B.Sc. in Computer Science and Engineering Thesis

# Archiving Medical Records in DNA Sequence


Submitted by

Md.Jakaria
1305024

Kowshika Sarker
1305034

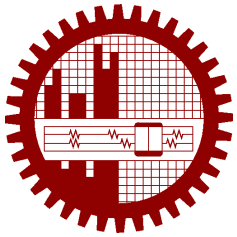
Mostofa Rafid Uddin
1305039

Md.Mohaiminul Islam
1305077

Trisha Das
1305098

Supervised by

Md. Shamsuzzoha Bayzid



**Department of Computer Science and Engineering**
**Bangladesh University of Engineering and Technology**

Dhaka, Bangladesh



April 2018

# CANDIDATES' DECLARATION

This is to certify that the work presented in this thesis, titled, "Archiving Medical Records in DNA Sequence", is the outcome of the investigation and research carried out by us under the supervision of Md. Shamsuzzoha Bayzid.

It is also declared that neither this thesis nor any part thereof has been submitted anywhere else for the award of any degree, diploma or other qualifications.

---

Md.Jakaria
1305024

---

Kowshika Sarker
1305034

---

Mostofa Rafid Uddin
1305039

---

Md.Mohaiminul Islam
1305077

---

Trisha Das
1305098

# CERTIFICATION

This thesis titled, **"Archiving Medical Records in DNA Sequence"**, submitted by the group as mentioned below has been accepted as satisfactory in partial fulfillment of the requirements for the degree B.Sc. in Computer Science and Engineering in April 2018.

**Group Members:**

**Md.Jakaria**

**Kowshika Sarker**

**Mostofa Rafid Uddin**

**Md.Mohaiminul Islam**

**Trisha Das**

**Supervisor:**

---

Md. Shamsuzzoha Bayzid
Asst. Professor
Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology

# ACKNOWLEDGEMENT

Setting off by thanking our mentor, adviser Dr. Md. Shamsuzzoha Bayzid would be the correct gesture. It was due to him we were introduced to this novel field of research. We are really grateful for the opportunities, direction and the freedom that was provided while we were undergoing our research. His invaluable supervisions and insights inspired us a lot in developing the work we have done.

We are thankful to our friends from CSE, BUET who constantly provided us a suitable environment for carrying out our research work.

Finally, we want to thank our families for always being supportive and being there when we needed them the most.

Dhaka
April 2018

Md.Jakaria

Kowshika Sarker

Mostofa Rafid Uddin

Md.Mohaiminul Islam

Trisha Das

# Contents

# List of Figures

# List of Tables

# List of Algorithms

# ABSTRACT

We present a proof-of-concept for archiving medical histories of a person in his DNA sequence. Functions of more than 98% of the human genome are currently unknown and these parts are called non-coding regions or Introns. We developed a lossless architecture for efficiently storing, managing and retrieving electronic health records (EHRs) within the non-coding regions of a patients genome sequence for efficient archival of both EHRs and genome sequences. We also proposed an efficient binary to ATCG conversion technique which is better than the traditional naive Binary-to-DNA mapping (where two binary bits are mapped to one of the four DNA bases). With the recent success in DNA storage system, which is more robust and long-lasting than the typical hardware storage systems, we believe our proposed system will open the possibility to store the medical records of an individuals entire life in his DNA sequence which may last for many years, and thus will contribute towards modern research in personalized medicine and future healthcare system in general. We developed a software implementing our proposed architecture.

# Chapter 1

# Introduction

Healthcare is moving towards personalized medicine which involves the use of individualized information so that medical treatment can be tailored to the specific characteristics of an individual. Modern research on healthcare requires access to biological samples, including DNA sequences and patients medical records. Therefore, central to personalized medicine is the ability to store and analyze DNA sequences and the medical records of the patients. Various retrospective studies related to personalized medicine will require access to both the DNA sequences and medical records of patients. While the demand for DNA banks is evident from the recent expansion of DNA sampling and collecting, efficient management, organization and protection of both DNA sequences and medical records of patients for sufficiently long enough time should be the cornerstone of the future healthcare system. However complete human genome consists of about 3 billion base pairs and storing them in FASTA format requires about 3GB space for each patient. Including this for all individuals requires a huge additional data storage. In our work we have addressed the problem and depicted an efficient solution with significant biological and computational insight. In our DNA sequence, only 1.5% is coding region or gene and the rest 98.5% is non-coding (which is often referred as Junk DNA). Among this non-coding part there is one inter genic portion and intragenic portion or introns. Though there is debate about functions of introns, it is somewhat established that the intergenic region doesnt have any sequence dependent function. But DNA is sequenced sequentially and parts of it cant be omitted from sequencing. So, we get the 'non-functional' sequences with Gene sequences and have to store both. We have come up with an idea that we may use this place for storing EHR files. We have developed a File Management System (FMS) inside human DNA sequence which is able to successfully insert, retrieve and delete a file inside the human DNA sequence. We also included compression in every step of our work so that we can store the EHR files of a patients entire lifetime in his genome sequence. The idea has come from the concept of DNA storage research which basically focuses on storing large amount of data in DNA sequence as small as possible. Hence, we do the same. We have also developed a new binary to DNA bit encoding

technique that works better for medical image files. Finally, we have created a software that works as the model weve created to establish our proof of concept.

## 1.1 Motivation

This work is motivated by the recent research done by Microsoft research in collaboration with University of Washington on how to take digital files and convert them into strings of DNA that can be easily stored and read back. It is also motivated by the recent attempts of storing genome sequences of individual patients in their corresponding EHRs (Electronic Health Records). From researches on DNA storage we have been acquainted with the idea of converting binary sequence to DNA bit sequences. Besides, in September 2003, The National Human Genome Research Institute (NHGRI) launched a public research consortium named ENCODE, the Encyclopedia Of DNA Elements to carry out a project to identify all functional elements in the human genome sequence. The result was published in 2007 and it was found that about 98.5 of the genome doesn't provide any function or they couldn't discover their function. Since 2007, many researches have been done to find out the functions of this non-coding DNA. Two subsets of non-coding DNA was found. One is intron (intra genic non-coding region) and other is called spacer DNA (intergenic non-coding region). Some functions have been identified for the introns, but the functions of spacer DNA is not found yet and it is thought that it may have sequence-independent function but has no sequence dependent function at all. All these facts paved our idea of inserting EHR files inside the spacer DNA parts of genome sequences. However, one may take into account the intron parts, too as our system is completely customizable.

## 1.2 DNA Storage

The world's data is increasing at geometric progression due to updated technologies (like high resolution camera phone, social media, digital information etc). Increased amount of data and energy conservation make it challenging to the data storing system available at present time. Hundreds of digital data storing system developed to store these data. But the most popular technologies (magnetic tape, magnetic disk, SSD) became saturated breaking the Moore's Law. Hence its became a matter of course to store these information in an optimum way i.e. low cost, high density. So scientists seeked for new technology to store this high volume of digital data. DNA storage system is considered an attractive solution to this problem.

## What is DNA?

Deoxyribonucleic acid or DNA is made up of molecules called nucleotides. Each nucleotide contains a phosphate group, a sugar group and a nitrogen base. There are four types of nitrogen bases in DNA : Adenine (A), Thymine (T), Guanine (G) and Cytosine (C). The order of these bases is what determines DNA's instructions, or genetic code. Human DNA has around 3 billion bases, and more than 99 percent of those bases are the same in all people.

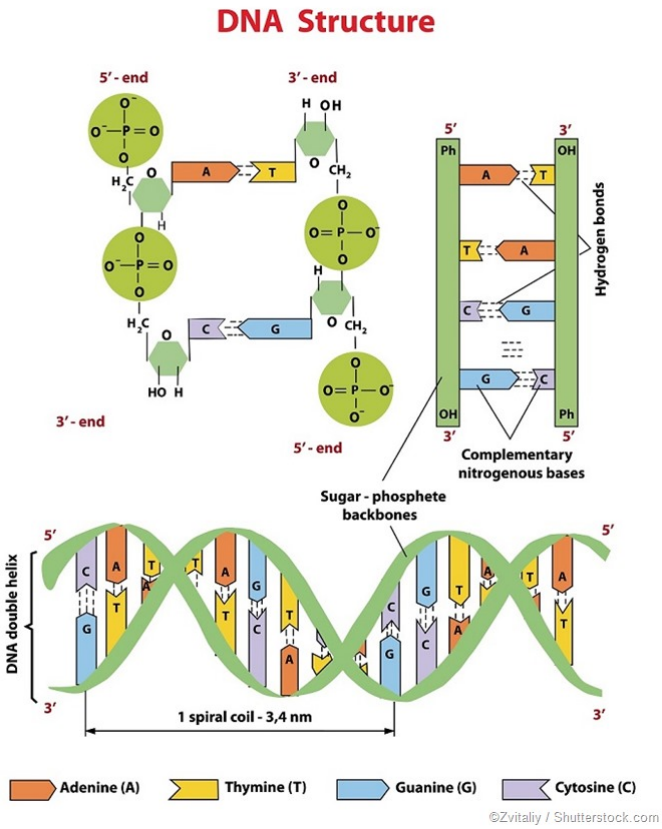



Figure 1.1: DNA structure

## Overview of DNA storage system

DNA digital data storage refers to any process to store digital data in the base sequence of DNA. This technology uses artificial DNA, which is made by using commercially available oligonucleotide synthesis machines for storage. DNA sequencing machines are used for retrieval [2]. Basically, DNA is itself a storage of information as it is actually a sequence of {A,T,C,G}*. But when we artificially store real life data into it by representing digital data with nucleotide sequences it may be referred as DNA digital data storage. DNA storage system is kept at very bottom of the storage hierarchy [3] because of its high density, high durability and high access time. Every DNA storage system architecture has three basic parts. First one is a DNA synthesizer. The work of DNA synthesizer is to encode binary data to nucleotide sequence to be

stored in DNA. The second component of storage system is DNA container that is divided into compartment. Each compartment of the DNA container store pools of DNA which contains previously saved data. The last principle component of DNA storage system is DNA sequencer which is reverse to the DNA synthesizer in nature. The work of this component is to decode the the pools of DNA nucleotide sequence to binary data and retrieve the original archived data.

### Benefits of DNA Storage

The capacity (or density) of DNA is far high compared to the technology used nowadays. It has been appeared as a durable data storage media that is non volatile in nature. Rapid growth of biotechnology of DNA synthesis and sequencing has paved the way to DNA system by reducing the cost. Recent lab experiments have proved the aptitude of the storage system. The storage capability is increasing more than 2 times per year immediately following the Moores law. In 2012, [1] demanded a storage density 700TB/gram achievable in lab in 2012. In the following year, the literature [2] demanded a storage density of 2PB/gram which is raised by more than two times within a year. Many architecture has been developed for DNA based archival storage, random access storage etc.

## 1.3 Problem Discussion

The trending precision medicine or personalized medicine concept emphasizes inclusion of genomic data into patient's health records. For it's importance, precision medicine is also named 'Genomic Medicine' [4] [5]. However, including genomic sequences in EHR has raised some complexities. Apart from security and ethics issues, there are practical challenges to integrate genomic data into electronic health records include size and complexity of genetic test results, inadequate use of standards for clinical and genetic data, and limitations in electronic health record capacity to store and analyze genetic data [6]. We addressed the problem of storage capacity of EHRs and also the huge size of the genomic data (about 3 billion base pairs).

## 1.4 Solution Overview

Our key idea is to store EHR records of patients inside their corresponding DNA sequence. EHR components e.g. prescriptions, reports, DICOM files, bills etc. are compressed with appropriate methods. For greater compression ratio in short time, we employ different compression techniques for DICOM images and other files. Binary bits of these compressed files are mapped to DNA base sequence {A, T, C, G}*. Resultant ATCG string is further reduced to a shorter

sequence.Next, we store the file i.e., the final ATCG chain in patient's genome sequence. This is challenging as we need to divide the resultant ATCG chains into smaller chunks so we can spread them over the inter genic regions of the genome. Also, we need to store the meta-data regarding the storage information of a file (which regions the resultant ATCG has been mapped to) in the DNA sequence as well so we can retrieve the data. We proposed an efficient in-DNA file management system. We maintain a dictionary of saved files and their positions in the genome to retrieve stored files correctly. The dictionary itself is converted to ATCG sequence and stored in the genome.

## 1.5 Our Contribution

- Analyzed the history of DNA storage system
- Addressed the data redundancy problem in storing patient genome sequences in EHR
- Proposed a concept of storing EHR files in genome sequence
- Developed a model of File Management System (FMS) inside FASTA / 2BIT file of human genome
- Included data compression in our architecture
- Applied and analyzed compression techniques over EHR files (specially DICOM files)
- Developed a new method of Binary to DNA bit conversion which works better for most medical image files
- Applied and analyzed compression techniques for DNA bits or ATCG sequences
- Created a working software with an user friendly GUI

## 1.6 Outline

This dissertation book is composed of total 10 chapters.

