Genomic Sequence Data Compression using Lempel-Ziv-Welch Algorithm with Indexed Multiple Dictionary

Keerthy A. S., S. Manju Priya

Abstract: ***With the advancement in technology and development of High Throughput System (HTS), the amount of genomic data generated per day per laboratory across the globe is surpassing the Moore's law. The huge amount of data generated is of concern to the biologists with respect to their storage as well as transmission across different locations for further analysis. Compression of the genomic data is the wise option to overcome the problems arising from the data deluge. This paper discusses various algorithms that exists for compression of genomic data as well as a few general purpose algorithms and proposes a LZW-based compression algorithm that uses indexed multiple dictionaries for compression. The proposed method exhibits an average compression ratio of 0.41 bits per base and an average compression time of 6.45 secs for a DNA sequence of an average size 105.9 KB.***

Keywords: Compression, lossless, LZW, DNA, Multiple Dictionary, Decompression.

I. INTRODUCTION

The past two decades have witnessed huge data evolution in many areas of science. Among them the most widely discussed is the discovery of sequencing genomic data. With the advent of technologies and HTS the data generation has become overwhelming challenging the scientists in the biological community to find ways of efficiently managing the sequences, especially for storing and transmitting it for further analysis. Data compression has been of interest to data scientists for a long time. Efficient algorithms for compression and decompression have been developed for textual, video and, image types of data. Since genomic sequence is composed of character set {A, C, G, T} corresponding to chemical compounds Adenine, Cytosine, Guanine and Thymine respectively and the data files are stored with text data, the text data compressions are the best suited for compression. Among the text compression techniques, lossy compression does not yield back the original data. The genomic data cannot afford a loss when decompressed as every character in the data file has equal importance and hence for compressing genomic data only lossless compression can be relied upon. The paper discusses different lossless compression techniques currently used for compressing text data as well as methods available for compressing genomic data.

Revised Manuscript Received on December 15, 2019.

* Correspondence Author

Dr. Keerthy A S[*], Research Scholar, Department of Computer Science, Karpagam Academy of Higher Education, Coimbatore (Tamil Nadu) India. E-mail: keerthysanthosh@gmail.com

Dr. S. Manju Priya, Professor, Department of CS, CA & IT, Karpagam Academy of Higher Education, Coimbatore (Tamil Nadu) India. E-mail: smanjupr@gmail.com)

The author [1] identifies dictionary method as the best suited for lossless compression of data and proposes a modified version of LZW with multiple dictionaries for efficient compression. Also, compression ratio achieved is compared with existing genomic and general purpose [1] compression algorithms.

II. LITERATURE REVIEW

The need for compressing genomic sequence data was identified and attempted from the 20th century where most attempts were made using text-based tools that already existed. In 1996, Rivals et al. [2] attempted to compress the genomic sequences by considering the regularities and approximate tandem repeats that appear in the DNA sequences. The author proposes an algorithm Cfact, a two-pass algorithm which locates repeated segments and measures their quantitative importance based on compression rate. They also propose a methodology to measure approximate tandem repeats which are of evolutionary importance to DNA sequences. GenCompress proposed by Xin Chen et al. is a one pass algorithm which looks for complemented palindromes in DNA sequences. When Cfact looks for global matches, GenCompress[3] searches for the best approximate match within the text under analysis. It also identifies regularities in the genomic sequences such as crossover and mutation. The authors claim that GenCompress performs better than Cfact. A combination of parallel dictionary and adaptive Huffman algorithm is proposed by Lin, Lee and Jan[4] for compressing text data. The usage of multiple dictionaries of variable length for compression and decompression is explained with a sample sequence. The fixed length codewords are converted to variable length codewords based on approximated AH algorithm. Variant approaches to AH (Adaptive Huffman) algorithm are discussed and their performance is evaluated. The output from Huffman tree is represented as canonical Huffman code. The authors bring out the advantage of having better compression with parallel dictionaries instead of a single dictionary for text data. NML (Normalized Maximum Likelihood) based model for DNA sequences is detailed and tested for genomic as well as non-genomic data sets by Korodi et al. [5]. The compression splits the DNA sequence into non-overlapping sequences and assigns a search dictionary for each of the blocks with subsequences collected from previous data and also complemented palindromes.

