A Pattern Matching Extended Compression Algorithm for DNA Sequences

Murugan. A[1] and Punitha. K[2],
amurugan1972@gmail.com punithsathish@gmail.com
Dr. Ambedkar Government College, Chennai, India, Agurchand Manmull Jain College, Chennai, India

Abstract
DNA sequencing provides fundamental data in genomics, bioinformatics, biology and many other research areas. With the emergent evolution in DNA sequencing technology, a massive amount of genomic data is produced every day, mainly DNA sequences, craving for more storage and bandwidth. Unfortunately, managing, analyzing and specifically storing these large amounts of data become a major scientific challenge for bioinformatics. Those large volumes of data also require a fast transmission, effective storage, superior functionality and provision of quick access to any record. Data storage costs have a considerable proportion of total cost in the formation and analysis of DNA sequences. In particular, there is a need of highly control of disk storage capacity of DNA sequences but the standard compression techniques unsuccessful to compress these sequences. Several specialized techniques were introduced for this purpose. Therefore, to overcome all these above challenges, lossless compression techniques have become necessary. In this paper, it is described a new DNA compression mechanism of pattern matching extended Compression algorithm that read the input sequence as segments and find the matching pattern and store it in a permanent or temporary table based on number of bases. The remaining unmatched sequence is been converted into the binary form and then it is been grouped into binary bits i.e. of seven bits and gain these bits are been converted into an ASCII form. Finally, the proposed algorithm dynamically calculates the compression ratio. Thus the results show that pattern matching extended Compression algorithm outperforms cutting-edge compressors and proves its efficiency in terms of compression ratio regardless of the file size of the data.

Key words: DNA sequence, pattern matching, lossless compression, compression ratio

1. Introduction

Sequencing techniques allow the DNA data to be represented as a long string made up from four nucleotide bases. This facilitates the computational analysis of our biological make-up for use in areas such as forensics applications, crime investigation or parental connection establishment [1]. Recent advancement in Next Generation Sequencing (NGS) techniques enables sequencing the individual genome in a fast and affordable manner. As a result, a large number of genomic data are produced for various studies. For example, the genomic data has been used for studying variations of genetic diseases on individuals so that personalised medicines and diagnostic tools can be developed [2]. Specific mutations within the genomic data have been used to study the risk of developing certain diseases [3]. The collection of data has also been used to study pattern about an organism's evolutionary history and aid in phylogenetic tree reconstruction [4].

As mentioned in [5], the genome sequence for an individual containing Adenine (A), Thymine (T), Cytosine (C) and Guanine (G) is almost incompressible. The amount of DNA being taken from organisms and order is increasing exponentially [6]. This gives in two questions- a place for storing and safe transmission. The hard question of place for storing while useful to the workplace is depending on the size of each base. The DNA order size varies from Megabyte (MB) to Terabyte (TB) annually [7]. The DNA contains some logical organization [8], hence data structure for storing, accessing and efficient processing tasks is challenging [9]. The DNA database requires an efficient compression algorithm for storing. Thus, special DNA sequence characteristics have to be explored in its compression. The basic idea in traditional DNA nucleotide sequence compression is to find identical sub-sequences within the target DNA sequence to be compressed so that they are encoded only once [10].

On the other hand, NGS techniques have introduced a new challenge in which a lot of DNA sequences have to be stored in various databases [11]. Large public databases for storing DNA sequences include the GenBank at the National Center for Biotechnology Information (NCBI) [12]. Also there are different file formats to store these DNA sequences. These file formats are used in different contexts, but they can be converted to one another easily. Although the exact details to be stored in these file formats may not be the same, they often contain two distinct parts. The first part is metadata such as sequence identifier, annotation and/or description. The second part is the actual nucleotide sequence. Currently, only general-purpose lossless compression methods such as gzip and bzip2 [13] are applied to reduce the storage.

Various approaches [14] have been adopted for DNA sequence compression. Compression plays a vital role in dealing with increasing size of sequencing data. DNA sequences can be compressed using generic approaches, as compression has natural representation as a string of characters for which a rich literature exists.

