SBVRLDNACOMP: AN EFFECTIVE DNA SEQUENCE COMPRESSION ALGORITHM

Subhankar Roy¹, Akash Bhagot², Kumari Annapurna Sharma² and Sunirmal Khatua³

¹Department of Computer Science and Engineering Academy of Technology, G. T. Road, Aedconagar, Hooghly-712121, W.B., India
²Master of Computer Application, Academy of Technology, G. T. Road, Aedconagar, Hooghly-712121, W.B., India
³Department of Computer Science and Engineer, University of Calcutta, 92 A.P.C. Road, Kolkata-700009, India

ABSTRACT

There are plenty specific types of data which are needed to compress for easy storage and to reduce overall retrieval times. Moreover, compressed sequence can be used to understand similarities between biological sequences. DNA data compression challenge has become a major task for many researchers for the last few years as a result of exponential increase of produced sequences in gene databases. In this research paper we have attempt to develop an algorithm by self-reference bases; namely Single Base Variable Repeat Length DNA Compression (SBVRLDNAComp). There are a number of reference based compression methods but they are not satisfactory for forthcoming new species. SBVRLDNAComp is an optimal solution of the result obtained from small to long, uniform identical and non-identical string of nucleotides checked in four different ways. Both exact repetitive and non-repetitive bases are compressed by SBVRLDNAComp. The sound part of it is without any reference database SBVRLDNAComp achieves 1.70 to 1.73 compression ratio after testing on ten benchmark DNA sequences. The compressed file can be further compressed with standard tools (such as WinZip or WinRar) but even without this SBVRLDNAComp outperforms many standard DNA compression algorithms.

KEYWORDS

DNA; Redundancy; Reference Base; Optimized Exact Repeat; Non-Repetition; LZ77; and Compression Ratio.

1. INTRODUCTION

The size of the genome database is increasing annually with a great speed. Each day several thousand nucleotides are sequenced in the labs. From 1982 to the present, the number of bases in GenBank has doubled approximately every 18 months. It is found that in Dec 1982, the number of bases and the number of sequence records were 680338 and 606 respectively for GenBank and none for WGS (Whole Genome Shotgun) and with Release 129 in Apr 2002, the number of bases
and the number of sequence records were 19072679701 and 16769983 respectively for GenBank and the WGS had 692266338 number of bases and 172768 number of sequences. Again in the Release 206 in Feb 2015, the number of bases and the number of sequence records were 187893826750 and 181336445 respectively for GenBank and the WGS had 873281414087 numbers of bases and 205465046 numbers of sequences.

High-throughput sequencing technologies [1] make it possible to rapidly acquire large numbers of individual genomes, which, for a given organism, vary slightly from one to another. Such repetitive and large sequence collections are a unique challenge for compression. Compressed data reduce the communication cost and storage cost. Furthermore, compressed sequence can be used to get the similarities within sequences.

The highly repetitive DNA sequences own some motivating properties [2], [3] which can be utilize to compress it. As DNA sequences consists of four nucleotides bases A, C, G and T called exon (i.e. coding regions or protein synthesis) or introns (i.e. non-coding regions or no protein synthesis) a, c, g and t in frequent cases, two bits is enough to store each base, in spite of this fact, the standard compression algorithm like “COMPRESS”, “GZIP”, “BZIP2”, “WinZip” or “WinRar” uses more than 2 bits per base [4]. Even both static and adaptive Huffman’s code fails badly on DNA sequences because the probabilities of occurrence of these symbols are not very different. In this paper we focus on compression of this particular data. DNA is a double stranded molecule with neighboring strands connected through hydrogen bonding between the bases. This hydrogen bonding is quite specific with Thymine (T/t) on one strand pairing with Adenine (A/a) on the other strand and Guanine (G/g) on one strand pairing with Cytosine (C/c) on the other and vice versa. All compression algorithms compress only one strand.