The results, when compared with bzip2, a general purpose encoder, showed sufficient advantage in compression. The authors further claim a compression ratio of 1.66 bits per symbol. The major drawback of LZW (Lempel Ziv Welch) based data compression is the size of the dictionary generated and that it gets filled up quickly while compressing. Parvinder et al.[6] propose a substitution of short sequence by long sequences as the dictionary gets filled up. An alternative suggestion is to use two dictionaries – a primary dictionary and a secondary dictionary. The primary dictionary will maintain frequently used entries. When it gets filled up, the entries would be moved to the secondary dictionary so that there will be more space in the primary dictionary to add new code words. XM (eXpert Model)[7] is a statistical methodology for compressing biological sequences such as genomic sequences and protein sequences. The method consists of using statistical properties and repetition within sequences for compression. The compressor calculates the probability of each symbol based on the observations from previous occurrences of the symbol, identifies the probability distribution of the symbol and compresses it using a general purpose method like arithmetic coding. A comparative analysis of different compression algorithms is presented, where XM claims to have an average compression rate of 1.69 bits per symbol. DNAEncodeWG[8] (DNA Encode for Whole Genome) is a reference-based compression method for Whole Genome sequences. The method identifies the presence of query sequence in the whole genome sequence and records the properties of the region and the differences identified between reference sequence and query sequence. The method claims to have an average compression of 0.19 bits per symbol. Heath et al. [9] proposes a four-stage algorithm for compression of genomic sequences, which also provides random accession of subsequences. The first step in the four-step compression strategy is preprocessing the target sequences, where the sequences are grouped based on segments or chromosomes, and then multiple sequence alignment is performed with the reference genome. In the second stage the difference between the target and reference sequences is identified. In the third step the differences are compressed with Huffman coding. In the last step the differences are used to identify mutations in the target genome. The authors claim a compression ratio of 0.98 for mitochondrial sequences. DNABIT[10] compress uses two phases of compression. In the first phase all single bases in non-repetitive regions are assigned two bit codes. In the second phase based on the number of bases repeated in each region, four different coding methods are used, namely 3 bit, 5 bit, 7 bit and 9 bit. For two or three similar bases 3 bit coding is used, for three to eight repeats of same base 5 bit coding is used, for two base repeat upto 8 times 7 bit coding is used, and if the consecutive 4 bases are the same in the subsequence under consideration, 9 bit coding is used. The authors conclude on an average compression ratio of 1.53 bits per symbol. Nishad and Chezian[11] proposes a two-stage dictionary-based compression technique for DNA sequence compression. In the first stage, a fragment of four characters is fetched from the sequence and converted to corresponding binary sequence. For conversion to binary sequence, each character is mapped to binary code as follows: A= 00, C= 01, G= 10 and T = 11. The binary string generated is added to the dictionary. In the second phase, a binary tree is constructed for the dictionary, where a child node is designed as a path taken from a parent node. New binary codes are generated for members of dictionary. Corresponding to each fragment in the sequence the new binary code is written to the output file. In their subsequent works, [12, 13], Nishad and Chezian, introduced compression based on dictionary. They described an implementation of LZW with binary searching that reduced the time complexity for searching a string later and also proposed to use multiple dictionaries in the place of single dictionary which would reduce the search complexity. The methodology is proposed for general purpose compression. The implementation is tested with genomic data as well. The decompression is also performed using multiple dictionaries and the authors claim a 94% compression for text data.

