Statistical Analysis and Compression of DNA using
Weighted Probability approach and Modified
Run-Length Encoding

Srinivasa K G\textsuperscript{1}, Sagnik Dhar\textsuperscript{1}, Venugopal K R\textsuperscript{2}, and L M Patnaik\textsuperscript{3}

\textsuperscript{1} Data Mining Laboratory, M S Ramaiah Institute of Technology
\textsuperscript{2} Department of Computer Science, University Visvesvaraya College of Engineering
\textsuperscript{3} Professor, Microprocessor Application Laboratory, Indian Institute of Science, Bangalore

Abstract. Rapid advancements in research in the field of DNA sequence discovery has led to a vast range of compression algorithms. The number of bits required for storing four bases of any DNA sequence is two, but efficient algorithms have pushed this limit lower. With the constant decrease in prices of memory and communication channel bandwidth, one often doubts the need of such compression algorithms. The algorithm discussed in this paper hence not only compresses the DNA sequence, but also allows one to generate finite length sequences, which can be used to find approximate pattern matches. DNA sequences are mainly of two types, Repetitive and Non-Repetitive. The compression technique used is meant for the non-repetitive parts of the sequence, where we make use of the fact that a DNA sequence consists of only 4 characters. The algorithm achieves bit/base ratio of 1.3 – 1.4(dependant on the database), but more importantly one of the stages of the algorithm can be used for efficient discovery of approximate patterns.

Keywords: Compression, DNA, Run-length Encoding, Pattern Discovery.

1 Introduction

The realization of the fact that the DNA is the prime genetic molecule, which carries the hereditary information in its chromosomes, focused attention on its structure. The late 1940s and 1950s saw a huge amount of research in this field when the structure of the DNA was analyzed. Researchers wanted to find out how the chromosomes replicated themselves to form two identical copies and how they carried genetic information.

The discovery of the double helix was a path breaking revelation, when researchers expressed the structure as a three dimensional, helical form, where the difference between two genes was in the order and number of 4 nucleotide building blocks along complimentary strands. Extensive research has shown that though the genetic structure remains to be the same, there is a lot of variation in the DNA of one organism to the other. For instance, the chromosomes of small viruses have single stranded DNA, instead of the usual double stranded molecules. The right-handedness in the twisting of
the helix is also replaced by a left-handed twist in some organisms. The shape of the DNA molecules varies from linear to circular and from mono-coiled to super-coiled.

The evolution of such a huge plethora of DNA sequences have led to Bio-informatics scientists developing algorithms and methods to compress and store the sequences. The final goal though is not only in compressing the sequences but identifying the properties and patterns from the compressed sequences, saving us the overhead of decompressing the sequence. The four bases found in a genomic sequence are:

- Adenine — A
- Cytosine — C
- Guanine — G
- Thymine — T

A Genomic sequence in a programming-sense is nothing but a string of A,C,T,Gs of an approximate length of 3 billion. A genomic sequence can be identified using various properties it possesses. Repetition is a very important property, which is exploited in compression [5]. Sequences can be classified as highly repetitive (tandem repeats), moderately repetitive (interspersed repeats) and single copy (no repeats). Other properties include approximate matches, palindromes and reverse matches.

A string that has a four-character alphabet requires 2 bits per base for storage. The bit/base ratio has been lowered to 1.76 - 1.5 [4], with the use of intelligent algorithms, which not only uses the redundancies in a genomic sequence to compress but also help in motif discovery. Our paper not only brings down the bit/base ratio to around 1.4 but also generates subsequences, which can be used to find approximate matches.

Scientists have often relied on statistical analysis to relate to identification of the various properties in a DNA sequence. The probability modeling approach has been used as an initial step to approximate the ratio of the four bases in the sequence. This is an important step as it indicates to the algorithm, what it should expect as it moves ahead in the various passes in the step-wise compression algorithm. The algorithm not only presents a compression model but also exposes its potential in Motif Discovery.