These behaviors are primary to substantial expansion in the size of DNA data sets, and are providing opportunities for novel compression techniques that take advantage of the characteristics of these new data. Our aspiration is to discover mechanisms for detecting this redundancy and use it in compression by searching optimal level of similarity within individual’s sequence.

The majority approach of compressing genomic data is to interpret the difference between the newly data that should be compressed with a reference sequence and then find out the differences [5], [6], [7]. This will be competent and possible when dealing with species that have a valid reference but is less satisfactory when approaching new species data due to the lack of a standard reference genome; for example, RNA sequencing data should be aligned against the entire transcriptome and is not feasible to account each possible substitute transcript splicing.

DNA sequences in higher eukaryotes contain many repetitive nucleotides and have several copies and also the Genes duplicate themselves for evolutionary purposes. To analyze, these sequences have to be stored. So, these facts conclude that DNA sequences should be compressed. Human DNA almost has 3 billion bases and among then more than 99% are the same in all human [8], [9]. Data compression reveals certain theoretical ideas such as entropy, mutual information and complexity between sequences of different genomes.

The most common and is very simple form of DNA compression is just binary encode the DNA sequence bases using 2 bit for each nucleotides i.e. by replacing A/a, C/c, G/g and T/t with “00”, “01”, “10” and “11” chosen abruptly. But in practically just by binary encoding the sequence, we can cut the file size near about 75% with slide more than 2 bit per base [10]. The advantage of
this method is that the file can be easily parsed without needing complicated compression algorithms, but it is not satisfactory because it does not use any property within a sequence. Traditionally, DNA data compression methods usually compress based on the different properties such as complementary string, complementary palindrome string or reverse complements, cross-chromosomal similarity, approximate repeat, direct repeat etc. [11] of DNA sequence. However a slight change in properties may give worst result. The preliminary summit of SBVRLDNAComp based on repeat checked in four different ways, which have been applied to each DNA sequence. SBVRLDNAComp encode both exact repetitive and non-repetitive parts without using exact Lempel-Ziv based compression algorithms or order-2 arithmetic encoding for former and later one respectively which are very common. So a sequence which is not so redundant also gives good compression ratio. SBVRLDNAComp permits independent compression and decompression of individual sequences.

The study is organized as follows: The description of a number of other specialized DNA compression algorithms (Section 2), the SBVRLDNAComp algorithm (Section 3), and experimental results (Section 4) are followed by concluding remarks of methods and their affect on compression (Section 5).

2. RELATED WORK

All Genome compression algorithms utilize redundancy within the sequence, but vary greatly in the way they do so. In general, compression algorithms can be classified into Naive Bit Manipulation, Dictionary-based, Statistical and Referential Algorithms.

Two best known lossless compression algorithms are LZ77 [12] and PPM [13]. PPM predicts the probability distribution of the next symbol based on all previously observed symbols. Both approaches follow sequential processing to encode a string. They test to see if the current substring has been seen formerly and, if so, encode it by reference to the earlier happening. A sliding window is maintained for newly observed text. If the text exceeds this window size, it cannot be used in compression. The application of LZ77 algorithm are gzip, 7-zip etc.

Compression algorithm using exact repeats are begin with BioCompress [14], BioCompress-2 [15], Cfact [16], Off-Line [17], DNASC [18] and B2DNR [19] use the common characteristic of a sequence.

Initial DNA compression algorithm based on exact repeat proposed by S. Grumbach and F. Tahi BioCompress search the regularities, such as the presence of palindromes. Although obtained result is not satisfactory but better than the existing general purpose compression technique. Extended version of BioCompress is BioCompress-2; later one is based on LZ77. It searches for longest exact repeats or longest palindrome or reverse complement in already encoded sequences, then encodes that repeats by repeat length \( l \) and the first position \( p \) of preceding repeat appeared i.e. a pair of integers \((l, p)\); when no repetition is found it uses order-2 arithmetic coding. The difference between BioCompress and BioCompress-2 is the addition of order-2 arithmetic coding in the later one.
Cfact is a two phase algorithm that executes sequentially. The parsing phase obtains the longest repeated factors using a suffix tree data structure. The encoding phase compresses first occurrences of repetitive segments and all non-repetitive segments using a 2-bit method. The repeated segment is replaced by a pointer in the form of (pos, len) tuples. But the suffix tree formation for large datasets is not possible in memory.

