International Journal of Hybrid Information Technology
Vol.8, No.11 (2015), pp.323-332
http://dx.doi.org/10.14257/ijhit.2015.8.11.28

An Efficient Compression Algorithm for Forthcoming New Species

Subhankar Roy[1], Sudip Mondal[2], Sunirmal Khatua[2] and Moumita Biswas[2]

[1]*Department of Computer Science and Engineering, Academy of Technology, G. T. Road, Aedconagar, Hooghly - 712121, W.B., India*
Department of Computer Science and Engineering, University of Calcutta, 92, A.P.C. Road, Kolkata - 700009, W.B., India
[1]*subhankar.roy2012@yahoo.co.in,* [2]*sudip.wbsu@gmail.com,*
[2]*skhatuacomp@caluniv.ac.in,* [2]*mmoumitabiswas@gmail.com*

Abstract

Genomic repositories gradually increase individual and reference sequences, which shares long identical and near-identical strings of nucleotides. In this paper a lossless DNA data compression technique called Optimized Base Repeat Length DNA Compression (OBRLDNAComp) has been proposed, based upon redundancy of DNA sequences. For easy storage, retrieval time reducing and to find similarity within and between sequences compression is mandatory. OBRLDNAComp searches long identical and near-identical strings of nucleotides which are overlooked by other DNA specific compression algorithms. This technique is an optimal solution of longest possible exact repeat benefits towards compression ratio. It scans a sequence horizontally from left to right to find statistic of repeats then follow substitution technique to compress those repeats. The algorithm is straightforward and does not need any external reference file; it scans the individual file for compression and decompression. The achieved compression ratio 1.673 bpb outperforms many non-reference based compression methods.

Keywords: *Redundancy, Reference genome, Longest Exact Repeats, Non-repeat, LZ77, and Compression Ratio*

1. Introduction

The rate of increasing of genome databases is the motivating factor towards compression. In the Release 210 in Oct 2015, the number of bases and the number of sequence records were 202237081559 and 188372017 respectively for GenBank and WGS [1] had 1222635267498 numbers of bases and 309198943 numbers of sequences.

Compressed sequence reduces the transmission cost over network have limited bandwidth and storage cost. It can be used to get the similarities within sequences efficiently.

DNA sequences consists of four nucleotides bases A, C, G and T *i.e.*, two bits is enough to code each base, in spite of this fact, the standard compression algorithm like "COMPRESS", "GZIP", or "BZIP2" uses more than 2 bits per base [2]. Due to the different probability of occurrence of the symbols both static and adaptive Huffman's code fails badly on DNA sequences. Each DNA has a double stranded molecule structure by hydrogen bonding between the bases. Bases A pair with T, C with G and vice versa. Only one strand compression is required. Our aspiration is to searching this redundancy and uses it in compression at optimal level.

*Corresponding Author: Subhankar Roy
Department of Computer Science and Engineering, Academy of Technology, Hooghly, India

ISSN: 1738-9968 IJHIT

The greater part of genomic data compressing focused on the difference between the newly data that should be compressed with a reference sequence and find out the differences [3-5]. But for new species due to the lack of standard reference databases it may give less satisfactory result.

The general approach of encoding the DNA sequence is used by 2 bits for each nucleotide i.e. A = 00, C = 01, G = 10 and T = 11 and it can cut the file size near about one fourth of total size [6]. Although it is very simple but gives poor result.

The different properties within and between sequences are complementary string, palindrome string or reverse complements, cross-chromosomal similarity, approximate repeat, direct repeat, *etc*. [7]. The proposed algorithm OBRLDNAComp based on optimized exact repeat length. Both repetitive and non-repetitive parts are compressed. By this approach even a non-redundant sequence gives a good compression ratio.

The description of a number of other genome compression algorithms (Section 2), our approach (Section 3), and results (Section 4) are followed by concluding remarks of methods (Section 5).

2. Background

Reference based genome sequence compression algorithms can't be applicable for new species. Sequence compression algorithms can be classified as Bit Manipulation, Dictionary-based, Statistical and Referential Algorithms. Now the different approach of sequence compression based on different properties are organized below.

