Comparative analysis of Data Compression and Pattern Matching Techniques for Biological Big Data

Keerthy A S, Research Scholar, Department of Computer Science, Karpagam Academy of Higher Education,Coimbatore.

Dr. S ManjuPriya, Associate Professor,Department of Computer Science, Karpagam Academy of Higher Education,Coimbatore.

Abstract—DNA is the basic blueprint of an organism. Analysis of DNA gives insight into the vitals of an organism. Millions of short reads of DNA molecules have been produced in last decade with the advancement of next-generation sequencing technologies. The abundance of data generated (50-100 PB per year) makes it essential for efficient storing, processing and transmitting the data to promote research and future medical applications. The rising amount of data, demands more space as well as time for analysis. The era of personalized genomics is promoting individual's genome being sequenced for disease prevention, diagnosis and treatment [1]. Hence compression of data is needed to reduce storage requirement, transmission cost and network congestion [2]. Also Genomic data comprises of common subsequences like promoters, functional motifs, etc within it [3]. Analyzing variations in sequence to understand the biological importance is current area of research and is challenge to biologists [4]. The era demands algorithms that match patterns in time and space proportional to compressed size i.e. without decompressing. Evidently pattern matching is unavoidable in the analysis of genomic data. Pattern searching in compressed data would be assisting analysis of repetitions or common subsequences at a reduced cost and time. While arguments are still going on in deciding upon whether compressed search is faster than regular decompression followed by search [5], this paper compares different compression techniques and analyses available pattern matching algorithms.

Keywords—Genomic Data, Sequence compression, LZW, Pattern matching, Sequence alignment.

I. INTRODUCTION

The reduction of cost in genomic sequencing and with the availability of efficient sequencing technologies have kicked off several whole genome sequencing projects. The growth rate of genomic data has outrun the rate of data growth predicted by Moore's law [Figure1].

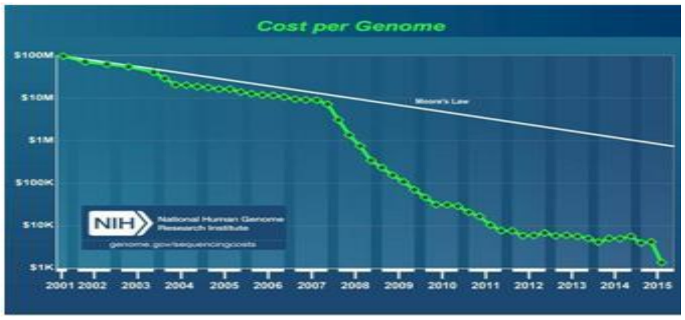

Figure 1 Source: N I H Website[41]

Data is generated faster than it can be analyzed [1]. The output of these varies widely with the viruses having a few thousands of bases to humans having approx 3 billion and amphibians up to even 120 bases. Analysis of this data gives insights into individual's health and benefits future medical research [6]. In higher eukaryotes, genomic data contains many copies of tandem repeats and essential genes. The analysis of repeats is vital as it affects gene regulation when present in regions with transcription factors [7].

The cost associated with processing, storage and transmission of data is a major challenge compared to cost of generating sequence. Discarding data after analysis is considered inappropriate by the research community; hence efficient methods to store data is inevitable. Centralized repositories makes data accession quick and easy; at the same time reduces duplication of data. Compressing data and storing them will increase efficiency with respect to storage and transmission.

The basic idea of compression is to take a stream of symbols and convert them into codes. Compression reduces the space requirement of storing data using mathematical algorithms which can be either lossy or lossless. Lossless compression ensures exact recreation of input file which is mandatory in the case of genomic data [6,8]. Compression algorithms utilize repetitive feature of inherent genomic sequences to achieve higher compression ratio. Traditional compression algorithms as gzip and bzip do not consider the biological characteristics like alphabet size, repeats and palindromes. Dictionary based algorithms, identify repetitions by book keeping previously occurring sequences [8]. DNA sequences may have long sequences of simple repeats in non-coding regions, reverse complements and some sequence conserved between individuals and between species [9]. Duplication in sequences is caused by retrotransposons, microsatellites, tandem repeats, etc which influence regulatory as well as evolutionary mechanisms [10].

