Higher Compression from Burrows-Wheeler Transform for DNA Sequence

Rexline S. J.
Department of Computer Science
Loyola College
Chennai, India

Aju Richard Gerard
Department of Electronics and Communication Engineering
SSN college of Engineering
Chennai, India

Trujilla Lobo F.
Department of Computer Science
Loyola College
Chennai, India

ABSTRACT
Large amount of space is required to store biological sequences in DNA database like GenBank sequence database. The data storage for biological sequences has become very essential in today's current situation. Standard compression algorithms are not competent enough to compress biological sequences. In recent times, special algorithms have been introduced specifically for the purpose of compressing the biological sequences like DNA and protein sequences. In this paper, the Burrows-Wheeler Transform (BWT) based approaches are explored to compress the biological sequences. In comparison with the existing general purpose compression algorithms, the proposed BWT based method compresses these types of sequences better and at the same time the cost of Burrows-Wheeler Transform is almost insignificant.

General Terms
Algorithms

Keywords
DNA sequence compression, Burrows-Wheeler Transform, BWT and genome.

1. INTRODUCTION
Bioinformatics deals with algorithms, databases and information systems in the field of biology and medicine. Modern Bioinformatics science produces enormous amounts of genomic sequences, such as nucleotide and amino acid sequences. Rapid growths of molecular research technologies and developments in information technologies have produced a significant amount of data associated with molecular biology. The human genome contains billions of deoxyribonucleic acid (DNA) base pairs. Downloading and maintaining the DNA sequences are much cost consuming factors due to the increasing amount of genome sequences. Hence, decreasing the space required to store the DNA sequences has become a very important new challenge faced by researchers. In addition to the need for efficient and effective algorithms for the analysis, annotation, interpretation and visualization of the data, there is also the need of effective techniques for the organization to store and transmit this mass amount of biological sequence data. In this paper, the problem of compressing biological sequences is considered and it plays a vital role in saving the storage space and transmission time required for DNA sequences and protein sequences. From a computational viewpoint, a biological sequence can be viewed mainly as a one-dimensional sequence of symbols, for instance with an alphabet of 4 symbols for DNA and 20 symbols for proteins.

For this effective improvement, the BWT (Burrows–Wheeler Transform), MTF (Move-To-Front encoding), RLE (Run Length Encoding) and Arithmetic Coding are efficiently introduced in the following sections.

2. METHODS

2.1 Burrows–Wheeler Transform
The Burrows–Wheeler Transform (BWT) was developed by Michael Burrows and David J. Wheeler in 1994[3]. The Burrows - Wheeler Transform (BWT) works on a block of data, where the input data is read block by block and each block is encoded separately as a single unit. BWT takes a block of data and rearranges it lexicographically using a sorting algorithm. It also passed through a Move-To-Front (MTF) stage, then the Run Length encoder Stage and finally applies Huffman coding or Arithmetic Coding. The transformation is reversible that the original ordering of the data can be restored with no loss of information. The method is also referred to as block sorting algorithm. Bzip2 compressing algorithm compresses files using the Burrows-Wheeler block sorting text compression algorithm, and Huffman coding. Bzip2 compresses large files in blocks.

The Burrows–Wheeler Transform (BWT) is not actually a compression scheme but a reversible transform that transforms the data into an intermediate format that is generally more compressible than the original data [2]. Attempted improvements on the original BWT algorithm have shown very limited success. The algorithm works by transforming a string S of N characters by forming the N rotations (cyclic shifts) of S, sorting them lexicographically, and extracting the last character of each of the rotations. A string L is then formed from these extracted characters, where the i^{th} character of L is the last character of the i^{th} sorted rotation. The algorithm also computes the index I of the original string S in the sorted list of rotations. With only L and I, there is an efficient algorithm to compute the original string S when undoing the transformation for decompression [4]. This BWT creates the transformed data even larger in size than its original size, but the transformed data is in need of less storage space [8]. Burrows and Wheeler explain that much of the time, the algorithm is performing sorts which may be another area where using a parallel sorting algorithm may increase the speed of BWT. They also explain that to achieve good compression, a block size of sufficient value must be chosen, at least 2 kilobytes [7]. Increasing the block size also increases the effectiveness of the algorithm at least up to the size of several megabytes.