Algorithm using exact repeats are starts with BioCompress [8], BioCompress-2 [9], Cfact [10], Off-Line [11], DNASC [12], DNABIT [13], B2DNR [14] and SBVRLDNAComp [15].

BioCompress proposed by S. Grumbach and F. Tahi search the presence of palindromes sequence. Even though result is less satisfactory but better than the existing general purpose encoding. Its second version is BioCompress-2 using LZ77 searches for longest exact repeats or longest palindrome or reverse complement in previously encoded sequences, then encodes by a pair of integers (l, p); where l is the repeat length and p is the first position of preceding repeat. For non repetition it uses order-2 arithmetic coding. The difference between Biocompress and Biocompress-2 is the addition of order-2 arithmetic coding.

Cfact is a sequentially two phase algorithm. The first phase is called parsing phase which obtained the longest repeated factors using a suffix tree. The second phase compresses first occurrences of repetitive segments and all non repetitive segments using 2-bit method. The repeated segment is represented by (*pos*, *len*) tuples.

Off-Line is analogous to Cfact. It uses a suffix tree to find out the exact repeated substring. But augmented suffix tree 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 characters. The bpb of Off-line is 1.97.

The algorithms discussed so far considered only the frequent bases *i.e.*, A, C, G and T. But DNASC have taken one of the infrequent nucleotides N; which has equal probability of being A, C, G or T. Both DNA and RNA can be compressed by this technique just by replacing T with U. Horizontally compressing are followed by vertically compression. For the former one it follows LZ style with window size 128 bases. Compression of the next block with respect to the current block is done by one of the 22 ways of redundancy.

Rajeswari et al. is proposed a bit oriented compression technique called DNABIT compresses into two phases: Even bit or 2-bit technique and one of the four Odd-bit

techniques namely 3bit technique, 5bit technique, 7bit technique, and 9bit technique.

B2 and B2DNR considering 4 frequent and 11 infrequent bases. The rare characters are {K, M, R, S, W, Y, B, D, H, V, N} [16]. It compresses a sequence using static LUT.

SBVRLDNAComp algorithm gives an optimal solution of four proposed methods by searching the exact repeats in different ways. Method1 is particularly significant for medium repeated segment ($r = 9$); whereas the second method is related for large *segment* ($r > 9$), next one is applicable for small r ($r = 2$) and the final method use for extremely uniform repetitive collections with the small segments ($r = 3$).

Approximate repeat using algorithms are GenCompress [17], DNACompress [18], DNAPack [19], and GeNML [20].

One of the LZ77 based algorithm is GenCompress. GenCompress uses both approximate repeats and reverse complements and also uses reverse complements that contain errors. Three standard edit operations Replace, Insert and Delete are used. The two versions of GenCompress are: GenCompress-1 and GenCompress-2. Former one uses Hamming distance, *i.e.*, searches approximate repeats with replacement or substitution operations only, and later one uses edit distance i.e. the operation insert and delete. Mutation and crossover are detected by this algorithm. Arithmetic order 2 encoding is used for non repetition region.

Another two phase's compression algorithm is DNACompress. Using a tool called PatternHunter [21] in the first phase it finds all approximate repeats with highest score including complemented palindromes and in the nest phase it compress the approximate repeat regions and non-repeat regions.

DNAPack 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. The non-copied regions are compressed 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. Both substitution and statistical styles are used. An inexact repeat is encoded using a pointer to an earlier instance of the subsequence followed by substitution, insertion or deletion operation. Using approximate repeat GeNML is better than the above three algorithms.

DNADP [22] compression algorithm is incorporated to DNAPack encoding schemes. It is used to compress the non-repeated region of a DNA sequence using Dynamic programming. A two pass compression scheme is used. In the first pass bases A and G replaced by A where T and C are represented by T. For second pass representation of A and C is A; G and T is T. In each pass each produces individual sequence. Each sequence is then converted to a square matrix row-wise, chosen greedily. The encoding function operates on that matrix i.e. an array of string. The stopping condition is identical nucleotides in each matrix.

CDNA [23], CTW+LZ [24], XM [25] and FCM-Mx [26] are through sequentially lossless compression algorithm such as PPM and other key family of this category.

CDNA based on statistical method by detecting the approximate repeat. It predicts the probability distribution by using partial matching of the current context to earlier seen substrings. To measure the inexact similarity CDNA use Hamming distance.

