Survey of Compression of DNA Sequence

Dhajvir Singh Rai
M.Tech(Computer Science & Engineering)
BTKIT, Dwarahat, Almora

R.K.Bharti,Ph.D
Asst. Prof.(Deptt. Of Computer Science)
BTKIT, Dwarahat Almora

Bhawana Parihar
Asst. Prof.(Deptt. Of Computer Science)
BTKIT, Dwarahat, Almora

ABSTRACT
Compression of large collections of data can lead to improvements in retrieval times by offsetting the CPU decompression costs with the cost of seeking and retrieving data from disk. In this paper, the author has study the different compression method which can compress the large DNA sequence. In this paper, authors have explored the DNA compression method that is COMRAD, which is used to compare with the dictionary based compression method i.e. LZ77, LZ78, LZW and general purpose compression method RAY. In this, authors have analyzed which one algorithm is better to compress the large collection of the DNA Sequence. Compression table and the line graph show that which compression algorithm has a better compression ratio and the compression size. It also shows that which one has better compression and decompression time.

Keywords
LZ77, LZ78, LZW, RAY, COMRA , DNA Sequence

1. INTRODUCTION
The increasing utilization of new sequencing technologies is leading to changes in the kinds of genetic data that are being gathered and stored. The Human Genome Project (HGP) produced a consensus sequence for much of the human genome, while similar work produced reference DNA data for other organisms. Recently, there has been a shift toward producing data that represent the sequences of individuals. In addition to the original HGP genome, there are now sequences for James Watson [1], two men of Nigerian [2] and Chinese [3] descent, and five southern African genomes [4], among many others. The falling cost of high-throughput sequencing is enabling more ambitious activities such as the 1000 Genomes Project,1 which aims to determine the variations in the human population by analyzing the genomes of at least 1,000 individuals; and the Personal Genomes Project,2 which aims to improve the understanding of how genetics and the environment affect human traits, beginning a gradual shift toward personalizing medical treatment.

In the past, researchers were able to rely on the trend of cheaper storage space to store the genomic data being generated. However, certain trends in sequencing, such as the maturation of second generation sequencing technologies, the creation of cheaper sequencing machines and then third generation of DNA sequencing technologies, are responsible for an ever increasing number of genomes being sequenced. These genomes are from both new species and more individual creatures having their genomes sequenced. This increasing rate of sequencing is outpacing Kryder's law even after general purpose compression algorithms are applied [5].

The genome of an organism is the DNA within that organism. DNA is comprised of nucleotides, also referred to as bases, which can be represented by the characters A, C, G and T for Adenine, Cytosine, Guanine and Thymine, respectively. In addition to those four letters specifying specific bases, there are letters which are wildcards and represent an arbitrary base or set of bases such as N for a non-specified string of bases and M for either Adenine or Cytosine. Converting the physical DNA to a data file is called sequencing. The human genome is about 3,000 megabytes of uncompressed data. In comparison, the complete works of William Shakespeare is about 5 megabytes.

DNA sequences may contain repeated substrings within a sequence; however, in database of sequences, the most significant repeats occur between sequences, usually those of the same or similar species. To help manage large genomic databases, compression algorithms that capture and efficiently encode this repeated information are employed. Compression algorithms specific to DNA sequences have been around for some time [6, 7, 8, 9, 10, 11, 12]. How- ever, most existing algorithms are unsuitable for compressing large datasets of multiple sequences. More recently, algorithms that compress large repetitive datasets, that also support random access and search on the compressed sequences, known as self-indexes, have emerged. Some of these algorithms are specific to DNA compression and support random access queries [13, 14]. Others can compress general datasets and also implement search queries on the compressed sequences [15].

The paper is organized as follows: Section I contains a brief Introduction about Compression of DNA Sequence, Section II presents a brief explanation about Dictionary based compression techniques, Section III discusses about RAY Algorithm, Section IV discusses about the COMRAD algorithm Section V has its focus on comparing the performance of Dictionary based compression technique, RAY and COMRAD algorithm and the final section contains the Conclusion.