For example, BWT is applied on the below DNA sequence: **ATGGTGCACCTGACT**

1. **Cyclic shifts of the above DNA sequence is given as follows**

ATGGTGCACCTGACT

TATGGTGCACCTGAC

CTATGGTGCACCTGA

ACTATGGTGCACCTG

GACTATGGTGCACCT

TGACTATGGTGCACC

CTGACTATGGTGCAC

CCTGACTATGGTGCA

ACCTGACTATGGTGC

2. After sorting them lexicographically, the above DNA sequence will be as follows.

ACCTGACTATGGTGC

ACTATGGTGCACCTG

ATGGTGCACCTGACT

CACCTGACTATGGTG

CCTGACTATGGTGCA

CTATGGTGCACCTGA

CTGACTATGGTGCAC

GACTATGGTGCACCT

GCACCTGACTATGGT

3. BWT sequence is CGTGAACTTTGCCGA, and I=3.

2.2 Move-to-Front Encoding

Move-to-Front algorithm [1] converts the data into a sequence of integers, with the expectation that the values of integers are small and could be effectively transformed using a statistical coding algorithm. The MTF encoder retains a list of symbols, called MTF list, which is initialized with all the symbols that occur in the data to be compressed. Then, for each symbol from the data, the encoder provides its position on MTF list in the form of an integer and updates the MTF list. A currently encoded symbol is moved from the current position in the MTF list to the beginning of the list .The most important property of this technique is that recently used symbols are near to the beginning of the list. Equal symbols will frequently appear close to each other in the data and therefore these symbols will be converted to small integers. In general, small integers appear more frequent so that they are encoded in fewer bits than larger integers using a statistical coding like the Huffman or the arithmetic coding. Move-to-front encoding algorithm transforms a DNA sequence S into a sequence of numbers, provided that the alphabet is known before hand.

Here is the algorithm for Move-to-front encoding method.

1. Initialize string E to contain each letter in the alphabet once.
2. Read the letters of *S* one at a time. For a character A that was just read, write down the index of A in E and move A in E to the front of E. So E becomes different permutations of the letters in the alphabet as S is processed, and obtained a sequence of indices.

Here is an example of the encoding of the string **"GGGTTTAATTCCC"** using the alphabet {**A,C,G,T**}. Table 1 shows the Move-to-Front encoding table.

Table 1. Move-to-Front Encoding Table

Letter of S	E (before)	Index	E (after)
G	TACG	3	GTAC
G	GTAC	0	GTAC
G	GTAC	0	GTAC
T	GTAC	1	TGAC
T	TGAC	0	TGAC
T	TGAC	0	TGAC
A	TGAC	3	ATGC
A	TGAC	0	ATGC
T	ATGC	1	TAGC
T	ATGC	0	TAGC
C	TAGC	3	CTAG
C	CTAG	0	CTAG
C	CTAG	0	CTAG

Hence, the encoding is **3001003010300**. One way to get good compression is to first do move-to-front encoding on the transformed string, then, do arithmetic or Huffman encoding in combination with run-length encoding for zero strings.

2.3 Run Length Encoding

Run Length Encoding (RLE) compression technique [6] is used when a given file contains too many redundant data or long run of similar characters. The repeated string or characters present in the input file or message is called a run which is encoded into two bytes. The first byte represents the value of the character in the run and the second byte contains the number of times given character appears in the run.

This Run Length Encoding algorithm consists of replacing large sequences of repeating data with only one item of this data followed by a counter showing how many times this item is repeated.

Algorithm for the general Run Length Encoding (RLE) is as follows:

```
Loop: count = 0
    REPEAT
        get next symbol
        count = count + 1
     UNTIL (symbol unequal to next one)
    output symbol
    IF count > 1
       output count
    GOTO Loop
```

Let's see a **sample DNA sequence** as follows:

AAAAAAAAAAGGGACCCCTTTTTC

This **DNA sequence** length is 24 and there are lots of repetitions. Using the Run Length Encoding (RLE) algorithm, repetitive runs can be replaced with shorter symbol followed by a counter.