A non-greedy algorithm CTW+LZ 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. PPM is used to predict the probability of next symbol using preceding symbols.

XM estimates the probability of recent bases with multiple "experts" but based on PPM. Once the symbol's probability distribution is determined, it is compressed by using a primary compression algorithm such as arithmetic coding.

Using eight finite-context models FCM-Mx compress a sequence, with orders from 2 to 16 in step 2. Through it is a recursive procedure; using weight calculation it finds out the average probabilities of A, T, C and G.

Compression algorithm based on inter sequence comparison are GRS [27] and COMRAD [28].

Genome ReSequencing data (GRS) is a compression tool for storing and analyzing the sequence. It compressed a sequence based on reference genome sequence without dealing with any other information about those sequences. GRS does not use reference single nucleotide polymorphisms (SNPs) map to process the genome and information on deletions and insertions.

COMRAD algorithm is based on RAY [29] which is a general-purpose compression algorithm. COMRAD reduce the costs for DNA compression compares to RAY. It identifies the long range repetitions. It constructs a dictionary for repeat identification in large DNA data sets. Search for exact repeated content in collections input sequences. Although COMRAD compresses the data is an expensive multiple passes process, but the execution time is reasonable and low space requirement.

An efficient compression tool for new species is KungFQ [30] compresses the fastq files. It takes the advantages of fastq characteristics and compresses the files in an optimized binary format which can be further compressed by standard tools (such as GZIP or LZMA). This approach is useful for new species because it does not need any reference file. After recognizing the ID it compresses the fixed part only once while the new portion is encoded for every read. It has almost constant memory requirement and have both the option of lossless and lossy compression.

Our algorithm OBRLDNAComp overcomes the optimal exact repeat searching within a sequence. In the following section we clarify OBRLDNAComp algorithm in details and all the associated components methods that OBRLDNAComp invoke; and then experimental results. Comparison with other algorithm is also enlightened in the result.

3. Proposed Method

The rope of nucleotides making up a DNA sequence can be represented as a string of symbols of alphabet $\sum$ = {A, C, G, T or U}, which are the standard DNA characters. The algorithm OBRLDNAComp is specifically designed for encoding those alphabets. It is also suitable for RNA sequence but not for proteins. It is a two-pass algorithm for compressing a set of DNA sequence. In the first pass it searches the optimal repeat length r_{opt}, which will give maximum profit in compression i.e. minimum number of compressed bits. A sequence is scanned horizontally from left to right followed by vertical scanning from top to bottom. The actual encoding is happened at the second pass.

3.1. First Pass

3.1.1. Longest Repeated Base Searching: Let the number of bases within one strand of a DNA is *n*. For each sequence *S*, from least 2 to longest possible repeat length *l* being search initially, where $2 <= l <= n-1$. Starting at point *i* and ending at

index *j*, searching stops when no more consecutive next identical character is found. Same repeated segment may have multiple copies. For each sequences it proceed in the same way. So there have no dependency between sequences.
For example in the following sequence:

AACCTTTTTGGGGGCTTTTACCCCCCCCCCG

The repeated segments lengths are $l = \{2, 4, 5, 10\}$, where $l_{max.} = 10$.

3.1.2. Optimal Repeat Length: After getting all l values, the number of occurrence c; of each repeat segment of length l is obtained. So in a sequence the number of the identical bases is $l_i * c_i$ and non-identical bases is $n - l_i * c_i$,, where $2 =< i <= l_{max.}$. The number of bits needed to encode the former one is constant i.e. 3 bits, where 2 bits for individual alphabets and one flag bit either 0 or 1 to distinguish between two parts. To encode partial or fully in-exact segments the number of bits needed have l_i dependency using the formula $1 + l_i * 2$. Number of bits required to encode each exact segments are $c_i * 3$. For each in-exact region necessary bits is $(n - l_i * c_i) / l_i * (1 + l_i * 2)$ and for last if there in-exact region it is $(n - l_i * c_i) \% l_i * 2$. Therefore total number of bits $T_i = (c_i * 3) + (n - l_i * c_i) / l_i * (1 + l_i * 2) + (n - l_i * c_i) \% l_i * 2$. Therefore $T_2 = 43$, $T_4 = 54$, $T_5 = 52$ and $T_{10} = 47$ respectively.