A. Genomic Data Compression

The advanced technologies of genome sequencing has reduced cost and increased quantity of genomic data. The effect is the increase in cost of storage and transfer of data and has made compression of data inevitable. Development of novel compression algorithms is the current area of interest in the field of Computation.

Though compression of textual data has been an interesting area of research for past decade, genomic data cannot be compressed efficiently using the standard text compression methodologies and tools. The main reason behind it is the repetitive nature of genomic data which demands more versatile compressing techniques.

The last decade of genomic research has taken interest in developing algorithms for DNA compression. Majority of them are based on reference based compression where a reference genome is used to identify similarities between target sequences. The similarities and differences between the reference and target genome need only be stored along with reference genome. This being the advantage of the system, the main disadvantage is the requirement of a reference genome as well as storage of reference genome.

Another aspect of data compression is data indexing. Since individual genome is static data, indexing is applicable and makes pattern searching easy. A notable indexing technique involves the LZ-based indexes which are efficient in removing redundancy to a great extent [11].

The peculiarity of genomic data is the high level of similarity between individuals of same species. This factor of similarity can be used for efficient compression of data by detecting redundancy and constructing dictionary while allowing fetching of individual items in any order [9].

Substitution compression identifies repetitive subsequences and stores them in a codebook or dictionary (Figure 2). Each repeated subsequence in target is replaced by corresponding encoded subsequence in the codebook [1].

.

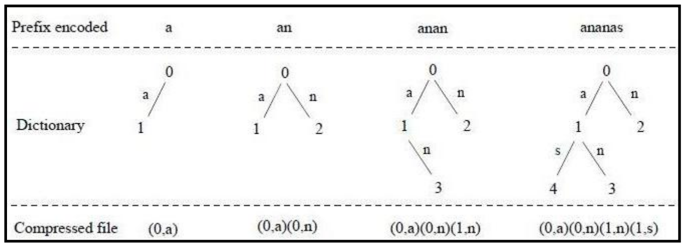

Figure 2: Compressing word ananas using LZ78. [27]

B. LZW based Text Compression

Substitution coding proposed by Lempel and Ziv uses pointers to previously compressed words or parts of words for compressing text data. Welch modified this compression by constructing a dictionary of words (from words or parts of words) in the text data and setting pointers to the words in the dictionary. LZW initiates a dictionary with characters in Σ, the character set. The text to be compressed is parsed into phrases and replaced by pointers to the dictionary. The longest match in the current position is identified and dictionary is updated by concatenating the next character to the match [5].LZW uses this dictionary as a tool to generate compressed output. New patterns are added to the dictionary by the revised algorithm. When a string already present in dictionary is encountered its index is output. To overcome the size limitation of the dictionary, suggested improvisations are to flush out the dictionary when it reaches maximum size or overwrite least recently used entries [12].

C. Pattern Discovery in Compressed Sequences

Pattern discovery in genomic data reveals the relevant patterns in biological sequences. These patterns may be related to gene regulation, DNA repair, etc. The pattern matching relates to identifying subsequences of statistical relevance, extracting significant motifs, molecular duplications like tandem repeats, microsatellites etc and identifying MicroRNAs [10].

Efficient pattern matching with compressed sequences aids in fast retrieval of patterns with efficient storage utilization. Comparing across the species helps us to identify how the species vary and hence how they diverged. Analyzing genomic data from population with in a species reveals how the species is evolving; comparing DNA of individuals within a population explains why individuals differ from one another. Finally, comparing DNA cells within an individual reveals how tissues develop and make different organs and muscles and what (when goes wrong) leads to diseases like cancer.

