International Journal of Applied Engineering Research ISSN 0973-4562 Volume 12, Number 16 (2017) pp. 5610-5615

Lempel-Ziv-Welch Compression of DNA Sequence Data with Indexed Multiple Dictionaries

Keerthy A.S.[1]
Research Scholar, Dept. of Computer Science, Karpagam Academy of Higher Education, Karpagam University, Coimbatore, India 641021

Orcid-id: 0000-0002-8641-4970

Dr. S. Manju Priya[2]
Associate Professor, Dept. of Computer Science, Karpagam Academy of Higher Education, Karpgam University, Coimbatore, India 641021

Orcid-id:0000-0002-2390-9693

Abstract

Liv-Zempel-Welch (LZW) is a globally used lossless method for compressing the data, especially when data contains repetitive information. LZW makes use of unique dictionaries built at the time of compression and decompression. Genomic sequence data of organisms is represented by A, G, C, T bases which are the basic building blocks of DNA double helical structures. The genomic sequence data is used by the biologists across the globe for experimental analysis with respect to anomaly detection, disease prediction, personal medicine, etc. These sequences generated in different laboratories worldwide have to be shared or transferred to the scientific community across the globe. This can be effectively achieved by an efficient lossless compression methodology. This paper attempts to compress/ decompress the biological DNA sequence data using the multiple dictionaries.

Keywords: Biological sequence data; DNA; LZW; Multiple Dictionaries; Compression; Decompression.

INTRODUCTION

Data compression aims at representing the data with the reduced resource requirements. Compression can be performed either in a lossy manner or in a lossless manner [1]. Lossy compression does not ensure the revival of exact copy of original file upon the decompression. Hence the lossless compression becomes mandatory when the data needs to be revived as an exact copy of the original data upon the decompression. One of the most important data set under the analysis in the current era among the different research communities is the biological sequence data [2]. Biological sequence data is the result of DNA sequencing experiments conducted in various biological research labs across the globe [3].

The four characters A, G, C and T represent the four bases Adenine(A), Guanine(G), Cytosine(C) and Thymine(T) which are the basic nucleotides that form the building blocks of cells in every organism. The genomic sequence of every living thing comprises of these bases in different frequency and order. The size of whole genome of the individual organisms is of the order of Terabytes (TB) to Petabytes (PB). Efficient storage and secure transfer of these data is unavoidable for the biological research community. To store them effectively, computational compression techniques have been used [4].

THE MATERIAL AND METHOD

A. Literature Review

The most widely used lossless Compression and Decompression technique is LZW, (named after Lempel, Ziv and Welch), developed as LZ1 and LZ2 in the 1970s[5] and modified by Terry A. Welch in 1984.The basic methodology of LZW makes use of the repetitive nature of sequences in the data. A unique dictionary is generated at the time of compression as well as decompression. The dictionary is initiated with the character set of size 256. Each character taken from the input file is attached with the previously read character and its presence/absence is verified in the dictionary by a sequential search. On a successful search, the search key is appended with the next character from input file and the search process is continued. This is done to verify the presence of a longer matching subsequence in the dictionary. If the search is unsuccessful, the dictionary is updated with the previously compared string and a unique code is assigned to that string. The unique code of previous string that returned a successful search will be send to the output file. Decompression in LZW also starts with a dictionary initiated with the character set of size 256. Whenever a code is read from the compressed file it first looks into the dictionary for the code. On finding the code from the dictionary, the corresponding subsequence will be returned. Further, the entries in the dictionary are done based on the character read as well as the previously read character [6].

LZW has been widely used in the textual data compression. It has been evident from the studies of many scholars that the LZW effectively compresses the data losslessly; it becomes

slow when the file size increases [7]. Further the improvisations by Parvinder Singh et. al. was to use a Bi-mode encoder which treated the individual character sequences of the bytes differently [8].

The major drawback of these methodologies are the huge time required for the dictionary generation and searching the dictionary for subsequences. Many modifications for the basic LZW algorithm have been proposed and is widely implemented for the textual data [9] [10].

Ming-Bo Lin replaces the single dictionary concept with a parallel VLSI architecture with multiple small variable-word-width dictionaries. The resulting scenario consists of multiple dictionaries with different word length which reduces the search time as well as operates in parallel. The authors' claims reduction in hardware cost as well as ease in implementation [11]. The detailed architectural design of Parallel Dictionary based LZW (PDLZW) is demonstrated in [12]. The dictionary is initiated with all single characters forming the character set of the input file. The input string is mapped to fixed-length code words based on dictionary set. The current substring and next character in the input stream is inserted as new entry into the dictionary [12].