Manuscript received August 5, 2021
Manuscript revised August 20, 2021
https://doi.org/10.22937/IJCSNS.2021.21.8.26

However, genomic data are inherently redundant because same species share significant part of the genome among individuals [15]. Compressing DNA sequence can take advantage of certain biological characteristics, as repeat content and relationship to existing sequences. Reviews in [16] describe prior work on DNA sequence compression and exploiting redundancy within DNA sequences, respectively. This paper is summarized as follows. Section 2 provides review of the related work; Section 3 presents the description of this research work. In Section 4, proposed comparison of different techniques that are described in detail by different authors in different papers, results are presented and discussed. In section 5, Conclusion is provided.

2. Related work

Compression algorithms for DNA or genome generally fall into four categories. The first category is naive-bit encoding where one or more characters are represented by a certain code word. For example, the simplest encoding for DNA sequence can be obtained by assigning 2 unique pair of bits to each of the unique alphabet present in a DNA sequence like A=00, C=01, G=10, and T=11 [17]. The second category is a dictionary-based or substitutional compression. It is mostly observed that a DNA sequence consists of a lot of repeated sequences. As a result, the repeated sequences can be replaced by references to a dictionary that is either built offline or is maintained at runtime [18]. The most common dictionary based algorithms include LZ77 and LZ78. Further, there are statistical methods that achieve extremely good compression rates by generating probabilistic models based on genome datasets [19].The fourth category is referential compression where any repeated sequence in an input dataset is replaced with a reference to one or more external DNA sequences [20]. A data structure plays a critical role in any algorithm designed for achieving good compression ratios, fast searching of patterns inside sequences, or both. Many algorithms proposed in the area of bioinformatics make use of self-index-based data structures to achieve above mention goals. They can be used to avoid the need of keeping large files of text along with the index.

The index itself contains sufficient information that any part of the text can be recreated. Some of the examples of self-indexes include Compressed Suffix Array (CSA), Succinct Suffix Array (SSA), and FM-index [21]. There are various string matching techniques which mainly deal with problem of identifying occurrences of a substring in a given string or locate the occurrences of specific pattern in a sequence. In this section, we explore these different types of string matching techniques. Some techniques are based on algorithms of exact matching in string, such as Brute-force algorithm, Bayer-Moore algorithm, Knuth-Morris-Pratt algorithms [22] and some are based on approximate string matching algorithms, dynamic programming is mostly used approach. In an index based K-Partition Multiple Pattern Matching Algorithm (IBKPMPM) [23] chooses the value of k and divides both the string and pattern into number of substring of length k, each substring is called as a partition. The proposed work compare all the first characters of all the partitions, if all the characters are matching while searching then go for the second character match and the process continues till the mismatch occurs or total pattern is matched with the sequence. In index based forward backward multiple pattern matching algorithm (IFBMPM) [24] patterns matching technique the characters in the given patterns are matched one by one in the forward and backward until a mismatch occurs or a whole pattern matches. In the Multiple Skip Multiple Pattern Matching Algorithm (MSMPMA) [25] technique the algorithm search the input text to find the all occurrences of the pattern based upon the skip technique.

To get starting location of the matching Index is used; it compares the text characters from the well-defined point with the pattern characters, and based on the match numbers decides the skip value (ranges 1 to m-1). In IBSPC [26] indexes have been used for the DNA sequence. Least occurring character index will be used to search for the pattern in the string. In an Index Based Algorithm, on the basis of frequently occur character index table is created and then align pattern with string and matched occurrence of patterns with multiple times one by one from left to right in the file. This paper proposed the most efficient approach for finding similarity between multiple patterns, till the end of the sequence. To further increase the performance of pattern matching an ASCII based multiple pattern matching algorithm using ASCII value comparison between pattern and substring is proposed. It is a simple approach for finding multiple occurrences of patterns from a given file. This algorithm gives better results when compare it with existing algorithms. This approach provides best results with the DNA sequence dataset. Experimental results of applying technique to DNA sequences show effectiveness of the proposed technique.

3. Proposed work

A double helix holds the DNA together with hydrogen bonding. Each strand of the helix is a biomolecule made up of many nucleotides that are bonded together. Adenine (A), Cytosine (C), Guanine (G), and Thymine (T) are the four types of nucleotides. The helix's two strands are the exact opposites of one another. Each nucleotide on one strand corresponds to its complement on the other strand, where A corresponds to T and G corresponds to C. DNA strands that are complementary to themselves are referred to as self-complementary or palindromes. The bases at every location of the target are compared with the corresponding bases at the reference to obtain an operation code indicating differences between a target sequence and its reference.

