct block *bi*, and a weight equal to the occurrence probability of block *bi* is associated with the corresponding node. The pair of nodes with the smallest weights is selected first, and a parent node is generated with weight equal to the sum of the weights of these two nodes. The previous step is repeated iteratively, selecting each time the node pair with the smallest sum of weights, until only a single node is left unselected, i.e., the root (we note that each node can be chosen only once). Starting from the root, we visit all the nodes once, and we assign to each left-child edge the logic 0 values and to each right-child edge the logic 1 value. The codeword of block *bi* is the sequence of the logic values of the edges belonging to the path from the root to the leaf node corresponding to *bi*. If *c*1, *c*2, . . . , *ck* are the codeword lengths of blocks *b*1, *b*2, . . . , *bk*, respectively, then the average codeword length is

$$C(T) = \sum_{i=1}^{k} P_i C_i$$

The size of a Huffman decoder depends on the number of distinct blocks that are encoded. Increased encoded-block volumes lead to big decoders due to the big size of the corresponding Huffman tree. For that reason, a selective Huffman approach was adopted in our project, according to which only the most frequently occurring blocks are encoded, whereas the rest are not.

Compression & selection encryption techniques for the general purpose of sequence data delivery to the client. Existing DNA search engines do not utilise DNA sequence compression algorithms & encryption for high security for client side decryption & decompression, i.e. where a encrypted compressed DNA sequence is decrypted & decompressed at the client end for the benefit of faster transmission & information security. Because most of the existing DNA sequence compression algorithms aim for higher compression ratios or pattern revealing, rather than client side & decryption decompression, their decompression times are longer than necessary information security. This makes these compression techniques unsuitable for the "on the fly" decompression. We use a encrypted compression technique designed for client side decrypted followed by decompression in order to achieve faster sequence secure data transmission to the client.

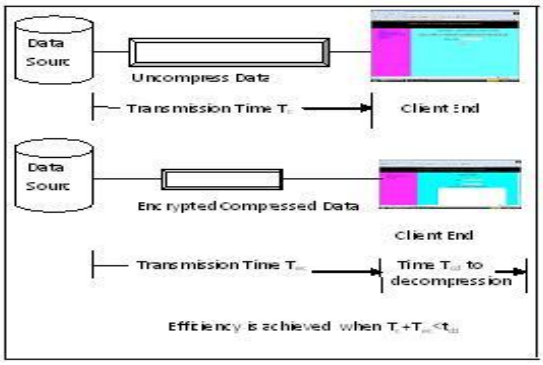


Fig-3

If encrypted compressed sequence data is sent from the data source to be decrypted decompressed at the client end and the decryption to decompression time along with the encrypted compressed file transmission time is less than the transmission time for uncompressed data transfer from the source to the client, then efficiency is achieved. Figure-2 illustrates the situation. Note that the sequence data should be kept pre-compressed within the data source.

A Sequence compression algorithm with reduced decompression time and moderately high compression rate is the preferred choice for efficient sequence data delivery with faster data transmission. As our target is to minimize decompression time and high information security, we use similar compression techniques to those used in [11], based on a "Two Pass" approach, meaning, that the file is compressed followed by encryption or decrypt followed by decompressed while reading it. Unlike "four pass" algorithms there is no need to re-read the input file. Our compression technique is essentially a symbol substitution compression scheme that encodes the sequence by replacing four consecutive nucleotide sequences with ASCI characters. Our technique to find the best solution for a client side decompression technique.

2. FLOW CHART:

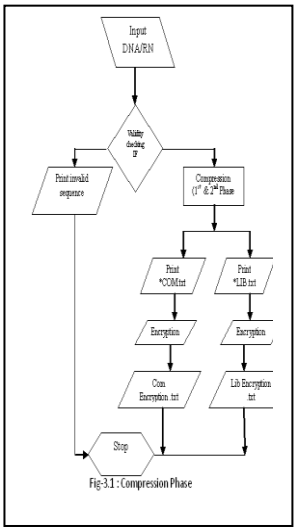


Fig-3.1 : Compression Phase

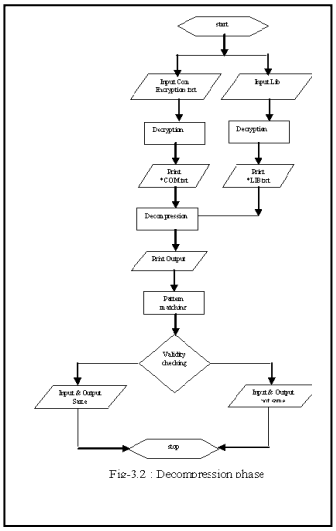
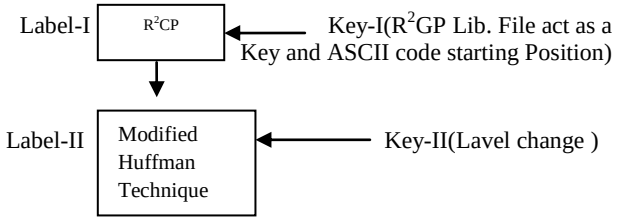


