IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

Fast and efficient two compression Programmes for DNA Sequences

Syed Mahamud Hossein

Department of Computer Science & Technology

I.C.V.Polytechnic

Abstract

As more and more DNA sequences are becoming available day by day, storing and transmitting them may require a huge amount of space. This paper introduces a new DNA sequence compression programme which is based on Huffman Technique of compression routine. DNA and RNA sequences can be considered as tests over a four letter alphabet, namely {a,t,g and c} (note that t is replaced with u in the case of the RNA). For complete genomes, these texts can be very long. 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[1]. Huffman coding is known to compress not very efficiently [2].This program is developed mainly to merge with dynamic Look up Table method, where sixty eight kind symbols are present. This merge techniques are developed to increase the compression rate/bases. This algorithm can approach a compression rate of 2.1 bits /base and even lower. This compression rate is defined as the compressed file size divided by base number[3]. This algorithm can approach a compression ratio is better than other method. The definition of the compression ratio is the same as in [4] 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. Compression of these genome sequences will help us increase the efficiency of their use. We use Huffman coding to encode the nucleotide genome sequence , a greedy algorithm that constructs an optimal prefix code called a Huffman code. The algorithm builds the tree T corresponding to the optimal code in a bottom-up manner. It begins with a set of |c| leaves and perform |c|-1 "merging" operations to create the final tree. The total running time of Huffman on the set of n characters is O(n log n). We tested the program on standard benchmark data used in[4]. The greatest advantage of this progrmme is fast execution, small memory occupation and easy implementation. Since this program have been written both in the C language, (Windows platform, and TC compiler) and MATLAB, 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, without additional hardware except for disk drives and printer. The execution is quite fast, all the operations are carried out in fraction of seconds, depending on the required task and on the sequence length.

Keywords : Huffman algorithm, Biology and genetics, Data Compression.

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 [5]. 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 [6-12]. 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 [6], 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. Some experiments indicate that the compression ratio is 2.6 bits/base.

2. METHOD

The first optimal code was developed by David Huffman. Before suggesting an algorithm for generating an optimal code, he pointed out that an optimal code has some characteristics which are summarized in the following theorem that uses a source alphabet S = {x_1……x_n), a set of associated probabilities p = {^{p}I……^{p}n} and code-words {c_1….. c_n} with corresponding lengths {l_1,. . ., ln}.

HuffmanAlgorithm( )

for each letter create a tree with a single root node and order all trees according to the probability of letter occurrence;
while more than one tree is left
take the two trees t_l, t_2 with the lowest probabilities p_1,p_3 and create a tree with probability in its root equal to $p_I + p_2$ and with t_l and t_2 as its subtrees; associate 0 with each left branch and 1 with each right branch;
create a unique codeword for each letter by traversing the tree from the root to
the leaf containing the probability corresponding to this letter and putting all encountered 0s and Is together;
The resulting tree has a probability of 1 in its root.
Huffman saw the structure resulting from application of his algorithm as a net of tributary rivers eventually flowing into a large river. He thought associating Is and Os with branches was analogous to "the placing of signs by a water-borne insect at each of these junctions as he journeys downstream" with right turn posts (marked with I) and left turn posts (marked with 0), which would allow the insect to return back to the starting point.

It should be noted that the algorithm is not deterministic in the sense of producing a unique tree because, for trees with equal probabilities in the roots, the algorithm does not prescribe their positions with respect to each other either at the beginning or during execution. If t_1 with probability P_1 is in the sequence of trees and the new tree t_2 is created with $p_1 = p_2$, should t_2 be positioned to the left of t_l or to the right? Also, if there are three trees t_1,t_2 and t_3 with the same lowest probability in the entire sequence, which two trees should be chosen to create a new tree? There are three possibilities for choosing two trees. As a result, different trees can be obtained depending on where the trees with equal probabilities are placed in the sequence with respect to each other. Interestingly, however, regardless of the shape of the tree, the average length of codeword remains the same.