2. DICTIONARY BASED COMPRESSION
A dictionary-based compression scheme [18] reads in input data and looks for groups of symbols that appear in a dictionary. If a string match is found, then a pointer or index into the dictionary can be output instead of the code for the symbol[21]. The longer the match, the better the compression ratio.

2.1 Static Dictionary

Choosing a static dictionary technique is most appropriate when considerable prior knowledge about the source is available. This technique is especially suitable for use in specific applications. For example, if the task were to compress the student records at a university, a static dictionary approach may be the best. This is because we know ahead of time that certain words such as "Name" and "Student ID" are going to appear in almost all of the records. One of the more common forms of static dictionary coding is diagram coding. In this form of coding, the dictionary consists of all letters of the source alphabet followed by as many pairs of letters, called diagrams, as can be accommodated by the dictionary. For example, suppose we were to construct a dictionary of size 256 for diagram coding of all printable ASCII characters. The first 95 entries of the dictionary would be the 95 printable ASCII characters. The remaining 161 entries would be the most frequently used pairs of characters. The diagram encoder reads a two-character input and searches the dictionary to see if this input exists in the dictionary. If it does, the corresponding index is encoded and transmitted. If it does not, the first character of the pair is encoded. The second character in the pair then becomes the first character of the next diagram. The encoder reads another character to complete the diagram, and the search procedure is repeated.

2.2 Adaptive Dictionary

Most adaptive-dictionary-based techniques have their roots in two landmark papers by Jacob Ziv and Abraham Lempel in 1977 [20] and 1978. These papers provide two different approaches to adaptively building dictionaries, and each approach has given rise to a number of variations. The approaches based on the 1977 paper are said to belong to the LZ77 family (also known as LZl), while the approaches based on the 1978 paper are said to belong to the LZ78, or LZ2, family. The transposition of the initials is a historical accident and is a convention we will observe in this book. In the following sections, we first describe an implementation of each approach followed by some of the more well-known variations.

2.2.1 LZ77 Approach:

The first compression algorithm described by Ziv and Lempel is commonly referred to as LZ77. In the LZ77 approach, the dictionary is simply a portion of the previously encoded sequence.

The encoder examines the input sequence through a sliding window as shown in Figure 1. The window consists of two parts, one is a *search buffer* that contains a portion of the recently encoded sequence, and another is a *look-ahead buffer* that contains the next portion of the sequence to be encoded. In Figure 1, the search buffer contains eight symbols, while the look-ahead buffer contains seven symbols

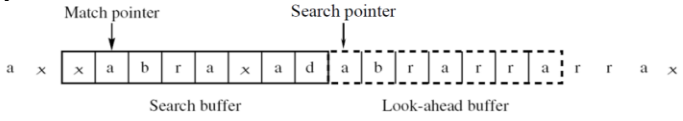

Fig 1 "sliding window"

The encoding algorithm

1. Set the coding position to the beginning of the input stream;
2. find the longest match in the window for the lookahead buffer;
3. output the pair (P,C) with the following meaning:
   - P is the pointer to the match in the window;
   - C is the first character in the lookahead buffer that didn't match;
4. if the lookahead buffer is not empty, move the coding position (and the window) L+1 characters forward and return to step 2.

Decoding

The window is maintained the same way during encoding. The algorithm reads a pair (P,C) from the input in each step. It outputs the sequence from the window specified by P and the character C.

For many types of data, compression ratio this method achieves is very good, but the encoding can be quite time-consuming, since there is many comparisons to perform between the lookahead buffer and the window. On the other hand the decoding is very easy to apply and fast. Low memory is required for both the encoding and the decoding.

Advantage:-

1- It able to compress the text well using a small amount of memory and fast speed.
2- Compression ratio of this method is achieved very good of many type of data.
3- It compress speed and decompress speed of data is very good.
4- Memory requirement are low for both encoding and decoding.

Disadvantage:

1- Encoding of the data can be time consuming.