Fig-3.2 : Decompression phase

Two Tier Security Label



3. METHODS

3.1: File Format: We will begin discussing file type is text file (file extraction is dot text).

3.2 :**Generating the substring from input sequence use the techniques of as follows,** describe in BLAST algorithms[12]

a. Break the query sequence into words

b. Search for word matches (also called high –scoring pairs, or HPSs) in the database sequences

c. Extend the match until the local alignment score falls below a fixed threshold (the most recent version of BLAST allows gaps in the extended match)

3.3: searching for exact R^2CP subsequences

If not otherwise mentioned, we will use lower case letters u, v to denote finite strings over the alphabet {a, t, g, c}, |u| denotes the length of u, and the number of characters in u.u_i is the i^{th} character of u. $u_{i:\ j}$ is the substring of u from position I to position j. The first character of u is u_i. Thus $u=u_{1:|u|-1}$, where $u_{i:\ j}$ represents the original substring and |v| denotes the length of v, the number of characters in v. v_i is the i^{th} character of v. $v_{i:j}$ is the another substring of v from position i to position j. The first character of v is v_1. Thus $v=v_{1:|u|-1}$. $u_{i:j}$ match with $v_{i:j}$. The minimum difference between u-v is of substring length. The $v_{i:\ j}$ represents the R/R/C/P substring. The match found if $u_{i:\ j}= v_{i:\ j}$ and count exact maximum R^2CP of $u_{i:j}$. We use Є to denote empty string and Є =0.

Consider a finite sequence s over the DNA alphabet {a, t, g, c}. As exact R^2CP is a substring in s that can be transferred from another substring in s with edit operations (on repeat, reverse and complement, insertion). We encode these substrings only to match approximate maximum that provides profit on overall compression.

This method of compression is as below:

1. Run the program and output all exact R^2CP into a list s in the order of descending scores.
2. Extract a R^2CP r with highest score from list s, and then replace all r by corresponding ASCII code into another intermediate list o and place r in library file. Where r is R/R/C/P substring.
3. Process each R^2CP in s so that there's no overlap with the extracted R^2CP r.
4. Goto step 2 if the highest score of R^2CP in s is still higher than a pre-defined threshold; otherwise exits.

3.4 : Encoding R^2CP

An exact R^2CP can be presented as two kinds of triplets, first is (l,m,p), where l means the R^2CP substring length, m and p show the starting position of two substrings in a R^2CP respectively. Second: replace this operation as expressed (r; p; char), which means replacing the exact R/R/C/P substring at position p by ASCII character char.

In order to recover an exact R^2CP correctly the following information must be encoded in the output data stream:

Encoding Analysis

So, we can write s=atggtagtaatgtacatg……n n>0 and l<=i<=i-L+1

Consider the sequence is defined by s, the match substring is stored in u[m] and all matches R/R/C/P substring is stored in v[p].

After breaking the sequence(s) into substring of three bases we can get the result as below. So we can get u[m]-u[l]…….u [n*] and R/R/C/P substring are v[p] =v[l]……..v [n-l+1] l<=p<=n-l+1

If the number of substring is u[m], total number of subsequences are generated by (n-2*l+i) and R/R/C/P substring are v[p], total match R/R/C/P substring are (n-l+1).

As per above example u[m] ->u[l] =atg and so on.

And v[p] ->v[l] =gta and so on.

This substring method is required to reduce the complexity of the programmer's execution.

3.5: Each substring is matched with all other substring for finding the exact maximum R^2CP substring.

Match condition occur if u[m] =v[p] p=l+1,

Step-1

S[l] match { R^2CP } with u[p] to v [n-1+l] and count u [1]

{As for example, u [1] =atg, where substring size=3

And v [4] =gta, v [5] =tag……v [19] =atg

So, u [1] substring repeats at 3 places, reverses at complement.