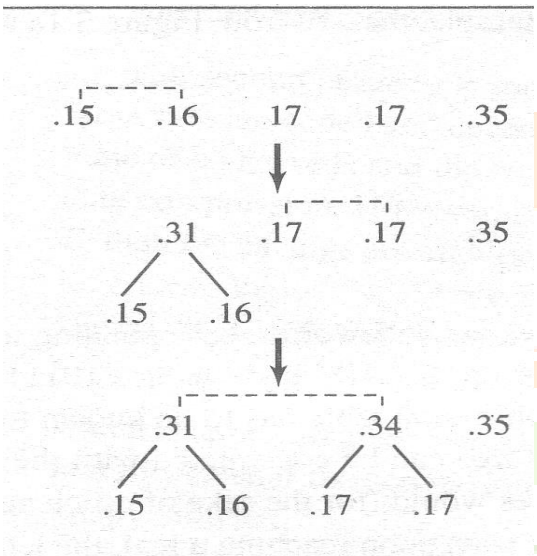

Fig.-A

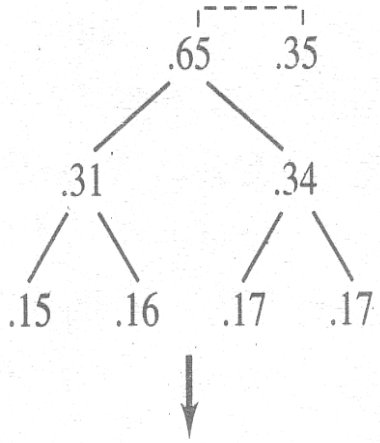

Fig.-B

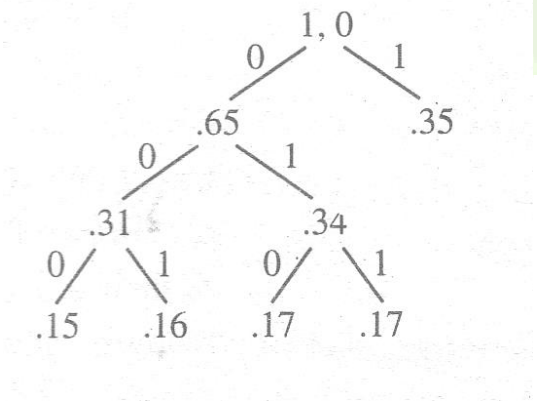

Fig.-C

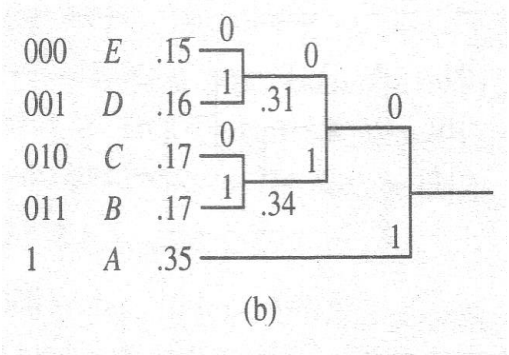

Fig.-D

Above Figure contain five letters A,B,C,D and E with probabilities .35, .17,.17,.16 and .15 , shows step-by step how a Huffman tree is generated

```
Data Structure used: Priority queue = Q
Huffman I
n = |c|
Q = c
for  i =1  to   n-1
   do   z = Allocate-Node ()
          x = left[z] = EXTRACT_MIN(Q)
          y = right[z] = EXTRACT_MIN(Q)
         f[z] = f[x] + f[y]
         INSERT (Q, z)
return EXTRACT_MIN(Q)
Pseudo code  for TC compiler: -
      (a)   Encoding  -

Procedure :
Var
I , r=0,j,sub,sos :Long Integer ;
S[4],x[100],c[100] :Long Integer;
Ch:  Character
a[100],b[100],infilename[20],outfilename[20] :
Character
fp,fpt :File pointer
time,time1:structure
begin:
display”Enter file name”;
accept(infilename);
   gettime(&t);
display "The current time is:
%2d:%02d:%02d.%02d\n”,
   t.ti_hour, t.ti_min, t.ti_sec, t.ti_hund;
   fp:=fopen (infilename,"r");
if fp=NULL then
begin:
display "unable to open file";
exit(0);
end:
for(i=0;i<=3;i++)
s[i]:=0;
while   1
 begin:
 ch:=fgetc(fp);
 if ch=EOF then
   break;
 switch(ch)
   begin:
    case 'a':
      s[0]++;
      break;
    case 'c':
      s[1]++;
      break;
    case 'g':
      s[2]++;
      break;
    case 't':
      s[3]++;
      break;
   end:
end:
fclose(fp);
break;
end::
            end:
   else
   begin
   display"%d",c[y];
   break;

   end:
```

```
for(i=0;i<=3;i++)
 r:=r+s[i];
 for(i=0;i<4;i++)
display"\t%d",s[i] ;
 display"\n\n\n\n";
for(i=0;i<=3;i++)
 s[i]:=(s[i]*100)/r;
  for(i=0;i<4;i++)
 display"\t%d",s[i];
 display"\n\n\n\n";
for(i=0;i<=2;i++)
  begin:
  for(j=0;j<=2-i;j++)
    begin:
     if s[j]<s[j+1] then
       begin:
            temp:=s[j];