The reduced DNA sequence is

A10G3A1C4T5C1

The length of this DNA sequence is 13, which is approximately 70% of the initial length.

2.4 Arithmetic Coding

Arithmetic coding has been efficiently developed in the place of Huffman coding. In Huffman coding, every input symbol has been replaced by a specific code whereas in arithmetic coding, a stream of input symbols has been replaced by a single floating-point output number. Depending on the length of the message, more bits are needed in the output number. Arithmetic coding is particularly constructive when dealing with symbols with high probabilities. It is also useful that output from an arithmetic coding process is a single number varying from 0 to 1 , not including 1. This single number can be uniquely decoded to create the exact stream of symbols that has gone into its construction. In order to construct the output number, the symbols being encoded should have a set of probabilities assigned to them. Once the character probabilities are known, the individual symbols need to be assigned a range along a "probability line", which is nominally 0 to 1. It does not matter what characters are assigned to which segment of the range, as long as it is done in the same manner by both the encoder and the decoder. Each symbol is allocated the value in the range of 0-1 that matches the probability appearance of the symbol. It is understood that the symbol holds the values of all except the higher number. Consequently, the last symbol has the range of values between 0.9-0.9999 and not 1[12]. In order to properly decode the first character, the final coded message has to be a number greater than or equal to the range of the first character of the actual stream. The range, that this could fall in, should be maintained to encode this number. So, after the first character is encoded, the algorithm must continue with the next character in actual stream. After the first character is encoded, the low and the high of the first character now bound to the range for the output number. The remaining of the encoding procedure is that each new symbol to be encoded will further control the possible range of the values. If it was the first number in the text, then low and high ranges of values are set directly to those values.

Algorithm for the Arithmetic Coding is as follows:

begin
count source symbols
interval *Ivalue*: = **new** interval 0..1
divide *Ivalue* according to rate of source symbols.
readSymbol(*X*)
while (*X*!=EOF) **do**
begin
new *Ivalue* := subinterval *Ivalue* matching *X*
divide *Ivalue* according to rate of source symbols.
readSymbol(*X*)
end
output(best number from *Ivalue*)
end

2.5 Huffman Coding

Huffman coding, introduced by David Huffman in 1952 , compresses texts by assigning shorter codes to more frequently used symbols and longer codes to less frequently used ones. This coding is an entropy encoding algorithm used for lossless data compression. A specific method in this coding is used to choose the representation for each symbol, resulting in a prefix-free code that expresses the most common characters using shorter strings of bits. To create Huffman code, the symbols present in the source file are sorted in increasing order by frequency. Merging the two least frequency used symbols into a new symbol can be constructed and its frequency is the sum of the frequencies of its two child symbols. In the way of replacement, a smaller set of symbols is obtained and this operation n-1 times can be repeated until all symbols have been merged. A node is created through every merging operation in a binary tree. The left or right choices on the path from root-to-leaf define the bit of the binary code word for each symbol. Though the usage of Huffman code is wide and frequent, these codes have three major disadvantages. Firstly, two passes are required over the document. Secondly, the coding table is stored along the document in order to reconstruct it.

The approach of Huffman's algorithm is illustrated in the following plain text with 24 symbols.

y z y x y x w z x y x z x w x w x y x y x x x y

Figure 1 illustrates an example of building the Huffman tree. The algorithm starts with a list of nodes 'x', 'y', 'z', and 'w', with frequencies 11, 7, 3, and 3 respectively. The frequencies of each symbol are calculated and a prefix-free tree labeled with 0 (left child) or 1 (right child) shown in figure1 is created for these symbols.