It is evident that pattern discovery is vital in the analysis of genomic data. Compressing genomic data efficiently, will substantially reduce the storage space as well as increase the speed of transmitting data across a network. When pattern matching is performed using compressed data, it helps to achieve results faster. The computational overhead of decompressing the compressed file before pattern matching can be avoided on performing the pattern matching in compressed sequences itself.

II. LITERATURE REVIEW

A. Compressing Genomic Data

Studies have been made by the renowned researchers in developing techniques of data compression and pattern matching. This paper analyzes more relevant and recent available methodologies for data compression.

ERGC: (Efficient Referential Genome Compression) algorithm uses a reference genome for compressing all given sequences. It stores the difference between reference sequence and target sequences by splitting reference and target sequence into equal sized strings. Each pair of strings is processed sequentially by aligning them using a greedy algorithm based on hash function. S.Saha and S.Rajasekaran claims a compression time of target sequence as O(nk) where n denotes the length of target sequence and k denotes largest value used in hashing [2].

Bakr and Sharawi have conducted a review of algorithms for DNA compression based on horizontal and vertical modes. Different horizontal mode compression algorithms are analyzed based on substitution methods, statistical methods, etc. The analysis concludes LZ as an efficient compression method suitable for compressing genomic data [13].

Zhu et al has done a comprehensive review of reference based and reference free compression methods based on the compression ratio, memory usage, compression/decompression time. They observed that with availability of appropriate reference sequence is favorable whereas reference free methods are self-contained and balance time and compression ratio efficiently [1].

NGC, reference based compressor proposed by Popitsch and Haeseler, takes in an alignment file, reference sequence and a set of parameters and outputs a lossless or lossy compressed file. They claim 30%-60% space saving in lossless compression and upto 98% saving in lossy compression. Still the method fails in the case of unmapped reads and stores the unmapped reads in separate files in BAM(Binary Sequence Alignment/Map) format[14].

A novel dictionary based approach proposed by Nishad and ManickaChezian combines fixed length binary code with LZW (Lempel-Ziv-Welch) compression algorithm for genomic data. In the first phase, the input sequence is divided into fragments of four nucleotides and each fragment is assigned a binary code. In the second phase a binary tree is constructed for dictionary. The authors claim an efficiency of 94.59% for repetitive as well as non-repetitive DNA sequences [15].

GRS (Genome ReSequencing data) is a tool implemented to compress genome resequencing data without SNP(Single Nucleotide Polymorphism) maps. The reference chromosome and input chromosome is split into same intervals. The varied sequence percentage is calculated based on each nucleotide frequency. Wang and Zhang claim a 195 fold compression for GRS when tested on a Korean personal genome [4].

DNABIT Compress proposed by Rajarajeswari and Apparao assigns binary bits for smaller segments of DNA bases to compress both repetitive and non-repetitive DNA sequence. Also it is designed to find tandem repeats, which aids phylogenetic analysis and disease diagnosis [7].

Mehta and Patel demonstrate a data compression using hash based data structure. It builds a hash table and assigns a unique character to each hash keys. The entire DNA sequence is spilt into DNA fragments, hash it into single character and finally compress [16]

GenCompress proposed by Chen et al is a dictionary based compression method that detects irregularities in DNA such as mutation and crossover. GenCompress works by identifying part of input that is already compressed and finding encoding the remainder. It also detects complemented palindrome in DNA sequences [17].

COMpression using RedundAncy of DNA (COMRAD) proposed by S. Kruppu, et. al. based on RAY, is a general purpose compression algorithm which identifies repeats in large DNA data sets. COMRAD compress data over multiple passes by creating a frequency dictionary. Random access to individual sequences and subsequences are allowed by the algorithm. Though it compresses upto 0.25 bits per base, since the number of iterations depend on number of substitutions in each of previous iterations, the cost of compression cannot be predicted exactly. COMRAD is effective for long range repetition detection and compressing large DNA sets but it is memory intensive. [9]

LZWGA (Liv-Zempel-Welch Genetic Algorithm) combines genetic algorithm with LZW compression technique to represent chromosome data in compressed format. The method proposed by Kunasolet. al. needs to decompress the data before fitness evaluation [18].