Chapter 1 is the current chapter and introduces the topic of the thesis. The motivation of our works is the main focus of this chapter. An overview of DNA storage, problem definition, solution and our contribution are described in this chapter.

In chapter 2, we discuss about beginning of the DNA storage system, its evolution and some other branches of works.

Chapter 3 discusses the existing tools and technologies that are currently used for storing digital information in DNA. This chapter focused on medical files specially DICOM file and its structure and Electronic Health Records (EHR).

Chapter 4 describes the proposed pipeline of this project.

In chapter 5, we presented different DICOM file compression techniques, file conversion procedures and compression results.

Chapter 6 discusses the previous methods and our approaches of converting binary stream of medical files to nucleotide sequences.

The compression processes of nucleotide sequences are described in chapter 7. This chapter includes the traditional compression processes and our proposed method.

In chapter 8, we elaborate the file management system of proposed architecture. This chapter includes system structure and file operations allowed in this structure.

Chapter 9 contains the implementation informations of our system.

Finally, in chapter chapter 10, we conclude our research work and explore the future possibilities of our approach.

# Chapter 2

# Evolution of DNA Storage Concept

## 2.1 The Beginning

DNA storage have long been considered as the future of data storage. It dates back to 1988 when the concept of storing messages in DNA came into light for the first time via Microvenus project. In 1988, Joe Davis, an artist collaborating with molecular biologist Dana Boyd in Jon Beckwith's lab at Harvard Medical School (and currently a research affiliate in George Church's lab), designed and synthesized an 18 base-pair message encoding the image of the ancient Germanic rune representing life and the female earth. [7] The Microvenus message was then pasted into a vector and transformed into E. coli, creating a living work of art. Microvenus project was inspired by the binary message sent by Carl Sagan and Frank Drake from the Arecibo radio telescope in 1974, an attempt to open up communication with extraterrestrial intelligence. [1] Sagan and Drake converted an image to a single stream of binary digits. The image is a 23x73 rectangle (having the dimensions be two prime numbers makes it easier to decode the single stream of binary digits) showing pictures of the telescope, a person, and information about solar system and DNA.

Microvenus was coded with a similar principle. It encoded a simpler image of ancient Germanic rune representing life and the female earth. The lines of the image translated to ones and zeros in a 5x7 grid, converted to nucleotide sequence with phase-change values rather than numerical values. The DNA bases are arranged by size for example C= 1, T=2, A =3, G=4 and represent the number of bits needed before you switch to the opposite bit. For example, 10101 translates to CCCCC because each digit occurs once before it switches, and 00011 would be AT because there are three 0 before it switches to two ones.
Joe Davis addressed the problem of stability and duration of message in space. Most sophisticated modern information storage media, such as optical disks, integrated circuit "chips," and magnetic tapes, cannot permanently withstand exposure to heat, light, moisture, and oxidizing

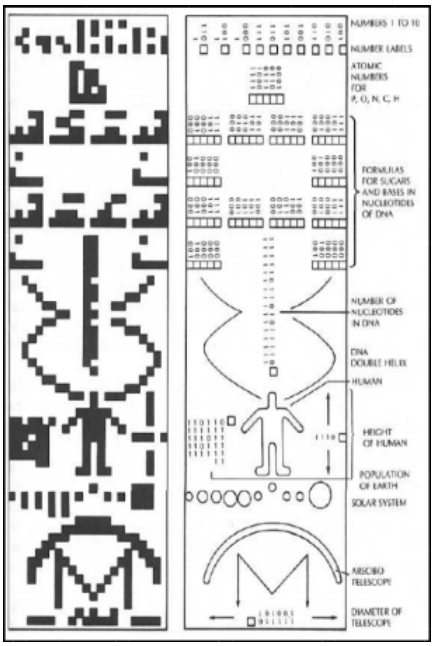



Figure 2.1: "Arecibo message", a interstellar radio message sent by Carl and Sagan carrying basic information about humanity and Earth in 1974 [1].

chemical activity encountered in normal terrestrial environments. DNA itself is actually a relatively stable chemical, yet like conventional communications media, it will ultimately fall apart in "shirt-sleeve" environments because most terrestrial environments contain enzymes that will degrade DNA. Kept within a bacterial carrier however, especially in spore-forming bacteria, it might survive intact for an indefinite period of time. Thats why the microvenus DNA sequence was synthesized and kept in the plasmid of cell of E.Coli Bacteria. The plasmid is a good vector for DNA. Restriction enzymes were used to cleave plasmids. Other enzymes (ligase and polymer) were used to attach Microvenus DNA to plasmid DNA to "glue" the recombinant plasmids back together again. Microvenus plasmids were cloned into several laboratory strains of E.Coli. The Microvenus project originally emphasized biological and artistic solutions for problems associated with the search for extraterrestrial intelligence. However, it also paved the way to the further research and motivation of DNA storage.

## 2.2 Cryptography with DNA Strand

The need of data security is increasing with the increasing data size and data sensitivity. The methods and processes of keeping sensitive data secure is called cryptography. Three components of cryptography are: Plain text: non encrypted form of message, Encryption: Process of transforming plain text message into encrypted message called cipher, and Decryption: process of transforming encrypted cipher into plain text message. Cryptography has a long history begins from Julius Caesar, popularly known as Caesar's cipher, or the shift cipher. Widespread use


Figure 2.2: The bitstream version of the message containing 1079 bits

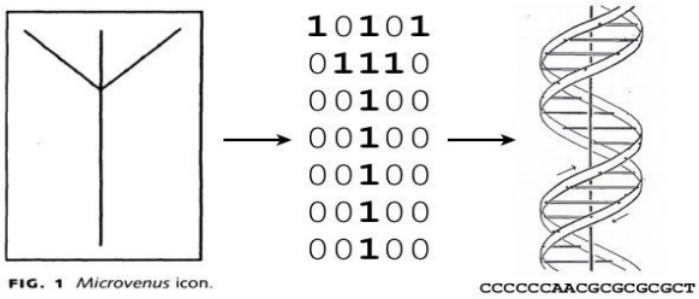



Figure 2.3: MicroVenus Encoding

of internet accelerate the advancement of field of cryptography. Many cryptography methods has been developed and multiple methods are used simultaneously to get more reliability and security. The traditional cryptography methods are symmetric key or private key cryptography, and asymmetric key or public key cryptography. These cryptographic algorithms are week and breakable at some extent. The more advance cryptographic methods are quantum cryptography and DNA cryptography. Quantum cryptographic system relies more on physics, opposite to the traditional cryptography, which are based on mathematics. Its founded on quantum mechanics and Heisenbergs uncertainty principle [8].

Two separate laboratories outlined architecture for DNA cryptographic system in [9] and [10]. The first architecture assumed that plain text message data are encoded in DNA strands by use of public key of short oligonucleotide sequences. They proposed a number of methods that use pre-assembled library of one-time-pads in the form of DNA strands which are in principle unbreakable. They also present a DNA cryptosystem for 2D image by use of photo sensitive DNA-on-a-chip technology. At the end of the paper, they present a DNA Steganography tech-

nique that hide secret message within other messages.

The second architecture used microdots DNA for hiding messages. Microdot DNA is a molecular version of steganography for sending secret messages. They represent messages by nucleotide sequences generated by PCR which was amplified by 20 base forward and reverse primers. They made it challenging for the adversary who might detect the microdot to decrypt the plain message. They proved the feasibility of their proposed technique by synthesizing a secret-message of 69 nucleotides long flanked by forward and reverse PCR primers.

Literature [11] proposed 2 methods of cryptography to encrypt the spy message using DNA. In the first method, they transform messages into ASCII code in decimal format and group the decimal numbers in block and encrypt them using traditional encryption methods. The encoded binary cipher transformed into nucleotide string using naive transformation. Forward and reverse primers are then added to the encoded nucleotide string and then then flanked by many sequences of DNA or by confining it to a microdot in the microarray. After the final step, this cipher is now securely transferable through internet or physically. The second method is based on DNA transcription and DNA translation and skipped thereby!

## 2.3 Works Afterwards

### Improved Huffman Coding (2009)

In 2009, Human coding approach was used for archiving text, music and image characters in DNA by Ailenberg et al. [12]. They converted an image of a lamb and the song “Mary had a little lamb” in DNA strands. They created separate DNA codons for text, image and music data. While designing codons, they emphasized on how less number of nucleotides can be used to express how much amount of data. They described the rationale of their method as their method serves one of the prerequisite of good DNA coding method - “economical use of nucleotides”. They synthesized a sequence of 844-bp DNA that stores information for the text, music, and image of the nursery rhyme Mary Had a Little Lamb described in the text. They created DNA codons based on frequency. For text data, DNA codons (G, TT, and TA) were placed as group headers, and the rest of the codons were placed in each group in decreasing frequency order, allowing for 69 unambiguous codons. Characters of the computer keyboard were assigned a unique DNA codon in decreasing frequency order. For image and music, they used one column modified huffman coding.

## Creation of a bacterial cell controlled by a chemically synthesized genome (2010)

In 2010, Gibson et al. successfully recovered 1280 characters encoded in a bacterial genome as watermarks. [13] But this approach was in vivo (inside an organism). They designed, synthesized and assembled *M. mycoides* genome from digitized genome sequence information and Transplanted it into a *M. capricolum* recipient cell. To differentiate between the synthetic genome and the natural one in the host cell, the included four watermark sequences in the genome. They could successfully recover the watermarks and was also successful in controlling the new cell only by the synthetic chromosome.

### Concept of Next generation digital Information storage system (2012)

In 2012, [14] developed a DNA synthesis and sequencing strategy to encode arbitrary digital information. They converted an html-coded draft of a book that included 53,426 words, 11 JPG images and one javascript program into 54898, 159-nt oligonucleotides. Each oligonucleotide consist of 96 nt data block, 19 nt address block and forward and reverse amplifier of 22 nt of each. They synthesized the oligo library by using ink-jet printed, high-fidelity DNA microchips. The library was then amplified by limited-cycle polymerase chain reaction and sequenced on a single lane of an Illumina HiSeq. To reduce the effect of sequencing error, they joined overlapping paired-end of 100 nt. Using this encoding process, the authors were able to recover the actual data with a total of 10 bit errors out of 5.27 million.

This newly developed next generation digital Information storage system had several improvement over the previously developed methods. To exclude the difficulty of reading and writing of some extreme content, they used a nucleotide to encode a bit instead of two. They also introduced the addressing scheme of DNA block so that it was no longer necessary to constructs long DNA strands. They synthesized, stored, and sequenced many copies of each individual oligo to avoid cloning and sequence verifying constructs. Most importantly they leveraged next-generation technologies which cost less than first-generation encodings.

### Towards practical, high-capacity, low maintenance information storage in synthesized DNA (2013)

In 2013, Goldman et al. [15] developed a scalable and reliable method to synthesize and store the information in vitro using shorter DNA fragments. This aid the facility of manipulating isolated DNA fragments and the routine recovery of intact fragments from thousand years old samples. This yields a low low-maintenance environments with long lifespan. To prove the concept, they encoded five computer data files totaling 739 kilobytes of hard-disk storage and

with an estimated Shannon information of 5.2 $\times 10^6$ bits into a DNA code. They encode each byte by a single DNA sequence with no homopolymers to reduce the error rate. Then each DNA sequence was split into overlapping segments with fourfold redundancy. They represented all five files by 153,335 strings of DNA, each 117 nucleotides. The oligonucleotides were synthesized by Agilent Technologies OLS (Oligo Library Synthesis) process. They created 1.2 $\times 10^7$ copies of each DNA string with an error rate 1 per 500 bases. To decode the nucleotide sequence they reconstructed full-length (117-nt) DNA strings in silico which yielded 79.6 $\times 10^6$ read-pairs of 104 bases in length. Reverse of the encoding procedure systematically discarding strings containing errors due to high level of redundancy. After manually inserting 50 bases, the original file had been reconstructed with 100% accuracy. The new approach allows for storage density of 2.2 PB/gram while consuming just 10% of the library produced from the synthesized DNA. They claimed that new methods would improve efficiency, cost and reliability, as well as storage time.

**A Rewritable, Random-Access DNA-Based Storage System (2015)**

In 2015, Yazdi et al [16] first developed a method for rewritable random-access DNA-based storage. They overcame the drawbacks of previously developed read only archival storage architecture that require sequencing the whole genome structure to retrieve even a unit of data and no way of editing synthesized data. Its encoding is dictionary-based and focuses on storage of text. They developed a new constrained coding techniques and DNA editing method to achieve data reliability, specificity and sensitivity of access and exceptionally high data storage capacity. The architecture was based on specialized address strings with inherent error-correction capabilities which is used for selective information access. This architecture use a special form of prefix-synchronized coding [17] to encode given address sequences. The address space is made explicit by choosing addresses to be mutually uncorrelated and at large Hamming distance from each other. The prefix encoding format is used for selection of the blocks to be rewritten via two DNA editing techniques, the gBlock and OE-PCR (Overlap Extension PCR) methods. They encoded parts of the Wikipedia pages of six universities in the USA, and selected and edited parts of the text written in DNA corresponding to three of these schools. This effort succeed with 100% accuracy.