COMRAD[14] works iteratively to compress a set of DNA sequences using the length of substring and a minimum frequency threshold as parameters for first iteration. In each subsequent iteration, a frequency dictionary is created and substitutions are done. In the first iteration, frequency dictionary creation step calculates the frequency of each substring of pre-specified length. In the first substitution step, the substrings that are repeated most frequently are replaced by symbols. The result from first iteration would be a mix of nucleotides and symbols used in the substitution. In the subsequent iterations, these steps are repeated with frequency dictionary generation and substitution. The iterations terminate when no further substitution is possible. From the final frequency dictionary, all those substitutions with frequency of less than a threshold value are eliminated by replacing the substitution with the original string in the sequence. The frequency dictionary generated and the substitution strings are encoded using Huffman coding as a final step. The compression cost of the method depends on the number of iterations performed. The decompression works in the reverse order, the first step being Huffman decoding and the second step COMRAD decoding. COMRAD permits random access across the sequence that is compressed. Though it assures pretty good compression, the algorithm is memory intensive. SCALCE (Sequence Compression Algorithm using Locally Consistent Encoding)[15] is a boosting method that works based on local parsing method that rearranges the reads to improve compression rate and speed with or without a reference sequence. SCALCE is combined with Arithmetic coding for compressing quality scores and gzip to achieve considerable compression on read names resulting in a good compression rate and improved running time.

Giancarlo, Rombo and Utro[16] have listed various methods of compression used widely based on research areas as collection of sequences, collection of HTS (High Throughput Sequence) reads and compressive sequence analysis. The compression techniques are compared based on the type of compression, the method of compression and availability of random access.

Huffman coding and arithmetic coding are identified as statistical methods of compression. Lempel-Ziv data compression is a dictionary-based one, which is used along with Huffman coding by COMRAD. For an analysis of relative compression, the authors compare GRS, GReEN and many other techniques that use reference sequence for accomplishing compression. The growth of raw sequence data over the years has been analyzed by Deorowicz and Grabowski [17], along with the various methods of data compression techniques available for genomic data, comparing the existing general purpose methods of compression using genomic sequences. Reference-based compression using FASTQ files and SAM files is performed. As data deluge is becoming a fact in the biological scientific community, the authors discuss the need for compression techniques. They look into the available compression methods and their suitability with respect to genomic sequence data. They also discuss different file formats for data storage.

Zhu et al.[18] selected a set of genomes from different species with respect to origin, length and repeat content. The performance evaluation of various compression techniques, both reference-based and reference-free, was performed based on compression ratio, memory usage and compression and decompression time. The author suggests CRAML, which is a lossy compression method to give the best compression ratio and the best compression and decompression time and memory. The authors also point out that high throughput sequencing has made personalized genomic sequencing affordable, and that encryption techniques must be deployed to protect the privacy of personalized data.

Wandelt et al.[19] discuss the key methods of genomic data compression. Naïve bit manipulation replaces every base by a two bit code. Dictionary-based method adds the entry of a codeword to dictionary when encountered. These book-keeping details are further used for the decompressing purpose. Since high amounts of repetition of bases are the prominent features of genomic data, statistical methods are also employed in compressing genomic information. Huffman coding or arithmetic encoder works based on the principle of statistical algorithms. Referential algorithms are used only when there is a standard sequence that can be kept as reference for compressing similar genomes. The authors discuss the usage of each of these techniques in whole genome compression as well as read compression. The lack of benchmark datasets as well as metrics for comparing performance of different methods has been pointed out. They suggest rate of compression, time taken for compression and decompression and memory usage while compressing and decompressing as metrics of performance analysis.

Pratas and Pinho[20] propose an asymmetric compressor for genomic data sequences which parallelize the tasks using two FCMs. The sequences are preprocessed by calculating probability estimates using symbol counts. The preprocessing helps in identifying low complexity areas in the sequence. Two finite context models (FCMs) of high and low order are run parallel to compete with each other. The outputs from each of the FCMs are stored and processed separately. The higher order FCM is used to compress regions with low information content. The higher order FCM also considers the possible inverted repeats (IRs). The high FCM consists of a regular chain and an IR chain. The authors point out that preprocessing can substantially improve the memory usage, especially while decompressing. Parallelization assures faster compression. GeCo (Genomic Compression)[21], a statistical method for genomic data compression, uses FCM based on Markov models. Patras, Pinho and Ferreira extend the FCMs as XFCM, where the probability estimate varies as the conditioning context is different. They use the most probable symbol as the conditioning context. A pseudo-random synthetic sequence is generated and used as reference sequence. It is mutated with a predefined substitution rate and genomic sequence with several degrees of mutation generated. These resulting sequences are compressed using the synthetic reference sequence generated. A cache-has memory approach, which keeps only the last hashed entries in memory, is used so that memory usage is further reduced.