2.2.2 LZ78 Approach:

The LZ78 program takes a different approach for building and then maintaining the dictionary. LZ78 [21] makes its dictionary out of all of the previously seen symbols in the input text instead of having a limited-size window into the preceding text. A dictionary of strings make a character at a time, instead of having carte blanche access to all the symbol strings in the preceding text,. The first time the string "Size" is seen, for example, the string "Si" is added to the dictionary[18]. The next time, "Siz" is added. If "Size" is seen again, it is added to the dictionary.

LZ78 inserts single or multi-character, non-overlapping, different patterns of the message to be encoded in a Dictionary. The multi-character patterns are in the form: $C_0C_1 \ldots C_{n-1}C_n$. The prefix of a pattern consists of all the pattern characters except the last: $C_0C_1 \ldots C_{n-1}$.

The encoding algorithm

1. At the start, the dictionary and P are empty;
2. C := next character in the charstream;
3. Is the string P+C present in the dictionary?
   a. if it is, P := P+C (extend P with C);
   b. if not,
      i. output these two objects to the codestream:
         - the code word corresponding to P (if P is empty, output a zero);
         - C, in the same form as input from the charstream;
      ii. add the string P+C to the dictionary;
      iii. P := empty;
   c. are there more characters in the charstream?
      - if yes, return to step 2;
      - if not:

i. if P is not empty, output the code word corresponding to P;
ii. END

The decoding algorithm

1. At the start the dictionary is empty;
2. W := next code word in the codestream;
3. C := the character following it;
4. output the string.W to the codestream (this can be an empty string), and then output C;
5. add the string.W+C to the dictionary;
6. are there more code words in the codestream?
   i. if yes, go back to step 2;
   ii. if not, END.

Advantage:

1- The biggest advantage over the LZ77 algorithm is the reduced number of string compression in each encoding speed.
2- It compress the large amount of the data into smaller amount which have the less memory required for storing the compress data.
3- Compression ratio is better than the LZ77.
4- Compression and decompression speed is more better than the LZ77.

Disadvantage:

1- It is expensive to store all symbols from the input.
2- Substring are added to the dictionary arbitrarily. Therefore, some entry may newer be referred to, so the dictionary consume more space than necessary.

2.2.3 LZW Approach:

LZW is a loss less compression algorithm. This method was invented and published by Lempel and Ziv, which is known as LZ78. LZW [22] algorithm replaces strings of characters with single codes. It doesn't do any analysis of the incoming text. Instead, it adds every new string of characters to a table of strings. When a single code is output instead of a string of characters then compression occurs.

The code that LZW algorithm outputs can be of any arbitrary length but instead of a single character it must have more bits in it. By default, the first 256 codes are assigned to standard character set and then he remaining codes are assigned to strings as the algorithm proceeds [23]. This means codes 0-255 refer to individual bytes where as codes 256-4095 which is the maximum limit of the dictionary refer to sub strings.

LZW compression provides a better compression ratio in most applications, that is why it became the first widely used general-purpose method on computers. It typically reduces to about half of its original size On large English texts. Other kinds of data are also quite usefully compressed in many cases.

```
LZW encoding algorithm:
Initialize Dictionary with 256 single character strings and their corresponding ASCII codes;
Prefix ← first input character;
CodeWord ← 256;
while(not end of character stream){
    Char ← next input character;
    if(Prefix + Char exists in the Dictionary)
        Prefix ← Prefix + Char;
    else{
        Output: the code for Prefix;
        insertInDictionary( (CodeWord , Prefix + Char) ) ;
        CodeWord++;
        Prefix ← Char;
    }
}
Output: the code for Prefix;
```

Advantage:

1- It is a lossless compression algorithm. Hence ho information is lost
2- One need not pass the code table between the two compression and the decompression.
3- Simple fast and good compression.
4- There is no need to analyze the incoming text.

Disadvantage:

1- What happen when the dictionary become too large.
2- One approach is to throw the dictionary array when it reaches a certain size.
3- Useful only for a large amount of text data where the redundancy is high.
4- Although the algorithm is pretty simple but implementation is complicated mainly because management of string table.
5- The method is good for the text file but not for the other type of file.