After getting all repeat length *l* and the corresponding total number of bits *T;* the optimal repeats length $l_{opt.}$, is obtained from minimum *T* value. *lopt.* = 2 for the above sequence example.
Total number of bits after compression is obtained by the following formula,

$T_{min.} = (c_{opt.} * r') + (t_2 / l_{opt.} * r'') + (t_2 \% l_{opt.} * 2)$

Where:

$l_{opt.}$ = Optimal segment repeat length
$c_{opt.}$ = Optimal number of segments of length $l_{opt.}$
$t_1 = l_{opt.} * c_{opt.}$ = Total number of repetitive bases
n = Number of bases in a sequence
$t_2 = n - t_1$ = Total number of non-repetitive bases
$r' = (1+2) = 3$ = Number of bits for repetitive segment
$r'' = (1 + l_{opt.} * 2)$ = Number of bits for non-repetitive segment
$n_1 = c_{opt.} * r'$ = Total number of bits for repetitive segments
$n_2 = (t_2 / l_{opt.} * r'') + (t_2 \% l_{opt.} * 2)$ = Total number of bits for non-repetitive segments

3.2. Second Pass

3.2.1. Binary Stream Formation: The actual encoding using $l_{opt.}$, is happened in this pass. Any sequence is divided into segments of length $l_{opt.}$. Then encode each segment using the coding rule as explained above for identical and non-identical parts. A temporary encoded file stores the bits stream before converting to final compressed file.

3.2.2. Substitution: The final compressed file is obtained by the 8 bits to 1 character mapping rule dynamically for *e.g.*, $(01000001)_2 = (65)_{10} = A$. The motive behind this concept is that any general purpose compression algorithm for e.g. LZ77 can be use on the generated file for final phase compression. As LZ77 is a window based algorithm, for DNA sequence it is more appropriate. In DNA sequence redundancy may occur at very long distances whereas for plain text it is local. So a variable window size as input is taken to obtain the best result from a set of experiment.

3.3. OBRLDNAComp

An outline of general algorithm needed when this technique is applied on some standard DNA sequence has been discussed in this section. The first pass includes Longest Repeated Bases (LRB) algorithm followed by Optimized Repeat Length (ORL). LRB returns $l_{max.}$; within individual sequence, where 2 =< max <= n-1. From each possible repeat length l = {l_1, l_2, l_3… l_{n-1}}; the calculated optimum repeat length $l_{opt.}$; is returned by ORL. Each segment is $R_{seg.}$; and its length $L_{seg.}$, where $l_{opt.}$ => $L_{seg.}$.

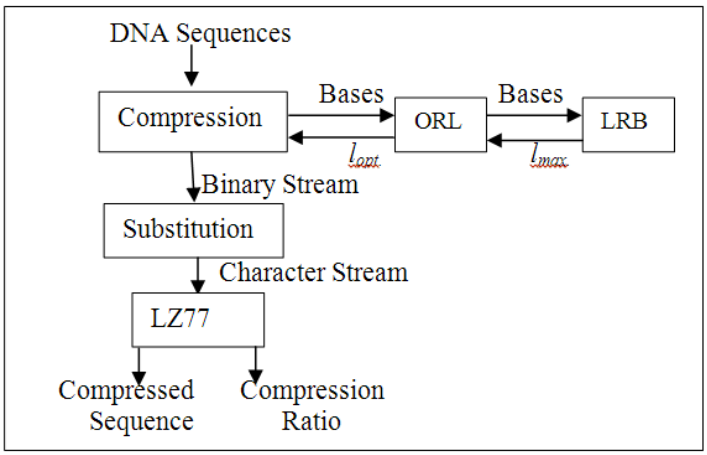

Figure 1. OBRLDNAComp Compression Structure

Algorithm (1): Procedure of Compression (OBRLDNAComp)
Input:
1: A DNA sequence *S*
2. Flag variable *v* for repeat B_0 and non-repeat B_1 segments
3: Two bits coding rule (A – 00, C – 01, G -10 and T - 11)
4: Count bits *c*
Output:
1: Compressed sequence *S*'
2: 'α' bits/bases (bpb)
Algorithm:
1: ORL algorithm
2: Divide each sequence of segment length $l_{opt.}$
3: **while** $R_{seg.}$!= null **do**
4: **if** $L_{seg.} = l_{opt.}$ **then**
5: **if** $R_{seg.}$ is identical **then**
6: Assign control bit followed by 2-bit code for the base
7: **else**
8: Assign control bit followed by 2-bit code for each base
9: **end if**
10: **else**
11: 2-bit encoding for each base
12: **end if**
13: **end while**
14: Fixed length Substitution algorithm
15: Variable length LZ77 algorithm