**DNA-based archival storage system (2016)**

In 2016, researchers from the University of Washington and Microsoft Research [18] have collaborated in working on how to take digital files and convert them into strings of DNA that can be easily stored and read back. They proposed an architecture for a DNA-based random access archival storage system.This architecture was modelled as a key-value store. In this architecture, they present some new encoding scheme to provide controllable redundancy to tolerate

errors without much affecting data density. They introduced rotating encoding to nucleotides to avoids homopolymer and so reduce the chance of sequencing errors. They synthesized DNA segments as separate strands and tagged strands with identifying primers. Hence size of storage increased and random access to isolate molecules of interest became viable. For reliability issue, the authors used XOR encoding which provide high reliability without incurring significant overhead. The reliability is made tunable so that different data gets level of reliability. They introduce Polymerase Chain Reaction (PCR) for achieving random access. They carried series of wet lab experiments to demonstrate feasibility, random access, and robustness of the proposed architecture. Whereas Yazdi et al. worked for text data, Bornholt et al. worked for binary data. They synthesized and sequenced four image files of size varying from 5kB to 84 kB. They were successful in recovering all four files where three files recovered without any error and the last file incurred a one-byte error.

### 2.3.1 Current Situation

DNA has many advantages for storing digital data. It's ultra compact, and it can last hundreds of thousands of years if kept in a cool, dry place. And as long as human societies are reading and writing DNA, they will be able to decode it. The only problem now is the synthesis and sequencing limitations. But they are being removed away fastly. We can observe that from Carlson Curve. The Carlson curve is a term to describe the rate of DNA sequencing or cost per sequenced base as a function of time. [19] It is the biotechnological equivalence of Moore's law.

So, we may hope that we'll see practical usuage of DNA storage technologies in our usual life in near future.

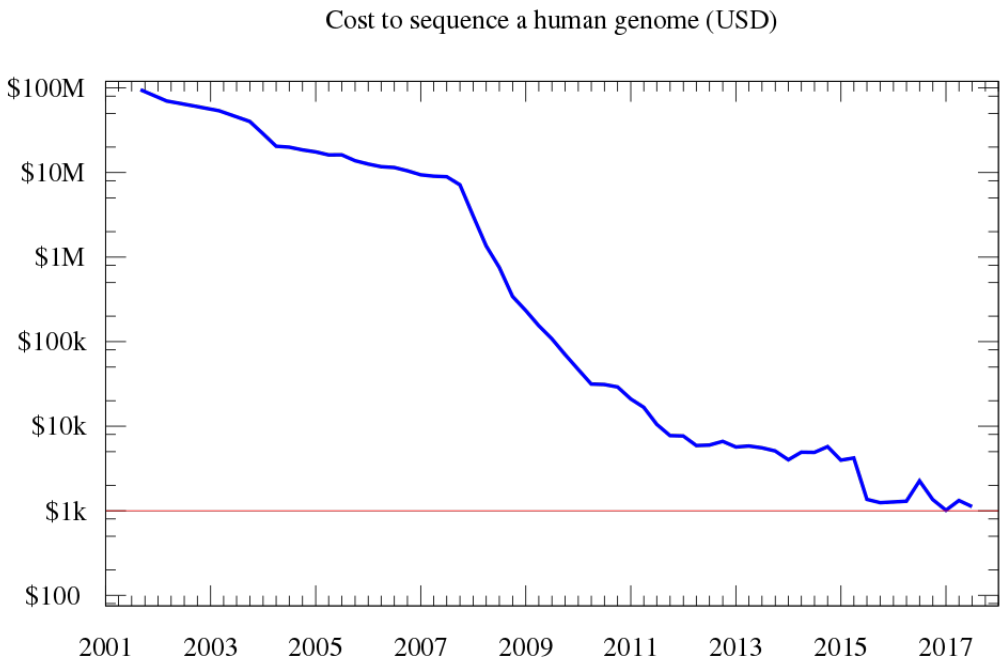



Figure 2.4: Total cost of sequencing a human genome over time as calculated by the NHGRI.

# Chapter 3

# Background

In this chapter, we include the descriptions of the necessary terms that are used throughout the book.

## 3.1 Genome and genome sequence

A genome is an organisms complete set of DNA, including all of its genes. Each genome contains all of the information needed to build and maintain that organism. In humans, a copy of the entire genomemore than 3 billion DNA base pairsis contained in all cells that have a nucleus. This sequence is made of A(Adenine), T(Thymine), C(Cytosine) and G(Guanine). In some places it is denoted N representing gaps that we can't fill yet with any nucleotides.

## 3.2 Exon, Intron and Spacer DNA

Among all the 3 billion base pairs, there are mainly two regions. One is Gene, another is Inter Genic Region that is situated between consecutive genes. The gene has two parts again. The portions of the gene that code for proteins via converting into messenger RNA or mRNA are called exon. On the other hand, the portions of gene that do not code for proteins are called intron. The word *"intron"* derived from the term *IntraGenic Region* or region inside a gene. This part is removed by RNA splicing during the final step of transcription (process of conversion of DNA into m-RNA). American biochemist Walter Gilbert termed them as 'introns' and the regions that are expressed was named as 'exons'. [20] The Inter Genic Region (IGR) is often called spacer DNA. In bacteria, spacer DNA sequences are only a few nucleotides long. But in eukaryotes like humans, they can be extensive and include repetitive DNA, comprising the majority of the DNA of the genome. [21]

About 57 % of human DNA is spacer DNA and 43 % consists of all gene. Among all genes, 96 % is intron and 4 % is exon. Overall, in human DNA, 98.5 % is non-coding (consists of intron and IGR) and only 1.5 % is coding (exon) region.

## 3.3 Personalized Medicine

Personalized Medicine or precision medicine is a medical process that classifies patients into different groups based on their risk of disease and predicted response to disease and based on that group individual patients are tailored with medical decisions, practices, interventions and products. [22] The term has risen in usage in recent years given the growth of new diagnostic and informatics approaches that provide understanding of the molecular basis of disease, particularly genomics.It is developing more and more with a goal of developing drug discovery and patient care system. [23]

## 3.4 Electron Health Record (EHR)

An electronic health record (EHR) is a digital version of a patients paper chart. EHRs are real-time, patient-centered records that make information available instantly and securely to authorized users. [24] While an EHR does contain the medical and treatment histories of patients, an EHR system is built to go beyond standard clinical data collected in a providers office and can be inclusive of a broader view of a patients care. EHRs can:

- Contain a patient's medical history, diagnoses, medications, treatment plans, immunization dates, allergies, medical images, and laboratory and test results
- Allow access to evidence-based tools that providers can use to make decisions about a patients care
- Automate and streamline provider workflow

The main benefit of EHR is that it makes all the details of patient easily shareable from one provider to another and thus creating a way towards Personalized Medicine.

## 3.5 DICOM

DICOM (Digital Imaging and Communications in Medicine) is a protocol used worldwide to store, exchange, and transmit medical images. It was developed by American College of Ra-

diology (ACR) and National Electrical Manufacturers Association (NEMA). DICOM has been central to the development of modern radiological imaging. DICOM incorporates standards for imaging modalities such as radiography, ultrasonography, computed tomography (CT), magnetic resonance imaging (MRI), and radiation therapy. DICOM is the most adopted format for medical images and gaining more acceptance everywhere day by day as a universal standard. It provides all the necessary tools for diagnostically accurate representation and processing of medical imaging data. Moreover, DICOM is not just an image or file format. It is an all-encompassing data transfer, storage and display protocol built and designed to cover all functional aspects of contemporary medicine. So, DICOM is not a single standard, rather it is a set of standards. Today DICOM truly governs practical digital medicine. [25]

### 3.5.1 DICOM File Structure

DICOM file structure follows a specific and strict set of norms fixed by the American College of Radiology (ACR) and the National Electrical Manufacturers Association (NEMA). All DICOM files have mainly 4 parts :

1. **File Preamble** This portion is fixed for every dicom file. It consists of the first 128 bytes (0th to 127th bytes) which is defined for application-specific purpose. If the file doesn't contain any, then these 128 bytes are all filled with '0's.
2. **Prefix** From 128th to 131th bytes there are 'D','I','C','M' respectively. This is mandatory for every DICOM file. DICOM files are identified by checking these 4 bits.
3. **Header** The size of this part varies among files and is a little bit complicated. It starts from 132th byte and contains the meta data part. It also contains the transfer syntax of the file. This part is divided into groups and each group consists of several data elements. Each group is defined a separate meaning according to its number. However, detail description of the data elements is complex and out of the scope of this thesis.
4. **Pixel Data** This part contains the image part. Each pixel contains an intensity value. Images may be monochrome or RGB. This part is also variable among files and doesn't have a fixed starting point.

### 3.5.2 DICOM features and Variations

DICOM files varies with different features. Some of the features are :

- *Mediastorage* which defines what type of medical image is that- a MRI, CT Scan, X-Ray or something else.

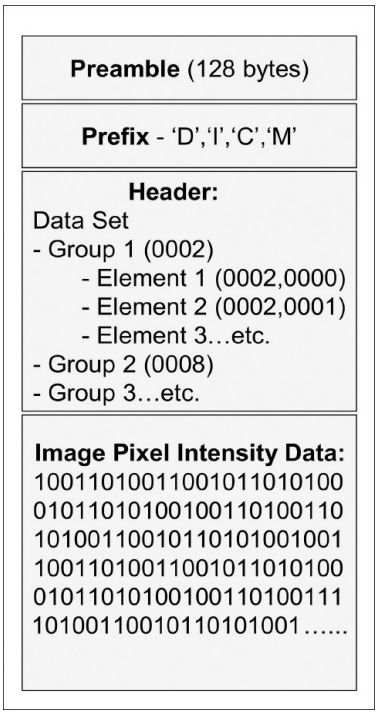



Figure 3.1: DICOM file structure

- *Photometric Interpretation* which defines whether the image is Monochromic or RGB
- *TransferSyntax* defines different types for DICOM standard details covered in 5.1.4
- *Number of Dimensions* defines the number of dimension of the picture
- *Samples per pixel* defines how many samples are taken for each pixel
- *Bits allocated per pixel* defines how many bits are needed to store each pixel
- *Scalar Type* defines the data type of the scalar for each pixel
- *Orientation Label* defines whether the orientation of the object on image is Sagittal, Axial or Coronal.
  Some samples of DICOM images are attached here.