2 Related work

The very first algorithm for DNA sequence compression was that of Grumbach and Tahi [2] [3], called BioCompress, was mainly aimed at Nucleic Acid sequences. It used a basic Lempel Ziv [10] [11] style substitution algorithm that detected exact matches and complementary palindromes. A complementary palindrome is that in which, the reverse of a particular subsequence, along with the complementary base interchanges, is an exact match of the original subsequence. Such kind of redundancy is quite frequent in DNA sequences and is often subjected to high compression. The updated version of the first algorithm was called Bio-Compress-2. The only major difference was that, this algorithm could detect regions where redundancy was absent and applied Arithmetic Coding of order 2 in these areas.

An efficient encoding scheme, Cfact was brought forward by Rivals. It had a two-pass technique, where the first pass, was used to detect exact repeats, and the second pass was used to encode the repeats. Regions with no repeats were coded at two bits per
base. Though the encoding scheme was useful, the algorithm, which used this scheme, did lossy compression [12], which is of little use for DNA sequences.

GenCompress [1] [13] [14] developed by Chen, had a better performance than its previous algorithms, Cfact and BioCompress-2 [3]. The algorithm created a Suffix Tree in the first pass, and the encoding was done in the second pass. The algorithm looked for optimal prefixes at every step, and gave a guaranteed gain in the bits/base ratio. If no gain was achievable, it used 2 bits/base. There were two variants of this algorithm, namely, GenCompress-1 that used hatting distances for the repeats, while GenCompress-2 used Edition Distance (Insertion, Deletion, Substitution) at each step.

CTW+LZ [4] algorithm came about as a very efficient compression algorithm, but it had very slow execution time. Developed by Matsumoto, this algorithm uses CTW (Context Tree Weighting) method and local heuristics for resolving the Greedy Selection Problem. Though this algorithm was not practically used because of the high time consumption, Lempel-Ziv algorithms have formed the crux of most Gene Compression algorithms.

DNACompress [15] developed by Chen, was a 2-phase algorithm, which used the Lempel-Ziv algorithm for compression. In the 1st phase, special software called PatternHunter [16] was used to find the exact repeats and the complimentary palindromes. The matches were sorted in descending order of size or some gain function. The second phase was finding non-repeating regions and approximately repeating regions. Pattern-Hunter developed by Ma, Tromp, Li used strings for Non Consecutive systems as a seed for search. The algorithm had a good execution time, and hence was regarded highly.

A departure from the Lempel-Ziv method was the use of Normalized Maximum Likelihood. NMLComp developed by Tabus, encoded the NML Model for discrete regression. The algorithm is suitable for encoding approximate block matches, based on replacement operations. The algorithm has a low complexity level and is quite light in terms of computational requirements.

The Sequitur (Nevill Manning and Witten 1997) used Digram Uniqueness and Rule Utility. Digram Uniqueness says no pair of adjacent symbols appear more than once in the grammar, while Rule Utility says that each rule is used at least twice (except for the start rule). The DNASEquitur(Cherniavsky and Ladner 2004) was an improvement on the previous Grammar-based compression algorithm, and it used Reverse Compliments.

The usual approach used by most algorithms is to find exact repeats and approximate repeats. The largest subset of compatible repeats is generated and encoded using algorithm-specific methods. Our algorithm is a departure from the usual first step of searching. It uses three step encoding process, to compress the whole sequence. The advantage in not using repeats is that the sequence is compressed irrespective of whether the sequence has redundancy or not.

The first step in our algorithm Gene-Compressor is a calculating the probability of the bases, followed by Huffman coding them. The second step encodes the sequence further using a transformation that aims at grouping the repetitions in the sequence. The DNA sequence is decompressed in the future for analysis and use; hence the transformation has been designed with an Inverse Transformation process in mind. The third step uses the localized repetition property that is present in the encoded sequence and uses a slightly modified version of Run Length Encoding.
3 Probability Model

In this section, we present a model which is used to calculate the probability of A, C, T and G in the sequence. Calculating the probability is the first step in the algorithm, as the probability values of the 4 bases are used to encode the sequence in a Huffman Code manner. As the length of the sequences is of the order of billions, we cannot do a full sequential scan of the sequence. This brings to light the need of a probability model, which can be used to calculate the occurrences of the bases.