            s[j]:=s[j+1];
            s[j+1]:=temp;
      end:
    end:
  end:
  for(i=0;i<4;i++)
display”t%d",s[i];
  display"\n\n\n\n";
/*tree const*/
for(i=0;i<=100;i++)
begin:
x[i]=0;
end:
y:0;
x[0]:100;
for(i=0;i<3;i++)
begin:
  x[2*y+1]:=s[i];
  x[2*y+2]:=x[y]-s[i];
  c[2*y+1]:=0;
  c[2*y+2]:=1;
  y:=2*y+2;
end:
for(y=0;y<=30;y++)
display"\t%d",x[y]:

/*traversal*/
for(i=0;i<4;i++)
begin:
display"\n%d=",s[i];
for(y=1;y<100;y=2*y+1)
  begin:
  if x[y]!=s[i] then
   begin:

            y:=y+1;
            if x[y]!=s[i] then
              begin:
               display"%d",c[y]:
              end:
            else
            begin
            display"%d",c[y];
```

```
  end:
end:
fclose(fp);
gettime(&t1);
   dispaly"The current time is:
%2d:%02d:%02d.%02d\n",t1.ti_hour, t1.ti_min,
t1.ti_sec, t1.ti_hund:
   display"\n\tCOMPLETION
TIME=%.1f",float((t1.ti_sec*100)+t1.ti_hund
  -float((t.ti_sec*100)+t.ti_hund)
end:
```

Table-I

3. Experimental Results

We tested this progrmme 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 3.0 GHz core 2 duo(1024FSB), Intel 946 original mother board, IGB DDR2 Hynix, 160GB SATA HDD Segate.

Sequence	Base pair	File Size (byte)	Reduce file size (byte)	Compression ratio[1]	compression rate(bits /base)	compare with gzip-9	Improvement
MTPACGA	100314	100314	24439	-2.54999	1.94900		
MPOMTCG	186608	186608	46719	-0.14361	2.00287		
CHNTXX	155844	155844	39028	-0.17196	2.00343		
CHMPXX	121024	121024	29274	-3.24563	1.93508		
HUMGHCSA	66495	66495	16691	-0.40454	2.00809		
HUMHBB	73308	73308	18394	-0.36558	2.00731	4.62605	0.56877
HUMHDABCD	58864	58864	14783	-0.45528	2.00910		
HUMDYSTROP	38770	38770	9760	-0.69641	2.01392		
HUMHPRTB	56737	56737	14252	-0.47764	2.00955		
VACCG	191737	191737	48002	-0.14133	2.00282		
HEHCMVCG	229354	229354	57406	-0.00117	2.00235		
Average					1.99486		