ERGC (Efficient Referential Genome Compression) [22] algorithm works by keeping a sequence as referential sequence and the sequence to be compressed as target sequence. The algorithm works by splitting the reference sequence and target sequence into equal-sized strings. A greedy algorithm generates k-mers one at a time, and hashes it to a hash table. If no match is found, k-mer is extended by one, otherwise it aligns the reference sequence and target sequence and extends the alignment until there is a mismatch. The mutations and insertions in reference and target sequence are also taken care of.

Though general purpose compression techniques like gzip can compress and assist in efficient storage and transmission of genomic data, they do not take into advantage the general features of genome sequence such as tandem repeats, microsatellites, etc., whose presence increases the possibility of compression even better. Hence, the scientific world is interested in identifying special purpose software that are tailored for genomic data compression. Mince's algorithm[23] works by grouping similar sequences. The compression is carried out in multiple phases. In the local bucketing phase, Mince places similar reads into a bucket. All k-mers of a read of length r are checked to see if any of the k-mer or its reverse complement has a label matching the existing ones. If none matches a new bucket is created. If it matches, the read of length 'r' is assigned to that bucket using an encoding transformation. Comparison with the existing methods assures a better compression ratio. The k-mer redundancy check exploits the possible sequence similarity between the reads, and the read order is chosen randomly while compressing.

The literature review extensively studies various methodologies used in compressing genomic sequence data. It is evident from a comparative analysis of the existing general purpose methods that they do not efficiently compress genomic data. Statistical methodologies reap the advantages of high degree of repetition of bases in the data, but they are mostly memory intensive. Reference-based methods claim to produce maximum compression, but it is possible only if a valid reference sequence is available.

Dictionary-based methods are fast and prompt in compression as well as decompression, but size and maintenance of dictionary are drawbacks. Parallel dictionaries claim to ease the time complexity in compression procedure with reduced number of shifts and comparisons. Also, the methodologies discussed are experimented in different datasets due to lack of benchmark dataset. A quantitative comparison of different methods is restricted to a few sequences that overlap different methodologies.

III. METHODS AND METHODOLOGY

An analysis of different compression techniques brings out the pros and cons of the various techniques used. Most of the methods described for compression are laid down for compressing text data. Of the methods discussed for text data compression usage of multiple dictionary in compression claims to be of less overhead and lossless. Since lossless compression is unquestionable LZW is the best option for compression. The LZW with single dictionary [24] is laid down for compressing text data. Since the character set is smaller for genomic data, the basic algorithm is modified to suit the character set. Also, for faster search and retrieval of data, multiple dictionaries[25] are used in compression and decompression, which facilitate faster and efficient storage and transmission of genomic data.

A. Compression

Each character read from the input file is concatenated with the previous char or subsequence; its presence is verified in the corresponding dictionary. If the string is present in the dictionary DL where L is the length of the previous substring and the character read from the input file, the subsequence is extended by reading in the next character from the file. Otherwise, a new entry corresponding to the subsequence is made in the dictionary DL with the index calculated using the subsequence. The procedure is laid down in following steps:

```
1. Set dictionary $D_1$ with initial character set A, G, C and T and assign to them CODE 1, 2, 3 and 4 respectively.
2. Read first character from input file to STRING
3. Initialize M:= 2, CODE := 5
4. Repeat the following steps till end of file
5. Assign CHAR := Next character from input file
6. Assign
L:= Lengthof(STRING+CHAR)
7. Assign
INDEX:=CALCULATE_INDEX(STRING+CHAR, M)
8. Search for INDEX in Lth dictionary
    a. If INDEX found
        AssignFLAG:= Search(STRING+CHAR, L)
    b. OTHERWISE
        i. Create INDEX in Lth dictionary
9. If FLAG is TRUE
    a. STRING := STRING+CHAR
10. ELSE
    a. Write CODE to output file
    b. Add CODE. STRING+CHAR to Lth dictionary
    c. CODE:= CODE+1
    d. STRING:=CHAR
11. Repeat from step 5
12. STOP
13. Function CALCULATE_INDEX(X,M)
Return($\sum_{i=1}^{M}\left(HEX(tolower(X_i)MOD5)\right) * 4^{M-i}$)
```