A two stage compression combining features of PDLZW with adaptive Huffman Algorithm have been proposed by Ming-Bo Ling, et al. Adaptive Huffman algorithm with dynamic-block exchange (AHDB) takes the output codewords from PDLZW and encodes them. The algorithm swaps the most frequently used symbol to the top of the ordered list and the index is encoded into canonical Huffman codeword. The authors claim that the method reduces hardware cost and can easily be implemented on to VLSI architecture [13].

Another implementation of PDLZW suggested by Perapong V, et. al, uses a barrel shifter to loading a new input string onto the shift register. Also the dictionary update is performed using Windowed Second Chance Updating Technique (WSC) that partitions dictionary into windows as k-size phrases and a three bit flag associated with each phrase. Test results exhibit compression ratio better than that of conventional PDLZW and also PDLZW with WSC technique [14].

Static Text Compression Algorithm (STECA) is a language dependent method that uses multiple static dictionaries for compressing text files. The dictionary is pre- constructed and used for all suitable situations. The method is appropriate only when prior knowledge of source is available. Compression is performed using most frequently occurring digrams and trigrams. Indexes of digrams and trigrams are encoded using hash tables. The high memory requirement for has tables is considered as the main drawback of STECA [15].

The analysis of Lempel-Ziv compression in parallel and distributed systems conducted by Agostino suggests that parallel compression is possible with LZW only with the usage of FREEZE deletion heuristic. The method after constructing the dictionary freeze it and the frozen dictionary will be used for parallel compression [16].

Combining PDLZW with wavelet transform on effective compression of EGC data is proposed by Sukesh and Jayashree [17] The ECG signals are wavelet transformed and then given to PDLZW for compression. The compressed output has relevance in storage and transmission of ECG data.

Nirali and Malay [18] combine the PDLZW with arithmetic coding and compare the performance with Deflate. From the input file, PDLZW takes out all the characters and forms initial dictionary. The output of PDLZW is given to the arithmetic encoder as input. Arithmetic encoder computes the frequency of digits, using it calculates the boundary values and generates the compressed bit file. The decoder follows the reverse order of calculations to obtain original string from bit file. The comparative analysis with Deflate shows better performance for proposed algorithm in the case of large files [19].

Nishad and Manikachezian propose a variant of LZW compression by using multiple dictionaries, each of which is sorted. Since the dictionaries have entries in sorted order, the presence of patterns can be identified with simple binary search. The encoding algorithm works in two phases [20] of (1) switching and coding and (2) searching and updating. The initial dictionary contains all characters in the input file in the ascending order. Subsequent dictionaries are constructed in the process and whenever a string is to be searched, the corresponding dictionary alone is scanned. In phase 2 the pattern search takes place in the chosen dictionary using binary search. If pattern is not found in the dictionary, the position to which insertion has to be done is identified and shifts are done for enabling insertion. The method reduces the number of comparisons made and the number of shifts made. Hence it assures better time complexity as well as compression ratio [21].

A novel attempt with the multiple dictionaries and binary search has been proposed by Nishad and Manickachezhian for compressing the textual files effectively with LZW [20][21]. This paper attempts an enhancement on their work by tailoring it for the DNA sequences. A recent survey by Keerthy A S and Manju Priya S over the usage of multiple dictionaries instead of a single dictionary in LZW based compression points towards the increased efficiency of compression as a result of reduced number of shifts and comparisons [22].

B. Proposed Enhancement over LZW

1) Compression Algorithm:

The compression algorithm initiates a dictionary D_1 with the character set A, G, C and T and assign the codes 1, 2, 3 and 4 to them respectively. A variable length STRING variable is

initialized with the first character from the input file and its code is fetched from the dictionary and sent to the output file. Each of the next character from the input sequence is read into a CHAR variable and the STRING is appended with the character in CHAR. A search for STRING+CHAR is executed in the respective dictionary D_L where L is the length of (STRING+CHAR). An index function calculates the unique index value in the respective dictionary. If the search is successful, the variable STRING is assigned with the concatenated STRING+CHAR; and next CHAR will be read from the input sequence and the search will be initiated based on the new index calculated for the dictionary D_{L+1}. At this point no updating of dictionary or output file takes place. The algorithm attempts to find the longest matching subsequence present in the dictionary in the subsequent iterations.

