International Journal of Information Technology and Knowledge Management
July-December 2010, Volume 2, No. 2, pp. 383-386

# DNA COMPRESSION USING HASH BASED DATA STRUCTURE

Ateet Mehta[1] & Bankim Patel[2]

DNA Sequences making up any organism comprise the basic blueprint of that organism so that understanding and analyzing different genes within sequences has become an extremely important task. Biologists are producing huge volumes of DNA sequences every day that makes genome sequence database growing exponentially. The databases such as EMBL, GenBank represent millions of DNA sequences filling many thousands of gigabytes computer storage capacity and the databases are doubled in size every 6-8 months. Hence an efficient algorithm to compress DNA sequence is required. Though there are many text compression algorithms, they are not well suited for the characteristics of DNA sequences. There are algorithms for DNA compression which takes advantage of repetitive nature of DNA fragments within the sequence where as few of the other algorithms are written for the non repeated patterns within DNA sequences. In this paper, we represent an algorithm which is based on hash based data structure to compress DNA sequences. The proposed algorithm performs equally well for both repeated and non-repeated patterns within the DNA sequence.

Keywords: DNA Sequence, Hashing, Data Structure, Compression, Huffman Encoding, Arithmetic Coding

## 1. INTRODUCTION

DNA Sequences making up any organism represent the basic blueprint of that organism so that understanding and analyzing different genes within the DNA sequences has become an extremely important task. Analyzing such different types of sequences help in many areas like customizing the medical treatment and therapy according to the genetic attributes of specific human, discovering new drug solutions etc. [1]. Hence DNA sequences play an important role in medical research, disease diagnosis, and the design and development of new drugs. [2]. Biologists are producing huge volumes of DNA and Protein Sequences every day. Genome sequencing projects are producing vast amounts of biological data for different organisms and storing them in databases. [3]. with almost every new scientific publication in genetics and related sciences, a new sequence is added and the rate at which the data is accumulating is on the rise. Currently there are global databases like NCBI, GenBank, and SWISS PROT storing such biological data. These biological databases are growing exponentially and compressing such data is important and it helps not only in reduction of storage space but also in exchanging data between systems through web services over internet.

There are many text compression algorithms available having quite a good compression ratio. But they have not been proved well for compressing DNA sequences as the algorithm does not incorporate the characteristics of DNA sequences even though DNA sequences can be represented in simple text form. DNA sequences are comprised of just four different bases labelled A, T, C, and G (for adenine, thymine, cytosine, and guanine respectively). T pairs with A, and G pairs with C. Each base can be represented in computer code by a two character binary digit, two bits in other words, A (00), C (01), G (10), and T (11). At first glance, one might imagine that this is the most efficient way to store DNA sequences. Like the binary alphabet {0, 1} used in computers, the four-letter alphabet of DNA {A, T, C, G} can encode messages of arbitrary complexity when encoded into long sequences. Shown in the figure 1 is how the bases pairs with each other. Human genome contains about 3 billion base pairs out of which 3% encodes proteins. There are 25000 genes in human genome which can encode nearly 100000 proteins.

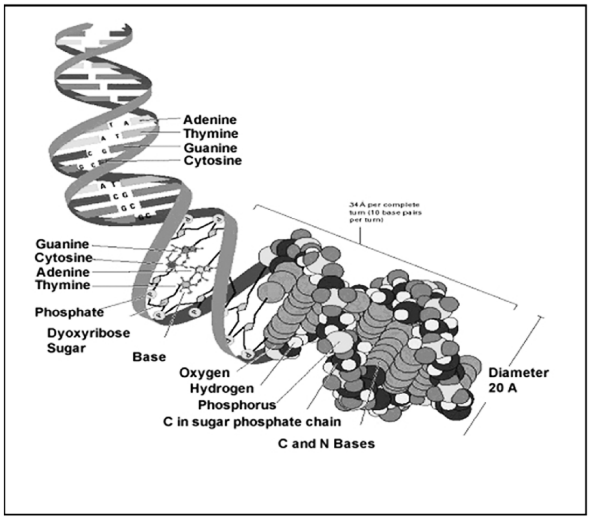


Fig. 1

Shrimad Rajchandra Institute of Management and Computer Application, South Gujarat University, Gujarat, India
Email: ateet.mehta@gmail.com[1], bankim_patel@srimca.edu.in[2]

DNA sequences, however, are not random; they contain repeating sections, palindromes, and other features that could be represented by fewer bits than is required to spell out the complete sequence in binary. [5]. A repeat pattern could be abbreviated to say the binary equivalent of "six times G" for instance, which would be a few bits shorter than explicitly writing "GGGGGG" in binary. Similarly, palindromes could be abbreviated in code relative to their complementary pattern in the DNA sequence.