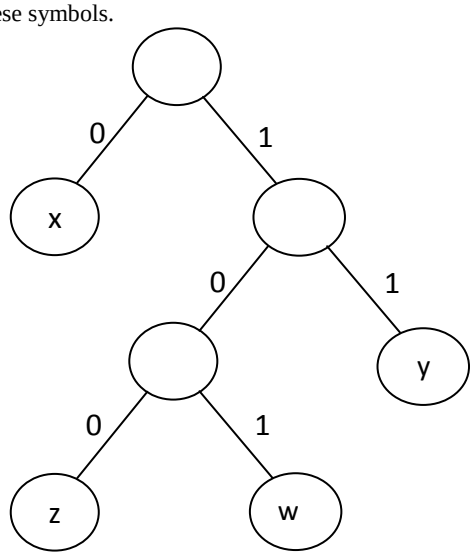

Fig 1: Prefix-Free Tree

From the Prefix-Free Tree, the algorithm creates a prefix code for each symbol by traversing the binary tree from the root to the end node, which corresponds to the symbol. It gives 0 for a left branch and 1 for a right branch. The Huffman algorithm, using the Huffman tree from Figure 1, assigns codes '0', '11', '100', and '101' to symbols 'x', 'y', 'z', and 'w' respectively. The generated Prefix code table is given below in Table 2:

Table 2. Prefix Code Table

Symbol	Frequency	Codeword
x	11	0
y	7	11
z	3	100
w	3	101

The plaintext is encoded with 43 bits as follows:

11100110110101100011010001010101101100011

In standard text storage, 8 bits per symbol are given and hence, there are 192 bits required for the above plaintext.

With the code table of Table 2, only 43 bits are achieved. Moreover, the Huffman tree with the Huffman codes for symbols must be stored together with the compressed output. This information is needed for the decoder and it is usually placed in the header of a compressed file.

3. PERFORMANCE ANALYSIS

The compression algorithm consists of two components, an *encoding* algorithm that takes input data and generates a "compressed" representation and a *decoding* algorithm that reconstructs the original data from the compressed representation. To experiment the BWT based algorithms, a standard set of DNA and protein sequences are compressed with these algorithms, and results are compared with the other standard general purpose compressors.

The following 7 benchmark standard data set of DNA sequences are used in this paper for the purpose of analysis:

- One chloroplast genomes (CHMPXX)
- Four human genes (HUMDYSTROP, HUMHDABCD, HUMHBB and HUMHPRTB)
- Two virus genomes (HEHCMVCG and VACCG).

The following 4 standard data set of protein sequences are used for the purpose of analysis:

- Haemophilus Influenzae(HI)
- Human(HS)
- Methanococcus Janaschii(MJ)
- Saccharomyces Cerevisciae(SC)

The above data sets are used by many researchers who work in DNA sequence compression [5]. The DNA Sequences can be downloaded from the GENBANK database: The DNA sequences are made available in FASTA file format in DNA databases which can also retrieved by any text processor. The chemical composition of the DNA is the same for all living organisms [10]. The DNA of every living organism contains four basic nucleotide bases: **adenine, cytosine, guanine,** and **thymine,** usually abbreviated using the symbols **A, C, G** and **T** respectively [11]. A typical structure of DNA sequence file is that in the appearance of the nucleotide bases having no spaces and an end of line symbol, with a restriction that a nucleotide may appear only nine consecutive times. Efficiency of the proposed method is measured in terms of bits per base (BPB)[9]. The main concern of the compression algorithm for these sequences is with the compaction ability but not consider with the time taken for the compression.

Table 3. Compression ratio of the general purpose compression algorithm for DNA Sequences

Sequence Name	File Size (Bytes)	WinRAR (BPB)	Gzip (BPB)	Bzip (BPB)
Chmpxx	121024	2.25	2.28	2.12
Humhbb	73308	2.22	2.25	2.15
Humghcsc	66495	2.32	---	---
Humhdabcd	58864	2.19	2.24	2.07
Humhprtb	56737	2.23	2.27	2.09
Hehcmvcg	229354	2.32	2.33	2.17
Vaccg	191737	2.23	2.25	2.09
Average		2.25	2.27	2.16