If the search for STRING+CHAR in the dictionary D_L is unsuccessful, the concatenated string is assigned with the next available CODE value. In the proposed enhancement the CODE value for the dictionary entries start from 5 onwards. The concatenated string and the unique CODE value are added as a new entry under the respective index in the dictionary D_L. The unique CODE value corresponding to the last successful search i.e. the code value of STRING is read from the dictionary D_{L-1} and added to the output file. Also STRING variable is reset with the value of CHAR variable. The compression procedure is executed till the last character is read and processed from the input sequence. Figure1 gives the step by step process in compression.

1. Initialize dictionary with character set and set codes to each.
2. STRING = first character of I/P file
3. Set M=2, CODE = 5
4. While not EOF
5. CHAR = Next character in I/P file
6. L = length(STRING + CHAR)
7. INDEX = CALCULATE_INDEX(STRING + CHAR, M)
8. SEARCH L^{th} dictionary (INDEX)
    a. If INDEX found
        i. FLAG = Search(STRING + CHAR, L)
    b. ELSE
        i. Create New INDEX in Lth dictionary
9. If FLAG == TRUE
    a. STRING = STRING + CHAR
10. ELSE
    a. Update Output file with CODE of STRING
    b. Add CODE, STRING + CHAR to Lth dictionary
    c. CODE = CODE + 1
    d. STRING = CHAR
11. Continue from 5
12. STOP
13. Function CALCUALTE_INDEX(X, M)

    Return ($\sum_{i=1}^{M}(HEX(to\ lower(Xi))MOD\ 5) * 4^{L-i}$)

Figure 1: LZW encoding with multiple dictionaries

2) Decompression Algorithm:

In the decoding also a dictionary is initiated at the beginning with the four characters A, G, C and T with CODES set as 1, 2, 3 and 4 respectively. Read the first character from the compressed file, retrieve the character corresponding to the code that is read from the dictionary and store it in variable OCODE. Write it into the output file. Store each of the subsequent characters read from the compressed file is into a variable NCODE. Search the dictionary for the code, if found store the string value corresponding to the NCODE into a variable STRING, write the value of STRING variable into the output file. If the dictionary search is fail, assign the STRING value as OCODE+CHAR. CHAR is reset as the first character of STRING variable. OCODE+ CHAR is assigned in the next available CODE and added to the dictionary. Once all the characters in the input compressed file are processed, the process ceases. Figure 2 gives the step by step process of decompression technique used.

1. Initialize dictionary with character set and set codes to each
2. SET M=1, CODE =4, L=1
3. OCODE = first character of I/P file
4. Write OCODE into O/P file
5. Continue till EOF
    a. NCODE = read next character from I/P file
    b. FLAG = SEARCH DICT(L, NCODE)

i.If FLAG == FALSE

1. STRING = OCODE + CHAR

ii. ELSE,

1. STRING = NCODE
2. L=L1+1
3. Write STRING to O/P file

c. CHAR = STRING[1]

d. CODE =CODE + 1

e. Update Lth Dictionary with CODE, OCODE+CHAR

f. OCODE = NCODE

g. L=2

6. STOP

7. Function CALCUALTE_INDEX(X, M)

Return

$$\left(\sum_{i=1}^{M}\left(HEX\left(to\ lower(Xi)\right)MOD\ 5\right)*4^{L-i}\right)$$

Figure 2: LZW decoding with multiple dictionaries

RESULTS AND DISCUSSION

An illustration of the compression process using the proposed algorithm is given in the Table 1. The sample sequence used has 18 characters.

TABLE 1: ILLUSTRATION of MULTIPLE DICTIONARY COMPRESSION of SAMPLE DNA SEQUENCE

INPUT SEQUENCE:ACGACCACCCGACAGGT						
SLNo:	CHAR	STR + CHAR	IN DICT	ADD TO DICT	STRING	O/P
1	A	A			A	
2	C	AC	N	5=AC	C	1
3	G	CG	N	6=CG	G	3
4	A	GA	N	7=GA	A	2
5	C	AC=5	Y		AC	
6	C	ACC	N	8=ACC	C	5
7	A	CA	N	9=CA	A	3
8	C	AC=5	Y		AC	
9	C	ACC=8	Y		ACC	
10	C	ACCC	N	10=ACCC	C	8
11	G	CG=6	Y		CG	
12	A	CGA	N	11=CGA	A	6
13	C	AC=5	Y		AC	
14	A	ACA	N	12=ACA	A	5
15	G	AG	N	13=AG	G	1
16	G	GG	N	14=GG	G	2
17	G	GG=14	Y		GG	
18	T	GGT	N	15=GGT	T	14
19	EOF	T	N			4
O/P : 1325386512144						

The sample sequence of 18 characters was chosen for testing the applicability of the proposed enhancement. Detailed performance evaluation of the method will be executed at the implementation level. The output file shows a reduction in the number of characters to 12 which gives a compression factor of 33.3%. It is evident from the illustration that as the size of input file increases; the compression will also get better.