Off-Line compression algorithm approach is quite similar to Cfact. It uses a suffix tree to find out the exact repeated substring. But unlike Cfact it uses an augmented suffix tree which reduces the time and space complexities to $O(n \log^2 n)$ and $O(n \log n)$ from both $O(n^2)$, where $n$ is the number of bases in a sequence. It encodes most frequent non-overlapping substrings of a sequence. The bpc of Off-line is 1.97, which is not better than any DNA specific compression algorithm but it is a general purpose compression algorithm.

Most of the DNA compression techniques considered frequent occurred of bases i.e. A, C, G and T. But DNASC have taken one of the infrequent occasion nucleotides N; which can be either A, C, G or T with equal probability. It is used to compress both DNA and RNA the former one can be converted to later just by replacing T with U. Bases are compressed by first horizontally and then vertically. In vertical process it follows LZ style representation of nucleotides with window size 128 bases i.e. 1024 bits and block size of 6 digits as a combination of 2 i.e. 21 bits using extended LZ style. Compression of the next block with respect to the current block is done by one of the 22 ways of redundancy.

Two algorithms B2 and B2DNR considering frequent and all infrequent bases. The rare characters are {K, M, R, S, W, Y, B, D, H, V, N}. They have formed nucleic acid sequences fragments of {A, C, G, T} and {K, M, R, S, W, Y, B, D, H, V, N} into $15^2=255$ combinations and then converting them into 255 ASCII characters out of 256. For repeat count in the later method they have used 9 characters from digit ‘1’ to ‘9’. If repeat is greater than 9; then it recounts the repeats.

Algorithms using approximate repeat detection, start with GenCompress [20], followed by DNACompress [21], DNAPack [22], GeNML [23] and DNAEcompress [24] show that even improved results can be achieved by exploiting the approximate nature of the repeated regions in DNA sequences.

GenCompress based on LZ77. GenCompress uses both approximate repeats and reverse complements and also uses reverse complements that contain errors. It considers three standard edit operations Replace, Insert and Delete for approximate matching. There are two versions of GenCompress: GenCompress-1 uses the Hamming distance i.e. searches for approximate repeats with replacement or substitution operations only, and GenCompress-2 uses edit distance that searches for approximate repeats based on the operation insert and delete. This algorithm is able to detect more properties in DNA sequences, such as mutation and crossover. In addition, they use arithmetic coding of order 2 if no significant repetition is found i.e. for non-profit encoding by these properties.
DNACompress algorithm compresses a sequence in two phases. In the first phase it finds all approximate repeats with highest score including complemented palindromes using a software tool called Pattern Hunter [25] and in the second it encode approximate repeat regions and non-repeat regions. It encodes those approximate repeats that give profits on overall compression.

DNAPack compression algorithm compresses both the repeat segments and the non-repeat segments. It detects the long approximate repeats and the approximate complementary palindrome repeats using dynamic programming. Both GenCompress and DNACompress use the greedy approach for selection of the repeat segments. DNAPack used Hamming distance, i.e. the approximation is only done by substitution. The non-copied regions are encoded by the best choice from an Order-2 Arithmetic Coding, Context Tree Weighting Coding (CTW) and naive 2 bits per symbol methods.

The GeNML algorithm split the sequence into fixed size blocks. It encodes the block by maximum likelihood model. GeNML combines both substitution and statistical styles. An inexact repeat is encoded using a pointer to an earlier instance of the subsequence followed by substitution, insertion or deletion operation. In compare with the above three algorithms; it produced better compression results than using approximate repeat.