Steps to finding probability:

1. Let the length of the sequence be \( L \). Using statistical methods, generate \( N \) such that \( 1 < N < (L/10) \). \( N' \) denotes the partition length to be considered. The value of \( N' \) should be restricted to \( L/10 \) as we want to calculate using a minimum of 10 samples.

2. Generate another number, \( R' \) such that \( 1 < R < N' \). \( R' \) denotes the sample size of each sample.

3. Partition the sequence into groups of \( N \) bases. Let the groups be called \( G_1, G_2, G_3, \ldots, G_n, \ldots G_N \).

4. A sample of each partition of length \( R \) can be selected in \((N - (R - 1))\) ways. The sample is selected using a \( \text{Rand()} \) function with seed value equal to system time (which assures uniqueness). The \( \text{Rand()} \) function chooses any one of the Samples. The length of the sample \( R' \) affects the accuracy of the Probability Model. The graph (as shown in Figure 1) displays values corresponding to partition size of 2000.

5. The probability of the four bases is calculated in these samples of \( R \). Let the probability be defined as

\[
(\text{PG}_{1A}, \text{PG}_{1C}, \text{PG}_{1T}, \text{PG}_{1G}) \text{ for the 1st group}
\]

\[
(\text{PG}_{2A}, \text{PG}_{2C}, \text{PG}_{2T}, \text{PG}_{2G}) \text{ for the 2nd group}
\]

\[
(\text{PG}_{3A}, \text{PG}_{3C}, \text{PG}_{3T}, \text{PG}_{3G}) \text{ for the 3rd group}
\]

\[
\vdots
\]

\[
\vdots
\]

\[
(\text{PG}_{iA}, \text{PG}_{iC}, \text{PG}_{iT}, \text{PG}_{iG}) \text{ for the ith group}
\]

6. The groups are classified into three types according to the conditions given below:
Condition 1: Let the difference between all pairs \((PG_{iX}, PG_{iY})\) where \(X, Y = A, C, T, G\) and \(X! = Y\) be equal to \(D\). \(i'\) signifies the group number. Condition 1 states that all values of \(D\) should be less than 0.05. This condition signifies “No Base Dominance”, a condition where the probability of occurrence of all bases tends to 0.25 and they have equal share in the sequence. This group is assigned a weight of 5.

Condition 2: The probability of any two of the four bases \(PG_{iX}\), where \(X = A, C, T, G\) should be greater than 0.4. Condition 2 signifies “Two Base Dominance”, a situation in which where two complimentary bases dominate the sequence. This group is assigned a weight of 3.

Condition 3: The probability of any 1 of the bases \(PG_{iX}\), where \(X = A, C, T, G\) should be greater than 0.8. Condition 4 signifies “Single Base Dominance”, a situation where one base dominates the sequence. In the absence of a complimentary pair, the base forms Hydrogen Bonds to attain stability. This group is assigned a weight of 2.

7. Multiply the probability of each base in a group with the corresponding multiplier of that group, i.e. \((PG_{iX} \times W_i)\) where \(X = A, C, T, G\) and \(i'\) denotes the group number.

8. To find the probability of a particular base, \(X\)
\[ P_x = (\Sigma(PG_{ix} \times W_i))/\Sigma W_i \] where \(X = A, C, T, G\)
The group length is selected as 1000 and the subsequence in that group is of length 100. This gives a Reliability Co-efficient of 0.1. These values are not standardized and can be manipulated according to the reliability of statistical analysis required. The reliability coefficient with changing values of \( R \) and constant \( N = 2000 \). (As shown in Fig 2).

4 Algorithm

*Problem Definition:* Given a genomic sequence, the objectives are:
(i) Compress the sequence so as to obtain an effective bit/base ratio.
(ii) Generate subsequences, which can be used for Pattern Matching.

*Assumptions:*
(i) The probability of each base calculated using the above Probability model described in section III, tends to the actual probability in the sequence.