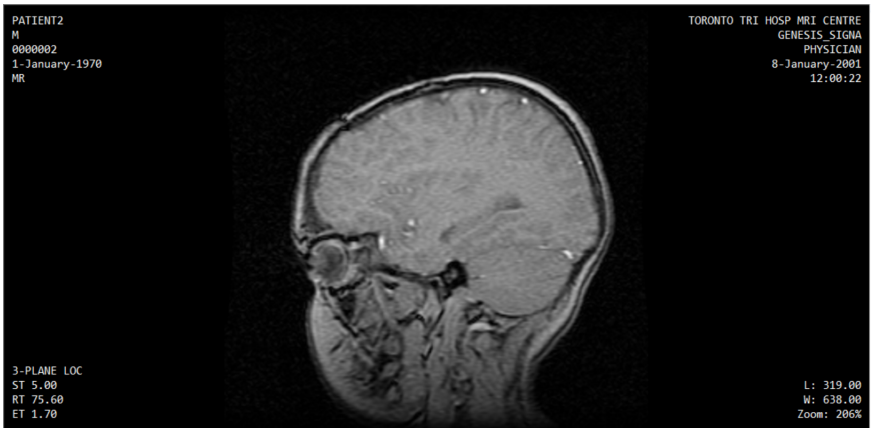



Figure 3.2: A DICOM showing MRI of Brain

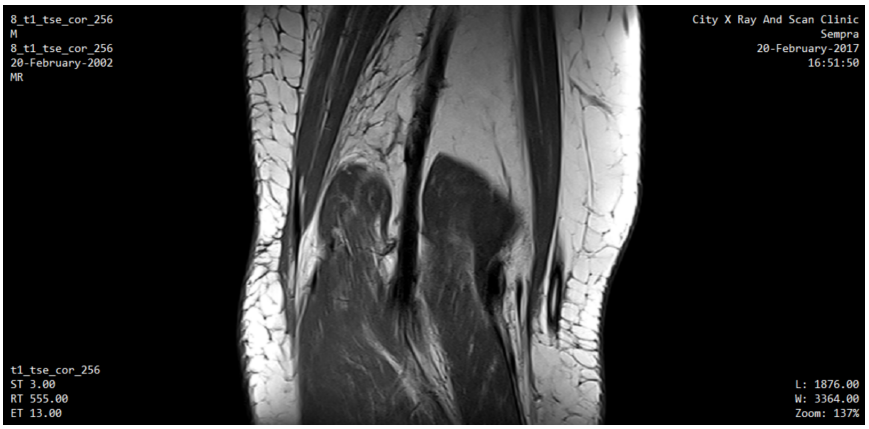



Figure 3.3: A DICOM showing X-Ray Result

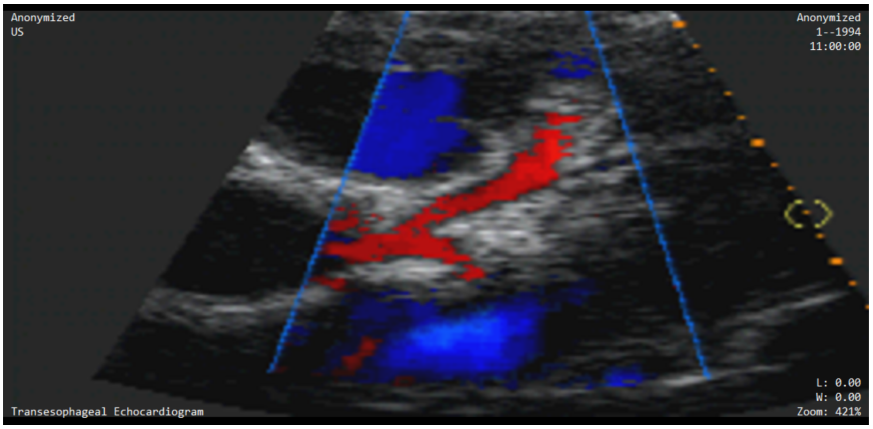



Figure 3.4: A RGB Colored DICOM showing ECG report


Figure 3.5: A DICOM showing CT Scan result

# Chapter 4

# Archiving Medical Records in DNA

EHR components e.g. prescriptions, reports, DICOM files, bills etc. are compressed with appropriate methods. For greater compression ratio in short time, we employ different compression techniques for DICOM images and other files. Binary bits of these compressed files are mapped to DNA base sequence {A, T, C, G}*. Resultant ATCG string is further reduced to a shorter sequence.

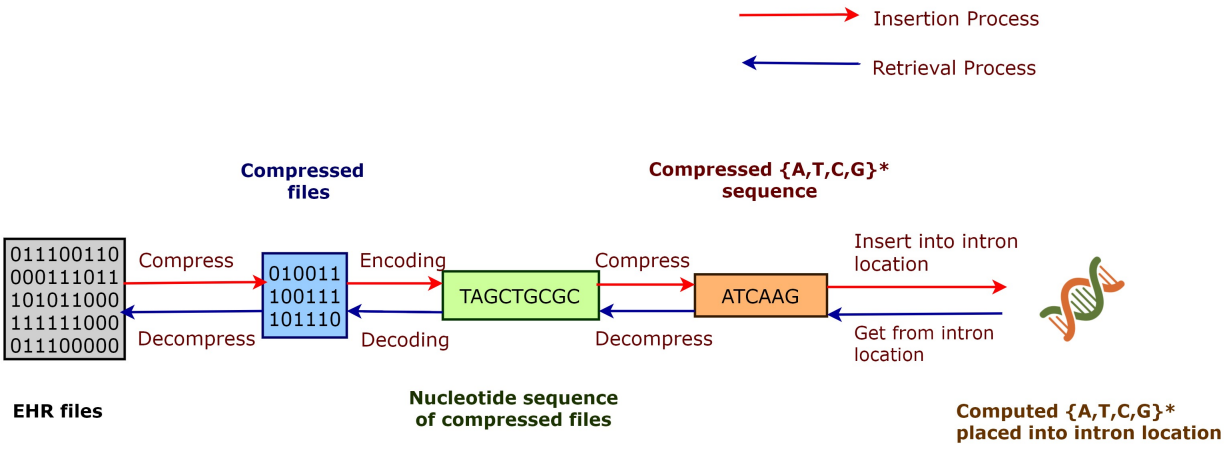



Figure 4.1: Pipeline

Next, we store the file content i.e., the final ATCG chain in patients genome sequence. This is challenging as we need to divide the resultant ATCG chains into smaller chunks so we can spread them over the intronic regions of the genome. Also, we need to store the meta-data regarding the storage information of a file (which regions the resultant ATCG has been mapped to) in the DNA sequence as well so we can retrieve the data. We proposed an efficient file management system in DNA intergenic regions. We maintain a dictionary of saved files and

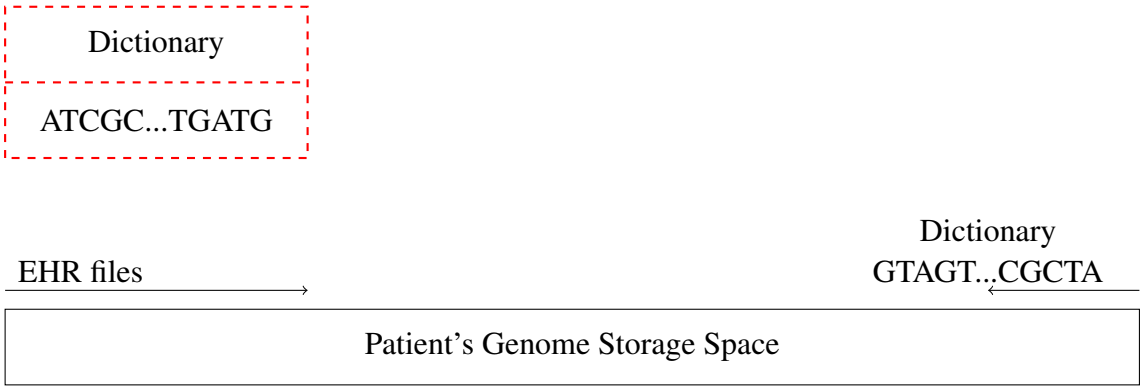



Figure 4.2: Patient's Genome Storage Space

their positions in the genome to retrieve stored files correctly. The dictionary itself is converted to ATCG sequence and stored in the genome which adds another level of complexity. Note that the size of this dictionary will grow as we store more files. Thus we have to store and manage the dictionary efficiently so we do not lose any meta-data and at the same time we have to make sure it does not grow and overwrite the contents of a file that have already been written into the DNA sequence. Similar to the stack in random access memory (RAM), dictionary is kept in reverse order from the last end of genome. EHR files are saved from the beginning of the genome.

Keeping dictionary from last of genome is for ease of maintenance. Storage space in genome is fixed but depending on number and sizes of stored files and dictionary size will vary. So we need to keep aside some space for dictionary beforehand. For space efficiency in this case, dictionary can be kept in several chunks, which will occupy storage alternately with stored files. Then we would also need to maintain a chain of pointers (genome location) among these chunks. We preferred to avoid these hassles using virtual memory concept. We have proposed and developed and efficient architecture to avoid these challenges. In genome, we use junk DNA locations (intergenic DNA and introns) to store EHR. If given genome sequence has enough room to insert ATCG of the new file, we search in dictionary for positions to insert new data and replace junk DNA sequence in appropriate location with EHR files ATCG content. Positions for next file insertion is updated in the dictionary. To retrieve a file saved in the patient's genome, we find the file's storage locations from the genomes dictionary and read ATCG contents from corresponding locations. These ATCG strings are then converted back to original binary files following decompression and reverse mapping of the insertion process.

# Chapter 5

# Compression of mMedical Files

Electronic Health Records occupies a large space as a whole. In order to be able to store large EHRs, we need to compress the medical records. In particular, as DICOM files are the most space consuming and have special file formats, we proposed customized compression techniques suitable for DICOM files. We performed an extensive evaluation study to identify the techniques suitable for DICOM. Finally, we customize the existing technique so that they perform well on DICOM.

## 5.1 DICOM Files Compression

The DICOM files are usually much bigger in size. So, before converting them into bitstreams and then nucleotide sequence, we plan of compressing them using the technologies available. However, Compressing medical imaging data is a tricky thing to do. The American College of Radiology (ACR) don't provide clear recommendation for compression. The US FDA does not permit compression storage or transmission of breast imaging. For other categories of image, we should carefully choose the type of image compression. So, if there is a viable need for compression due to congested bandwidth or limited storage, we should stick to lossless compression techniques. Hence we had to use the compression that serves the best compression ratio maintaining the DICOM standards.

### 5.1.1 Previous Works

A lot of works have been done in the field of image compression. However, usual image compressions are not applicable for DICOM images as DICOM images are not only images, they also contain many useful metadata and fixed by a well-defined structure. So, compression of DICOM files is different than compression of non-DICOM binary images.

However, DICOM being the universally adopted standard for medical images, its compression has caught attention of researchers worldwide. Recent works have shown some successful algorithms that has achieved significant achievements in compression. Among them mentionables are "multi wavelet and hybrid speck-deflate algorithm" which combines the wavelet encoding and deflated algorithm [26], high efficiency video coding which uses the principle of H.264/MPEG 2 AVC ( Advanced Video Coding) and is an improvement over them [27]. Significant amount of work has been also done to improve the HEVC Technique, like HEVC intra coding [28]. It is also explicit from some experimental results that, HVEC works much better for high bit-depth medical images [29]. However, inspite of being a huge surge for including HEVC in DICOM standard, it is not widely addopted yet. The widely adopted compression techniques that current DICOM standard supports RLE (Run Length Encoding), Deflate, JPEG, JPEG LS, and JPEG 2000 compression techniques. Among these, JPEG is lossy while RLE, Deflate, JPEG LS and JPEG 2000 are lossless compression techniques. Since we need our architecture to be lossless, we only evaluated the lossless compression techniques recommended by DICOM standard. In order to simulate various compression algorithms we used Grassroots DICOM (GDCM). GDCM is an open source C++ library, which implements DICOM standard. It is accessible from Java, PHP, Python, and C#. With gdcm, we checked with DEFLATE, RLE, RLE LS, JPEG, and JPEG 2000 compressions.

Here is a brief description of each of them :

**RLE**

Run-length encoding (RLE) is a very simple form of lossless data compression in which runs of data (that is, sequences in which the same data value occurs in many consecutive data elements) are stored as a single data value and count, rather than as the original run.

**Deflate**

Deflate is a lossless data compression algorithm and associated file format that uses a combination of the LZ77 algorithm and Huffman coding. There are three modes of compression that the compressor has available:

1. Not compressed at all. This is an intelligent choice for data that's already been compressed. Data stored in this mode will expand slightly, but not by as much as it would if it were already compressed and one of the other compression methods was tried upon it.

2. Compression, first with LZ77 and then with Huffman coding. The trees that are used to compress in this mode are defined by the Deflate specification itself, and so no extra space needs to be taken to store those trees.

3. Compression, first with LZ77 and then with Huffman coding with trees that the compressor creates and stores along with the data.

The data is broken up in "blocks," and each block uses a single mode of compression.

#### JPEG Lossless

Lossless JPEG is a 1993 addition to JPEG standard by the Joint Photographic Experts Group to enable lossless compression. It uses a predictive scheme based on the three nearest (causal) neighbors (upper, left, and upper-left), and entropy coding is used on the prediction error. Lossless JPEG is actually a mode of operation of JPEG. This mode exists because the discrete cosine transform (DCT) based form cannot guarantee that encoder input would exactly match decoder output. Unlike the lossy mode which is based on the DCT, the lossless coding process employs a simple predictive coding model called differential pulse code modulation (DPCM). This is a model in which predictions of the sample values are estimated from the neighboring samples that are already coded in the image. Most predictors take the average of the samples immediately above and to the left of the target sample. DPCM encodes the differences between the predicted samples instead of encoding each sample independently. The differences from one sample to the next are usually close to zero.

#### JPEG 2000

JPEG 2000 is an image compression standard and coding system created by the Joint Photographic Experts Group committee in 2000 with the intention of superseding their original discrete cosine transform-based JPEG standard (created in 1992) with a newly designed, wavelet-based method. Like the Lossless JPEG standard, the JPEG 2000 standard provides both lossless and lossy compression in a single compression architecture. Lossless compression is provided by the use of a reversible integer wavelet transform in JPEG 2000. We used the lossless version of JPEG 2000. JPEG 2000 is a much better image solution than the original JPEG file format. Using a sophisticated encoding method, JPEG 2000 can compress files with much better ratio.

### 5.1.2 A Brief Description of GDCM

Whenever medical data, especially medical image data, is generated in a clinical environment, that data must be stored such that it can be retrieved by the same hospital either immediately, or after several years to determine the effectiveness of a course of treatment and to allow comparisons of multiple images for the same patient.