The sensitivity and efficiency of a pattern are used in detecting approximate repeats and complementary palindromes in the input DNA sequence in the first phase. The proposed algorithm uses the standard parameters for pattern. These parameters were set and optimized based on the number of bits needed to encode a mismatching base as well as to encode a match. The output pattern contains information about each repeat such as its pattern count, start and end position, a set of edit operations, and whether it is an approximate repeat etc. The Figure 1 depicts the work flow of a proposed algorithm.

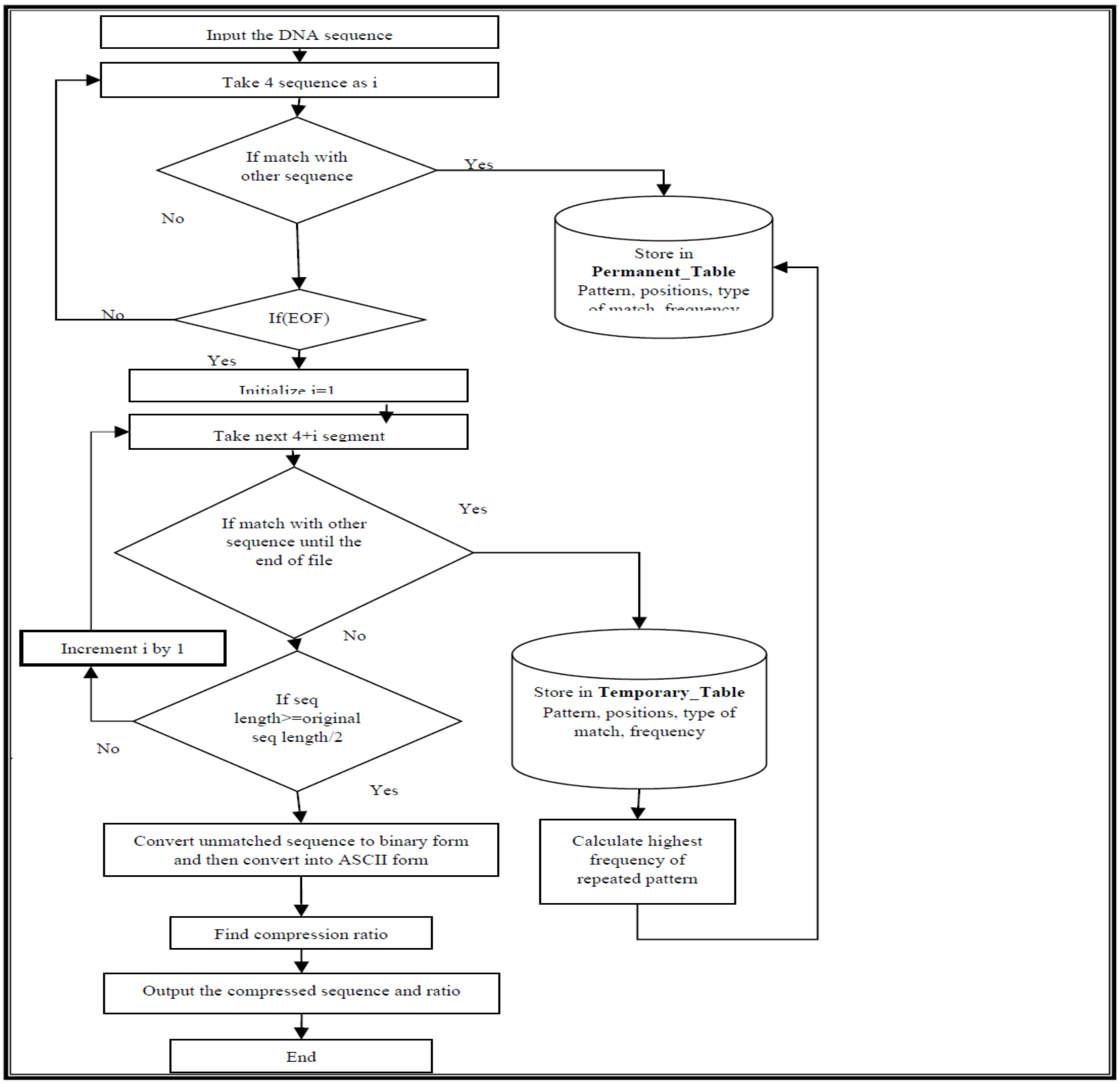
Input the DNA sequence
Take 4 sequence as i
If match with other sequence
Yes
No
Store in Permanent_Table Pattern, positions, type of match, frequency
No
If(EOF)
Yes
Initialize i=1
Take next 4+i segment
If match with other sequence until the end of file
Yes
No
Increment i by 1
If seq length>=original seq length/2
No
Yes
Store in Temporary_Table Pattern, positions, type of match, frequency
Calculate highest frequency of repeated pattern
Convert unmatched sequence to binary form and then convert into ASCII form
Find compression ratio
Output the compressed sequence and ratio
End