Giancarlo et al reviews different aspects of bioinformatics and computational biology where compressionis used. Pattern discovery in biological sequences involves analysis of tandem repeats, microRNA, extraction of significant motifs, etc.The study suggests that versatility, parameter free data, association mining and speed are improved by compressing in biological data for investigation [10].

Toshiko et al proposes effective compression of DNA sequences by combining Context Tree Weighting Method (CTW) and LZ. The algorithm explores reverse complements and approximate repeats in the sequence employing hash tables and dynamic programming. On successful identification, the algorithm represents the subsequence by storing its length and distance [19].

B. Using LZW in Compressing text data

Wang proposes LZW for compressing large files by table pruning. Table pruning improves efficiency of LZW dictionary and is carried out by choosing Least Recently used entry from am self-organizing file. Aging replacement is also used along with it for improving efficiency. LZW/aging and LZW/LRU was tested using text files and E.Coli sequence data. LZW/aging gave better performance compared to LZW/LRU only if an ideal TTL is computed which aids table pruning to overcome the disadvantages of table flushing [20].

LZW++ proposed by Kamiret. al. provides an enhancement over LZW by taking three characters at a time instead of one. The triplets are assigned codes and stored to data dictionary. The experimental results claim to exhibit better performance when tested with text and pdf file formats [21].

A comparative analysis of flexible parsing by LZ-77, LZW and LZW-FP is carried out by Matiaset. al. from data coming from three different sources viz data to measure asymptotic redundancy in data by generating pseudo random numbers, biological data as protein and DNA sequences as well as CT and MRI (Magnetic Resonance Imaging) scans and text files from two data compression tools. The test results points out to a dictionary size of 2^{24}giving optimal performance to all three types of data [22].

Rajpoot and Yao compared the standard LZW with variations such as LZW with PB code, LZW with DS code, LZW with balanced code and LZW with dynamic block shifting. The Phase in Binary trees has all sub trees as complete binary trees. PB(Phase-in-Binary) codeword is defined by the path traversed from root to nth leaf node. The Depth Span tree is a rooted binary tree whose left child is a complete binary tree and right child has maximum depth. The code words ensure maximum code length contrast. The Balanced Code tree is a rooted binary tree which is either complete binary tree or complete up to a depth. Code words produced will be of average length. The comparison was done using data from ordinary corpus that held data from multiple sources, large corpus having file size >1MB including E.Coli sequence, and two text files, and an artificial corpus having files with no repetition and files with repetitions. The experimental results claim that LZW with DS and LZW with balanced codes perform optimally with respect to code length contrast and average code length respectively [23].

Rahul Gupta et al proposed a LZW based compression algorithm for compressing dynamic textual data. The input text is compressed and stored as blocks of data. The algorithm maintains an indexed lexicon table for storing already encountered substrings and information about blocks. Whenever new data is to be appended; the respective block is decompressed,new data is appended to the block and compressed. In standard LZW algorithm the entire compressed file has to be decompressed to append any additional data [24].

Kodituwakku and Amarasinghe performed experimental analysis of lossless data compression algorithms based on compression ratio, compression factor, saving percentage, compression time entropy and code efficiency for textual data. For non statistical based algorithms like RLE and LZW, entropy and code efficiency could not be calculated. It was also observed that LZW does not work well for large files as the dictionary size for compression and decompression was huge [25].