In computer science and information theory, data compression or source coding is the process of encoding information using fewer bits (or other information-bearing units) than an unencoded representation would use, through use of specific encoding schemes. [4]. Compression is useful because it helps reduce the consumption of expensive resources, such as hard disk space or transmission bandwidth. Compressed data must be decompressed to be used, and this extra processing may be detrimental to some applications. Therefore, the design of data compression schemes therefore involves trade-offs among various factors, including the degree of compression and the computational resources required to compress and uncompress the data. Further some compression algorithms can introduce distortion in data which are known as lossy compression. Lossless compression algorithms usually exploit statistical redundancy in such a way as to represent the sender's data more concisely without error. [4]. Lossless compression is possible because most real-world data has statistical redundancy. For example, in English text, the letter 'e' is much more common than the letter 'z', and the probability that the letter 'q' will be followed by the letter 'z' is very small. Lossless compression exploits the repeats, palindromes and patterns present in the digital data to reduce the overall size.

## 2. RELATED WORK

The compression of DNA sequences is considered as one of the most challenging tasks in the field of data compression. [6]. Standard compression algorithms are not able to compress DNA sequences. [11]. Rapid advancements in research in the field of DNA sequence discovery has led to a vast range of compression algorithms. The number of bits required for storing four bases of any DNA sequence is two. With the constant decrease in prices of memory and communication channel bandwidth, one often doubts the need of such compression algorithms. There are many text compression algorithms implemented in various tools like winzip, winrar, gzip, bzip2 but they cannot compress DNA sequences. It is very well known that one of the main features of DNA sequences is that they contain substrings which are duplicated except for a few random mutations. For this reason, most DNA Compressors are built to searching and encoding these repeats. However, searching for repeated patterns takes a long time and takes more memory. The algorithms designed specifically to compress DNA sequences assuming the frequent occurrences of repeated patterns in DNA sequence may not efficiently compress with non repeated patterns in the DNA sequences where as some of the algorithms work well but built for short sequences. Therefore, the results for these algorithms under the best and worst case vary in a high degree.

There are several approaches for encoding of texts which are Huffman Encoding, Adaptive Huffman Encoding, Arithmetic coding, Arithmetic adaptive coding, Context Tree weighted method etc. Algorithms designed for DNA Compression like GenCompress, BioCompress, DNA Compress, CTW+LZ, Cfact have achieved an approximate compression ratio of 22% use the above mentioned approaches. With BioCompress, at each step, the longest factor beginning at the current position which matches with a factor starting before is chosen. BioCompress-2 uses arithmetic coding of order 2. Cfact performs in two pass in that it looks for longest exact matching repeat and uses a suffix-tree for finding the longest repeat. GenCompress works on approximate repeats in which at each step, it looks for the optimal prefix of the not yet encoded part of the DNA sequence. It uses Hamming distance (v1) and edit distance (v2) for approximate repeats. CTW+LZ is a Combinaison of GenCompress and CTW uses Context Tree Weighting method. DNACompress uses PatternHunter as preprocessing. Found repeats are sorted in decreasing order of size. Greedy approach of GenCompress and DNACompress has not been proved well. DNAPack was made using dynamic programming instead of greedy approach and proved better than above mentioned algorithms. Common components of most of DNA compression algorithms are

- Finding the candidate repeat segments.
- Considering approximate repeats.
- Encoding of the repeat segments.
- Encoding of the non-repeat segments.

We have attempted to take another approach to present a hash based data structure to compress DNA Sequences which is a lossless compression algorithm and will get compression ratio to be 75%. Another important feature of our algorithm is that it compresses long sequences of hundreds of mega byes long.

## 3. PROPOSED WORK

We use hash based data structure to compress DNA sequences. As explained earlier, DNA sequence can consist of four letters {A, C, T, G}. Let the set S represent {A, C, T, G}. If m is the total elements in the set and n is the length of the word that should be constructed using the set s, possible words that can be generated would be mPn where element

cannot be repeated. In case of DNA sequences, the bases can be repeated. So the possible words of the length n with m characters in the set S would be m^n where in any character in the set can be repeated any number of times within the word length n. Considering this, we build the hash keys of word length=4 with no collisions within the hash table. Each hash keys will be represented by single character. We scan the whole DNA sequence and fragment it in small DNA fragments of length=4 and hash this fragment into hash table resulting in one character. The result would be compressed DNA. Therefore, the algorithm initially builds hash table and assigns unique character to each of the hash keys and then scan entire DNA sequence into small DNA fragments, hash it into a single character finally resulted into compressed DNA.

ALGORITHM TO COMPRESS

```
begin
 let h be an empty hash table
 let c be a collection of single byte character codes.
 Let s be the set of {A,C,T,G}.
 Let seq be the DNA sequence, cseq be the compressed DNA sequence.
 Initialize index by 1
 Initialize segment by length(seq)/4
 [step to construct hash table]
 repeat while i<=4 do
   repeat while j<=4 do
        repeat while k<=4 do
             repeat while l <=4 do
 hashkey= substr(s,i,1) || substr(s,j,1) || substr(s,k,1) || substr(s,l,1);
 h[hashkey] :=c[n++];
             end do
 end do
 end do
 end do
 [step to scan dna sequence in fragments and hash it within hash table]
 repeat while index <= segment do
 cseq := cseq || h[substr(seq,index++,4)];
 end do
 append(seq,length(cseq)-mod(cseq,4)) to cseq;
end
```

Decompression is done exactly in the reverse manner to compression. We scan the compressed sequence one character at a time. This character is hashed in the hash table; the key of the hash table for the particular character represents the original 4 character fragment. Repeating this process for the entire compressed sequence and adding these fragments result in an original DNA sequence without any loss of information.