3. RAY ALGORITHM

Ray[16] is a general-purpose compression algorithm that is possible to use compressing the DNA sequence. The ray algorithm is similar to Re-pair, except the occurrence od many. The algorithm also supports random access into the compressed data. Unlike Re-pair and Sequitur, RAY is a multi-pass algorithm. The algorithm has follows:

Input:
-input string
-frequency threshold f
Algorithm:

Step 1. Create frequency dictionary of symbols pairs.

Step 2. Determine the symbol pairs that could be replaced (Candidates) by through triplet and if the left most pair has higher frequency then the right most pair and count of the left most pair is at least f, then increment the candidates count of the left most pair by one.

Step 3. The symbols pair with counts of at least two from the step 2 are selected to replaced so they are added to the dictionary.

Step 4. Update the frequency dictionary to be consistent with a new string. Step2-4 are repeated until a terminating condition is satisfied.

The decompression algorithm of RAY is similar to Re-pair. The dictionary can either be stored in memory as the hierarchy of rules, or the right-hand sides of rules can be expanded to support fast decompression at the cost of storing expanded substrings. As each symbol is decoded, if it is a non-terminal, the dictionary is used to retrieve the substring represented by that non-terminal.

Advantage:

1- Ray detect the global repeats and use less iteration
2- Ray compresses better than gzip and compress
3- It has faster decompression
4- Ray can access substring from the dataset faster than accessing the same segments from the uncompress collection on disk
5- It can randomly access the substring from the dataset.

Disadvantage:

1- Algorithm required multiple passes through the input to find repetition.
2- Repeats are detected globally, the memory usage of dictionary may be high for the large input.
3- It has slower compression

4. COMRAD ALGORITHM

Comrad [17] is a dictionary compression algorithm that detects repeated substrings in the input, and encodes them efficiently to achieve compression. Comrad also operates in multiple iterations, however, it is a DNA-specific disk-based algorithm designed to compress large DNA datasets. Instead of replacing pairs of frequent symbols, Comrad replaces repeated substrings of longer lengths to reduce the number of iterations. The first iteration of COMRAD counts distinct L length substrings and the repeated substrings from most frequent to least frequent are replaced with nonterminals and a dictionary is formed. The input sequence now consists of a combination of terminals and non-terminals. In subsequent iterations, the counts of distinct substrings that satisfy a certain set of patterns is recorded (see [13]), and again substrings from most frequent to least are replaced with non-terminals. The iterations continue until there are no substrings of the above form remaining with at least a count of F (only substrings with frequency F are eligible for replacement). The algorithm outputs the input sequence with repeated substrings replaced by non-terminals, and like Re-pair, a dictionary

containing the non-terminals mapping to the substrings they replace. As with the Re-pair dictionary, we expand non-terminals and append them to create a reference sequence.

Algorithm:

Input:

1: Set of DNA sequences S_0
2: Iteration 1 substring length L
3: Minimum frequency threshold F
4: Set of patterns P

Output:

1: Compress DNA sequences S_k
2: Dictionary of symbols D

Algorithm:

1: Create the frequency dictionary D_1 of all L length substring, with frequency of at least F, for the input DNA sequences S0
2: Encode the input sequences S_0 to get sequences S_1
3: k<- 2
4: **while** the dictionary continues to grow do
5: Create the frequency dictionary D_k of all substring matching pattern in P, with the frequency at least F, for the input sequences S_{k-1}
6: Encode the input sequences S_{k-1} to get sequences S_k
7: k<- k+1
8: **end while**
9: Cleanup Dictionary D to remove infrequent non-terminals and make numbering consecutive

Advantage:

1- Less iteration required
2- It is fast to compress smaller collection
3- Decompression speed is very fast.

Disadvantage:

1- It is slower to compress large collection
2- It is need more memory during the compression

5. RESULT AND COMPARISON

In this section we focus our attention to compare the performance of Dictionary based algorithm (like LZ77, LZ78 & LZW), Ray and COMRAD algorithm. Research works done to evaluate the efficiency of any compression algorithm are carried out having three important parameters.

Table1: comparison of compression size and compression ratio of an algorithm are achieved.