Parvinder Singh et al proposed an enhancement for improving LZW algorithm by eliminating frequent flushing of dictionary to reduce processing time. The authors point out the short comings of LZW algorithm for text data compression and suggest improvisations. Whenever a dictionary gets filled a replacement strategy is initialized which replaces shorter strings by longer string. This provides efficient compression ratio. Another suggestion was a bi-level dictionary modification scheme using two dictionaries. Primary dictionary stores frequently used entries and have smaller code size. Secondary dictionary stores codes with larger size. As the primary dictionary gets filled up the replacement strategy removes nodes from primary dictionary to secondary dictionary. This can be further enhancedwith the usage of Bi-mode Encoder. Individual bytes are sent in uncompressed mode (as it is) and sequence of bytes are sent in compressed mode (compressed using LZW)[26].

C. Compressed Pattern Matching

G. Navarro and M. Raffinothas presented a hybrid compression technique that allows fast searching as LZ78 and at the same time maintains many of the features of LZ77. They compressed text is maintained as a sequence of blocks and searching is done without decompressing it [27].

Kida et al proposed to incorporate Shift-And approach to pattern matching in text compressed using LZW. They give experimental evidence of 1.5 time faster comparison compared to uncompressed file. They also propose extensions to generalized pattern matching, pattern matching with k mismatches and multiple pattern matching. The generalized pattern matching is identifying patterns where pattern element is a set of characters. Pattern matching with k mismatches is carried out by counting up the number of mismatches instead of directly comparing pattern and text. Multiple patterns are identified by keeping one bit vector per pattern and performing Shift-and in parallel [5].

The leftmost occurrence of a pattern in a compressed text can be identified in O(n+m) time and space as proposed by Gawrychowski. He proposes to simulate the Knuth-Morris-Pratt algorithm by matching snippets instead of single characters [28].

Amir et. al. attempts pattern matching in z-compressed files using a dictionary trio. Their proposed algorithm consists of a pattern preprocessing part and a text compressed string scanning part. In the text scanning construction and updating of dictionary and pattern search is carried out. The expected runtime is $O(n\log m+m^2)$ and space complexity is $O(m^2)$[29].

Gonzalo Navarro has attempted to solve the search of regular expressions in Ziv-Lempel compressed text. The compressed text is represented as a sequence of blocks, $z=b_1b_2b_3\ldots b_n$ where each block is a substring. The blocks are formed by concatenating previous block and an explicit letter. Searching for regular expressions in this LZW compressed text claims to achieve twice faster search results than decompressing and searching [30].

A proposal to solve pattern matching allowing k errors was proposed by Navarro et. al. The algorithm claims to find R occurrences of a pattern of length m with k errors over n blocks of LZ78/LZW compressed text. The algorithm claims O(kmn+R)worst case time and O(k^2 n +R) average time[31].

A bit parallel approach for approximate string matching proposed by Matsumoto et. al claims to achieve a time and space complexities of the order of O(k^2n + km) and O(k^2n) respectively for LZW compressed data [32].

Shibata et. al presents an algorithm or pattern matching in compressed text files using Byte Pair Encoding(BPE) . BPE uses substitution table to map every substitution made during compression. For every iteration of compression the most frequently occurring pairs of characters are identified and it is replaced by a character that is none occurring in text. The authors claim time and space complexity of the order O(||D||+|S|+m^2+r) and O(||D||+m^2) respectively. ||D|| is the size of dictionary D and |S| is the length of sequence S [33].

Navarro et. al. proposes two search algorithms for compressed natural language based ob Huffman compression. The first one automaton based, plain filter less algorithm overcomes the disadvantages of previous version that compressed the pattern also before initiating the search. The second one called plain filter tries to match patterns directly in Huffman compressed text as well as verify for false matches using automaton based algorithm. Though the algorithms claim to perform better than Compress and Gzip tools, the search strategy used is sequential which is unfortunate [34].

Boyer-Moore algorithm proposed by Shibata et. al. was an initial attempt to perform compressed pattern matching. The method focuses on a token at a time and shift focus to the right until a mismatch occurs. Upon encountering a mismatch, focus is shifted to the left. The algorithm works successfully for small patterns [35].

Based on Amir's approach on Pattern Matching, Tao Tao et.al proposed, Aho-Corasick based pattern matching algorithm for LZW compressed sequences. The space and time complexity reported by the authors are O(mt) and O(n+mt+r) respectively [36].