We calculate the probability of each base and then apply Huffman coding to the Sequence. Huffman Coding helps us convert the file into 0s and 1s, the basic requirement, if we want to bring down the bit/base ratio from 2 to 1.4.
**Algorithm: HuffmanTree-Creator:** Creates a Huffman Tree

1. Let the base probability values be leaf nodes or independent subtrees.
2. Sort the sub-trees according to their values.
3. Combine the sub-trees with two lower-most values by a root node.
4. Eliminate the two sub-trees chosen from the list and insert the new sub-tree.
5. Repeat Steps ii to v till we obtain a single Sub-tree.
6. Mark all the left edges as zero and all the right edges as one.
7. The path from the root to a particular base is the Huffman Code for the corresponding base.

The new file will definitely be approximately double the size of the original sequence, but it brings the alphabet size down to 2.

**Algorithm PacketGen:** Used to generate intermediate packets

1. The sequence is broken down into packets of length $'p'$. 
2. All $'p'$ length rotations of each packet $P$ is generated.
3. The last base of each rotation is extracted and put in another packet $P'$. 
4. This packet is interposed into the sequence in place of Packet $P$.

Using the PacketGen algorithm, the sequence is then broken down into packets of length $'p'$. The value of $'p'$ is an important tradeoff and deserves discussion. The various values of $'p'$ and its effect on the algorithm are discussed in section V. For the time being, we assume that a $'p'$ value of eight is optimal and hence used.

The packets $P_i$ are stored in a buffer, along with all its $'p'$ length rotations. The rotations of a particular packet can be used in any Motif Discovery Algorithm, which searches for approximate repeats.

Once all the rotations are generated, we take the last character of each rotation and form a new $'p'$ length packet, $P_i'$. This packet is put instead of the original packet. The new packet, $P_i'$, has two very important properties which are exploited. $P_i'$ has very high localized repetitions, because it has been constructed from a sorted order of strings. This property can be used to compress the packet, but the main question is how do we get back the original packet $'P_i'$. The method used above to encode a packet using its rotations can be reversed to get back the original packet. This is done by a series of sorting operations.

**Algorithm Gene-Compressor**

Input: Genomic Sequence $G$
Output: Compressed Sequence $GC$
1. Calculate the probability of each base in the sequence using the probability.
2. Calculate encryption codes according to the probability calculated in Huffman’s coding algorithm.
3. Encrypt the input file $G$ by replacing each of the Bases by their corresponding Huffman Codes.
4. The new file $G'$ consists only of $0's$ and $1's$ and $Length(G') = 2 * Length(G)$, approximately.
5. The sequence in $G'$ is broken into small packets, $P_i$ of length $'p'$. 
6. Each packet $P_i$ is put in a buffer $B$ and all $p$ rotations of the buffer are generated and sorted in ascending order.
7. The last character of each rotation is extracted and a string $P_i'$ is made out of it.
8. The packet $P_i$ is replaced by $P_i'$ in the file $G'$.
9. The file $G'$ now consists of packets which have a high percentage of repetitions of $0's$ and $1's$.
10. Using this to our benefit, let us do run length encoding of consecutive repetitions of four and five bases only.
11. Store the run-length encoded sequence in a file called $G_c$, which is the output file.

Once we generate the encoded sequence, we apply a modified version of Run-length encoding, where only 4 and 5 length repetitions are encoded. This has two reasons. First, we cannot increase the no of alphabets used in the final compression. It has to remain at 4. Hence we just add two more characters, $'4'$ and $'5'$, along with the characters $'0'$ and $'1'$. Along with that, a close observation has showed that the frequency of localized rotations occurs rarely.

5 Optimization of $'p'$

The value of $'p'$, or the length of the packets can vary from two, three, four to around sixteen. With increase in the value of $'p'$, the buffer length increases meaning we can generate motifs of a larger length. But on the other hand, using large value of $'p'$ increases the overhead of sorting and generating full-length rotations. Using very low values of $'p'$ means that the repetitions are of length two to three, which does not give any compression sense when we apply run length encoding.

The algorithm was tested with $'p'$ values of four, six, eight and it was found that eight was the most optimal value for compression.

At $p = 8$, the buffer length to be sorted is not too high, and also it gives repetitions of length 4 to 5, which is the ideal for the modified run length encoding process that is used in this algorithm.