Then m and p is incremented by one}

Step-2

Match u [2] match with u[p] to v [n-l+1] and count u [2]

{As for example u [2] =tgg and v [5] =tag, v [6] =agt

So, v [2] substring R^2CP at one place

U [2] substring repeats at 3 places, reverses at complement

Then m and p increments by one}

Step-3

This method will continue till u [n-l+1]

So, u [n-2*l+1] matches with u[p] to v [n-2*l+1] and count u [n-2*l+1]

So, u [n-2*l+1] R^2CP at only one place if match occur.

U [4] substrings repeats at 3 palaces and reverses at complement.

Step-4

Store all R^2CP count in decreasing order and find all exact maximum R/R/C/P count.

Step-5

Replace exact maximum R/R/C/P substring by corresponding ASCII code and place R/R/C/P substring in library file, and create an on line look up table.

Step-6

Repeat: R/R/C/P Step-1 to step-5 excluding ASCII code.

Step-7

If the highest score of R/R/C/P in 's' is still higher than a pre-defined threshold; otherwise exit.

As per above example: Now we find maximum repeat, reverse and complement's probability. These substrings are replaced first. Here we can get u [2] = (atg) substrings are repeated, reversed and complement 3 times in the sequence.

These substrings are placed in Look Up Table, corresponding ASCII characters replaces all the R/R/C/P substrings by ASCII characters. A library file creates the online Look Up Table.

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

The encoding procedure follows this rule and produces compressed output file[m], that matches { R^2CP } with v[p] to v [n-1+l], places.ASCII characters in the output files i^{th} position. In each match the value of m is incremented by: m= number of unmatched character+ (number of substring match* substring length+1).

Otherwise u[m] =! v [p] to v[n+1] place base pair in output files i^{th} position. If unmatch occurs, the value of m and p is incremented by one.

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

At the end we can get the compressed file, corresponding input sequence. So, o=!""!tac……..n_1,where n_1 is the length of output file. Output file size is n_1 byte. And library file:!atg"gta.

3.6: Decoding

Decoding time first requires online Library file, which was created at the time of encoding the input file.

On this particular value, the encoded input string is decoded and produces the original files output.

O=!""!tac!.....n_1, where n_1 is the length of the output string ($n>n_1$).

At the time of decoding, each ASCII character is replaced by corresponding base pair i.e. O [M] =L [K].

In case of repeat L [K] =

In case of reverse L [K] =

In case of complement 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 [33] to L [256] with O [M], place ASCII equivalent substring in i^{th} places in the output file. The value of m is incremented by one. If unmatch found in between L[33] to L[256] with O[M], place base pair in i^{th} position in output file./ The value of M is incremented by one. This process will continue until $M=n_1$ position will appear.

The decoding processes mention this rule and produce the original output string.

Match is found if o[m] =L [33] to L [256] place ASCII character equivalent substring in i^{th} position. If match found, the value of m is incremented by one.

Otherwise o[m] =! l [33] to L[256]place base pair in i^{th} position in output file. If mismatch occurs, the value of m 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 "atggtagtaatgtacatg……..n" has been read, in the destination file, we represent them as "!""!tac!.....n1" obviously the destination file is case sensitive.

We know that each character requires 1 byte (8 bit) for storing. In the above example, string length= 18, that means 18 byte is required for storing this string. After encoding on the basis of R^2CP techniques of 3 substring length, reduced string length is 8, requires 8 byte for storing this string.

3.7 Information security

This technique can provide two phase information securities.

In phase one, the input sequence contain 4 bases (a, t, g, c), after compression the file size is compressed as well as file contain is converted into 256 characters including a,t,g &c i.e. one substring contains 3 characters, is replaced by single ASCII code, so the output file is information secure than input file. This technique can provide a information security.

3.7.1: METHODOLOGY OF EXPERIMENTS PERFORMED

We have conducted our experiments in normal text files of different sizes and on the basis of the statistical property generated by Huffman tree for each text files. Since our objective is to find out the selective portion i.e. R part (discussed previously) from the text message we made swapping of the branches in the Huffman tree on at a particular level on the basic of a key and decode the encoded symbols using the modified Huffman tree which are specified in scheme I and II. In scheme-I we apply swapping method on two nodes at specified level on Huffman tree, and in scheme-II we perform swapping method between two specified nodes at different level on Huffman tree. When we generate the Huffman tree using the statistical property of symbols, first we consider each character of input text message as a symbol and later each word as a symbol.