The codes are written in the output file as ascii characters corresponding to their values. This eliminates the possibility of wrongly reading the codes while decompressing.

B. Decompression

The decompression starts by reading in codes from the compressed file. An initial dictionary D1 is created with character set A, G, C, T assigned with codes 1, 2, 3, 4 respectively. For each code read from the compressed file, if it is updated in the dictionary, write the corresponding character or subsequence to the output file. Otherwise assign the previous subsequence value to the dictionary and update the code value.

```
1. Set dictionary $D_1$ with initial character set A, G, C and T and assign to them CODE 1, 2, 3and 4 respectively.
2. Initialize M: = 2, CODE:= 4, L: = 1
3. Assign OCODE:= Character(First character read from compressed file)
4. Write OCODE to output file.
5. Repeat the steps till end of file
    a. Assign NCODE := Next character from input file
    b. Assign FLAG:= Search Dictionary(L, NCODE)
        i. If FLAG is FALSE
            Assign STRING:= OCODE + CHAR
        ii. ELSE
            1. Assign STRING := NCODE
            2. L=L+1
            3. Write STRING to output file
    c. Assign CHAR=STRING[1]
    d. CODE = CODE+1
    e. Assign L = Lengthof(OCODE+CHAR)
    f. Update Lth dictionary with CODE,
       OCODE+CHAR
    g. OCODE = NCODE
6. STOP
14. Function CALCULATE_INDEX(X,M)
Return ($\sum_{i=1}^{M}\left(HEX(tolower(X_i)MOD5)\right) * 4^{M-i}$)
```

IV. RESULTS AND DISCUSSION

The proposed algorithm is implemented on Intel® Core™ i7 (2.40GHz 8GB RAM) running on Windows 10 with Python 3.6. The experimental analysis was carried out on Nvidia GeForce GTX 1060 GPU (Intel Core i7, 32GB RAM). A set of standard DNA sequences that were tested with other compressing algorithms were compressed and compared with available compression ratio of WinZip and CTW, which are general purpose compression algorithms, and CTW+LZ, BIOCOMPRESS, GENCOMPRESS, DNACOMPRESS, XM and DNAEncodeWG which are genomic compression algorithms. The test data includes five human gene sequences (HUMDYSTROP, HUMGHCSA, HUMHBB, HUMHDABCD AND HUMHPRTB) that are commonly used in most of DNA compression publications. Along with these MOUSE CHROMOSOME2 and BAKER's YEAST CHROMOSOME 2 were also compressed, for which comparative values for other algorithms were unavailable. The compressions were compared using compression ratio calculated as number of bits/base. Table 1 gives the comparison of sequence size before and after compression, and the time taken for compression.

Table 1: Comparing sequence size before and after compression using multiple dictionaries based on LZW

Name of Sequence	Uncompressed size (in KB)	Compressed Size(in KB)	Compression Ratio	Exec time (in sec)
MOUSECHR2	218.1	86	0.394314535	6.342
BAKERSYEASTCHR1	231	94	0.406926407	6.898
HUMDYSTROP	38.77	16	0.412690224	6.283
HUMGHCSA	64.495	24	0.37212187	6.139
HUMHBB	73.323	31	0.422786847	6.408
HUMHDABCD	58.864	25	0.424707801	7.15
HUMHRPTB	56.737	24	0.423004389	5.917
Average	105.9	42.86	0.41	6.45

The compression time (in secs) for seven sequences are presented graphically as follows:

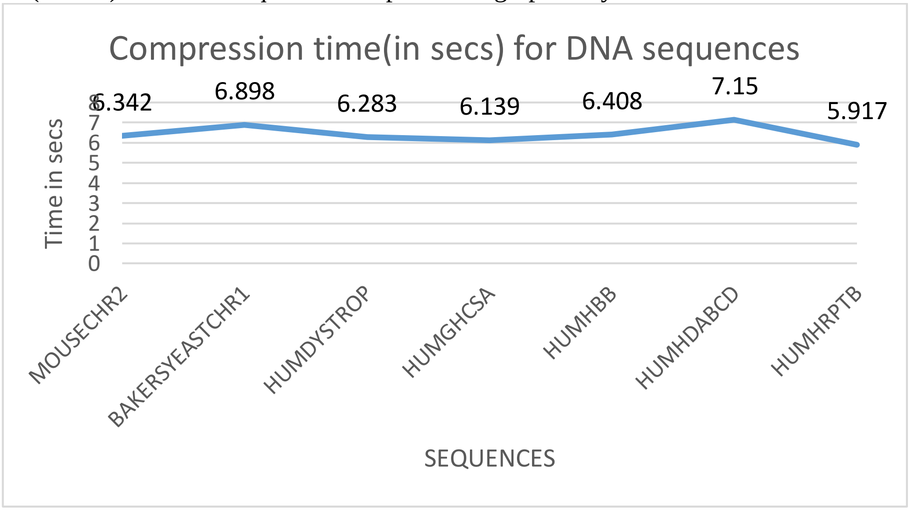

Figure 1: Graph presenting compression time (in secs) for the seven sequences compressed

A. Comparing the proposed implementation (MDLZW) with the existing algorithms:

Table 2: Compression ratios given by different general purpose and genomic compression algorithms for compressing five human genes.

DNA SEQUENCE	Win Zip	BIO COMPRESS	GEN COMPRESS	Normal CTW	CTW + LZ	DNA COMPRESS	XM	DNA Encode WG	MDLZW
HUMDYSTROP	2.38	1.9262	1.9231	1.92	1.9175	1.9116	1.9031	0.1729	0.4069
HUMGHCSA	2.34	1.3074	1.0969	1.3638	1.0972	1.0272	0.9828	0.2732	0.4127
HUMHBB	2.33	1.88	1.8204	1.8928	1.8082	1.7897	1.7513	0.2744	0.3721
HUMHDABCD	2.29	1.877	1.8192	1.8973	1.8218	1.7951	1.6671	0.1422	0.4247
HUMHRPTB	2.32	1.9066	1.8466	1.9132	1.8433	1.8165	1.7361	0.1111	0.423

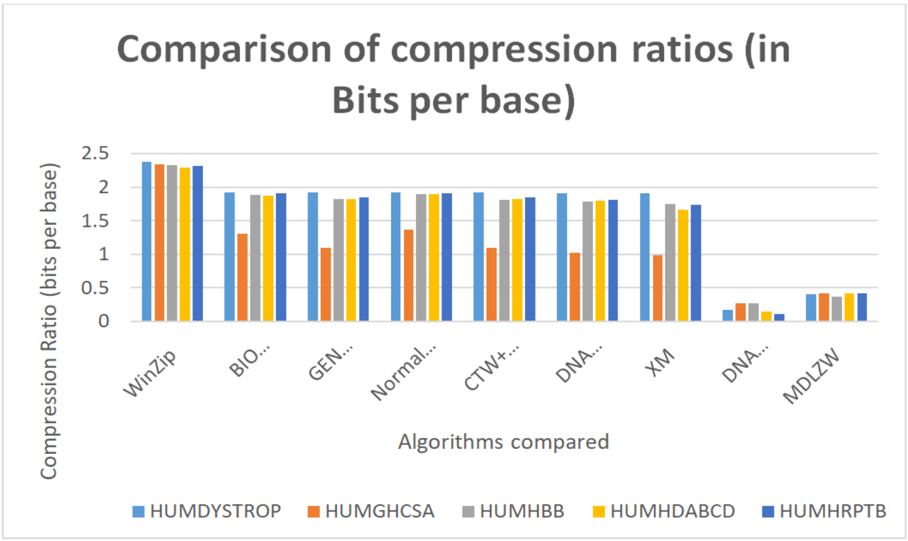

Figure 22: Comparing compression ratio (bits per base) obtained by different algorithms on five human genes