At $p = 4$, the repeat sequences was of length 2, which did not provide any decrease in the bit/base ratio, when encoded using run-length encoding.

At $p = 6$, the buffer length is too less for later use in motif discovery.
6 An example

Let a sequence of length 250 bases be:

```
atattcgttaatgcagatagtctgtcgcggatcaacccctttgggagatatgcattccttagtgcagatctgatcgac
cggggcttcccttacgaatggtttagcgagaatcgggaccgagggctatatgtcgcagaaatcagcgccgagtctgcagc
tacctaaacgcataagatgtatcctgaggggtatcgagactcgcagactgcgcacccttgccaaagattgcaaacgtagccaaagatg
agacgctg
```

Calculating the probabilities using statistical measures and the probability model given in section III:

\[
\begin{align*}
P_A &= \frac{66}{250} = 0.264 \\
P_C &= \frac{59}{250} = 0.236 \\
P_T &= \frac{60}{250} = 0.24 \\
P_G &= \frac{65}{250} = 0.26
\end{align*}
\]

Taking the example of the first few bases: “atattcgttta...”
The subsequence is Huffman coded as: “0111011111010001111101”
According to Fig 3, dividing into packets of Length ‘\(p\)’ = 8

Packet I: 01110111
Packet II: 11101000
Packet III: 11111101

Generating all \(p\) length rotations for Packet I:

<table>
<thead>
<tr>
<th>Random Order</th>
<th>Sorted Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>10111011</td>
<td>01110111</td>
</tr>
<tr>
<td>11011101</td>
<td>01110111</td>
</tr>
<tr>
<td>11101110</td>
<td>10111011</td>
</tr>
<tr>
<td>01110111</td>
<td>10111011</td>
</tr>
<tr>
<td>10111011</td>
<td>11011101</td>
</tr>
<tr>
<td>11011101</td>
<td>11011101</td>
</tr>
<tr>
<td>11101110</td>
<td>11101110</td>
</tr>
<tr>
<td>01110111</td>
<td>11101110</td>
</tr>
</tbody>
</table>

Extracting the last character from the Buffer, we get:

Transformed Packet \(P'_{I}\): 11111100
Similarly

Transformed Packet \(P'_{II}\): 10100110
Transformed Packet \(P'_{III}\): 11111110
Applying modified Run-length encoding to the packets we obtain:
Packet $P'_1$: 51100
Packet $P'_2$: 10100110
Packet $P'_3$: 51110
Applying the algorithm to the sequence of 250 bases we get:

511001010011011101100101010101010110110011001101105040101101101111
401000111011401101010105111101111151010100011111011101501000100
111100101110110101010105101011150101011011110110150010150100011011
10101110011501000101111150510004100101110101011011105100
10101100101501041001140100110104100111115010114011001010100011001
0101001011100140111015010

Length of the above sequence: 367

Compression achieved (bit/base ratio): 1.472

7 Performance Analysis

The Compression algorithm, Gene-Compressor, is used for compression of Repetitive and Non-repetitive Sequences. During the development of the algorithm, we used a
DNA Sequence generator to generate sample input and test the code. After the development of the algorithm, it was tested using standard DNA databases. The results of the tests have been given in Table 1 and Table 2. Table 1 gives the comparative analysis of Gene-Compressor with other text-based compression algorithm. Table 2 gives the comparative analysis with other DNA-specific compression algorithms. All the values given are in bits/base. The algorithm was implemented on a Pentium 4 machine with 512MB Ram running Fedora Core 3. The algorithm performs better than most other algorithms in terms of compression. The algorithm does require a large amount of Buffer for storing the intermediate encoding stages. It also requires a high amount of Buffer for storing the full-length rotations of the packets.