SCHEME-I: SWAPPING NODES AT SPECIFIED LEVEL

Our objective is to find out the selective portion i.e. R part (discussed previously) from the text message. We made swapping of the branches in the Huffman tree on at a particular level on the basis of a key and decode the encoded symbols using the modified Huffman tree. Now for selecting the R part, my first experiment is swapping two nodes at specified levels. Here I exchange left most node with right most node at specified level. So only those nodes, which are changed their positions after swapping, are affects and also corresponding codes are also altered. Remaining other nodes is kept unchanged. Hence selective bits are altered.

Let illustrate this with an example, Fig.4.2 is the original tree and Huffman codes of W=00, X=01, Y=10, Z=11.

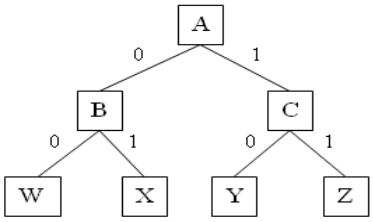


Figure 4

Suppose we apply swapping at level 1 then we find out A is single node i.e. root node at level 0 i.e. root node at level 0. Then we interchange the position of left and right child node with their sub tree as shown in fig.4.3. So corresponding code of W, X, Y, Z are totally changed. W=10, X=11, Y=00, Z=01. So if the original text is "WWXYXZ" then it will be encrypted "101011001101". If we decode it without change the level the text will be "YYZWZX". D_{SID} in this case is 6.

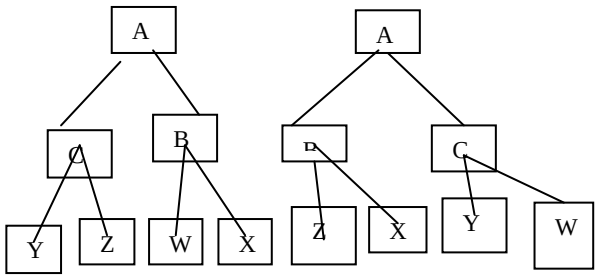


Figure 5

Figure 6

Character	Before Encrypt	After Encrypt	
		Swapping at Level 1	Swapping at Level 2
W	00	10	11
X	01	11	01
Y	10	00	10
Z	11	01	00

Table- 1

Now we apply swapping at level 2(here B, C) then we interchange the position of left child of B and right child of C and with their sub tree as shown in fig.4.4. In this case corresponding codes are selectively changed. i.e. since in fig.4.4 we see on W and Z are interchange their positions so code of Z and W are only changed, other should be unchanged according to table 4.1. So if the original text is "WWXYXZ" then it will be encrypted "111101100100". If we decode it without change the level the text will be "ZZXYXW". D_{SID} in this case is 3.

The results diverged from our expectations in some simple cases due to the complexities in the alignments of characters when calculating D_{SID}. In our experiment, when we measure Lavenstein distance and effect ness on actual text we face some minor problems.

Suppose we compress a text file-applying node swapping method at particular level. But if we measure effect ness on actual text by decoding the encoded text without any swapping method apply, then in some cases all codes may not be retrieved, for e.g. suppose frequency of A=2, B=1 and C=1 then tree will be generate like

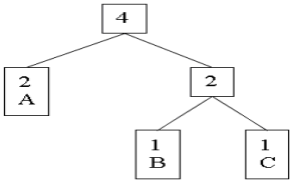


Figure 7

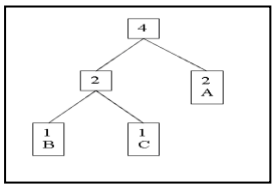


Figure 8

and their corresponding codeword A=0, B=10, C=11.

And suppose the string 'AABBC' would be encoded as 00101011.

Now if we apply swapping method at level 0 then the tree will generate like fig. 4.6

And their corresponding code A=1, B=00, C=01.

And same string then encoded as 11000001.

Now if we measure Lavenstein distance & % of effect ness on actual text then we must decode without apply swapping method at level 0. Then that encoded string is decoded using original tree (fig.4.5).

1	1	0	0	0	0	0	1
C	C	A	A	A	A	A	-

Table-2

The last 1 is left over because there is no such code of only 1. So for measurement purpose the size of original text is altered in these cases.

SCHEME-II-SWAPPING BETWEEN TWO SPECIFIED NODES AT DIFFERENT LEVEL

In my second experiment we get another approach for selecting the r part (discussed previously). In this approach swapping can be perform at any specified two nodes. By this approach we can interchange any two nodes with its subtree of the Huffman tree at any level. Hence this scheme has flexibility to modify Huffman tree and also use more than one key so it obviously increase security concern. In this scheme we need to specify two level values of two nodes and two binary values. Number of binary digit must be same with level value with respect to nodes. If we consider above specified two values as a key then security concern is improved than before experiment. E.g. Fig.4.7 is the original tree and Huffman codes of W=00, X=01, Y=10, Z=11.