Algorithm (2): Procedure of ORL

1: LRB algorithm
2: $l \leftarrow 2$
3: **while** $l <= l_{max.}$ **do**
4: $c \leftarrow 0$
5: **while** $R_{seg.}$!= null **do**
6: **if** $L_{seg.} = l$ **then**
7: **if** $R_{seg.}$ is identical **then**
8: $c \leftarrow c+1$
9: **end if**
10: **end if**
11: **end while**
12: Bits required for each l is obtained by the above formula
13: **end while**
14: return $l_{opt.}$

Algorithm (3): Procedure of LRB

1: Store the current reference base in R and next base to T
2: $l \leftarrow 0$
3: **while** R != null **do**
4: **if** $R = T$ **then**
5: $l \leftarrow l + 1$
6: **else**
7: Update R and T
8: Store l in a list
9: **end if**
10: $l \leftarrow 0$
11: **end while**
12: return $l_{max.}$

3.4. Decompression

The first character of the compressed file is $l_{opt.}$; is the control character for that particular sequence. Apply LZ77 decompression algorithm before substitution. After reverse substitution algorithm $l_{opt.}$; come to act. The original file is obtained in accordance to that.

4. Experimental Results

The previous section has demonstrated the optimal number of bits required by the proposed methods. This section evaluates the performance of the methods by applying on ten standard DNA sequences [31], which are summarized in Table 1.

In the following other concerns of data compression such as compression ratio will be discussed. The definition of compression ratio α is the sequence length after compression divided by the sequence length before compression.

Although the compression ratio of OBRLDNAComp is not competitive with reference based compression algorithm like COMRAD, but it is extremely good for new species.