The comparative analysis of compression ratio by different algorithms is presented in the Figure 2. The compression rates in Figure 2 are obtained from existing literature[8] except the last column, which gives the experimental results in the proposed method. The general purpose compression algorithms give the highest compression ratio. The best compression among the compared algorithms is given by DNAEncodeWG with an average compression ratio of 0.1948 bits per base, which is a reference-based algorithm. A reference-based algorithm works well only if a valid reference is available. Hence, it cannot be considered as a convincing method for compressing sequences resulting from experiments. Hence, the proposed algorithm gives a better compression ratio than other compression algorithms discussed and compared. Also, it is evident that general purpose algorithms are not suitable for compressing the genomic sequences as they do not take in to account the high amount of repetitiveness in the sequence data.

V. CONCLUSION

The paper presents an efficient compression algorithm based on LZW with the modification of using multiple dictionaries. LZW has been proven as the best among the compression algorithms for lossless compression. For compressing DNA sequence data, lossless compression is mandatory and hence LZW is the basic methodology chosen. The main drawback of LZW is the dictionary size as well as time consumption in searching the dictionary. This is overcome by using multiple dictionaries that are indexed. Also, the dictionaries are created dynamically during compression as well as decompression, and the dictionary is not stored for later use. Hence, memory overhead of dictionary storage is overcome. Only the compressed file is stored and transmitted. The proposed method MDLZW for Genomic Sequence data has an average compression rate of 0.41bits per base, which is the best among the compression rates compared. The decompression algorithm performs a lossless decompression in comparable time.