DNA Sequence		tatsgs.txt	atef1a23.txt	atrdnai.txt	chmpxx.txt	humdystrop.txt	humghcsa.txt
Input Size		9647	6022	5287	15180	38770	66495
LZ77	Compression Size(Byte)	3971	2579	2209	5793	15101	25300
	Compression Ratio	2.428981	2.334222	2.392984	2.620404	2.567358	2.628261
LZ78	Compression Size(Byte)	3190	2148	1876	4473	10755	17188
	Compression Ratio	3.023654	2.830354	2.817479	3.393696	3.604835	3.868575
LZW	Compression Size(Byte)	3190	2148	1876	4473	10755	17188
	Compression Ratio	3.023654	2.830354	2.817479	3.393696	3.604835	3.868575
RAY	Compression Size(Byte)	1753	869	799	3175	7873	5922
	Compression Ratio	5.503137	6.929804	6.617621	4.781102	4.924425	11.22847
COMRAD	Compression Size(Byte)	3594	1695	1563	6110	16240	29356
	Compression Ratio	2.685691	3.552802	3.382597	2.484452	2.387021	2.265125

One is the compression ratio and second is the compression sized achieved and the other is the time used by the encoding and decoding algorithms.

The compression sized and compression ratio achieved for the LZ77, LZ78 and LZW is presented in table1. The compression ratio of LZ77 fall in the range of 2.334222 to 2.628261.

Compression ratio achieved for the LZ78 fall in the range of 2.817479 to 3.868575. so compression ratio achieved by this algorithm is better than the LZ77.

Compression ratio achieved for the LZW fall in the range of 2.817479 to 3.868575. the compression ratio is achieved by this algorithm is better than the LZ77 & LZ78. After comprehensive analysis, LZW is better than LZ77 & LZ78 in case of compression ratio.

LZW requires less memory space to store compressed data than LZ77 and LZ78.

The compression time and decompression time achieved for the LZ77, LZ78 and LZW are presented in Table2.

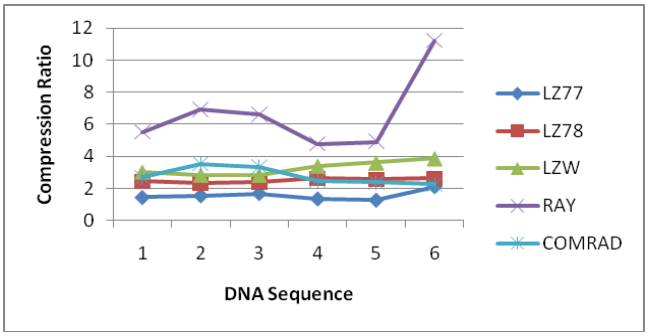

Figure2.(a): Line chart Shows the comparison of compression ratio of above algorithm in table1

After comprehensive analysis compression time of the LZ77, LZ78 and LZW are similar but LZW has better decompression time than LZ77 and LZ78.

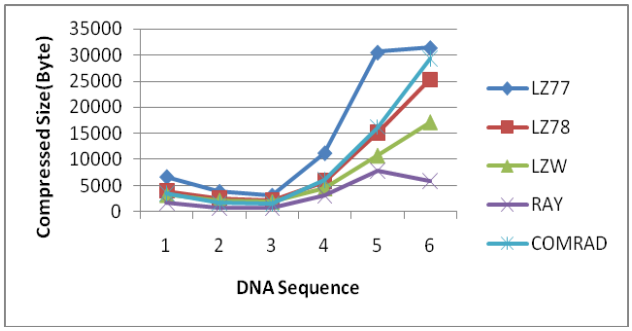

Figure2.(b): Line Chart shows the comparison of Compressed size of algorithms in table1

The compression sized and compression ratio achieved for the RAY and COMRAD is presented in table1. Line chart shows compression algorithm (LZ77,LZ78 & LZW), RAY and COMRAD in figure2.(a) and Compression size in figure2.(b). decompression time achieved for the RAY and COMRAD are presented in Table2. COMRAD is very slow to compress the data but decompression is better than RAY algorithm. Finally, we analyzed that COMRAD algorithm is very slow to compress the data but it is very fast during the decompression than LZ77, LZ78, LZW and RAY.