α = (Sequence length after compression/Sequence length before compression)*8 bpb

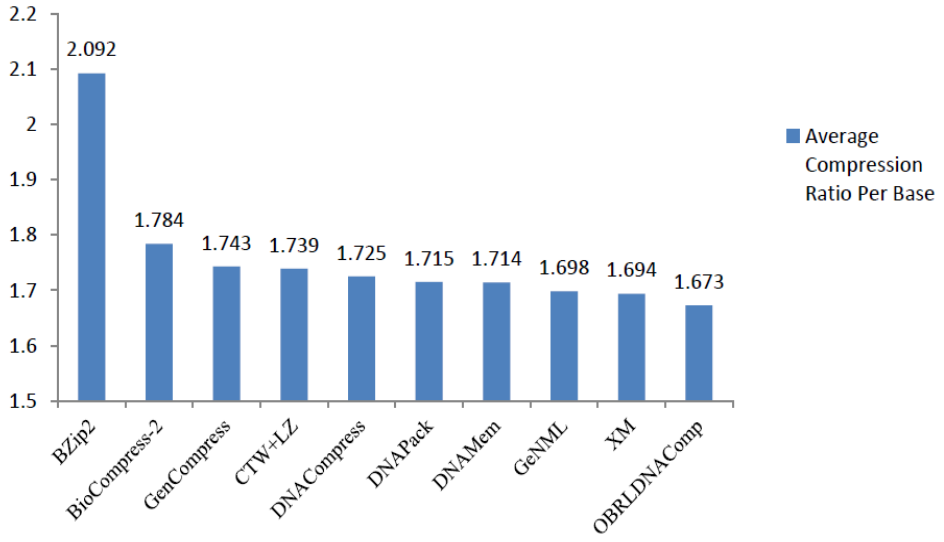

Figure 2. Average Bits per Base for DNA Compression Algorithms

Table 1. Bits Per Base for the Benchmark DNA Sequences After Compression

Sequences Name	Number of characters before compression (n)	Number of characters after compression (n')	$T = n'*8$	Compression ratio ($\alpha = T/n$)
Chmpxx	121024	25262	202098	1.6699
Chntxx	155,844	33084	264670	1.6983
humdystrop	38,770	8211	65684	1.6942
Humghcsa	66,495	13934	111472	1.6764
humhdabcd	58,864	12244	97956	1.6641
Humhprtb	56,737	11926	95409	1.6816
Mpomtcg	186,608	39897	319174	1.7104
Mtpacg	100,314	20549	164375	1.6386
Hehcmvcg	229,354	47270	378159	1.6488
Vaccg	191,737	39409	315274	1.6443

5. Concluding Remarks

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 a new compression method specialized on searching redundant optimal substrings on highly repetitive sequences. So for any type of exact base repeat OBRLDNAComp surpass the other standard techniques operating on newly created genome sequence.

References

[1] http://www.ncbi.nlm.nih.gov/genbank/statistics.
[2] T. V. Mridula and P. Samuel, "Lossless segment based DNA compression", Proceedings of the 3rd International Conference on Electronics Computer Technology, IEEE Xplore Press, **(2011)**, pp. 298-302.
[3] C. Kozanitis, C. Saunders, S. Kruglyak, V. Bafna and G. Varghese, "Compressing Genomic Sequence Fragments Using Slimgene", Research in Computational Molecular Biology, vol. 6044, **(2010)**, pp. 310-324.