Fig. 1 Architecture for a Pattern Matching Extended Compression Algorithm

The proposed Pattern Matching Extended Compression Algorithm (PMECA) takes into account the unique properties of DNA sequences and employs a novel encoding method. PMECA have been divided into different phases, as seen in Figure 1. The phases are: 1) From the whole DNA sequence pattern file it takes every four, five, six, seven and so on segments which is not considered in the existing Pattern Matching Algorithm and discover matching patterns with the remaining input sequences. 2) If 4 segment matched, then the patterns have been removed from the input sequence and save it in the permanent table, if number of bases in the segment is greater than 4 then store the segment in the temporary table with pattern, position, and type of Match as EXACT or REVERSE. 3) Finding the maximum frequency for the type of Match and it is been stored in permanent table. 4) Finding matching with the remaining input sequence for each and every 5, 6, 7… segment patterns till the segments file length is less than half of the input sequence's length. The proposed algorithm makes a binary representation of the remaining unmatched sequence for the Encoding process. 5) Additionally, Convert binary bits to ASCII and display as an output by grouping binary bits into 7 bits. 6) Finally, the compression ratio is used to test the efficiency of the efficient Pattern Matching Extended Compression technique, which is defined as the rate between the size of the dictionary file and the size of the output sequence divided by the size of the original sequence multiplied by hundred.

The goal of data compression is to reduce the number of bits which are useful in storing or transporting of data. The Pattern Matching Extended Compression technique for compressing DNA sequences that is more efficient than prior techniques. To accommodate the massive genetic data, the new approach is designed to require less storage with less compression - decompression time. The current level of specialization in compression algorithms is aimed at exploiting the specific properties of each type of data to be compressed, allowing for performance optimization, which is a specialized compressor for genomic data have been developed.

3.1 Compression Algorithm

The proposed compression algorithm is the extension of the Improved_Compress algorithm[28] which gives better compression when compared to Improved_Compress algorithm by covered more number of segments.

Algorithm 1: Ext_Improved_Compress

(input_sequence)

```
Begin

        Read the first 4 bases as segment from
        input_sequence
        Call Improved_Compress(input_sequence)
        Set i = 1;
        For(x=5;x<=(input_sequence.length)/2;x++)
        {

        Read x as number of bases in the segment from
        input_sequence
        Call Improved_Compress(remaining sequence)

        }

End
```

3.2 Implementation

Step 1: Read the input sequence.

Step 2: Take the 4 segment and find the matching with remaining input sequence
AAGTGACGTATTTCCCTTTAGGGACGTTGGAGC
ATTGACCTAGGTTAGCATTACGTTCTAAAGTGTG
CAATGCCCCAAAATGGGCAAGTCGTTGCCCTTT

Step 2.1: Remove the matching pattern and store it in the permanent table

Step 3: Take 5 segment from the remaining sequence and find the matching, if match occurs then store it in the temporary table.

GATTTTTTGGGCCGTTGGAGCATTGACCTAGGTT
AGCATTACGTTCTAGATGCCCCAAAATGGGCAA
GTCGTTGTTTGGA

Step 3.1: Remove the pattern from the sequence and Store the TTGGA segment in the permanent table.

GATTTTTCGGCATTGACCTGATTTTTCGGCATTG
ACCTAGCATTACGTTCTAGATGCCCCAAAA

Step 4: Next take 6 segment and find the matching and repeat the process as in step 3 till the length of the segment is lesser than the half of the length of the input sequence.

Step 5: Encode the remaining unmatched sequence into the binary form and then group the binary bits into 7 bits and convert that into the ASCII form.