DNAEcompress compression algorithm for DNA sequence uses three standard edit operations; replacement, insertion and delete which are extended to five operations. These are Complementary replace define by crep (pos), Insert string represented by inss (pos; str), Delete string i.e. dels (poslen), Exchange expressed as exch (pos1; pos2) and Inversion define by inv (poslen) respectively. The matched patterns both exact and inexact are encoded by LZ algorithm and unmatched pattern are order-2 arithmetic encoding. So it is like GenCompress algorithm.

Sequentially lossless compression algorithm such as PPM and the other key family of this category are the basis of the DNA compression algorithms CDNA [26], CTW+LZ [27], and XM [28].

The first compression algorithms based on statistical method by detecting the approximate repeat within DNA sequences is CDNA. It predicts the probability distribution of each nucleotide by using partial matching of the current context to earlier seen substrings. To measure the inexact similarity CDNA use Hamming distance.

CTW+LZ is a non-greedy algorithm which searches for exact and approximate repeats; exact and approximate reverse complements or complementary palindrome using hash table and dynamic programming. It follows time consuming greedy search to get the longer repeat. LZ77 algorithm is used to compress long exact or approximate repeats. Short repeats are encoded by order-32 Context Tree Weighting (CTW) and edit operations are encoded by arithmetic coding. It uses PPM a statistical compression algorithm to predict the probability of next symbol using preceding symbols.

The best compression algorithm compared to other two recent above algorithms is expert model (XM). It estimates the probability of recent bases with multiple “experts” but based on PPM. An example of an expert is an order-k Markov model where k>0. Based on the k preceding symbols it estimates the probability of recent nucleotides. Once the symbol’s probability distribution is
determined, it is compressed by using a primary compression algorithm such as arithmetic coding. Another is a copy expert that gives a probability based on whether the next symbol is part of a copied region from a particular offset.

There is one important feature that has not observed by all of the above algorithms based on exact repeat. They have not checked all promising types of exact repeats from very small to maximum possible and uniform of particular size. Our algorithm overcomes that.

In the following section we clarify SBVRLDNAComp algorithm in details and all the associated components methods that SBVRLDNAComp invoke; and then experimental results. Comparison with other algorithm is also enlightened in the result.

3.PROPOSED METHODS

The algorithm SBVRLDNAComp is designed for the compression of DNA. It can also used for RNA sequence but not for proteins. It encodes a text of characters $\sum = \{A, C, G, T \text{ or } U\}$. This algorithm is an optimal solution of four proposed methods compressed any sequence by searching the repeats in four different ways. It is a sequential compression algorithm. After getting the bits sequence form Method 1 and Method 2 it compares the bit length and chooses the optimal one dynamically before going to the subsequent method. All methods allows for access to individual sequences in the compressed data.

Therefore, $N_{opt.} = \text{Min}(N', N'', N''', N'''' )$, Where $N'$, $N''$, $N'''$ and $N''''$ are the number of bits by four proposed method and $N_{opt.}$, the optimal number of bits before mapping to character. The character mapped intermediate compressed file is finally compressed by LZ77 [12]. The Fig. 1 shows the general structure of the SBVRLDNAComp algorithm. Following sub-sections discussed the components method of SBVRLDNAComp and general algorithm.

![Fig. 1. SBVRLDNAComp structure](image)
3.1. Components methods of SBVRLDNAComp

The compression process by all methods is given below. Decompression methods are just reverse of compression process. All methods will use a self dynamic reference variable \( R \) which stores the current character i.e. A, C, G or T, a temporary variable \( T \) and scanning the sequence horizontally from left to right and top to bottom. Each module search repeated regions from different aspect then encode them to their individual logic, however, the non-repeated region and \( R \) are coded just by straightforward 2bit coding rule assuming A/a = 00, C/c = 01, G/g = 10 and T/t or U/u =11 respectively. Result of each method is a binary stream and the optimal one mapped to ASCII character from fixed window of size 8. Method 1 (M1) gives profitable encoding when the segments length is form 4 to 9. Whereas Method 2 (M2) for longest repetition. Method 3 (M3) act well for 2 successive identical bases throughout the sequence. For uniform repeats of segment size 3, Method 4 (M4) performed well. In the following sub-sections the details of each method have been discussed.