[4] K. Daily, P. Rigor, S. Christley, X. Xie and P. Baldi, "Data Structures and Compression Algorithms for High-Throughput Sequencing Technologies", BMC Bioinformatics, vol. 11, no. 1, article 514**, (2010)**.
[5] M. H.-Y. Fritz, R. Leinonen, G. Cochrane and E. Birney, "Efficient Storage of High Throughput DNA Sequencing Data Using Reference-Based Compression", Genome Research, vol. 21, no. 5, **(2011)** May, pp. 734-740.
[6] S. Roy, S. Khatua, S. Roy and S. K. Bandyopadhyay, "An Efficient Biological Sequence Compression Technique Using LUT and Repeat in the Sequence", IOSRJCE, vol. 6, Issue 1, **(2012)** September-October, pp. 42-50.
[7] S. Roy and S. Khatua, "DNA DATA COMPRESSION ALGORITHMS BASED ON REDUNDANCY", IJFCST, vol. 4, no. 6, **(2014)** November, pp. 49-58.
[8] S. Grumbach and F. Tahi, "Compression of DNA sequences", IEEE Symp. on the Data Compression Conf., DCC-93, Snowbird, UT, **(1993)**, pp. 340–350.
[9] S. Grumbach and F. Tahi, "A new challenge for compression algorithms: genetic sequences", Info. Process. & Manage, Elsevier, **(1994)**, pp.875-866.
[10] E. Rivals, J. Delahaye, M. Dauchet and O. Delgrange, "A Guaranteed Compression Scheme for Repetitive DNA Sequences", DCC '96: Proc. Data Compression Conf., **(1996)**, pp. 453.
[11] A. Apostolico and S. Lonardi, "Compression of Biological Sequences by Greedy Off-Line Textual Substitution", DCC '00: Proc. Data Compression Conf., **(2000)**, pp. 143-152.
[12] K. N. Mishra, A. Aaggarwal, E. Abdelhadi and P. C. Srivastava, "An Efficient Horizontal and Vertical Method for Online DNA Sequence Compression", IJCA, vol. 3, no. 1, **(2010)** June, pp. 39-46.
[13] P. R. Rajeswari and Dr. A. AppaRao, "DNABIT Compress – Genome compression algorithm", Bioinformation,vol. 5 no. 8, **(2011)** January, pp. 350-360.
[14] S. Roy and S. Khatua, "Compression Algorithm for all Specified Bases in Nucleic Acid Sequences", IJCA, vol. 75, no. 4, **(2013)** August, pp. 29-34.
[15] S. Roy, A. Bhagat, K. A. Sharma and S. Khatua, "SBVRLDNAComp: An Effective DNA Sequence Compression Algorithm", IJCSA, vol. 5, no. 4, **(2015)** August, pp. 73-85.
[16] Department of Chemistry, Queen Mary University of London, "Nomenclature for Incompletely Specified Bases in Nucleic Acid Sequences".
[17] X. Chen, S. Kwong and M. 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, **(2001)** July/August, pp. 61-66.
[18] X. Chen, M. Li, B. Ma and J. Tromp, "DNACompress: Fast and Effective DNA Sequence Compression", Bioinformatics, vol. 18, **(2002)** June, pp. 1696-1698.
[19] B. Behzadi and F. L. Fessant, "DNA Compression Challenge Revisited: A Dynamic Programming Approach", CPM '05: Proc. 16th Ann. Symp. Combinatorial Pattern Matching, **(2005)**, pp. 190- 200.
[20] G. Korodi and I. Tabus, "An Efficient Normalized MaximumLikelihood Algorithm for DNA Sequence Compression", ACM Trans. Information Systems, vol. 23, no. 1, **(2005)**, pp. 3-34.
[21] B. Ma, J. Tromp and M. Li, "PatternHunter—faster and more sensitive homology search", *Bioinformatics*, vol. 18, **(2002)**, pp. 440–445.
[22] K. G. Srinivasa, M. Jagadish, K. R. Venugopal and L. M. Patnaik, "Efficient Compression of non-repetitive DNA sequences using Dynamic Programming", IEEE, **(2006)**.
[23] D. Loewenstern and P. Yianilos, "Significantly Lower Entropy Estimates for Natural DNA Sequences", DCC '97: Proc. Data Compression Conf., **(1997)**, pp. 151.
[24] T. Matsumoto, K. Sadakane and H. Imai, "Biological Sequence Compression Algorithms", Genome Informatics, **(2000)**, pp. 43–52.
[25] M. D. Cao, T. Dix, L. Allison and C. Mears, "A Simple Statistical Algorithm for Biological Sequence Compression", DCC '07: Proc. Data Compression Conf., **(2007)**, pp. 43-52.
[26] A. J. Pinho, D. Pratas and P. J. S. G. Ferreira, "Bacteria DNA sequence compression using a mixture of finite-context models", IEEE Statistical Signal Processing Workshop (SSP), IEEE Xplore Press, DOI: 10.1109/SSP.2011.5967637, **(2011)**, pp. 125-128.
[27] C. Wang and D. Zhang, "A novel compression tool for efficient storage of genome resequencing data" Nucl. Acid Res., vol. 39, no. 7, **(2011)** January, e45. DOI: 10.1093/nar/gkr009.
[28] S. Kuruppu, B. Beresford-Smith, T. Conway and J. Zobel, "Iterative dictionary construction for compression of large DNA data sets", Proceedings of the IEEE/ACM Transactions on Computational Biology and Bioinformatics, vol. 9, no. 1, **(2012)** January/February, pp. 137-149.
[29] A. Cannane and H. Williams, "General-Purpose Compression for Efficient Retrieval," J. Am. Soc. for Information Science and Technology, vol. 52, no. 5, **(2001)**, pp. 430-437.
[30] E. Grassi, F.D. Gregorio and I. Molineris, "KungFQ:A simple and powerful approach to compress fastq Files", IEEE/ACM Trans. on Comput. Biol. and Bioinform., vol. 9, no. 6, **(2012)** November/December, pp. 1837-1842.
[31] The GeNML homepage: http://www.cs.tut.fi/~tabus/genml/results.html.

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.

Sudip Mondal, is currently working as a Lecturer in the Department of Computer Science, Acharya Prafulla Chandra College, India. He has received his M. Sc and M .Tech degree in Computer Science and Information Technology from West Bengal State University and University of Calcutta, India in 2011 and 2014 respectively. He is doing research work from University of Calcutta under the guidance of Mr. Sunirmal Khatua. His research interests include Bioinformatics and Computational Biology.

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.

Moumita Biswas, is currently a Guest Lecturer in the Department of Computer Science at Vidyasagar College and Panihati Mahavidyalaya, India. She has received her M.Tech in CSE, M.Sc. in CIS and B.Sc in CS from Scottish Church College, University of Calcutta, India. Her research interests are in the areas of Database Management Systems, Object Oriented Programming and Bioinformatics.