3.3 Compression Ratio

The compression ratio is calculated by using the following formula:

$$1 - \frac{(\text{Dictionary\_Size} + \text{Output\_Size})}{\text{Original DNA Input sequence\_Size}} \times 100 \qquad (1)$$

3.4 Decompression Algorithm

The decompression algorithm is responsible for getting back the original content from the compressed content. The compressed output sequence can be decompressed to the original input sequence based on the patterns stored in the permanent table's details. Decompression algorithm is the reverse of the compression process. The file matching algorithm is used to find the matching between the original input sequence of compression algorithm and the output sequence of decompression algorithm to determine that the proposed algorithm is a lossless compression algorithm.

4. Results and Discussions

To test the efficiency of the proposed Pattern Matching Extended Compression algorithm, it is been tested to a large data set of DNA sequences from NCBI repository and compared the findings to those reported for other existing DNA Compression algorithm. Human dystrophin (HUMHDYSTROP), human growth hormone (HUMGHCSA), human beta globin region on chromosome 11 (HUMHBB), human DNA sequence (HUMHDABCD), human hypoxanthine phosphoribosyl transferase gene (HUMHPRTB), and the vaccinia virus Copenhagen full genome are the reference sequences (VACCG). The suggested Pattern Matching Extended Compression technique was used to test the above data for varied sequence lengths.

4.1 Comparative Analysis

Table 1 and figure 2 shows a comparison of the proposed algorithm compression ratio with various existing compression techniques [27] [28]. The proposed algorithm gives average compression ratio of 91%.

Table 1: Comparison of proposed Algorithm with Existing Algorithms

Sequence	Lengt h	Optimal Seed based algorith m	Improv ed Compre ss algorith m	Ext_Imp roved Compres sion algorith m
HUMHDYST ROP	38,770	82%	90%	93%
HUMGHCSA	66,496	90%	89%	90%
HUMHBB	73,308	82%	89%	90%
HUMHDABC D	58,563	84%	88%	91%
HUMHPRTB	56,832	84%	89%	90%

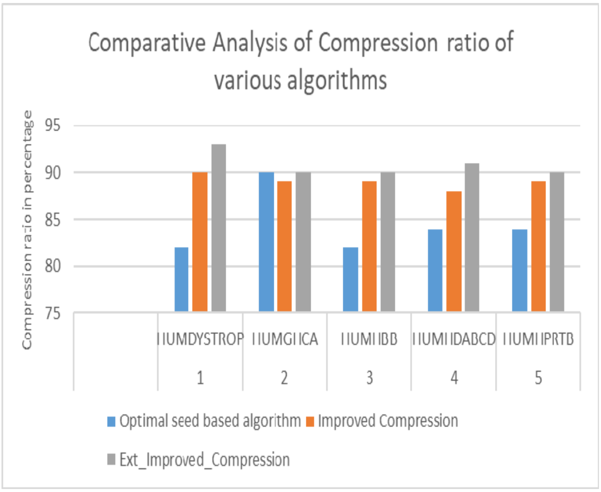

Fig. 2 Compression ratio of various compression algorithms

The Proposed algorithm computational cost is proportional to the time it takes to compute the solution for genomic compression. This computational complexity modelling of sequences is based on their similarity and coding to lower bit size. Table 2 and Figure 3 compares the proposed Pattern Matching Extended Compression algorithm to the existing approach [27] [28] in terms of time execution (in seconds). The proposed algorithm has a faster response time than the existing approach.

Table 2: Execution Time Comparison of proposed and existing algorithms

Sequence	Length	Optimal seed based algorithm	Improved Compress algorithm	Ext_Improved_ Compression algorithm
HUMHDYSTROP	38,770	1.5	1.3	1.1
HUMGHCSA	66,496	2.5	2.4	2.1
HUMHBB	73,308	2.8	2.6	2.4
HUMHDABCD	58,563	2.2	2.1	1.9
HUMHPRTB	56,832	2	1.8	1.6

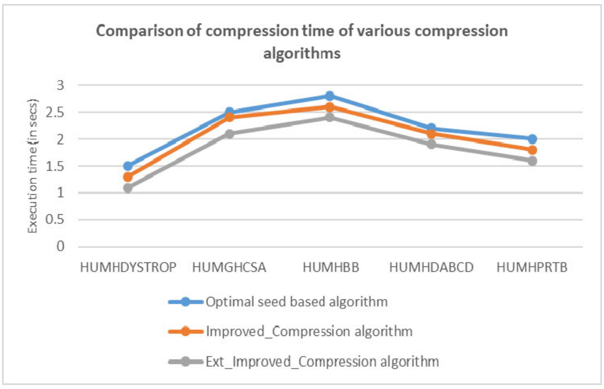

Fig. 3 Execution Time of various Compression Algorithms

The storage and processing are not unknown in the field of bioinformatics due to the large volume of DNA sequence data. In this paper, proposed a Pattern Matching Extended Compression algorithm which compresses the original size of the file with a lesser compression ratio without any data losses which is been described in Table 3.