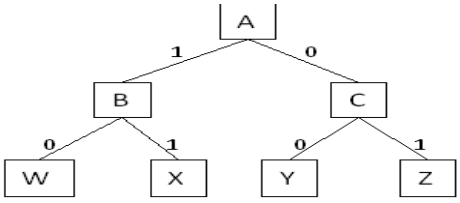


Figure 9

Suppose I want to swap between two nodes B and Z then we need to specify first, level number, in this case 1 and binary digit 0 for B node and level number for Z, in this case 2 and binary digit 11. Then new tree will generate like

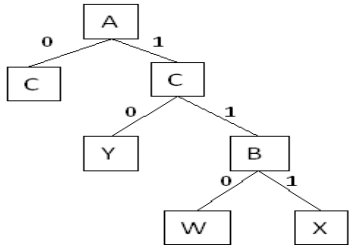


Figure-10

After interchanging the position of left and right child node with their sub tree as shown in fig.4.8 so corresponding codes of W, X, Y, Z are totally changed according to table 4.2. So if the original text is "WWXYXZ" then it will be encrypted "110110111101110". If we decode it without change the level the text will be decrypt like "ZXYZZXZ". D_{SID} in this case is 6.

Character	Before Encrypt	After Encrypt	
		Swapping Between B and Z	Swapping Between C and X
W	00	110	00
X	01	111	1
Y	10	10	010
Z	11	0	011

Table-3

Work already carried out : So many biological compression algorithm is available in market as in paper[100] where showing that Huffman's code also fails badly on DNA sequences both in the static and adaptive model, because there are only four kind symbols in DNA sequences and the probabilities of occurrence of the symbols are not very different. Here this two phase technique solved this problem because after the 1st phase compression we get the 252 ASCII characters' with nucleotide base pair a, t,g & c.

4. We have developed the following algorithms:

4.1 : Compression algorithm

Step 1: Here we use three text file. First is for take input of Genome sequence i.e, any combination of {a,t,g,c}. Second is for Dynamic look up table. And last one for out put.

Step 2: Store the input form the first text file to a buffer, say s.

Step 3: We have to check whether the first input sequence is {a,t,g,c} or not. If true do step 4 to step8 else increment the position by one.

Step 4: We have to match the whole input sequence according to the first four taking sequence.

Step 5: If the number of matching sequence foung greater than one then do step 6 to step 8 else increment the position by one.

Step 6: Write the ascii character with its corresponding matching sequence into the dynamic look up table.

Step 7: Replace the sequences with corresponding ascii character in the input string where the matching sequences are found.

Step 8: Increment the value of the ascii counter by one.

Step 9: Now we have to write the input buffer into the output file. After doing those steps successfully the input buffer will be the compressed genome sequence.

Step 11: Stop.

4.2 : Decompression algorithm

Step 1 : Declare three FILE pointer fp, fp1, fs and five character variable say ch1,ch2,ch3,ch4,ch5.

Step 2 : fp is required to point the encrypted file & retrive the encrypted genom sequece.

Step 3 : fs is required to point lookup table & retrive the genom sequeuence and croessponding ASCII character.

Step 4 : fp1 required to point the decripted file & used to store the decrypted genom sequence.

Step 5 : Store the encryped genom sequence in a temporary buffer say s.

Step 6 : Determine the number of neucleotide exist in the encrypted file (say len).

Step 7 : Initialize a counter (say i) is equal to 0 (zero). Repeat step 8 to step 15 until i lessthan number of neucleotide

exist in the encrypted file i.e, (i<len).

Step 8 : Seek the FILE pointer fs to the first neucleotide in the lookup table by rewind(fs).

Step 9 : Do step 10 to step 14 while(!feof(fs)).

Step 10: Initielize the following :

ch1=fgetc(fs) i.e, ch1 holds the first neucleotide present in the lookup table.

ch2=fgetc(fs) i.e, ch2 holds the second neucleotide present in the lookup table.

ch3=fgetc(fs) i.e, ch3 holds the thired neucleotide present in the lookup table.

ch4=fgetc(fs) i.e, ch4 holds the fourth neucleotide present in the lookup table.

ch5=fgetc(fs) i.e, ch1 holds the first neucleotide present in the lookup table.