Proteins are sequences consumed from amino acids [13]. There are 20 kinds of amino acids except for some abnormal ones; hence the size of alphabet of proteins is 20. It is recognized that the compression of proteins is also very difficult .Since the size of alphabet is 20, consequently the storage space required for proteins is equal to or less than log2 (20) = 4.322 per symbol. The compression ratios by the widely used compression algorithms compress or gzip are more than log2 (20) = 4.322 bits per symbol. The unit of compression ratio for protein symbol is bit per symbol. Compression results are best when an algorithm can compress a sequence less than log2 (20) =4.322 bits per symbol. The sequence file based on DNA sequence and protein sequence is passed through the BWT transform and then piped through a move-to-front stage, then a run length encoder stage, and finally an entropy encoder, normally arithmetic coding or Huffman Coding. In this section, to compare the performance of the proposed idea, the BWT transform, move-to-front, run length encoder and arithmetic coding is used.

Table 4. Compression ratios of the BWT based algorithm for DNA Sequences

Seq Name	Compressed File size					
	BWT,MTF, RLE,ARI		BWT,MTF, ARI		BWT,ARI	
	Bytes	BPB	Bytes	BPB	Bytes	BPB
Chmpxx	33133	2.19	30323	2.00	28622	1.89
Humhbb	20255	2.21	18505	2.02	18371	2.00
Humghcsc	15582	1.87	13314	1.60	16911	2.03
Humhdabcd	15910	2.16	14747	2.00	15030	2.04
Humhprtb	15507	2.19	14274	2.01	14320	2.02
Hehcmvcg	64269	2.24	58284	2.03	57801	2.02
Vaccg	52172	2.18	48345	2.07	46816	1.95
Average		2.15		1.96		1.99

Table 5. Compression ratio of the general purpose compression algorithm for Protein Sequences

Seq Name	Seq. Length (Bytes)	Compress (Bps)	Bzip2 (Bps)	Gzip9 (Bps)	PPM +
HI	509519	4.7702	4.324	4.6712	4.862
HS	3295751	4.7177	4.256	4.6054	4.641
MJ	448779	4.6459	4.269	4.5879	4.711
SC	2900352	4.7761	4.300	4.6397	4.686
Average (BPS)		4.727475	4.28725	4.62605	4.725

Table 6. Compression ratios of the BWT based algorithm for Protein Sequences

Sequence Name	BWT, MTF, RLE, ARI (BPS)	BWT, MTF, ARI (BPS)	BWT, ARI (BPS)
(HI)	4.4721	4.3039	4.1843
(HS)	4.3984	4.2438	4.1581
(MJ)	4.4253	4.2568	4.0959
(SC)	4.4603	4.3009	4.1897
Average (BPS)	4.439025	4.27635	4.157

The actual command line to perform the compression sequence will look like this:

BWT < input-file | MTF | RLE | ARI > output-file

The decompression is just the reverse process and command line to perform the decompression sequence look like this

UNARI input-file | UNRLE | UNMTF |UNBWT > output-file

According to the above said procedure, the average compression ratio in terms of BPB is 2.15 for the standard DNA files. Since the method compresses the DNA sequences very badly takes more than 2 BPB. The average compression ratio in terms of BPS is 4.439025 for the protein files. Since the method compresses the sequences very badly takes more than 2 BPB for DNA and more than log2 (20) = 4.322 per symbol for protein sequences.

Therefore the procedure given below is tried out to improve the compression ratio of the DNA and protein files. The sequence file is passed through the BWT transform and then piped through a move-to-front stage, and finally an entropy encoder, normally arithmetic coding.

The actual command line to perform the compression sequence will look like this:

BWT < input-file | MTF | ARI > output-file

The decompression is just the reverse process and command line to perform the decompression sequence look like this

UNARI input-file | UNMTF |UNBWT > output-file

The procedure without run length encoding gives slightly better compression improvement when compared with BWT then piped through a move-to-front stage, then a run length encoder stage, and finally an arithmetic coding. Again to get better improvement of DNA files, the sequence file is passed through the BWT transform and then applied an entropy encoder called arithmetic coding. This procedure gives the average compression ratio of 1.99 BPB for DNA files and 4.157 BPS for protein sequences.