ALGORITHM TO DECOMPRESS

```
Initialize index by 1
Let seq be the DNA sequence, dseq be the compressed DNA sequence, h be the hashtable
repeat while index <=length(dseq) do
   seq := seq || h.search(substr(dseq,index,1));
end do
end
```

Assume seq be the sequence having length=96 as shown below.

ACAA GATG CCAT TGTC CCCC GGCC TCCT
GCTG CTGC TGCT CTCC GGGG

CCAC GGCC ACCG CTGC CCTG CCCG GTGG
CGTA ATGC TCAC GCAA GTTT

Therefore, logically it needs 96 bytes to store this sequence in a text file. Normal text compression algorithm like winzip supplied with Windows or zip, tar supplied with UNIX cannot actually compresses this sequence. With the proposed algorithm, sequence will be compressed by 75% irrespective of the number of repeated or non-repeated patterns within the sequence. We tested this algorithm on a sequence database created in Oracle 10g Release 2 running on Windows XP. The algorithm is implemented as a stored procedure within Oracle Database. Machine has Pentium 4 processor with 1GB RAM, 2.80 GHz.

Following table shows execution timings with different sequence sizes.

Table 1

| Sequence Size | Time in seconds |
|---|---|
| 2.9 KB | < 1 second |
| 24 KB | < 1 second |
| 280 KB | < 1 second |
| 2.8 MB | 1 second |
| 28 MB | 4 seconds |
| 280 MB | 62 seconds |

While we execute this database stored procedure on any of the DNA sequences having size in mega bytes, the whole processing is done at a database layer. No data is pulled out of the database for processing and bringing the results into the database. Data fetching, compression of the sequences and putting the compressed data all is done within the database.

The algorithm is not built keeping the repetitive or non-repetitive patterns in mind. It performs equally well irrespective of the patterns within DNA Sequence. As shown in the above table, it has been tested for sequence from few

kilo bytes to few hundreds of mega bytes. The algorithm assumes the data within the sequence does not contain any junk or invalid characters. In case if it is not sure, it is wise to clean the sequence first for the invalid characters before running this algorithm.

## 4. CONCLUSION AND FUTURE WORK

Standard text compression algorithm cannot compress DNA sequences and there is need to develop compression algorithms specifically for DNA sequences considering the characteristics of DNA sequence being the collection of 4 bases {A,C,T,G} with many repeated and non repeated patterns within the sequence. Though they are algorithms designed for DNA sequences, some of them work well for short sequences where as some of them work well where in the sequences contains many repeated patterns. We attempted to design compression algorithm for DNA sequences which is lossless compression algorithm and will compress to 75% irrespective of the repeating or non repeating patterns within the sequence. The algorithm does not depend on the size of the sequence and capable to compress even larger sequences having sizing in hundreds of MBs. The algorithm assumes the input sequence to have valid characters. The algorithm does not account for the repetitiveness of the fragments. So if the fragment within DNA sequences appears to be AAAA…200 times, it would be compressed to AAAA..50 times. (75% compression). Algorithm can be extended to check if it finds such fragments in which same base is being repeated for multiple times, it can be compressed as base{n} where n is the frequency of repetition.

## REFERENCES

[1] Cynthia Gibas & Per Jambeck-Developing Bioinformatics Computer Skills.

[2] S.C.Rastogi, P.Rastogi- Bioinformatics Methods and Applications.

[3] David W. Mount- Bioinformatics: Sequence and Genome Analysis.

[4] http://en.wikipedia.org/wiki/Data_Compression.

[5] L. Allison, L. Stern, T. Edgoose, and T. I. Dix. Sequence Complexity for Biological Sequence Analysis. Computers and Chemistry, 24(1):43–55, 2000.

[6] A. Apostolico and S. Lonardi. Compression of Biological Sequences by Greedy Off-line Textual Substitution. In J. A. Storer and M. Cohn, Editors, Proceedings Data Compression Conference, Pages 143–152, Snowbird, UT, 2000. IEEE Computer Society Press.

[7] J. Bentley and D. McIlroy. "Data Compression with Long Repeated Strings". Information Sciences, 135:1–11, 2001.

[8] X. Chen, S. Kwong, and M. Li. "A Compression Algorithm for DNA Sequences and its Applications in Genome Comparison". In Proceedings of the 4th Annual International Conference on Computational Molecular Biology (RECOMB-00). ACM Press, 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] National Center for Biotechnology Information. http://www.ncbi.nih.gov.

[11] S. Grumbach and F. Tahi. A New Challenge for Compression Algorithms: Genetic Sequences Inf. Proc. and Management, 30(6):875–886, 1994.

[12] R. Karp and M. Rabin. "An Efficient Randomized Pattern-matching Algorithms". IBM Journal of Research and Development, 31(2) : 249–260, 1987.