Line chart shows the comparison of compression and of Dictionary based compression algorithm (LZ77,LZ78 & LZW), RAY and COMRAD in figure3.(a) and decompression time in figure3.(b)

Table2: comparison of compression Time and Decompressed Time of an algorithms are achieved

DNA Sequence		**tatsgs.txt**	**atef1a23.txt**	**atrdnai.txt**	**chmpxx.txt**	**humdystrop.txt**	**humghcsa.txt**
Input Size		9647	6022	5287	15180	38770	66495
LZ77	Compression Time(milis)	37767	20467	17566	64482	408439	1191156
	Decompression Time(milis)	312	125	78	624	2543	3838
LZ78	Compression Time(milis)	37003	14773	14211	82118	520198	1467105
	Decompression Time(milis)	265	124	125	468	2060	4868
LZW	Compression Time(milis)	29749	17535	15163	74365	544987	1492174
	Decompression Time(milis)	29765	13775	8050	82384	540213	1576992
RAY	Compression Time(milis)	7254	5319	9032	6131	545081	1464561
	Decompression Time(milis)	140	78	31	243	453	687
COMRAD	Compression Time(milis)	50684	68103	33571	308163	7122074	11525853
	Decompression Time(milis)	140	94	109	172	514	527

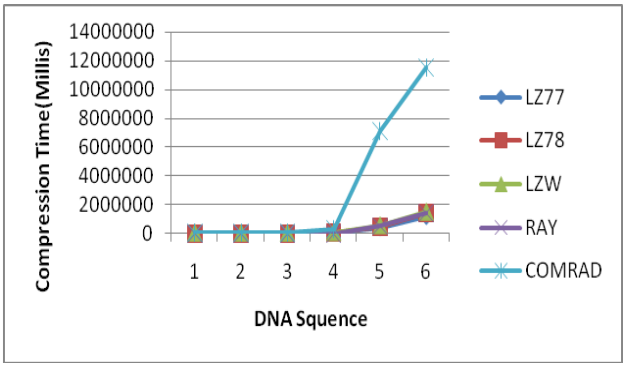

Figure3.(a): Line chart shows the comparison of compression time of algorithms in table2.

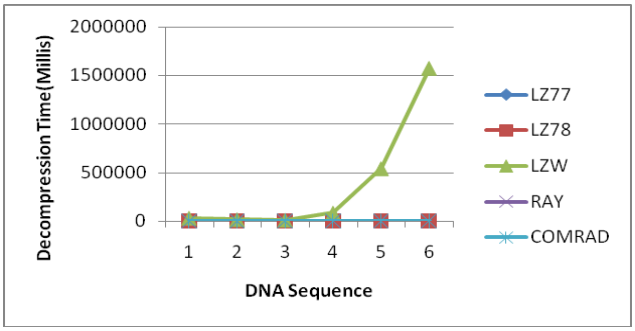

Figure3(b): Line chart shows the comparison of compression time of algorithms in table2.

6. CONCLUSION

We have taken up the dictionary based compression algorithm(LZ77,LZ78 and LZW), RAY and COMRAD algorithm for our study to examine the performance in compression. In this, we analyzed that LZW algorithm is better than the LZ77 and LZ78 in the compression ratio. It also require less memory to store the compress data. but it take too much time during the decompression data. other algorithm is COMRAD which is very slow during the compression but this algorithm take fewer during decompression than the other algorithm like LZ77, LZ78, LZW and RAY. Its compression ratio is good and required less memory to store the compressed data.

7. REFERENCE

[1] D. Wheeler et al., "The Complete Genome of an Individual by Massively Parallel DNA Sequencing," Nature, vol. 452, no. 7189, pp. 872-876, 2008.

[2] D. Bentley et al., "Accurate Whole Human Genome Sequencing Using Reversible Terminator Chemistry," Nature, vol. 456, no. 7218, pp. 53-59, 2008.

[3] J. Wang et al., "The Diploid Genome Sequence of an Asian Individual," Nature, vol. 456, no. 7218, pp. 60-65, 2008.