<table>
<thead>
<tr>
<th>DNA Sequence name</th>
<th>Sequence Length</th>
<th>gzip -9</th>
<th>lz (1M)</th>
<th>arith +1(1m)</th>
<th>PPMD+</th>
<th>adapted PPMD+</th>
<th>normal CTW</th>
<th>CTW -4</th>
<th>Gene Compressor</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHNTXX</td>
<td>121024</td>
<td>2.220</td>
<td>2.234</td>
<td>1.866</td>
<td>1.977</td>
<td>1.840</td>
<td>1.879</td>
<td>1.838</td>
<td>1.473</td>
</tr>
<tr>
<td>CHNTXX</td>
<td>155844</td>
<td>2.291</td>
<td>2.300</td>
<td>1.956</td>
<td>2.062</td>
<td>1.934</td>
<td>1.974</td>
<td>1.933</td>
<td>1.403</td>
</tr>
<tr>
<td>HEHCMVCG</td>
<td>229354</td>
<td>2.279</td>
<td>2.286</td>
<td>1.985</td>
<td>2.053</td>
<td>1.965</td>
<td>1.997</td>
<td>1.958</td>
<td>1.523</td>
</tr>
<tr>
<td>HUMDYSTROP</td>
<td>38770</td>
<td>2.377</td>
<td>2.427</td>
<td>1.948</td>
<td>2.237</td>
<td>1.921</td>
<td>1.960</td>
<td>1.920</td>
<td>1.534</td>
</tr>
<tr>
<td>HUMGHCSA</td>
<td>66495</td>
<td>1.551</td>
<td>1.580</td>
<td>1.438</td>
<td>2.077</td>
<td>1.694</td>
<td>1.376</td>
<td>1.363</td>
<td>0.901</td>
</tr>
<tr>
<td>HUMHBB</td>
<td>73308</td>
<td>2.228</td>
<td>2.255</td>
<td>1.911</td>
<td>2.116</td>
<td>1.921</td>
<td>1.917</td>
<td>1.892</td>
<td>1.653</td>
</tr>
<tr>
<td>HUMHDABCD</td>
<td>58864</td>
<td>2.209</td>
<td>2.241</td>
<td>1.950</td>
<td>2.130</td>
<td>1.948</td>
<td>1.909</td>
<td>1.897</td>
<td>1.661</td>
</tr>
<tr>
<td>HUMHPRTB</td>
<td>56737</td>
<td>2.232</td>
<td>2.269</td>
<td>1.942</td>
<td>2.130</td>
<td>1.932</td>
<td>1.922</td>
<td>1.913</td>
<td>1.532</td>
</tr>
<tr>
<td>MPOMTCG</td>
<td>186609</td>
<td>2.280</td>
<td>2.289</td>
<td>1.961</td>
<td>2.075</td>
<td>1.966</td>
<td>1.989</td>
<td>1.962</td>
<td>1.715</td>
</tr>
<tr>
<td>PANMTPACGA</td>
<td>100314</td>
<td>2.232</td>
<td>2.249</td>
<td>1.873</td>
<td>2.018</td>
<td>1.872</td>
<td>1.902</td>
<td>1.866</td>
<td>1.722</td>
</tr>
<tr>
<td>SCCHRIII</td>
<td>315339</td>
<td>2.265</td>
<td>2.266</td>
<td>1.935</td>
<td>2.023</td>
<td>1.950</td>
<td>1.976</td>
<td>1.945</td>
<td>1.732</td>
</tr>
<tr>
<td>VACCGB</td>
<td>191737</td>
<td>2.190</td>
<td>2.194</td>
<td>1.862</td>
<td>2.002</td>
<td>1.910</td>
<td>1.897</td>
<td>1.857</td>
<td>1.672</td>
</tr>
</tbody>
</table>

8 Conclusion

The algorithm presented in this paper was tested for around 12 sequences. It performs better than most of the text compression algorithm as it uses a upper limit of 2 bits/base. It performs better than most of the DNA compression algorithms also. It has a high execution time for Sequences of length greater than 2 billion. On the other hand, we have also seen that the Buffer used in the intermediate step can be used to generate full length and partial length rotations. These rotations can be used in Motif Discovery for finding approximate matches. We are trying to use this algorithm for compression of larger genes with an optimized execution time and also for Non-Repetitive genes like HUMDYSTROP, who failed to compress efficiently using most of the standard compressors.