Figure 3 illustrates the initial dictionary D1 with which the compression initiates and the subsequent dictionaries D2, D3 and D4 generated during the compression process.

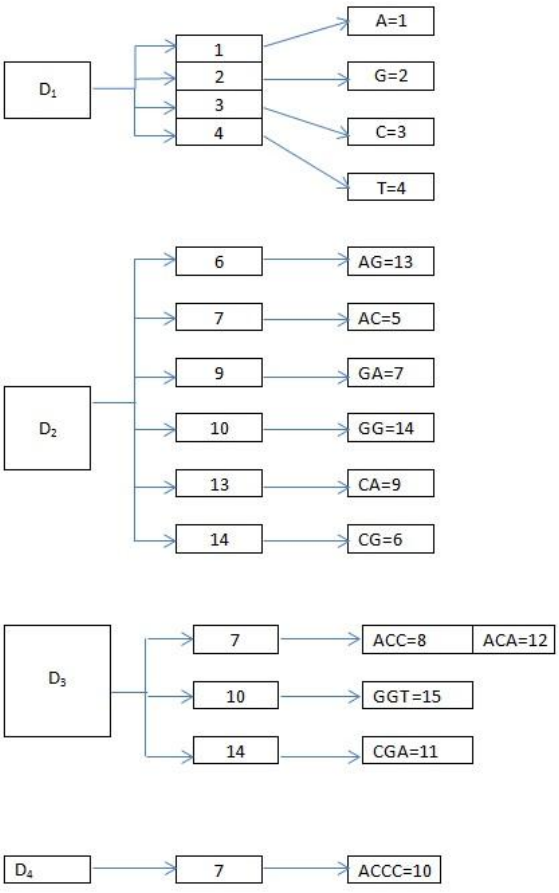

Figure 3: Dictionaries generated during compression

An illustration of the decoding procedure is given in the Table 2. Here the input file chosen is the output obtained from the illustration in the Table 1.

TABLE 2: ILLUSTRATION of DECOMPRESSION with RUN TIME DICTIONARY GENERATION

INPUT: 1325386512144						
SL NO:	OCODE	NCODE	STRING	CHAR	ADD TO DICT	O/P
1	A					A
2		3	C	C	5=AC	C
3	C	2	G	G	6=CG	G
4	G	5	AC	A	7=GA	AC
5	AC	3	C	C	8=ACC	C
6	C	8	ACC	A	9=CA	ACC
7	ACC	6	CG	C	10=ACCC	CG
8	CG	5	AC	A	11=CGA	AC
9	AC	1	A	A	12=ACA	A
10	A	2	G	G	13=AG	G
11	G	14	GG	G	14=GG	GG
12	GG	4	T	T	15=GGT	T
O/P : ACGACCACCGACAGGGT						

The decoding example illustrated gives a clear picture that the lossless compression is achievable by this method. Also the dictionary can be dynamically constructed at the time of decompression as well. The non-requirement of dictionary to be stored with the compressed file, reduces the memory overhead. Also the compressed file alone needs to be shared among the users who can decompress it with the mere knowledge of initial character set and code assignment.

CONCLUSION

LZW is the universally accepted algorithm for compressing the textual data. Its usage in the compressing the huge repositories of DNA sequence data is expected to be of great advantage to the biological research community [23]. The data under consideration has a limited character set of four characters. This adds to the advantage of allocating the codes starting from 5 instead of 256 as in the normal text compression. The usage of multiple dictionaries reduces the search time requirement and hence it speeds up the compression process.

The proposed work has to be implemented and tested for the authentication of the verified results.

REFERENCES

[1] Sayood, Khalid, 2012, Introduction to data compression, 4th ed. Morgan Kaufmann, Elsevier.

[2] Marx, Vivien, June 2013 "Biology: The big challenges of big data." Nature, 498.7453 pp 255-260.

[3] Vey G., 2009 “Differential direct coding: a compression algorithm for nucleotide sequence data”, Database: The Journal of Biological Databases and Curation.