**Grassroots DICOM (GDCM)** is an implementation of the DICOM standard designed to be

open source so that researchers may access clinical data directly. GDCM includes a file format definition and a network communications protocol, both of which should be extended to provide a full set of tools for a researcher or small medical imaging vendor to interface with an existing medical database.

GDCM is an open source implementation of the DICOM standard. It offers some compatibility with ACR-NEMA 1.0 & 2.0 files (raw files). It is written in C++ and offers wrapping to the following target languages

- Python
- C#
- Java
- PHP
- Perl

It attempts to support all possible DICOM image encodings, namely:

- RAW
- JPEG lossy 8 & 12 bits
- JPEG lossless 8-16 bits
- JPEG 2000 reversible & irreversible
- RLE
- Deflated
- JPEG-LS

### 5.1.3 GDCM Commands

We mainly used GDCM to compare various compressing algorithms of DICOM files. GDCM came very handy in order to implement variuos compressing algoritms. We just need to execute correct command to implement an algorithm. We can also extract useful informations of any DICOM file just by command using GDCM.

Here is the short description of GDCM commands we used:

1. **gdcminfo:** The gdcminfo command line program takes a DICOM file as input, or a directory and process it to extract meta- information about the DICOM file processed. We used this command to get the transfer syntax of a DICOM file.

   **SYNOPSIS:**
   gdcminfo [options] file-in

   **PARAMETERS:**
   file-in DICOM input filename

   **OPTIONS:**
   -r - -recursive recursive.
   -d - -check-deflated check if file is proper deflated syntax.
   - -resources-path Resources path.

2. **gdcmconv:** The gdcmconv command line program takes as input a DICOM file (file-in) and process it to generate an output DICOM file (file-out). The command line option dictate the type of operation(s) gdcmconv will use to generate the output file.

   **SYNOPSIS:**
   gdcminfo [options] file-in file-out

   **PARAMETERS:**
   file-in DICOM input filename file-out DICOM output filename

   **OPTIONS:**
   -X - -explicit Change Transfer Syntax to explicit.
   -M - -implicit Change Transfer Syntax to implicit.
   -w - -raw Decompress image.
   -d - -deflated Compress using deflated (gzip).
   -J - -jpeg Compress image in JPEG.
   -K - -j2k Compress image in J2K.
   -L - -jpegls Compress image in JPEG-LS.
   -R - -rle Compress image in RLE (lossless only).

### 5.1.4 Conversion Procedure

We used GDCM tool to compress DICOM files. After decompressing, the transfer syntax of the decompressed file will be the default transfer syntax for DICOM, which is Implicit VR Endian. So, we need to go back to the original transfer syntax. Hence a brief description on what transfer syntax is.

**Transfer Syntax**

A Transfer Syntax is a set of encoding rules to unambiguously represent one or more Abstract Syntaxes. It is actually an attribute which represent the file format of DICOM. Some DICOM transfer syntaxes are given below:

| Transfer Syntax UID | Transfer Syntax name |
|---|---|
| 1.2.840.10008.1.2 | Implicit VR Endian: Default Transfer Syntax for DICOM |
| 1.2.840.10008.1.2.1 | Explicit VR Little Endian |
| 1.2.840.10008.1.2.1.99 | Deflated Explicit VR Little Endian |
| 1.2.840.10008.1.2.2 | Explicit VR Big Endian |
| 1.2.840.10008.1.2.4.80 | JPEG-LS Lossless Image Compression |
| 1.2.840.10008.1.2.5 | RLE Lossless |
| 1.2.840.10008.1.2.4.91 | JPEG 2000 Image Compression |

**Compression and decompression algorithm**

Here is the Compression and decompression algorithm.

**Algorithm 1** Compression

**Input :** Dicom file $f$
**Output:** Compressed DICOM file
Transfer_sequence ← Transfer syntax of input DICOM file
Compress using any algorithm
**return** Compressed DICOM file

**Algorithm 2** Decompression

**Input :** Compressed Dicom file $f$
**Output:** Decompressed DICOM file
Decompress input file using the same algorithm used during compressing.
Convert the decompressed DICOM file's transfer syntax to the transfer syntax saved during compressing.
**return** Decompressed DICOM file

### 5.1.5 Result

We had a colorful data-set of various DICOM files. We collected around 100 DICOM files from various sources. They covered various types of medical images including:

- CT (computed tomography)
- MRI (magnetic resonance imaging)

- Ultrasound
- X-ray
- Fluoroscopy
- Angiography
- Endoscopy
- Microscop
- PET (positron emission tomography)

These files were of various sizes from few kilobytes upto 45 Megabytes.We simulated RLE, DEFLATE, JPEG LS and JPEG 2000 compression algorithm on these files. Among these JPEG 2000 gave the best compression ratio. Here is a graph of part of result of our simulation:

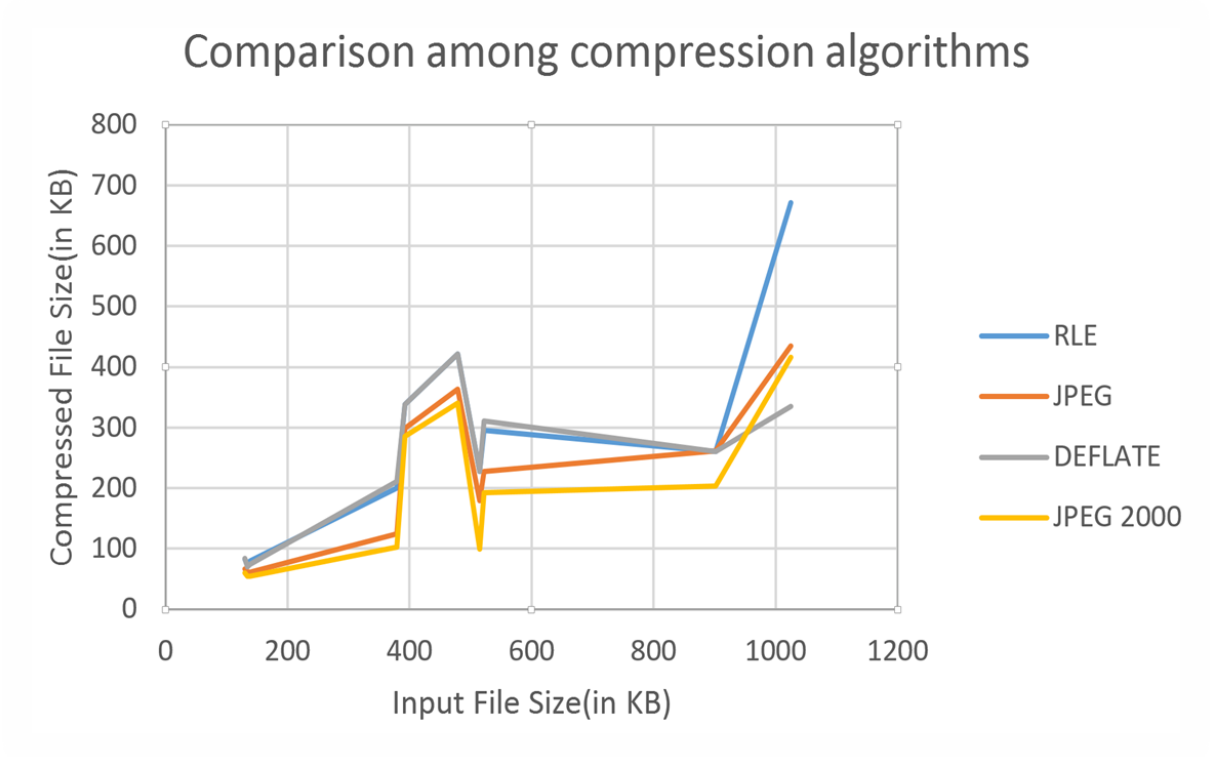



Figure 5.1: Comparison among compression algorithms

# Chapter 6

# Binary stream to nucleotide sequences

After compressing the medical files, they are converted into a continuous binary bit-stream. In DNA storage concept, the next work is to convert this binary bit-stream to DNA bit-stream. That means we need a method that converts a sequence X of {0,1}* to a sequence Y of {A,T,C,G}*. It can be also denoted as f(X) = Y. Suppose, the length of X is p and the length of Y is q. Then for all p, there will be a q. And our goal is to find the f such that for any X of length p, it minimizes the q. Also, the conversion must be completely reversible.

## 6.1 Previous Methods

Hence we discuss the existing methods of conversion between binary stream to nucleotide stream. From the history of DNA storage, we see that very few work has been done about this topic.

### Numerical/Usual Encoding

This is the most usual way of conversion. To map each of the different permutations of length 2 generated from 0,1* to any one element of the A,T,C,G set. There can be 4! or 24 different combinations for this sceme. One may be as follows :

| DNA bit | Binary |
|---|---|
| A | 00 |
| T | 01 |
| C | 10 |
| G | 11 |

Table 6.1: Numeral encoding of binary to {A, T, C, G}

Hence, a sequence like 01101100 will be converted to 'TCGA' $(01 \rightarrow T, 10 \rightarrow C, 11 \rightarrow G, 00 \rightarrow A)$
In this conversion, the value of q is fixed with respect to p and q = $\frac{p}{2}$.

### Phase-Change Encoding

In phase-change coding, we rely on the run-length of consecutive 0's and 1's while conversion. This was used in microvenus coding. Hence each base of A,T,C,G is used to indicate how many times each binary bit (e.g., zero or one) is to be repeated before changing to the other binary bits. This technique is used in many forms of computer-compression technologies. This can be represented as: C = X, T = XX, A = XXX, G = XXXX where X is 0 or 1. The first five digits of the Microvenus binary code were 10101. Using the phase-change coding method, corresponding genetic characters would be CCCCC, because each binary digit occurs only once before switching to the other digit. The next part of the bit-mapped Microvenus read was 0111000 so again, the first digit, 0, translated to the genetic C (0 occurs just once before switching to 1). Then 1 occured three times so these three binary digits could be represented by a single genetic A. The next change in the binary map was another triplet, 000, so again, a single A was used in the corresponding genetic sequence. The binary sequences 11 and 00 were signified by the genetic T; and G signified either 1111 or 0000. Coded in this way, the thirtyfive-digit Microvenus binary map translates to only eighteen DNA bases: "CCCCCCAACGCGCGCGCT". These could be then decoded into one of two binary codes: 10101011100010000100001000010000100 or 01010100011101111011110111011111011 However this ambiguity didn't raise any problem for microvenus icon as it was bilaterally symmetrical. But in general, we need to express explicitly whether the sequence starts with 0 or 1. Also, problem arises when the run of 0 or 1 is greater than 4. So, in that case, after each G, we need to explicitly express whether the next bit is same as previous or it changes. We can do that by inserting another bit after each 'G' to indicate bit change. However, in this conversion, the value of q is not fixed with respect to p and depends on X. However, for medical image files, this method doesn't work well and we achieve q of less value with numerical/usual way than this way.

## 6.2 Our Approaches

However, we tried with different approaches for converting binary bitstream to nucleotide sequences. Among them mentionable are: a) Quaternary Huffman tree b) Run-length encoding. However, none of them gave better compression for medical files than numerical usual encoding. Later, we came up with an approach that is a mixed form of run-length encoding and numerical encoding and found that approach to work better for most of the medical image files.

1. Quaternary Huffman
2. Run-length encoding
3. Specialized encoding for medical files

### 6.2.1 Quarternary Huffman

The prime approach to store digital data in DNA is to encode binary data to DNA sequence of four nucleotides. There exists many methods to encode binary (base 2) data to quaternary (base 4) data. The naive way of encoding of binary data to quaternary data is to represent four 2 bit data to four quaternary digit. This process produce a string of quaternary digit of length n/2 from binary string of length n. Then quaternary digits are mapped to DNA nucleotides. Some methods encode binary data to ternary data and map these digits to three nucleotides of DNA. The fourth nucleotide is used as a control digit. This process has some extra gain of control and error reduction due to homopolymers repetitions(?). As base 3 is not multiple of base 2 (m3 != n2) there is some waste of possible states. Another procedure to encode binary string to nucleotide sequence is Huffman code with rotating encoding[1]. Huffman code used to translate binary data to ternary data and the rotating encoding translate ternary digits to four nucleotides.

At the dawn of our work, proposed a encoding technique using quaternary huffman tree concept[2][3]. Here we considered a file (specially a dicom file) as ASCII byte stream. A dicom is consist of some images, patient data and some other metadata. The images are normally BW image(duran, define the image), hence the frequency of some pixels must be higher than other pixels. So, if we consider the the whole dicom file as a byte stream, frequency of some byte might be dominating. Hence we can take the advantage of the fact and assign a shorter code to the frequent characters. The naive technique assign a sequence of four nucleotides for each ASCII character and were supposed to get a variable length coding with average average length less than 4.