Table-II

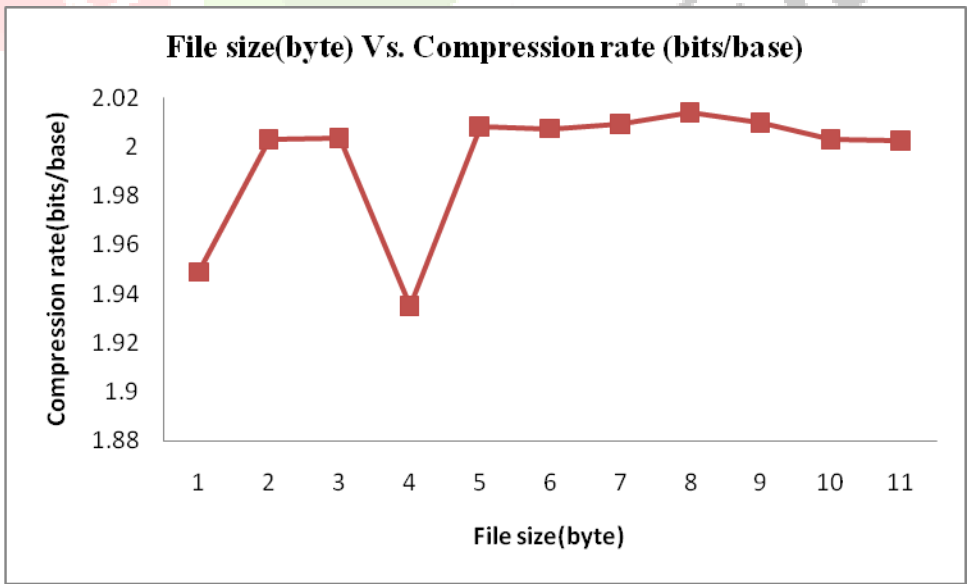

Fig.-I

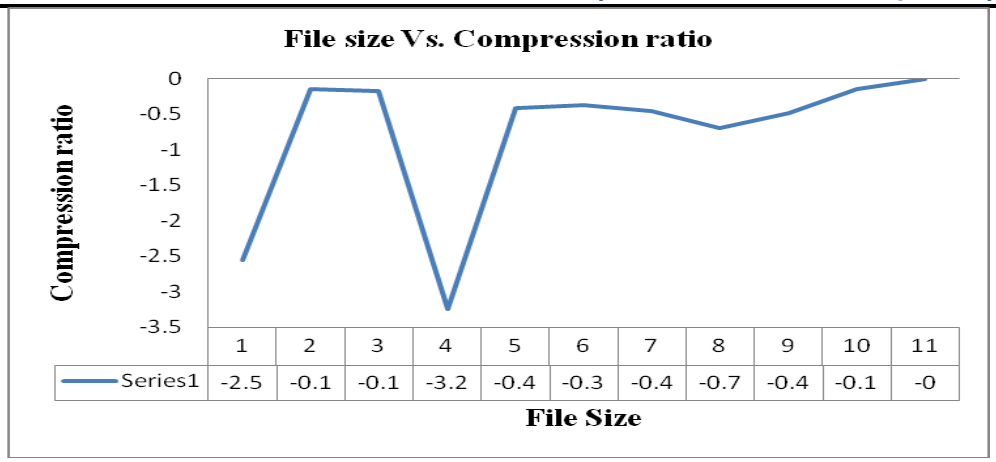

Fig.-II

Compression ratio and Compression rate for the DNA sequences shown in Table II. From top to bottom. Each row displays the result for this algorithm showing the compression ratio and rate for each sequence and the compression rate in bits per symbol. The last two columns show the compares of others compression techniques. Also presented the compression rate and ratio in fig.-I & II.

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, our algorithm performances better than it in both compression ratio and elapsed time. By just using the Huffman technique, 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.

5.Conclusion:

We discussed a DNA compression programming result using Huffman algorithm whose key idea is probabilities of occurrence of the symbols. This compression algorithm gives a good model for compressing DNA sequences that reveals the true characteristics of DNA sequences. 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.

6. Future work :

We are trying to do more, such as combining our dynamic LUT pre-coding routine with Huffman's compression technique, whose key idea is probabilities of occurrence of the symbols, to revise our algorithm in order to improve its performance.

Reference

[1] Matsumoto, T., Sadakane, K., and Imai, H., Biological sequence compression algorithms, GenomeInformatics, 11:43–52, 2000.
[2] 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.
[3] S.Bao, S.Chen, Z. Jing, and R. Ren, "A DNA Sequence Compression Algorithm Based on LUT and LZ77 , arXiv:cs.IT/0504100 v6 7 Jun 2005
[4] 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.
[5] M. Li and P. Vitányi, An Introduction to Kolmogorov Complexity and Its Applications, 2nd ed. New York: Springer-Verlag, 1997.
[6] R. Curnow and T. Kirkwood, "Statistical analysis of deoxyribonucleic acid sequence data-a review," J. Royal Statistical Soc., vol. 152, pp. 199-220, 1989.
[7] 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.
[8] 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.
[9] K. Lanctot, M. Li, and E.H. Yang, "Estimating DNA sequence entropy,"in Proc. SODA 2000, to be published.
[10] 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.)
[11] T.Matsumoto,K.Sadakame and H.Imani, "Biological sequence compression algorithm", Genome Informatics 11:43-52 (2000).
[12] X. Chen, M. Li, B. Ma, and J. Tromp, "Dnacompress:fast and effective dna sequence compression," Bioinformatics, vol. 18,2002.
[13]National Center for Biotechnology Information,http://www.ncbi.nlm.nih.gov
[14]Book : D. Adam, " Elements of Data Compression", published by Vikas Publishing House.