3.1.1. Method 1

This method finds the repeated optimal segments length of 1 to 8 characters with respect to the current reference base \( R \). Control bit 0 i.e. denoted by \( B_0 \) for repetitive \( R \) and bit 1 denoted by \( B_1 \) for non-repetitive \( R \) to distinguish between two, followed by the 2-bits representation of \( R \). Repeated variable length segment of characters (2 <= length <= 9) are encodes by the relation: \( S_r = r, R = 1, 2, 3, 4, 5, 6, 7, 8 \). Three bits coding rules form \( S_1 \) to \( S_8 \); for segments of length 2 to 9 in step of 1, are represented by \{000, 001, 010, 011, 100, 101, 110 and 111\}. The codeword is \( B_0 R S_1...r \) for repetitive segments and \( B_1 R \) for non-repetitive portion.

The total number of bits required to compressed any sequence by M1 is obtained by the following equation,

\[
N' = n \times c_0 + n' \times c_1
\]

Where:
- \( c_0 = \) Number of bits for repetitive bases = \((1 + 1 \times 2 + 3) = 6\),
- \( c_1 = \) Number of bits for non-repetitive bases = \((1 + 1 \times 2) = 3\),
- \( n = \) Total number of repetition,
- \( n' = \) Total number of non-repetition.

Then, \( N' = 6 \times n + 3 \times n' = 3(2n + n') \)

3.1.2 Method 2

For each repetition with respect to \( R \) the assigned bit is 1 and for end of repetitive base or bases bit is 0. It will search for maximum possible repetition. It uses dynamic programming approach. So if 4 consecutive bases appeared then the codeword is \( R B_1 B_1 B_1 B_0 \) and for non-repetitive \( R \) it is \( R B_0 \), where \( R \) is encoded by 2-bit coding. Then the following equation determines the number of bits desired by M2,

\[
N'' = n \times c_0 + n' \times c_1
\]

Where:
- \( c_0 = (1 + 2 + r + 1) = 3 + r \),
\( r_i \) = The number of base repetitions with respect to \( i^{th} \) reference base, 
\( c_j = (1 * 2 + 1) = 3. \)

Therefore, \( N'' = n * 3 + \sum_{i=1}^{n} r_i + n' * 3 \)

\[
= 3n + \sum_{i=1}^{n} r_i + 3n' \\
= 3n + \sum_{i=1}^{n} r_i + 3n' \\
= \sum_{i=1}^{n} r_i + 3(n + n')
\]

### 3.1.3. Method 3

Here the assigned bit for each individual repetitive base is 0 and bit 1 for each non-repetitive base. It follows greedy algorithm. The codeword for identical part is \( RB_0 \) and \( RB_1 \) for non-identical characters. The bits length by \( M3 \) follows the equation below,

\[ N''' = n * c_0 + n' * c_1 \]

Where:
\( c_0 = (1 * 2 + 1) = 3, \)
\( c_1 = (1 * 2 + 1) = 3, \)

Hence, \( N''' = 3n + 3n' = 3(n + n') \)

### 3.1.4. Method 4

Divide each sequence into disjoint segments size 3 using divide-and-conquer algorithm. Bit flag 0 for exact three repetitive bases and bit flag 1 for any unmatched base. The codeword of indistinguishable segments is \( B_0R \) and for distinguishable segments \( B_1RRR \). For the last segments of length < 3; if any the coding is \( R_k \), where \( 0 < k < 3 \). The total number of bits is obtained by the equation given below,

\[
N'''' = n * c_0 + n' * c_1 + R_k
\]