5. Conclusions

Problems related to storage and processing are not unknown in the field of bioinformatics due to the large volume of DNA sequence data. Since in this paper, Pattern Matching Extended Compression algorithm are been explored to provide a solution other than the existing compression of DNA sequences. The proposed approach allows finding the segments and finds the matching patterns and the number of times the pattern is repeated in the sequence. The remaining unmatched series is translated to binary, then divided into binary bits (eight bits) and converted to ASCII. Finally, the proposed algorithm calculates the compression ratio dynamically. The proposed solution brings highly promising results. Thus, regardless of higher data size, the findings show that the Ext_Improved Compression algorithm outperforms with a good compressed ratio, storage capacity. In future, the experiments will carried out on DNA sequences to provide a proof of concept application of the proposed Ext_Improved Compression algorithm in real life applications.

References

[1] Bradley, P., Den Bakker, H.C., Rocha, E.P., McVean, G. and Iqbal, Z.: "*Ultrafast search of all deposited bacterial and viral genomic data*". Nature biotechnology, vol. 37(2), pp.152-159(2019)

[2] Chikhi, Rayan, Jan Holub, and Paul Medvedev: "*Data structures to represent sets of k-long DNA sequences*". *arXiv preprint arXiv:*1903.12312(2019)

[3] Břinda, Karel, Michael Baym, and Gregory Kucherov: "*Simplitigs as an efficient and scalable representation of de Bruijn graphs*". Genome biology 22, vol. 1, pp.1-24(2021)

[4] Kryukov, K., Ueda, M.T., Nakagawa, S. and Imanishi, T.: "*Nucleotide Archival Format (NAF) enables efficient lossless reference-free compression of DNA sequences*". Bioinformatics, vol. 35(19), pp.3826-3828(2019)

[5] Al-Okaily, A., Almarri, B., Al Yami, S. and Huang, C.H.: "*Toward a better compression for DNA sequences using Huffman encoding*". Journal of Computational Biology, vol. 24(4), pp.280-288(2017)

[6] Solomon, B. and Kingsford, C.: "*Fast search of thousands of short-read sequencing experiments*". Nature biotechnology, vol. 34(3), pp.300-302(2016)

[7] Khairy, Reem, Mona Safar, and Watheq El-Kharashi. M.: "*Bloom filter acceleration: A high level synthesis approach*". In the proceedings of 30th IEEE Canadian Conference on Electrical and Computer Engineering (CCECE), pp. 1-6. Canada(2017)

[8] Deorowicz, S.: "*FQSqueezer: k-mer-based compression of sequencing data".* Scientific reports, vol. 10(1), pp.1-9 (2020)

[9] Bingmann, Timo, Phelim Bradley, Florian Gauger, and Zamin Iqbal: "*Cobs: a compact bit-sliced signature index*. In: Proceedings of International Symposium on String Processing and Information Retrieval, pp. 285-303. Springer, Cham(2019)

[10] Bradley, P., Den Bakker, H.C., Rocha, E.P., McVean, G. and Iqbal, Z.: "*Ultrafast search of all deposited bacterial and viral genomic data*". Nature biotechnology, vol. 37(2), pp.152-159(2019)

[11] Holley, G., Wittler, R. and Stoye, J.: "*Bloom Filter Trie: an alignment-free and reference-free data structure for pan-genome storage*". Algorithms for Molecular Biology, vol. 11(1), pp.1-9(2016)

[12] Marchiori, D. and Comin, M.: "*SKraken: Fast and Sensitive Classification of Short Metagenomic Reads based on Filtering Uninformative k-mers*". Bioinformatics, pp. 59-67(2017)

[13] Chikhi, R., Holub, J. and Medvedev, P.: "*Data structures to represent sets of k-long DNA sequences*", arXiv preprint arXiv*:* 1903.12312(2019)