This version of quaternary huffman coding assign DNA digits/nucleotide in such a way that length of DNA sequence code depends on relative frequency of a character. The DNA code is derived from a full quaternary or 4-ary tree, a special form of Huffman tree. Here each node has 0 4 children. We labeled them as LEFT child, LEFT MID child, RIGHT MID child, RIGHT child. The four edges correspond to these child are mapped with four nucleotide bases, A, T, C and G from left to right as shown in figure[?]. Each leaf node correspond to a ASCII character and labeled by this character. The node is weighted by the frequency or weight of the the character. Here the goal is to construct a tree with the minimum external path weight. This a quaternary tree with the minimum sum of weighted path lengths for the given set of leaves. The weighted path length of a leaf is defined as production of its weight and depth. . As a

consequence, a character with high weight should have low depth. On the contrary a character might be pushed deeper in the tree if it has less weight. The outline of creating a quaternary huffman tree is as follows: List all the possible characters with their frequencies. Create a forest of n initial huffman quaternary tree where each tree is a single node tree and correspond to a single character. Put the trees onto a priority priority queue organised by total weight of the tree. Remove four trees with smallest weights from the queue , make a new node and join these tree with new node to make a new tree. The new node is the root of the tree and its weight is the sum of the weights of the four trees. Put the newly created tree back in the queue. Continue the process until the queue has only one tree.

An example elaborating this process is shown in Table 6.2.

| Character | Frequency | Code |
|---|---|---|
| Space | 5 | TT |
| T | 5 | TA |
| E | 5 | AG |
| N | 4 | AG |
| A | 4 | AA |
| S | 4 | TGG |
| R | 3 | TGC |
| I | 3 | TGT |
| D | 3 | TGA |
| M | 2 | TCG |
| C | 2 | TCC |
| U | 1 | TCT |
| B | 1 | TCA |
| H | 1 | ACG |
| O | 1 | ACC |
| W | 1 | ACT |
| . (dot) | 1 | ACA |

Table 6.2: Generation of Codeword using Quaternary Huffman

The codeword for a symbol can be determined by traversing the tree. Let us consider a symbol si with frequency fi. We traverse the tree from the root of the tree to the leaf node corresponding to the symbol si. Let ci be the codeword for the symbol si. Before traversing the tree, ci is kept empty string. Each time we go through a edge of the tree, the label of the edge is added to ci.

Let us consider a set of characters C = $\{c_i \epsilon A : 0 <= i <= n\text{-}1\} = \{Space, T, E, N, ..\}$. Each character has a frequency or weight $w_i$.

This method does compression and encoding simultaneously. But this has some problems. The tree generated by this algorithm is not a balanced a tree. It does not ensure enough compression as does a balance tree. The ASCII byte stream found in a dicom file is equal probabilistic , i.e., all character has almost equal frequencies. Hence huffman compression did not worked well in

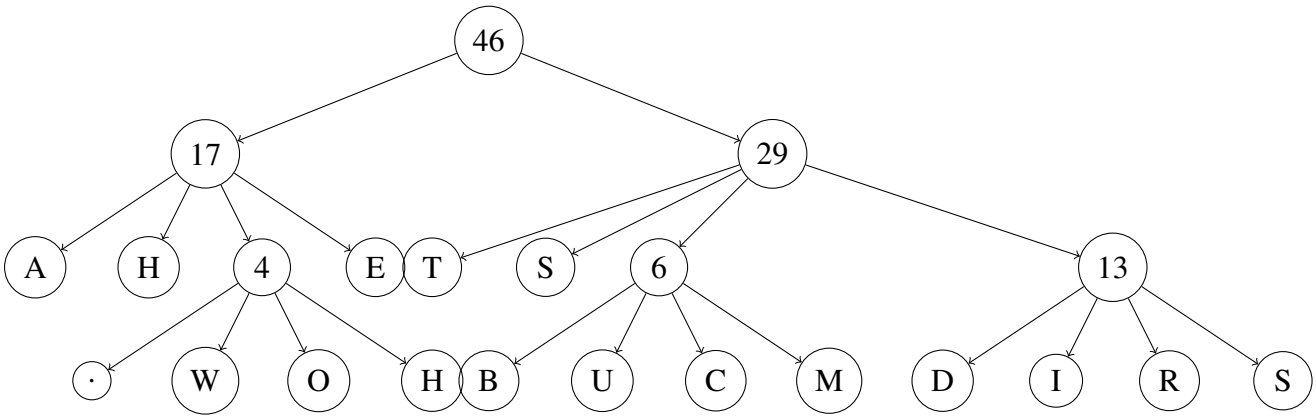



Figure 6.1: Construction of quaternary Huffman tree

this data.

### 6.2.2 Run-length encoding

In run-length encoding we made a variant of phase-change encoding. Hence, we use two different DNA bits from a set $\alpha \subset \Delta$ = {A,T,C,G} for denoting whether it is a run of 0 or run of 1. Then we convert the run-length into binary form and then assign each binary bit to a different value from set $\beta = \Delta$ - $\alpha$ . In our code, $\alpha$ = {A,G} and $\beta$ = {C,T}.

**Algorithm 3** Encoding using Run-length

```
ATCGSequence ← ϵ
while not end of bitstream do
    differBit ← read(bitstream)
    if differBit = '0' then
        ATCGSequence ← ATCGSequence + 'A'
    else
        ATCGSequence ← ATCGSequence + 'G'
    end
    nextBit ← differBit
    count ← 0
    while nextBit = differBit do
        nextBit ← read(bitstream)
        count ← count + 1
    end
    binaryCount ← base2conversion (count)
    read binaryCount char by char
    if char = '0' then
        ATCGSequence ← ATCGSequence + 'C'
    else
        ATCGSequence ← ATCGSequence + 'T'
    end
end
return ATCGSequence
```

**Algorithm 4** Decoding using Run-length

```
\* map(char c) converts 'A'or 'C' to 0 and 'G' or 'T' to 1 *\
bitStream ← ϵ
while not end of ATCGSequence do
    char ← read(ATCGSequence)
    nextChar ← readNext(ATCGSequence)
    run-lengthCode ← ϵ
    while nextChar ϵ set β do
        run-lengthCode ← map(nextChar)
        nextChar ← readNext(ATCGSequence)
    end
    run-length ← base10conversion (run-lengthCode)
    for i : 1 to run-lenth do
        bitStream ← bitStream + map(char)
    end
end
convert bitStream to file
return file
```

### 6.2.3 Specialized encoding for medical files

In the specialized encoding technique for medical files, we tried to utilize the frequent large run of 0s in the converted binary stream. The large run of 0's are obvious in DICOM files as usually in most of them, most of the pixels have gray scale values of 0 even after enduring lossless compression. Four sample histograms are provided for four compressed DICOM files. 6.2 The zero value skewed distribution proves the fact. So, we thought of utilizing this special property of dicom files by converting this sequences of long 0s using run-length encoding and others according to numerical encoding. But this also has a problem. Suppose, we express the length of the run using {C,T,G}* and then surround it with prefix and suffix of As at both end. But hence, the minimum number of As can be two which can be assigned for run of 0s of length 2 $(00 \rightarrow AA)$. But in usual way we needed only one A. So, for 2 0s the length was increased $\frac{2}{1}$ or 2 times and for 4 0s the length was increased $\frac{3}{2}$ or 1.5 times, for 6 0s the length was same as usual 3 DNA bits for runs of more than 6, the length was decreased than usual way and as the run increases the compression rate is much better. So, for runs of larger lengths the corresponding sequence will be much shorter than the usual way but for runs of short length the corresponding lengths gets more than in usual way. We analyzed the frequency of various run lengths of 0s in the files of dataset and the result was nearly similar. The runs of 0s varied from 2 to 1024 or more. But frequency of short runs were much greater than of long runs. Thats why if we choose to do special encoding for runs of all lengths results become much poorer than usual way for all files in our selected dataset.

We experimented with 30 different dicom files including colorful dicoms. The lower limit of

run-length for all dicom files were 2 and the upper limit was variable among them ( some had lengths upto 6784 or more). A single run of length 1024 was common at the beginning of all dicom files. This is due to the 128 dicom preamble defined for application specific purpose. [30]

After analyzing the frequency chart, we decided to do special encodings for runs greater than or equal some fixed amount rather than starting it from 2. It can be said we decided to shift the run-length by a fixed amount (shift). That means, the encoding style used for 2 will now be used for shift+2 , 4 will now be shift + 4 and so on.

However, for converting the run length to DNA bits, a common way was to convert to the length/2 (as we are working with pairs and there are always even number of bits in binary bitstream) in base 3 number and then map the 0 of base 3 number to T, the 1 of base 3 number to C and the 2 of base 3 number to G. But hence the 0(T), 00 (TT), 01 (TC) ,02 (TG), 000(TTT), 010 (TCT) etc. are never generated, but we couldve used them. In general, for n bits we should be able to use $3^n$ permutations, but if we do base 3 conversion directly, then well be able to use $2 * 3^{(n-1)}$ permutations. Thats why we developed a new conversion that is slightly different from base 3 encoding. We used usual way for runs of less than length of shift + 2 and for runs of length shift + 2 or more, we used a prefix of $\frac{(shift+2)}{2}$ number of consecutive As which is common and was only 1 A as suffix or as end marker for length bits. For length of shift + 2, no bits were used between prefix and suffix. Hence in base 3 system, we needed (1)10 = (1)3 = 1 bit to between the prefix and suffix which is $\frac{(shift+2)}{2}$ + 2 DNA bits in total. In our conversion we used $\frac{(shift+2)}{2}$ + 1 DNA bits. So, we are saving 20% in this case which is significant as the frequency of run of shift+2 0s is most among the runs of length greater or equal to shift+2. However, for run of length shift+4, we used 1 bit (T) in between, for run of shift+6, we used C, for shift+8, we used G. For run of length shift+10, we needed 2 bits (TT). From runs of length shift+10 to shift+26 we needed only 2 bits in between. From runs of length shift+28 to shift+80, we needed only 3 bits in between. Thus we find a common pattern, for shift + $3^n$ + 1 to shift+ $3^{(n+1)} - 1$ bits we need n bits between the suffix and prefix. Below is a table for those bits and their corresponding run-length meaning up to some extent (DNA bits are excluding prefix and suffix):

Suppose, the shift equals 6. Then, the encoding style used for 2 will now be used for shift + 2 or (6+2) or 8. In usual way, for 8 0s we needed 4 As. In our way, for 8 0s we needed 5 As so hence we increased the length of the sequence by $\frac{5}{4}$ or 1.25 which is much less than the increment for 2 0s. For 10 0s we needed 5 As with 1 T and in total 6 bits where we needed only 5 As in usual way. So, hence we increased size by $\frac{6}{5}$ or 1.2. For 12 0s the converted sequence size is same as the original one (6 bits) and later the size was much decreased in our special encoding. And using this technique the size of {A, T, C, G}* sequence was much lessened for most of the compressed dicom files in our dataset.