Compressed String Matching (CSM) and Fully Compressed String Matching (FCSM) differs in compressing text alone in CSM and both text and pattern in FCSM.Gasieniecet. al. propose sequential as well as parallel approach for pattern matching based on FCSM for LZW compressed sequences. The time complexity of the proposed algorithm is O((n+m)log(n+m)) [37].

Kida et al proposes a Collage system which is a framework based on existing dictionary based algorithm for compressed pattern matching. The system identifies all occurrences of a pattern in a text without decompression. They claim the algorithm to run in O((||D||+|S|)*height(D)+m^2+r) time and O(||D||+m^2) space. To identify multiple patterns the system needs to be modified [38].

Farach and Thorup presents a LZ1 compressed matching algorithm to perform string matching in a compressed text without uncompressing it. For a given compressed string of size N, representing a text of size U, and a given pattern of size p the algorithm runs in time O(N$\log^2$ U/N + p)[39].Comparison of the algorithms is as shown in Table I.

Table I: Comparison of different pattern matching algorithms.

Pattern matching Technique Used	Compression Technique	Space Complexity	Time Complexity
Knuth-Morris-Pratt[28]	LZW	O(n+m)	O(n+m)
Dictionary Trio[29]	Z compress	O(m^2)	O(nlogm+m^2)
Approximate string Matching[31]	LZ78/LZW	-	O(k^2 n +R)
Bit Parallel[32]	LZW	O(k^2n)	O(k^2n + km)
Approximate string Matching [33]	BPE	O(||D||+m2)	O(||D||+|S|+m2+r)
Aho-Corasick [36]	LZW	O(mt)	O(n+mt+r)
FCSM[37]	LZW	-	O((n+m)log(n+m))
Collage system [38]	Dictionary based	O(||D||+m^2)	O((||D||+|S|)* height(D)+m^2+r)
Compressed Matching[39]	LZ1	-	O(N$\log^2$ U/N + p)

A comparative analysis of the different pattern matching techniques discussed in this study points to the abundance of research that is going on in this area. It is well observed that majority of the algorithms use LZW/Dictionary based compression. The algorithms analyzed were all defined for text data compression and pattern matching. It is clearly evident from the Table:1 that LZW compressed text which uses Knuth-Morris-Pratt method for pattern matching gives the most efficient space and time utility. The major drawback is that the algorithm identifies only one occurrence of a pattern.

III. Future Work

The study points out the unavailability of a comprehensive system for compressing genomic data. Although many compression algorithms and tools are available for textual data and some of them are using genomic data for testing purpose there is no efficient tool specialized in compressing genomic data based on dictionary method. At the same time several researchers point out the need and advantages of pattern matching in the area of medical science and plant and animal development. This point to the need of an efficient algorithm to discover patterns from compressed genomic data.

The advancements in sequencing techniques have led to sufficient availability of genomic data. Hence for effective storage and transportation of Biological Big Data we propose to develop a novel algorithm based on LZW for compression of genomic data. We propose to modify the existing LZW algorithm by reducing the dictionary size and choosing a better dictionary updating methodology.

Pattern matching in biological sequences may aid to reveal many secrets and solves mysteries in the genomic data of individuals. Available pattern matching techniques works on uncompressed genomic data. Already compressed pattern matching is available for text data. We identify the need for efficient pattern matching technique in compressed genomic data and hence propose to develop a methodology to identify patterns in LZW based compressed genomic data without uncompressing it.

IV. CONCLUSIONS

A recent study brings out the need for a promising compression algorithm for compressing biological data. The study also point out the importance of pattern matching in biological data as well as advantage of compressed pattern matching [40]. The effectiveness of dictionary based compression algorithms in compressing textual data is evident from the literature referred to in this analysis.