[4] Grumbach, Stéphane, and Fariza Tahi., 1993, "Compression of DNA sequences." in proc Data Compression Conference, DCC'93, IEEE.

[5] Ziv, Jacob, and Abraham Lempel.,1977, "A universal algorithm for sequential data compression." IEEE Transactions on information theory 23.3 , pp 337-343.

[6] Kotze, H. C., and G. J. Kuhn., 1989, "An evaluation of the Lempel-Ziv-Welch data compression algorithm," in proc Communications and Signal Processing,COMSIG. Southern African Conference on IEEE, p.65.

[7] S. R. Kodituwakku, U. S. Amarasinghe, 2010, “Comparison of Lossless Data Compression Algorithms for text data”, in proc IJCSE, pg 416-425.

[8] Parvinder Singh, Manoj Duhan, Priyanka, 2006, “Enhancing LZW Algorithm to increase Overall Performance”, in Proc IEEE, pg1-4.

[9] Keerthy A S, Manju Priya,2016, ” Comparative analysis of Data Compression and Pattern Matching Techniques for Biological Big Data”, International Journal of Advanced Research in Computer Engineering & Technology (IJARCET) Volume 5 Issue 1.

[10] Keerthy A S, Appadurai, 2015, ―An empirical study of DNA compression using dictionary methods and pattern matching in compressed sequences‖, IJAER, vol 10, no:15, pg 35064-35067.

[11] Ming-Bo Lin, 1997, “A parallel VLSI Architecture for the LZW Data Compression Algorithm,” International Symposium on VLSI Technology , Sytems and Applications, June 3-5, Taiwan, pp 98-101.

[12] Lin, Ming-Bo.2000, "A hardware architecture for the LZW compression and decompression algorithms based on parallel dictionaries." Journal of VLSI signal processing systems for signal, image and video technology 26, no. 3, pp: 369-381.

[13] MLin, Ming-Bo, Jang-Feng Lee, and Gene Eu Jan, 2006, "A lossless data compression and decompression

algorithm and its hardware architecture." IEEE TRANSACTIONS on very large scale integration (vlsi) systems 14, no. 9, pp: 925-936.

[14] Vichitkraivin, Perapong, and Orachat Chitsobhuk., 2009, "An Improvement of PDLZW implementation with a Modified WSC Updating Technique on FPGA." World Academy of Science, Engineering and Technology 36, pp: 611-615.

[15] Carus, A., and A. Mesut., 2010, "Fast text compression using multiple static dictionaries." Information Technology Journal 9, no. 5, pp: 1013-1021.

[16] De Agostino, Sergio, 2011, "Lempel–Ziv data compression on parallel and distributed systems." Algorithms 4, no. 3, pp: 183-199.

[17] Jayashree, M. J., and A. Sukesh Kumar., 2011, "Resourceful scheme of ECG compression using different Wavelet transforms and PDLZW method." In Electronics Computer Technology (ICECT), 2011 3rd International Conference on, vol. 3, pp. 99-102. IEEE.

[18] Bhatt, Malay, 2012, "Cascading of the PDLZW compression algorithm with arithmetic coding." In International Journal of Computer Applications,pp 21-24.

[19] Thakkar, Nirali, and Malay Bhatt., 2012, "Two-stage algorithm for data compression." In Proc. of the Intl Conf. on Advances in Computer, Electronics and Electrical Engineering, pp. 350-354.

[20] Nishad P M, R. Manicka Chezian, 2012, "Enhanced LZW (Lempel-Ziv-Welch) Algorithm by Binary search with multiple Dictionary to reduce time complexity for Dictionary creation in encoding and Decoding" In International Journal of Advanced Research in Computer Science and Software Engineering (IJARCSSE), pp. 192-198**.**

[21] Nishad P M, 2014, "A novel approach to reduce computational complexity of multiple dictionary lempel ziv welch mdlzw using indexed k-nearest twin neighbor iknth clustering and binary insertion sort algorithm", Ph. D Thesis,Bharathiyar university, Tamilnadu, India, pp. 22-32.

[22] Keerthy A S and Dr Manju Priya S, 2017, "Lempel-Ziv-Welch Compression using Multiple Dictionaries", Karpagam Journal of Computer Science, Vol 11, Issue 2.

[23] Wang, Congmao, and Dabing Zhang., 2011, "A novel compression tool for efficient storage of genome resequencing data." Nucleic acids research, 39.7, pp: 45.