Where:
\( R_k = n'' * 2 \)
\( n'' = \) Number of bases (0 < Length < 3),
\( c_0 = (1 * 2 + 1) = 3, \)
\( c_1 = (3 * 2 + 1) = 7. \)

So, \( N'''' = n * c_0 + n' * c_1 + n'' * 2 \)

\[ = 3n + 7n' + 2n'' \]
3.2. Combined Method

The basic versions of the each method have been discussed in the previous section. The following section will discuss the combined edition i.e. SBVRLDNAComp algorithm. The space complexity is \( O(n) \), where \( n \) is the number of bases in a sequence. SBVRLDNAComp first calculates the number of bits needed by each component methods sequentially; then compares between the results obtained \( N', N'', N''', \) and \( N'''' \); chooses the optimal one and the corresponding optimal method suited for a particular sequence. It uses substitutional method; form the optimal generated bit pattern intermediate file. The number of repeated bases or segments \( r \) varies from method to method. For first component \( r = 8 \), but second module \( r \) is unpredictable, third one does not have \( r \) dependency and for last part \( r = 3 \) respectively. An outline is shown in Fig. 2.

Input:

1: A DNA sequence \( S \)
2: Three bits coding rules: \{\( S_1, S_2, S_3, S_4, S_5, S_6, S_7 \) and \( S_8 \)\}
3: Flag variable \( v \) for repeat \( B_0 \) and non-repeat \( B_1 \) segments
4: Two bits coding rule (A – 00, C – 01, G -10 and T - 11)
5: Count bits \( c \)

Output:

1: Compressed sequence \( S' \)
2: ‘\( \alpha \)' bits/bases (bpb)

Algorithm:

1: Store the current character of remaining \( S \) into \( R \)
2: Check the next character relative to \( R \), store it to \( T \)
3: \( c \leftarrow 0 \)
4: while \( R \neq \) null do
5: \hspace{1em} if \( R = T \) then
6: \hspace{2em} \( v = B_0 \)
7: \hspace{1em} else
8: \hspace{2em} \( v = B_1 \)
9: \hspace{1em} end if
10: The compressed codes are \((B_0R_{1, r})\) or \((B_1R)\); \((RB_{1, r}B_0)\) or \((RB_0)\); \((RB_1)\) or \((RB_0)\); \((B_0R)\) or \((B_1R_{1, r})\) or \((R_{1, r})\)
11: Update \( c \) according to coding Splice
12: end while
13: The total number of bits \( c \) and optimal method for a specific sequence is obtained
14: Compressed by optimal method
15: Use LZ77 for final stage compression

Fig. 2. Outline of the SBVRLDNAComp algorithm.
4. Results

This section evaluates the performance of the SBVRLDNAComp by applying on ten standards DNA sequences [14], details of these sequences are summarized in Table 1. Although SBVRLDNAComp performance has been test on 10 standard DNA sequence of small size; this algorithm can be applied on any DNA or RNA sequence of any size.

In the following Table 2 other concerns of data compression such as compression ratio has been discussed of different sequences. In best case it takes 1.7009 bpb. The definition of compression ratio $\alpha$ is the number of characters after compression $l$ divided by the same before compression $n$.

$$\alpha = \frac{l}{n}$$
$$= \frac{l \times 8}{n} \text{ bpb}$$
$$= \left(\frac{N_{opt}}{n}\right) \text{ bpb}$$

Where,
$$N_{opt} = l \times 8$$

The average bits per base of the proposed algorithm and other existing methods on DNA sequences are illustrated in Fig. 3, it shows that the SBVRLDNAComp achieves the best compression ratio among all other algorithms.

The algorithm is implemented by Java6, on a Core 2 Due processor with a 2 GB of RAM and OS is Fedora 19. The compression ratio considered to be excellent to the best of author’s knowledge. But it can vary slightly depends on the machine hardware and software.