Step 11: If s[i]=ch5 then print ch1,ch2,ch3,ch4 into the decrypted output file.

Step 12: else if(s[i]=='a' || s[i]=='t' || s[i]=='g' || s[i]=='c') then print s[i] into the decrypted output file.

Step 13: else continue.

Step 14: end if.

Step 15: end for.

Step 16: The number of neucleotide present in the encrypted file is equal to len.

Step 17: Calculate the total estimated time to decompress the encrypted file.

Step 18: Check that the decompression is lossless. If true then decompression is successful.

Step 19: Stop.

4.3 Proposed Algorithm for Scheme-I:

This algorithm recursively find a weighted binary tree with n given weights w_1, w_2, ….w_n. (Here weights mean frequency of n characters in text). LEVEL is the input where the tree is altered.

1.Arrange the weights in increasing weights.

2.Construct two leaf vertices with minimum weights, say w_i and w_j in the given weight sequence and parent vertex of weight $w_i + w_j$.

3.Rearrange remaining weights (excluding w_i and w_j but including parent vertex of weight $w_i + w_j$) in increasing order.

4.Repeat step 2 until no weight remains.

5.Find out left most node and right most node at specified LEVEL and interchange their position with respect to their parent node.

6.To find out code for each given weights (i.e. frequency of characters) traversing tree from root assign 0 when traverse left of each node & 1 when traverse right of each node.

4.4 Proposed Algorithm for scheme-II:

This algorithm recursively find a weighted binary tree with n given weights w_1, w_2, ….w_n. (Here weights mean frequency of n characters in text). LEVEL is the input where the tree is altered.

1.Arrange the weights in increasing weights.

2.Construct two leaf vertices with minimum weights, say w_i and w_j in the given weight sequence and parent vertex of weight $w_i + w_j$.

3.Rearrange remaining weights (excluding w_i and w_j but including parent vertex of weight $w_i + w_j$) in increasing order.

4.Repeat step 2 until no weight remains.

5.Find out two nodes at specified LEVEL by binary digits and interchange their position with respect to their parent node.

6.To find out code for each given weights (i.e. frequency of characters) traversing tree from root assign 0 when traverse left of each node & 1 when traverse right of each node.

4.5: Algorithm for file mapping:

Matching Percentage (String_1,String_2)

The function takes two strings as argument and returns the percentage of matching. Here first we calculate the number of errors i.e. miss match in between these two strings. After that we can calculate percentage by this following formula:

Percentage= [(Frame_size-Error_no)/ (Frame_size)]*100

Here Frame_size means the length of the String.

ALGORITHM:

1.Frame_size = LENGTH(String_1);

2.Repeat step 3 to 5 while Strng_1 is NULL;

3.Index=MISMATCH-INDEX(String_1,String_2);

4.If INDEX>Length(String_1)-1 then goto step 6;

5.If Index= Length(String_1)-1

Then String_1= NULL.

Else

String_1=SUBSTRING (String_1, (Index+1));

String_2=SUBSTRING (String_2, (Index+1));

6.Error _no=Error_no+1;

7.Percentage=((Frame_size-Error_no)/Frame_size)*100;

8.Return Percentage.

5. EXPERIMENTAL RESULTS

We tested R^2CP techniques on standard benchmark data, used in [13].For testing purpose we use two types of data, they comes under different sources.

These tests are performed on a computer whose CPU is Intel P-IV 3.0 GHz core 2 duo (1024FSB), Intel 946 original mother board, 1GB DDR2 Hynix, 160GB SATA HDD Segate.Since the program to implement the technique have been written originally in the C++ language[14-15],(Windows Xp platform and TC compiler) it is possible to run in other microcomputers with small chances (depending on the platform and compiler used) . The program run on the IBM personal computer requires 512K, without additional hardware except for disk drives and printer.

The definition of the compression ratio is [16]; 1-(|O|/2|I|), where |I| is the 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|/|I|), where |I| is the number of bases in the input DNA sequence and |O| is the length (number of bits) of the output sequence. The compression ratio and rate presented in tables Tadle-4 to Table-5.