During the design of the algorithm we have had to face a lot of design decisions, which we have taken after conducting exhaustive experiments and statistical analysis.
Table 2. Comparison of DNA-Specific Compression Algorithms

<table>
<thead>
<tr>
<th>DNA Sequence name</th>
<th>Sequence Length</th>
<th>Bio-Compress</th>
<th>Gen-Compress</th>
<th>CTW-LZ</th>
<th>DNA-Compress</th>
<th>DNA-Pack</th>
<th>DNA-MEM</th>
<th>Gene Compressor</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHNTXX</td>
<td>121024</td>
<td>1.6848</td>
<td>1.6730</td>
<td>1.6690</td>
<td>1.6616</td>
<td>1.6602</td>
<td>1.660</td>
<td>1.473</td>
</tr>
<tr>
<td>CHNTXX</td>
<td>155844</td>
<td>1.6172</td>
<td>1.6146</td>
<td>1.6120</td>
<td>1.6127</td>
<td>1.6103</td>
<td>1.610</td>
<td>1.403</td>
</tr>
<tr>
<td>HEHCMVCG</td>
<td>229354</td>
<td>1.8480</td>
<td>1.8470</td>
<td>1.8414</td>
<td>1.8492</td>
<td>1.8346</td>
<td>1.834</td>
<td>1.523</td>
</tr>
<tr>
<td>HUMDYSTROP</td>
<td>38770</td>
<td>1.9262</td>
<td>1.9231</td>
<td>1.9175</td>
<td>1.9116</td>
<td>1.9088</td>
<td>1.908</td>
<td>1.534</td>
</tr>
<tr>
<td>HUMGHCSA</td>
<td>66495</td>
<td>1.3074</td>
<td>1.0969</td>
<td>1.0972</td>
<td>1.0272</td>
<td>1.0390</td>
<td>1.030</td>
<td>0.901</td>
</tr>
<tr>
<td>HUMHBB</td>
<td>73308</td>
<td>1.8800</td>
<td>1.8204</td>
<td>1.8082</td>
<td>1.7897</td>
<td>1.7771</td>
<td>1.776</td>
<td>1.653</td>
</tr>
<tr>
<td>HUMHDABCBD</td>
<td>58864</td>
<td>1.8770</td>
<td>1.8192</td>
<td>1.8218</td>
<td>1.7951</td>
<td>1.7394</td>
<td>1.739</td>
<td>1.661</td>
</tr>
<tr>
<td>HUMHPRTB</td>
<td>56737</td>
<td>1.9066</td>
<td>1.8466</td>
<td>1.8433</td>
<td>1.8165</td>
<td>1.7868</td>
<td>1.788</td>
<td>1.532</td>
</tr>
<tr>
<td>MPOMTCG</td>
<td>186609</td>
<td>1.9378</td>
<td>1.9058</td>
<td>1.9000</td>
<td>1.8920</td>
<td>1.8932</td>
<td>1.892</td>
<td>1.715</td>
</tr>
<tr>
<td>PANMTLCPACGA</td>
<td>100314</td>
<td>1.8752</td>
<td>1.8624</td>
<td>1.8555</td>
<td>1.8556</td>
<td>1.8535</td>
<td>1.853</td>
<td>1.722</td>
</tr>
<tr>
<td>VACC_G</td>
<td>191737</td>
<td>1.7614</td>
<td>1.7614</td>
<td>1.7616</td>
<td>1.7580</td>
<td>1.7583</td>
<td>1.758</td>
<td>1.672</td>
</tr>
</tbody>
</table>

We have made assumptions on the basis of the fact that the results of the statistical study approximates the exact results to a great extent.

Acknowledgments

The project is supported by the AICTE, as a part of Career Award of Young Teachers (AICTE File No.: F. No. 1-51/FD/CA/(9)2005-06) to Mr. K.G. Srinivasa, faculty, Dept. Of Computer Science and Engineering, M.S. Ramaiah Institute of Technology, Bangalore : 560094, India.

References

1. Xin Chen, Sam Kwong, and Ming Li. A compression algorithm for dna sequences and its application in genome comparison. genomic, 12:512-514, 2001