[4] S. Schuster et al., "Complete Khoisan and Bantu Genomes from Southern Africa," Nature, vol. 463, no. 7283, pp. 943-947, 2010.

[5] L. Stein. The case for cloud computing in genome informatics. Genome Biology, 11(5):207, 2010.

[6] B. Behzadi and F. L. Fessant. DNA compression challenge revisited: A dynamic programming approach. In Proc. 16th Annual Symposium on Combinatorial Pattern Matching (CPM'05), pages 190{200, 2005.

[7] M. D. Cao, T. Dix, L. Allison, and C. Mears. A simple statistical algorithm for biological sequence compression. In Proc. Data Compression Conference (DCC'07), pages 43{52, 2007.

[8] X. Chen, S. Kwong, and M. Li. A compression algorithm for DNA sequences and its applications in genome comparison. In Proc. 4th Conference on Research in Computational Molecular Biology (RECOMB'00), pages 107-117, 2000.

[9] X. Chen, M. Li, B. Ma, and J. Tromp. DNACompress: fast and effective DNA sequence compression. Bioinformatics, 18(12):1696-1698, 2002.

[10] S. Grumbach and F. Tahi. Compression of DNA sequences. In Proc. Data Compression Conference (DCC'93), pages 340-350, 1993.

[11] S. Grumbach and F. Tahi. A new challenge for compression algorithms: Genetic sequences. Information Processing & Management, 30(6):875-886, 1994.

[12] E. Rivals, J. Delahaye, M. Dauchet, and O. Delgrange. A guaranteed compression scheme for repetitive DNA sequences. In Proc. Data Compression Conference (DCC'96), page 453, 1996.

[13] S. Kuruppu, B. Beresford-Smith, T. Conway, and J. Zobel. Iterative dictionary construction for compression of large dna datasets. IEEE/ACM Transactions on Computational Biology and Bioinformatics, 2011. To appear.

[14] S. Kuruppu, S. J. Puglisi, and J. Zobel. Relative Lempel-Ziv compression of genomes for large-scale storage and retrieval. In Proc. 17th Symposium on String Processing and Information Retrieval (SPIRE'10), pages 201-206, 2010.

[15] V. M akinen, G. Navarro, J. Sir en, and N. V alim aki. Storage and retrieval of highly repetitive sequence collections. Journal of Computational Biology,17(3):281-308, 2010.

[16] A. Cannane and H. Williams, "General-Purpose Compression for Efficient Retrieval," J. Am. Soc. for Information Science and Technology, vol. 52, no. 5, pp. 430-437, 2001.

[17] S. Kuruppu, B. Beresford-Smith, T. Conway, and J. Zobel. Iterative dictionary construction for compression of large dna datasets. IEEE/ACM Transactions on Computational Biology and Bioinformatics, 2011. To appear.

[18] MarkNelson and Jean-loup Gailly" The Data Compression Book" http://staff.uob.edu.bh/files/781231507_files/The-Data-Compression-Book-2nd-edition.pdf

[19] Reference Sequence Construction for Relative Compression of Genomes Shanika Kuruppuy Simon J. Puglisiz Justin Zobely arXiv:1106.3791v1 [q-bio.QM] 20 Jun 2011

[20] L. Felician and A. Gentili, A nearly optimal Huffman technique in the microcomputer environment, *Inf. Sys. 12*, 4 (1987), 371.

[21] Mark Nelson and Jean-loup Gailly, The Data Compression Book, http://read.pudn.com/downloads153/ebook/675728/The_Data_Compression_Book_By_Mark_Nelson.pdf

[22] Mamta Sharma," Compression Using Huffman Coding", IJCSNS International Journal of Computer Science and Network Security, VOL.10 No.5, May 2010

[23] Ziad M. Alasmer, Bilal M. Zahran, Belal A. Ayyoub, Monther A. Kanan," A Comparison between English and Arabic Text Compression", Contemporary Engineering Sciences, Vol. 6, 2013, no. 3, 111 – 119 HIKARI Ltd, www.m-hikari.com