Sequence	Base pair	File Size bits	Using R^2CP algorithm.				Using R^2CPHUFF algorithm.			Improvement over R^2CP
			Reduce file size bits	Library File size	Compression ratio	Compression rate(bits /base)	Reduce file size	Compression ratio	Compression rate(bits /base)	
MTPACGA	100314	100314	44982	133	-0.7936	3.5873	27657	-0.102817	2.205634	36%
MPOMTCG	186608	186608	83942	129	-0.7993	3.5986	53029	-0.136693	2.273386	
CHNTXX	155844	155844	70204	133	-0.8019	3.6038	43844	-0.12533	2.250661	
CHMPXX	121024	121024	53842	129	-0.7795	3.5591	32835	-0.085239	2.170479	
HUMGHCSA	66495	66495	30049	129	-0.8076	3.6152	19249	-0.157922	2.315843	
HUMHBB	73308	73308	33154	129	-0.809	3.618	21117	-0.152234	2.304469	
HUMHDABCD	58864	58864	26366	129	-0.7917	3.5833	17149	-0.16533	2.330661	
HUMDYSTROP	38770	38770	17472	129	-0.8026	3.6053	11544	-0.191024	2.382048	
HUMHPRTB	56737	56737	25821	133	-0.8204	3.6408	16554	-0.167069	2.334138	
VACCG	191737	191737	85921	129	-0.7925	3.585	52796	-0.101425	2.202851	
HEHCMVCG	229354	229354	102656	129	-0.7904	3.5807	64768	-0.129573	2.259145	
Average						3.5979			2.27484	

Table-4

Sequence	Base pair	File Size	Using R^2CP algorithm				Using R^2CPHUFF algorithm			Improvement over R^2CP
			Reduce file size	Library File size	Compression ratio	Compression rate(bits /base)	Reduce file size byte	Compression ratio	Compression rate(bits /base)	
atatsgs	9647	9647	4313	125	-.7883	3.5767	3626	-0.503473	3.006945	21%
atef1a23	6022	6022	2738	105	-0.8186	3.6373	2677	-0.778147	3.556294	
atrdnaf	10014	10014	4464	129	-0.7831	3.5662	3769	-0.505492	3.010985	
atrdnai	5287	5287	2319	35	-0.75449	3.509	2433	-0.840741	3.681483	
celk07e12	58949	58949	26583	133	-0.8038	3.6076	16864	-0.144311	2.288622	
hsg6pdgen	52173	52173	23399	129	-0.7939	3.5879	15282	-0.171641	2.343281	
mmzp3g	10833	10833	4869	117	-0.7978	3.5957	2952	-0.090003	2.180006	
xlxfg512	19338	19338	8658	129	-0.79088	3.5818	6186	-0.279553	2.559106	
Average						3.5827			2.82834	

Table-5

The results from Table 4 & 5 show our algorithms to be the best solution for client side decryption -decompression with the shortest and linearly increasing decompression time. However, our algorithms doesn't compress sequences as much as others for many of the cases in the compression ratio table of [17] but it provide high information security. This is because our algorithms uses 2 bits to represent one nucleotide

In order to compare the overall performance, we conducted further studies involving sending actual sequence files of varying sizes (without compression) to measure the calculated time (T_c) needed for the transmission from the source to the destination. Then we compressed those files using both compression & encryption algorithms. The total time T, defined as the sum of the encryption compressed file transmission time (T_{ec}) plus the client side decompression time (T_{dd}), is measured by both these methods.

6. RESULT DISCUSSION:

The result show that compression ratio are vary from each other due to the data set are come into different sources. 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 R^2CP, users can obtain original sequences in a time that can't be felt. Additionally, our algorithm can be easily implemented.

From these experiments, we conclude that internal R^2CP matching patter are same in all type of sources and Library file plays a key role in finding similarities or regularities in DNA sequences. Output file contain ASCII character with unmatched a, u, g and c so, it can provide information security which is very important for data protection over transmission point of view. This techniques provide the high security to protect nucleotide sequence in a particular source using modified Huffman Techniques.

The ratio of decompression time to original transmission time of the uncompressed sequence file (T_{dd} / T_c), reduces with increasing file size. This means our client side decryption decompression technique with our algorithm is a better choice for larger sequence files. Our client side decryption decompression technique can be implemented by a genome search agent and decryption decompression time can be estimated by two empirical equations according to our experiments.

Our algorithms combines moderate encryption compression with reduced decryption decompression time to achieve the best performance for client side sequence delivery compared with existing techniques. Its linearity in decompression time and close linearity in compression time make it an effective compression tool for commercial usage. Given, for a particular connection speed, the efficiency achieved using our algorithm, this compression technique is recommended for transmission of queried sequence files.