REFERENCES

1. Keerthy A S, Manju Priya S, 2016, Comparative analysis of Data Compression and Pattern Matching Techniques for Biological Big Data. International Journal of Advanced Research in Computer Engineering & Technology (IJARCET), 5(1).
2. Rivals, E., Dauchet, M., Delahaye, J.P. and Delgrange, O., 1996. Compression and genetic sequence analysis. Biochimie, 78(5), pp.315-322.
3. Chen, X., Kwong, S. and Li, M., 1999. A compression algorithm for DNA sequences and its applications in genome comparison. Genome informatics, 10, pp.51-61.
4. Lin, M.B., Lee, J.F. and Jan, G.E., 2006. A lossless data compression and decompression algorithm and its hardware architecture. IEEE TRANSACTIONS on very large scale integration (vlsi) systems, 14(9), pp.925-936.
5. Korodi, G., Rissanen, J. and Astola, J., 2007. DNA sequence compression-Based on the normalized maximum likelihood model. IEEE Signal Processing Magazine, 24(1), pp.47-53.
6. Singh, P. and Duhan, M., 2006, September. Enhancing LZW Algorithm to Increase Overall Performance. In 2006 Annual IEEE India Conference (pp. 1-4). IEEE.
7. Cao, M.D., Dix, T.I., Allison, L. and Mears, C., 2007, March. A simple statistical algorithm for biological sequence compression. In 2007 Data Compression Conference (DCC'07) (pp. 43-52). IEEE.
8. Do Kim, H. and Kim, J.H., 2009. DNA data compression based on the whole genome sequence. Journal of Convergence Information Technology, 4(3), pp.82-85.
9. Heath, L.S., Hou, A.P., Xia, H. and Zhang, L., 2010, August. A genome compression algorithm supporting manipulation. In Proc LSS Comput Syst Bioinform Conf (Vol. 9, pp. 38-49).
10. Rajarajeswari, P. and Apparao, A., 2011. DNABIT compress–genome compression algorithm. Bioinformation, 5(8), p.350.
11. Nishad, P.M. and Chezian, R.M., 2012. A vital approach to compress the size of DNA sequence using LZW (Lempel-Ziv-Welch) with fixed length binary code and tree structure. International Journal of Computer Applications, 43(1), pp.7-9.
12. Nishad, P. M. and R. Manicka Chezhian, 2012, Optimization of LZW (Lempel-Ziv-Welch) Algorithm to Reduce Time Complexity for Dictionary Creation in Encoding and Decoding, AJCSIT, 114-118.
13. Nishad, P. M., and Manicka Chezian R., 2012, Enhanced lzw (lempel-ziv-welch) algorithm by binary search with multiple dictionary to reduce time complexity for dictionary creation in encoding and decoding, IJARCSSE 2.3,192 -198.
14. Kuruppu, S., Beresford-Smith, B., Conway, T. and Zobel, J., 2012. Iterative dictionary construction for compression of large DNA data sets. IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB), 9(1), pp.137-149.
15. Hach, F., Numanagić, I., Alkan, C. and Sahinalp, S.C., 2012. SCALCE: boosting sequence compression algorithms using locally consistent encoding. Bioinformatics, 28(23), pp.3051-3057.
16. Giancarlo, R., Rombo, S.E. and Utro, F., 2013. Compressive biological sequence analysis and archival in the era of high-throughput sequencing technologies. Briefings in bioinformatics, 15(3), pp.390-406.
17. Deorowicz, S. and Grabowski, S., 2013. Data compression for sequencing data. Algorithms for Molecular Biology, 8(1), p.25.
18. Zhu, Z., Zhang, Y., Ji, Z., He, S. and Yang, X., 2013. High-throughput DNA sequence data compression. Briefings in bioinformatics, 16(1), pp.1-15.
19. Wandelt, S., Bux, M. and Leser, U., 2014. Trends in genome compression. Current Bioinformatics, 9(3), pp.315-326.
20. Pratas, D. and Pinho, A.J., 2014, September. Exploring deep Markov models in genomic data compression using sequence pre-analysis. In 2014 22nd European Signal Processing Conference (EUSIPCO) (pp. 2395-2399). IEEE.
21. Pratas, D., Pinho, A.J. and Ferreira, P.J., 2016, March. Efficient compression of genomic sequences. In 2016 Data Compression Conference (DCC) (pp. 231-240). IEEE.
22. Saha, S. and Rajasekaran, S., 2015. ERGC: an efficient referential genome compression algorithm. Bioinformatics, 31(21), pp.3468-3475.
23. Patro, R. and Kingsford, C., 2015. Data-dependent bucketing improves reference-free compression of sequencing reads. Bioinformatics, 31(17), pp.2770-2777.
24. Ziv, J., and Lempel A., 1977, "A universal algorithm for sequential data compression, IEEE Trans. Inf. Theory, 23.3, 337-343.
25. Keerthy, A. S., and S. Manju Priya, 2017, Lempel-Ziv-Welch Compression of DNA Sequence Data with Indexed Multiple Dictionaries. IJAER, 12.16, 5610-5615

AUTHORS PROFILE

Keerthy A. S. is currently pursuing Ph D in Karpagam Academy of Higher Education, Coimbatore, India, under the guidance of Dr. S Manju Priya. She has completed MPhil in Bioinformatics from Kerala University and MCA from Calicut University. Her area of interests include Data Mining, Graph Theory, Artificial Intelligence, Bioinformatics and Machine Learning. She has more than 8 years of teaching experience in post graduate level. She has published 4 Scopus indexed papers and presented papers in multiple conferences.She has also guided project students in post graduate level.

Dr. S. Manju Priya is working as a Professor in Dept of CS, CA & IT, Karpagam Academy of Higher Education, Coimbatore, India for the past 15 years. She has completed Ph D in 2014 in Karpagam Academy of Higher Education. She has attended various conferences and has published 42 papers in various National and International Journals. She has published on book on wireless sensor network. Under her guidance she has produced 5 Ph D scholars and 4 M Phil scholars. Currently 6 scholars are under her guidance. She is one of the associate Editor in Karpagam Journal of Computer Science. Her research areas are like Sensor Network, IOT, Data mining and Big Data Analytics.