**Algorithm 5** Encoding Algorithm

**Input:**Compressed Dicom File $f$ and Shift amount $s$ **Output:**ATCG Sequence

$ATCG\_sequence \leftarrow \epsilon$
convert the compressed dicom into bitstreams
read bitstream by 2 bits at once
**if** *two bits is '01' or '10' or '11'* **then**
  $ATCG\_sequence \leftarrow ATCG\_sequence + numericalEncoding(two\_bits)$
**else**
  calculate the run_length of 0s
  **if** *run_length* $< (s + 2)$ **then**
    **for** *each* $i$ *from* $1$ *to* $run\_len/2$ **do**
      $ATCG\_sequence \leftarrow ATCG\_sequence + numericalEncoding(two\_bits)$
    **end**
  **else**
    $prefix \leftarrow (s + 2)/2$ number of consecutive A's
    $suffix \leftarrow$ "$A$"
    $length\_bits \leftarrow \epsilon$
    **if** *run_length* $> s + 2$ **then**
      find '$len$' such that $3^n + 1 \leq len \leq 3^{n+1} - 1$
      $value \leftarrow (run\_length - s - 3^{len} - 1)/2$
      $value \leftarrow convertToBase3(value)$
      **for** $i$ *from* $1$ *to* $len - 1$ **do**
        $length\_bits \leftarrow length\_bits + EncoderMap(i^{th} bit of value)$
      **end**
    **end**
    $converted\_sequence \leftarrow prefix +$ "$length\_bits$" $+ suffix$ $ATCG\_sequence \leftarrow ATCG\_sequence + converted\_sequence$
  **end**
**end**
**return** ATCG_sequence

**Algorithm 6** Decoding Algorithm

**Input:**ATCG_sequence $ATCG\_seq$ and Shift amount $s$
**Output:**Dicom file $f$

$bitstream \leftarrow \epsilon$
read ATCG_sequence character by character
**if** *character is 'T' or 'C' or 'G'* **then**
  $bitstream \leftarrow bitstream + NumericalDecoding(character)$
**else**
  count the number of consecutive A's
**end**
**if** *count* $\leq s/2$ **then**
  add count number of '00's to bitstream
**else if** *count* $= (s+2)/2 + 1$ **then**
  add $(s+2)/2$ number of '00's to bitstream
**else**
  $specialEncodedString \leftarrow \epsilon$
  **while** *next 'A' is not found* **do**
    read character by character
    concatenate each character to specialEncodedString
  **end**
  $len \leftarrow lengthofspecialEncodedString$
  $base \leftarrow 3^{len} + 1$
  $numberofExtraZeros \leftarrow 0$
  **for** *each* $i$ *from* $len - 1$ *to 0* **do**
    $numberofExtraZeros \leftarrow numberofExtraZeros$ + $DecoderMap$($i^{th}$ character of specialEncodedString)$*(3^{len-1-i})$
  **end**
  $numberofExtraZeros \leftarrow numberofExtraZeros \times 2$ + base + s
  add numberofExtraZeros/2 number of '00' to bitstream
**end**
convert the bitstream to Dicom file
**return** Dicom file

| DNA Bits required | Run-length - *shift* | DNA Bits required | Run-length - *shift* |
|---|---|---|---|
| - | 2 | TTT | 28 |
| T | 4 | TTC | 30 |
| C | 6 | TTG | 32 |
| G | 8 | TCT | 34 |
| TT | 10 | TCC | 36 |
| TC | 12 | TCG | 38 |
| TG | 14 | TGT | 40 |
| CT | 16 | TGC | 42 |
| CC | 18 | TGG | 44 |
| CG | 20 | CTT | 46 |
| GT | 22 | CTC | 48 |
| GC | 24 | CTG | 50 |
| GG | 26 | ... | ... |

Table 6.3: DNA bit allocation in specilized encoding

## Improvement using this approach

We encoded the files using our special encoding methods and compared results with the usual encoding method. Without shifting the run-length, all dicom files gave poorer result than usual one (which is mentioned previously). However, with shifting (which is our final algorithm) we achieved better results than usual encoding for most of the dicom files. We plotted a bar diagram for length of encoded DNA Sequence vs value of shift equaling to 4,6,8,10 and 12 and also without shift ( Naive way). The bar diagram shows average over 75 dicom files with high standard deviation in size. 6.4

We achieved best results for shift equaling to 8 (though for shift equaling 10, results were better for files larger than 500KB but that was still poorer than usual encoding and it was also poorer than shift equaling 8 for smaller files which is more in number in our dataset). As for most dicoms in our dataset, shift equaling 8 gave best results, so we used shift = 8 in our final software.

### Reason of the diversified result

From 6.5 it is evident that for most files specialized encoding works better than the usual one ( q - $\frac{p}{2}$ >0). For some files, specialized works same as the usual one ( q - $\frac{p}{2}$ = 0). And also for some files it works poorer than the usual one ( q - $\frac{p}{2}$ < 0). The reason for this diversity lies in the structure of DICOM files.They mainly consists of two parts :-

1. Meta Data
2. Pixel Data

The pixel Data is the image part and it contains the large runs of 0's. The Meta data is like text part and doesn't contain large runs of 0's. So, the specialized encoding doesn't work better for

meta data part. Usually large file sizes contains a large portion of meta data and thus, specialized encoding doesn't minimize q more than $\frac{p}{2}$ for large dicom files. However, digging more into this and trying to ensure an encoding that must always work better than the usual encoding is out of the scope of this thesis. As, in our datset, we get better results for shift = 8 on average, so we used this sheme in our final software. That's why we can observe that for files larger than 10x50KB or 500KB, usual encoding is still better. But for less than that, our encoding give better results (for most of the files) or same result (for some of the files). It is a good thing for us as usually most compressed dicom files are of length less than 500KB.

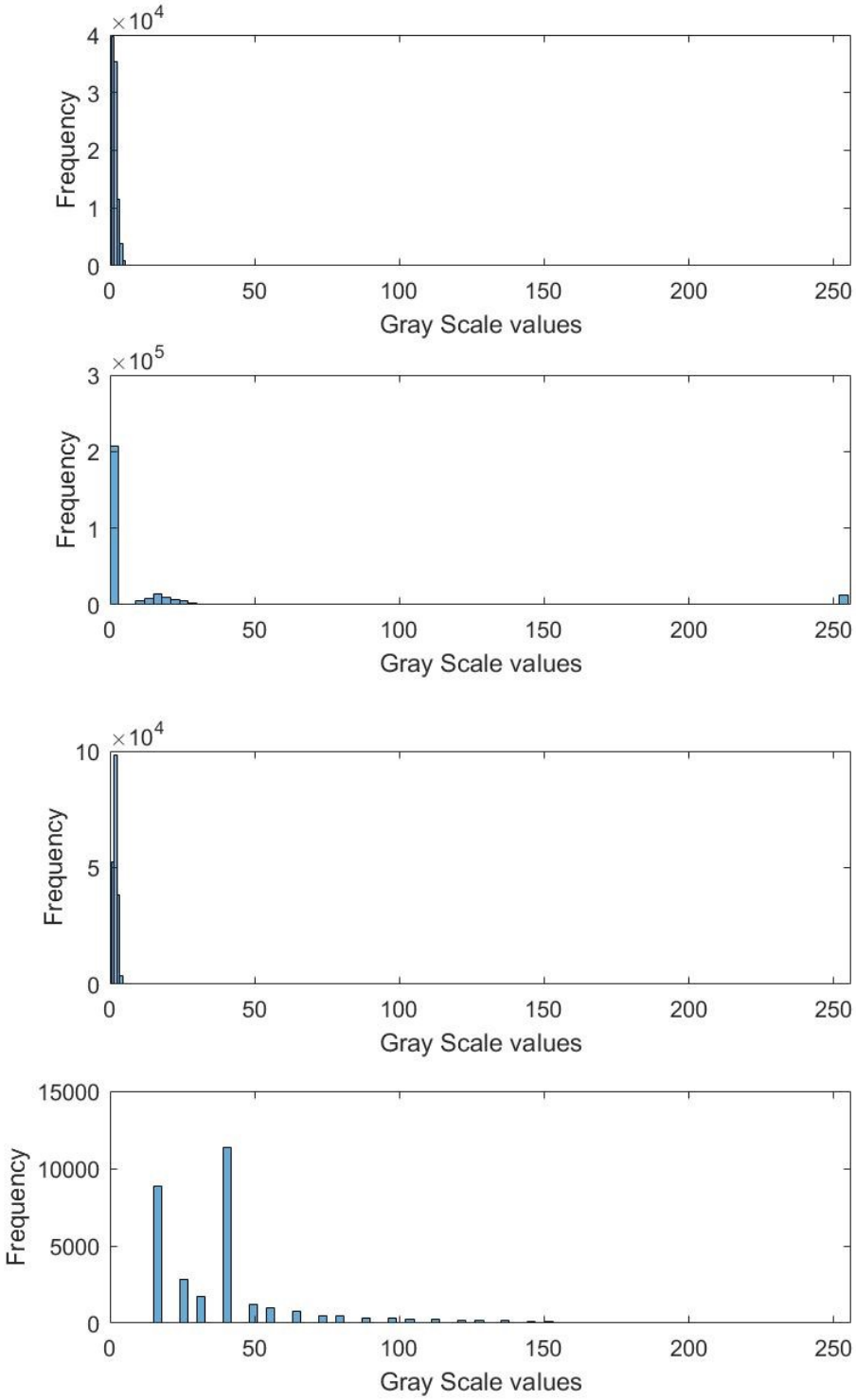



Figure 6.2: Histograms showing the frequency of pixel gray scale intensity values for four sample compressed Dicom files

Two of the files are RGB and two are monochrome. These files are available at: https://goo.gl/ruzozJ. All of them show right-skewed distribution. For RGB dicoms, one of the three slices was taken for histogram. The density of 0th value is much higher which reflects the presence of higher frequency or long runs of 0 in the binary stream of dicom files. This is due to the larger darker portion in dicom files which have grayscale value of 0, i.e., RGB value (0,0,0).

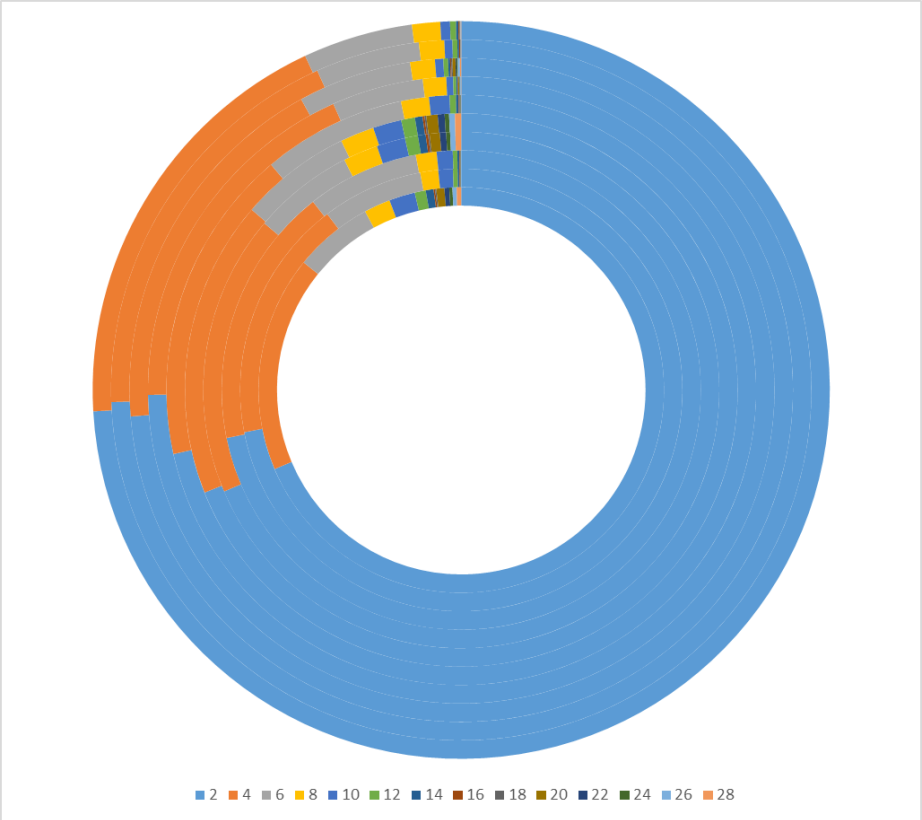



Figure 6.3: Run-length of 0s vs frequency Doughnut Chart
The chart is shown for only 10 files as it is explicit from only 10 of them, that the frequency of short run of 0's is much more greater than long run of 0's. The maximum run-length is limited to 50 in this graph for better view. For run-length of more than 50, the frequency is usually less than 10. Each circle depicts a different dicom file and the labels on bottom refer the run of consecutive 0's.

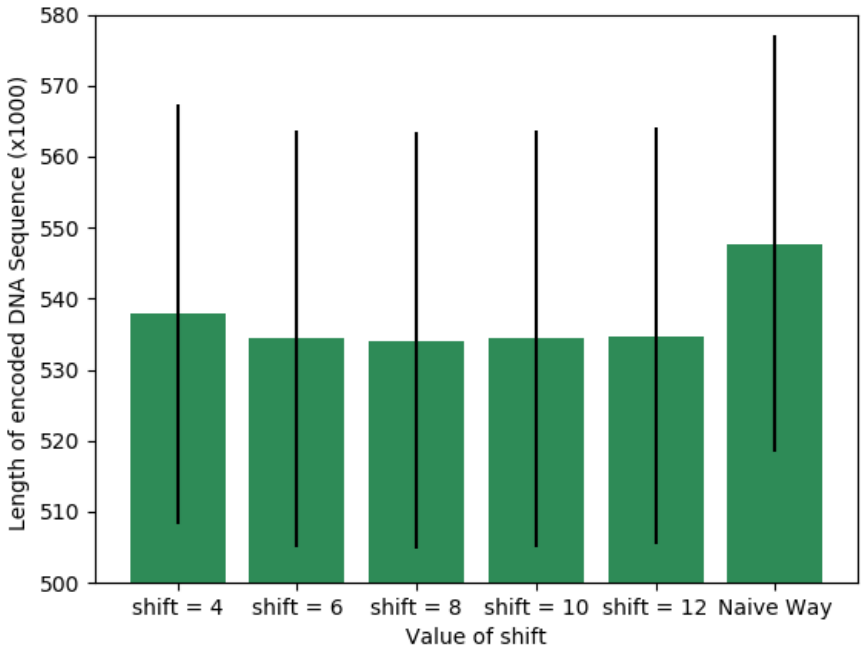



Figure 6.4: Comparison of our encoding technique with various shift values and the nave encoding (without shift parameter).