7. CONCLUSION:

Discuss a new DNA compression & security algorithm whose key idea is internal R^2CP. This compression algorithm gives a good model for compressing DNA sequences that reveals the true characteristics of DNA sequences including data security. 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 is fails to achieve higher compression ratio than others standard method, but it has provide very high information security.

Important observation are :

a)R^2CP substring length vary from 2 to 5 and no match found in case the substring length becoming six or more.

b)The substring length is three of highly repeated substring than substring length of four and five. That is why substring length of three is highly compressible over substring length of four and five.

c)This algorithm provide the better data security than other methods. If we use security directly on the cellular DNA sequence, we are getting very low label security because DNA sequence contain only four bases, anyone can hack the data by trial error methods where as our result show that after compression it has created four separate file first one is compress data contain 256 (ASCII) different characters, so it provide strong security label second file is library life, which is also contains more than four characters. At the time of transmission if two files are transmit one by one it is very hard to hack the data, these techniques has also provide data secure. Also if any one can apply simple Huffman techniques they fail to achieve the original DNA sequences without node label value.

In this work we have performed computational experiments to selectively encrypt the compressed text of different sizes generated through static Huffman encoding technique and compare the effectiveness in terms of dissimilarity from the original file if one has to decrypt without the key and the resistance of the cipher text from the attacks based on statistical property of the plain text. We have used two different schemes; in scheme-I swapping of nodes is done at specified level based on key and in scheme-II swapping is done between two specified nodes at different levels. We have found from our experiments, the effectiveness of the encryption system increases as the level at which swapping is done, increases. We have achieved in the both the scheme with x % encryption can achieve.

This approach has a good scope as a selective encryption scheme because of the fact that in a text of any language the articles, verbs, and prepositions have a higher frequency compared to the other words relevant to the core content of the

text. The problem small key space has to be sorted out to effectively apply this encryption system in real world..

In case of word consideration word's frequency are high but other word have very lower frequency. These lower frequency words are representing by higher bit. So percentage of compression is decrease. In terms of security word encryption is more effective than character encryption. In case of character encryption, we know there is only 256 characters are available and since workspace is short. So here is a possibility to break the security. But in case of word encryption, numbers of distinguishable words are huge, not known by all, so that workspace is also increased and breaking the security is not possible.

8. FUTURE WORK:

We try to reduce the time complexity and compression rate. Also we are try to apply another security method for getting better security.

Here in this work, we have taken into consideration the statistical property of a character or a word while doing compression. Instead, one can consider the statistical property of any number characters or bits, the number of bits may be provided by the user depending on the application or may be chosen automatically on the basis of entropy. In that case this encryption technique may be extended to any type of media. The effectiveness of selective encryption may be studied for the other statistical compression algorithms available.

9. ACKNOWLEDGEMENT

Above all, the authors are grateful to all our colleagues for their valuable suggestion, moral support interest and constructive criticism of this study. We would also like to thank our PCs.

10. REFERENCES

[1] M. Li and P. Vitányi, An Introduction to Kolmogorov Complexity and Its Applications, 2nd ed. New York: Springer-Verlag, 1997.

[2] Curnow, R. and Kirkwood, T., Statistical analysis of deoxyribonucleic acid sequence data { a review, J. Royal Statistical Society, 152:199{220, 1989.

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

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

[5] 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.

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

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

[8] H. Cheng and X. Li, "Partial Encryption of Compressed Images and Video," IEEE Transactions on Signal Processing, 48(8), 2000, pp. 2439-2451.

[9] C. E. Shannon, "Communication theory of secrecy systems," *Bell Systems Technical Journal*, v. 28, October 1949, pp. 656-715.

[10] D. A. Huffman, "A method for the construction of minimum-redundancy codes,"Proc. IRE, vol. 40, pp. 1098-1101,1952.

[11] Chen, L., Lu, S. and Ram J. 2004. "Compressed Pattern Matching in DNA Sequences". Proceedings of the 2004 IEEE Computational Systems Bioinformatics Conference (CSB 2004)

[12] S.F. Altchul, W. Gish, W.Miller,E.W. Myers, and D.J.Lipman,1990, A. Basic Local Alignment search tool, J.Mol. Biol. 215 :403-410.

[13] 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.

[14] E. Balagurusamy, Introduction to Computing. McGraw-Hill,1998

[15] K.R. Venugopal & S.R. Prasad, Mastering C. McGraw-Hill,1998

[16] 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.

[17] Chen, X., Li, M., Ma, B. and J. Tromp. 2002. "DNACompress: fast and effective DNA sequence compression". Bioinformatics. 18: 1696-1698.

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

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