The comparative compression ratio of the existing general purpose compression algorithm like WinRAR, GZIP and BZIP of the DNA test corpus are shown in Table 3. The detailed compression ratios in terms of BPB for the Burrows Wheeler Transform based algorithm of the DNA test corpus are shown in Table 4. The test results of protein sequences are shown in Table 5 and 6. It shows that the combination of Borrow Wheeler Transformation, move-to-front and arithmetic coding works better when compared with the existing general purpose compression algorithms.

4. CONCLUSIONS

After doing a detail study on various standard general purpose compression algorithms it is seen that a great improvement on DNA and protein sequences compression ratio after preprocessing with Burrows Wheeler Transformation. With these results it is understood that the combination of Burrows Wheeler Transform and Arithmetic Encoding gives us a best compression method for DNA and protein compression. The simplicity and flexibility of BWT based Compress algorithms could make it an invaluable tool for DNA compression in clinical research. Since the Standard general purpose compression algorithms are unsuccessful to get a good compression ratio, it is better to use pattern recognition method to create new compression algorithm for DNA and protein sequences to improve the compression ratio.

5. REFERENCES

[1] Arnavut. Z, "Move-to-Front and Inversion Coding", Proceedings of Data Compression Conference, IEEE Computer Society, Snowbird, Utah, pp. 193- 202, March 2000.

[2] M.P.Bhuyan, V.Deka, S.Bordoloi, "Burrows Wheeler based data compression and secure transmission", IJRET: International Journal of Research in Engineering and Technology, Volume: 02 Special Issue: 02 | Dec-2013.

[3] M. Burrows, and D.J. Wheeler, "A Block-sorting Lossless Data Compression Algorithm," Digital Systems Research Center Research Report 124, 1994.

[4] Chun Li , Huan Liu, Junhong Liu, Yuping Qin, Zhifu Wangb, "A Burrows-Wheeler Transform based method for DNA sequence comparison, Computational Biology and Bioinformatics, 2(3): 33-37, 2014.

[5] Jolanta Kawulok, "Approximate String Matching for Searching DNA Sequences", International Journal of Bioscience, Biochemistry and Bioinformatics, Vol. 3, No. 2, March 2013.

[6] Jouni Sir´en, Niko V¨alim¨aki, Veli M¨akinen, and Gonzalo Navarro, "Run-Length Compressed Indexes Are Superior for Highly Repetitive Sequence Collections", String Processing and Information Retrieval , pp 164-175, 2008.

[7] Juha K¨arkk¨ainen, "Fast BWT in Small Space by Blockwise Suffix Sorting", Preprint submitted to Elsevier Science, 16 March 2007.

[8] Nelson M, "Data Compression with the Burrows-Wheeler Transform", Dr. Dobb's Journal, Sept. 1996.

[9] Rahul Vishwakarma1 and Newsha Amiri, "High Density Data Storage in DNA Using an Efficient Message Encoding Scheme", International Journal of Information Technology Convergence and Services (IJITCS) Vol.2, No.2, April 2012.

[10] RAFAŁ POKRZYWA, "Searching for Unique DNA Sequences with the Burrows-Wheeler Transform", Biocybernetics and Biomedical Engineering, Volume 28, Number 1, pp. 95–104, 2008.

[11] Sebastian Wandelt, Marc Bux, and Ulf Leser," Trends in Genome Compression", Knowledge Management in Bioinformatics, Institute for Computer Science, Humboldt-Universit□at zu Berlin, Germany ,June 4, 2013.

[12] Witten, I.H., R.M. Neal and J.G. Cleary, "Arithmetic coding for data compression", Commun.ACM, 30: pp : 520-540,1987.

[13] Yong Zhang, Amar Mukherjee, Matt Powell and Tim Bell, "DNA Sequence Compression Using the Burrows-Wheeler Transform", Proceedings of the IEEE Computer Society Bioinformatics Conference, 2002.