[14] Kryukov, K., Ueda, M.T., Nakagawa, S. and Imanishi, T.: "*Nucleotide Archival Format (NAF) enables efficient lossless reference-free compression of DNA sequences*". Bioinformatics, vol. 35(19), pp. 3826-3828(2019)

[15] Pratas, D., Pinho, A.J. and Ferreira, P.J.: "*Efficient compression of genomic sequences*". In Proceedings of Data compression conference (DCC), pp. 231-240, IEEE, USA(2016)

[16] Chandak, S., Tatwawadi, K., Ochoa, I., Hernaez, M. and Weissman, T.: "*SPRING: a next-generation compressor for FASTQ data*". Bioinformatics, vol. 35(15), pp. 2674-2676(2019)

[17] Liu, Y., Yu, Z., Dinger, M.E. and Li, J.: "*Index suffix–prefix overlaps by (w, k)-minimizer to generate*

long contigs for reads compression". Bioinformatics, vol. 35(12), pp.2066-2074(2019)

[18] Hernaez, M., Ochoa, I. and Weissman, T.: "*A cluster-based approach to compression of quality scores*". In 2016 Data Compression Conference (DCC), pp. 261-270, IEEE, USA(2016)

[19] Pratas, D., Hosseini, M., Silva, J.M. and Pinho, A.J.: "*A reference-free lossless compression algorithm for DNA sequences using a competitive prediction of two classes of weighted models*". Entropy, vol. 21(11), p.1074(2019)

[20] Long, H., Sung, W., Kucukyildirim, S., Williams, E., Miller, S.F., Guo, W., Patterson, C., Gregory, C., Strauss, C., Stone, C. and Berne, C.: "*Evolutionary determinants of genome-wide nucleotide composition*". Nature ecology & evolution, vol. *2*(2), pp.237-240(2018)

[21] Hernaez, M., Pavlichin, D., Weissman, T. and Ochoa, I.: "*Genomic data compression*". Annual Review of Biomedical Data Science, vol. *2*, pp.19-37(2019)

[22] Hosseini, M., Pratas, D. and Pinho, A.J.: "*A survey on data compression methods for biological sequences".* Information, vol. 7(4), p.56(2016)

[23] Bonfield, J.K., McCarthy, S.A. and Durbin, R.: "*Crumble: reference free lossy compression of sequence quality values*". Bioinformatics, vol. 35(2), pp. 337-339(2019)

[24] Chandak, S., Tatwawadi, K. and Weissman, T.: "*Compression of genomic sequencing reads via hash-based reordering: algorithm and analysis*". Bioinformatics, vol. 34(4), pp. 558-567(2018)

[25] Ginart, A.A., Hui, J., Zhu, K., Numanagić, I., Courtade, T.A., Sahinalp, S.C. and David, N.T.: "*Optimal compressed representation of high throughput sequence data via light assembly*". Nature communications, vol. 9(1), pp. 1-9(2018)

[26] Ochoa, I., Hernaez, M., Goldfeder, R., Weissman, T. and Ashley, E.: "*Effect of lossy compression of quality scores on variant calling*". Briefings in bioinformatics, vol. 18(2), pp. 183-194(2017)

[27] Pamela Vinitha, E., Gopalakrishnan, G. and Karunakaran, M.: "*An optimal seed based compression algorithm for DNA sequences*". Advances in Bioinformatics, vol. 2016, Article ID 3528406(2016)

[28] Punitha K. and Murugan A.: "*Pattern Matching Compression Algorithm for DNA Sequences*", In: Proceedings of the International Conference on Sustainable Expert System, vol.176, pp. 387-402, Nepal(2021).

Profile of Authors:

Dr. A. Murugan is an Associate Professor and Head of Computer Science at Dr. Ambedkar Government Arts College, Chennai. He received his M.Sc Computer Science from Manonmaniam Sundaranar University and Ph.D from the University of Madras. His areas of interests are Algorithm analysis, Molecular computation and Data mining. He has over 25 years of experience in teaching. He has published 98 papers in International journals and 17 papers in National publications.

Ms. K. Punitha is an Assistant Professor at Agurchand Manmull Jain College, Chennai. She received her M.Sc Computer Science from University of Madras, M.Phil from the Periyar University, B.Ed from the University of Madras. Her areas of interests are Computation Algorithms, Computer Networks and Molecular Computation. She has over 16 years of experience in teaching. She has published 6 papers in International and National Journals.