In the current era of mounting volume of genome sequencing and resequencing, considering the cost of storing and transmitting this data, efficient compression tools are constantly in demand. These tools could assist in analysis of human genome variation between individuals and hence could be a key for progress in personal medicine effects. LZW has been successfully used in textual data compression, but its weakness is the increased dictionary size hence increased computational cost. A lossless dictionary based tool using LZW with reduced dictionary size would definitely help in achieving high compression ratio and reduced computational cost and accelerate pattern matching in compressed sequences.

REFERENCES

[1] Zhu, Z., Zhang, Y., Ji, Z., He, S., & Yang, X, "High-throughput DNA sequence data compression."*Briefings in bioinformatics* 16.1: 1-15, 2015.

[2] Saha, Subrata, and SanguthevarRajasekaran. "ERGC: an efficient referential genome compression algorithm." *Bioinformatics*: btv399, 2015.

[3] Eric C Rouchka, "Pattern Matching Techniques and their applications to Computational Molecular Biology", 1999.

[4] Wang, Congmao, and Dabing Zhang. "A novel compression tool for efficient storage of genome resequencing data." *Nucleic acids research* 39.7:e45-e45,2011.

[5] Kida, Takuya, et al. "Shift-And approach to pattern matching in LZW compressed text." *Combinatorial Pattern Matching*. Springer Berlin Heidelberg, 1999.

[6] Cánovas, Rodrigo, and Alistair Moffat. "Practical compression for multi-alignment genomic files." *Proceedings of the Thirty-Sixth Australasian Computer Science Conference-Volume 135*. Australian Computer Society, Inc., 2013.

[7] Rajarajeswari, Pothuraju, and AllamApparao. "DNABIT Compress– Genome compression algorithm." *Bioinformation* 5.8: 350, 2011.

[8] Sebastian Wandelt, Marc Bux and Ulf Leser, "Trends in Genome Compression", *Current Bioinformatics*, pg315-326,2014.

[9] ShanikaKuruppu, Bryan Beresford-Smith, Thomas Conway and Justin Zobel, "*Iterative Dictionary Construction for Compression of Large DNA Data sets*", Published by IEEE/ACM Transactions on Computational Biology and Bioinformatics, pg 137-149,2012.

[10] Raffaele Giancarlo, DavideScaturro and FilippoUtro, "Textual Data compression in computational biology: a synopsis", *Bioinformatics*, pg 1575-1586, 2009.

[11] Sebastian Deorowicz,Szymon Grabowski, "DNA compression for sequencing Data", *Algorithms for Molecular Biology*,2013

[12] Mohammed Al-Iaham, Ibraheim M. M. El EMary, "Comparitive study between various Algorithms of Data Compression Techniques", *International Journal of Computer Science and Network Security*., pg 281-29, 2007.

[13] Bakr, Nour S., and Amr A. Sharawi. "DNA lossless compression algorithms: review." *American Journal of Bioinformatics Research* 3.3: 72-8, 2013.

[14] Popitsch, Niko, and Arndt von Haeseler. "NGC: lossless and lossy compression of aligned high-throughput sequencing data." *Nucleic acids research* 41.1: e27-e27, 2013.

[15] Nishad PM. and Dr. R. ManickaChezian, " 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* (0975 – 8887) Volume 43– No.1, pp. 7-9, April 2012.

[16] Mehta, Ateet, and Bankim Patel. "Dna compression using hash based data structure." *International Journal of Information Technology & Knowledge Management* 3: 383-386, 2010.

[17] Chen, Xin, Sam Kwong, and Ming Li. "A compression algorithm for DNA sequences." *IEEE engineering in medicine and Biology* 20.4: 61-66, 2001.

[18] Kunasol, Naris, WorasaitSuwannik, and PrabhasChongstitvatana. "Solving one-million-bit problems using ZWGA." *Communications and Information Technologies, 2006.*

[19] Toshiko Matsumoto, KunihikoSadakane, Hiroshi Imai, "Biological Sequence Compression Algorithms", *Genome Informatics*, pg 43-52,2000.