Table 1. Information of ten standard benchmark DNA sequences.

<table>
<thead>
<tr>
<th>Sequences name</th>
<th>Source</th>
<th>File size(KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>chmpxx</td>
<td>Chloroplasts</td>
<td>118.1875</td>
</tr>
<tr>
<td>chntxx</td>
<td></td>
<td>152.1914</td>
</tr>
<tr>
<td>humdystrop</td>
<td>Human</td>
<td>37.8613</td>
</tr>
<tr>
<td>humghcsa</td>
<td></td>
<td>64.9365</td>
</tr>
<tr>
<td>humhdabcd</td>
<td></td>
<td>57.4844</td>
</tr>
<tr>
<td>humhprt</td>
<td></td>
<td>55.4072</td>
</tr>
<tr>
<td>mptomtcg</td>
<td>Mitochondria</td>
<td>182.2344</td>
</tr>
<tr>
<td>mtpacg</td>
<td></td>
<td>97.9629</td>
</tr>
<tr>
<td>hehmvvcg</td>
<td>Viruses</td>
<td>223.9785</td>
</tr>
<tr>
<td>vaccg</td>
<td></td>
<td>187.2431</td>
</tr>
</tbody>
</table>

Table 2. The compression ratio (bpb) for the ten benchmark DNA sequences after compression.

<table>
<thead>
<tr>
<th>Sequences name</th>
<th>Number of characters before compression ($N$)</th>
<th>Number of characters after compression ($L$)</th>
<th>$N_{opt}$</th>
<th>Compression ratio ($\alpha$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>chmpxx</td>
<td>121024</td>
<td>25746</td>
<td>205971</td>
<td>1.7019</td>
</tr>
</tbody>
</table>
5. Discussions

It is not likely that exactly one compression strategy will be optimal for diverse DNA sequence. Different experimental results are going to show various bases distributions whereby one compression strategy can be more efficient than another. We have proposed four new compressions methods specialized on searching redundant substrings on highly repetitive sequences. The first one is particularly significant for medium repeat value of $r = 9$; whereas the second one is relevant for large $r$ values $r > 9$, third one is applicable for small $r$ value $r = 2$ and the final one stands out on extremely uniform repetitive collections with the small segments of size $r = 3$. Depending on the repetition of bases one of the four modules gives extremely good quality result. So for any type of exact base repeat SBVRLDNABase surpass the other standard techniques.
Acknowledgements

This work is supported in part by the Bioinformatics Centre of Bose Institute, Computer Science and Engineering Department of University of Calcutta and Academy of Technology under WBUT. We thank Dr. Zumur Ghosh and Arijita Sarkar.

References


Authors

Subhankar Roy is currently an Assistant Professor in the Department of Computer Science & Engineering, Academy of Technology, India. He has received his B. Tech and M. Tech degree in Computer Science and Engineering both from University of Calcutta, India in 2010 and 2012 respectively. His research interests are in the areas of Bioinformatics and compression techniques. He is a member of IEEE.

Akash Bhagat is presently working as Faculty at Mahendra Educational Pvt. Ltd., Asansol, India. He has received his MCA degree from Academy of Technology, India in 2015. His research interests include Bioinformatics and DNA data compression techniques.

Kumari Annapurna Sharma is presently working as Project Engineer at WIPRO-Project SHELL, India. He has received his MCA degree from Academy of Technology, India in 2015. His research interests include Bioinformatics and DNA data compression techniques.

Sunirmal Khatua is currently an Assistant Professor in the Department of Computer Science and Engineering, University of Calcutta, India. He has received the M.E. degree in Computer Science and Engineering from Jadavpur University, India in 2006. He is also pursuing his PhD in Cloud Computing from Jadavpur University. His research interests are in the areas of Distributed Computing, Cloud Computing, Bioinformatics, and Sensor Networks. He is a member of IEEE.