[20] Wang, Chung-E. "Dynamic LZW for Compressing Large Files." 2010.

[21] Kamir, YusofMohd, Mat DerisMohdSufian, and Abidin Ahmad Faisal Amri. "Study of Efficiency and Capability LZW++ Technique in Data Compression."*World Academy of Science, Engineering and Science*, 2009.

[22] Matias, Yossi, NasirMahmoodRajpoot, and SuleymanCenkSahinalp. "The effect of flexible parsing for dynamic dictionary based data compression."*Proceedings. Data Compression Conference,IEEE*, 1999.

[23] Yao, Zhen, and NasirMahmoodRajpoot. "Less redundant codes for variable size dictionaries." *Proceedings DCC 2002: data compression conference*. Institute of Electrical and Electronics Engineers, 2002.

[24] Rahul Gupta, Ashutosh Gupta, SunnetaAgarwal, "A Novel Data Compression Algorithm for Dynamic Data", IEEE, pg 266-271, 2008.

[25] S. R. Kodituwakku, U.S. Amarasinghe, "Comparison of Lossless Data Compression Algorithms for text data", *Indian Journal of Computer Science and Engineering*, pg 416-425, 2010.

[26] Parvinder Singh, ManojDuhan, Priyanka, "Enhancing LZW Algorithm to increase Overall Performance", IEEE, pg1-4, 2006.

[27] Navarro, Gonzalo, and Mathieu Raffinot. "A general practical approach to pattern matching over Ziv-Lempel compressed text." *Combinatorial Pattern Matching*. Springer Berlin Heidelberg, 1999.

[28] Gawrychowski, Paweł. "Optimal pattern matching in LZW compressed strings." *ACM Transactions on Algorithms (TALG)* 9.3: 25, 2013.

[29] Amir, Amihood, Gary Benson, and Martin Farach. "Let sleeping files lie: Pattern matching in Z-compressed files." *Journal of Computer and System Sciences* 52.2: 299-307, 1996.

[30] Navarro, Gonzalo. "Regular expression searching over Ziv-Lempel compressed text." *Combinatorial Pattern Matching*. Springer Berlin Heidelberg, 2001.

[31] Kärkkäinen, Juha, Gonzalo Navarro, and EskoUkkonen. "Approximate String Matching over Ziv—Lempel Compressed Text." *Combinatorial Pattern Matching*. Springer Berlin Heidelberg, 2000.

[32] Matsumoto, Tetsuya, et al. "Bit-parallel approach to approximate string matching in compressed texts." *String Processing and Information Retrieva[SPIRE}l, 2000.*

[33] Shibata, Yusuke, et al. "Speeding up pattern matching by text compression."*Algorithms and Complexity*. Springer Berlin Heidelberg, 306-315, 2000.

[34] Silva de Moura, Edleno, et al. "Fast and flexible word searching on compressed text." *ACM Transactions on Information Systems (TOIS)* 18.2: 113-139, 2000.

[35] Shibata, Yusuke, et al. "A Boyer—Moore Type Algorithm for Compressed Pattern Matching." *Combinatorial Pattern Matching*. Springer Berlin Heidelberg, 2000.

[36] Tao Tao, Amar Mukherjee, "Pattern Matching in LZW Compressed Files", IEEECS, pg 929-937, 2005

[37] LeszekGasieniec and WojciechRytter, "Almost optimal fully LZW-compressed pattern matching", Proceedings Data Compression Conference, pg 316-325, 1999.

[38] Takuya Kida, Yusuke Shibata, Masayuki Takeda, Ayumi Shinohara, SetsuoArikawa, "A unifying framework for Compressed Pattern Matching", IEEE, pg 89-96, 1999.

[39] Martin Farach, MikkelThorup, "String Matching in Lempel-Ziv Compressed Strings", ACM, pg 703-712, 1995.

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

[41] https://www.genome.gov/images/content/costpergenome2015_4.jpg