We show the average length of the encoded DNA base stream (over 75 dicom files) with different values of shift (4, 6,8,10 and 12), and in nave way (without shift). We observe that minimum average DNA base string length is obtained for shift = 8. In nave encoding, the performance is much worse than the performance with shift. The dataset consists of 75 Dicom files of a single patient which is available at: https://goo.gl/gMXJMa .

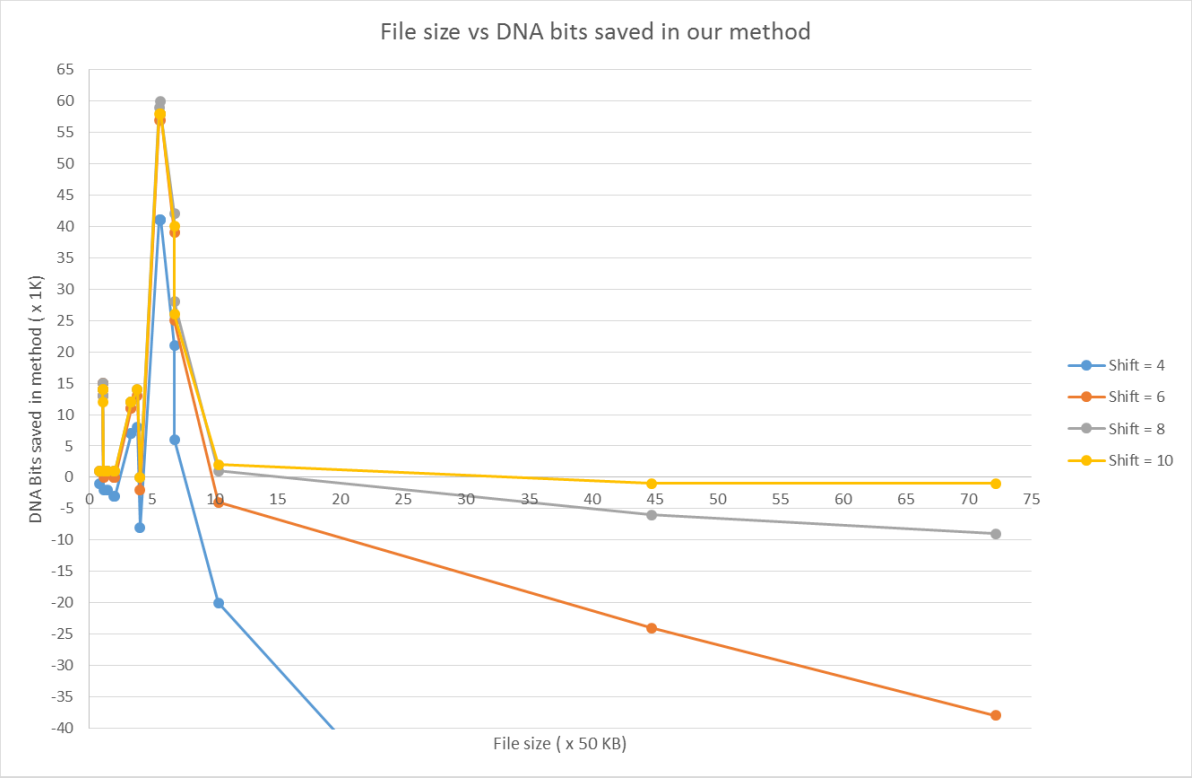



Figure 6.5: Comparison among results with different values of shift with different dicoms of different file sizes

# Chapter 7

# Nucleotide Sequence Compression

Once we have a nucleotide base sequence from EHR components, we target to reduce it into a sequence of shorter length. As human genome sequence has a fixed number of bases and size of EHR of a patient may practically expand throughout entire lifetime, we try to to gain format specific compression in several stages, to have a final base string of as small size as possible. In this stage, we explore numerous techniques of compression for different kinds of sequencing data. Raw and annotated sequencing data are nowadays the greatest challenge for storage and transfer.

## 7.1 Traditional Nucleotide Sequence Compression

The raw reads from sequencing are usually stored as records in textual FASTA format. General-purpose compression algorithms like gzip, bzip2 etc. are frequently used for efficient text storage and transferring, are directly applicable to genomic sequences. They do not take into account biological characteristics such as small alphabet size, rich in repeats and palindromes. Only when longer genomic sequences were obtained and the storage and transferring of these data become routine, specialized compression algorithms were developed. Compared with general-purpose text compression methods, these specialized algorithms better utilize repetitive subsequences inherent in genomic sequences to achieve higher compression ratio. Compression algorithms tailored for such properties are more likely to achieve better performance.

In a dictionary, highly repetitive subsequences are stored in these processes. Compression is achieved by replacing each repetitive subsequence in the target genomic sequence by the corresponding encoded subsequence in the dictionary. This representation has a much smaller size. The performances of different algorithms differ mainly based on how many repetitive subse-

quences can be identified and how efficiently they can be.

## General Purpose Compression Techniques

### Huffman Coding

Huffman Coding [31] is a lossless data compression algorithm. The idea is to assign variable-length codes to input characters, lengths of the assigned codes are based on the frequencies of corresponding characters. The most frequent character gets the smallest code and the least frequent character gets the largest code. The variable-length codes assigned to input characters are Prefix Codes, means the codes (bit sequences) are assigned in such a way that the code assigned to one character is not prefix of code assigned to any other character. This is how Huffman Coding makes sure that there is no ambiguity when decoding the generated bit stream.

### Burrows- Wheeler Transform

The Burrows-Wheeler transform (BWT, also called block-sorting compression) [32], is an algorithm used in data compression techniques such as bzip2. When a character string is transformed by the BWT, none of its characters change value. The transformation rearranges the order of the characters. If the original string had several substrings that occurred often, then the transformed string will have several places where a single character is repeated multiple times in a row. This is useful for compression, since it tends to be easy to compress a string that has runs of repeated characters by techniques such as run-length encoding.

### Run-length Encoding

Run-length encoding (RLE) is a simple lossless data compression method which replaces runs of data with a single count value and data symbol. It achieves much better compression if the original input stream is first passed thorugh BWT.

### Arithmetic Coding

Arithmetic coding [33]is a form of entropy encoding used in lossless data compression. When a string is converted to arithmetic encoding, frequently used characters will be stored with fewer bits and not-so-frequently occurring characters will be stored with more bits, resulting in fewer bits used in total. Arithmetic coding differs from other forms of entropy encoding, such as Huffman coding, in that rather than separating the input into component symbols and replacing each with a code, arithmetic coding encodes the entire message into a single number,

an arbitrary-precision fraction q where 0.0 q ¡ 1.0. It represents the current information as a range, defined by two numbers.

A slight variant form of Arithmetic Coding is range coding, in which the input stream of symbols encoding is done with digits in any base, instead of with bits, and so it is faster when using larger bases (e.g. a byte) at small cost in compression efficiency.

### Asymmetric Numeral Systems

Asymmetric Numeral Systems or ANS [34] coding is a hybrid of advantages of two principal general purpose data compression approaches to entropy coding: Huffman Coding and Arithmetic Coding. Huffman is much faster, but it cannot achieve theoretical limit for inputs in which symbols have probabilities of exact power of 2. For other combinations, it approximates actual probabilities with power of 2, which results in relatively low compression ratio. Arithmetic Coding uses nearly exact probabilities and easily approaches theoretical Shannon Entropy, but has mush larger computational cost.

ANS is a recent approach to accurate entropy coding which represents properties of input probability distribution with a considerably small table: defining entropy coding automaton. besides, a number of variances of this table can be employed. For 256 size alphabet, this table takes about a few kilobytes. ANS represents a state by a single natural number instead of a range as in Arithmetic Coding. This greatly improves compression time, about 50% for 256 size alphabet at the same time achieving compression ratio of Arithmetic Coding.

### Lempel-Ziv-Welch

LZW [35]is a data compression method that takes advantage of this repetition. The original version of the method [36] was created by Lempel and Ziv in 1978 (LZ78) and was further refined by Welch in 1984, hence the LZW acronym.LZW is a ”dictionary”-based compression algorithm. This means that instead of tabulating character counts and building trees (as for Huffman encoding), LZW encodes data by referencing a dictionary. Thus, to encode a substring, only a single code number, corresponding to that substring’s index in the dictionary, needs to be written to the output file. Although LZW is often explained in the context of compressing text files, it can be used on any type of file. However, it generally performs best on files with repeated substrings, such as text files.
These kinds of compressions are combined in various forms and are given various names which we use for general purposes, such as gzip (LempelZiv ”LZ” + Huffman) and bzip2 (BurrowsWheeler transform ”BWT” + Move-to-Front ”MTF” + Huffman).

## Genomic Sequence Specific Compression Techniques

### BioCompress

The first special purpose DNA compression algorithm is BioCompress [37] developed by Grumbach and Tahi in 1993.BioCompress is a method which uses Lempel-Ziv (LZ) substitutional algorithms. This compression technique compresses the exact repeats and palindromes.First, it detects exact and reverse complement repeats in the DNA and stores it using 4-ary tree, and then it encodes them by the repeat length and the position of a previous repeat occurrence. For non-repeat regions, it is encoded by 2 bpb. BioCompress-2 [38] is the upgraded version of BioCompress. The algorithm combines substitutional and statistical methods. It utilizes arithmetic encoding scheme to store the non-repetitive regions.An average compression ratio for Biocompress is of 1.850 bpb and for Biocompress 2 it is 1.783 bpb , compared to the general-purpose algorithms compact and compress, which used more than 2 bpb.

### GenCompress

GenCompress [39] is a one-pass algorithm. It proceeds as follows: For input w, assume that a part of it, say v, has already been compressed, and the remaining part is u, i.e. w = vu. GenCompress finds an optimal prefix of u such that it approximately matches some substring in v so that this prefix of u can be encoded economically. After outputting the code of this prefix, remove the prefix from u, and append it to the suffix of v. Continue the process till u = .

### DNACompress

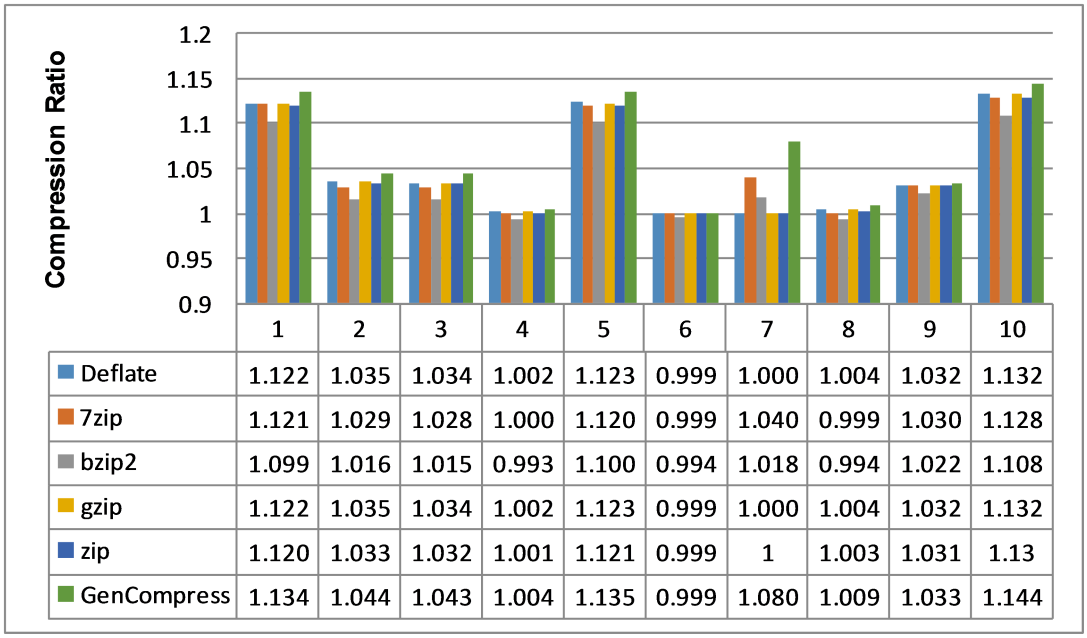
DNACompress [40] uses LZW compression scheme.Typically much faster than GenCompress. There are two phases:

- Find all approximate repeats including complementary palindromes.
- Encode approximate repeat regions and non-repeat regions.

DNACompress checks each repeat to see whether it saves bits to encode. If not, it will be discarded. At the end, all the remaining regions other than repeats are concatenated together and then sent as input to a two-order arithmetic coder.DNACompress uses almost the same encoding as GenCompress.

## 7.2 Our Proposal

Now we are in a stage of our pipeline where DICOM files have been passed through J2K lossless compression and Binary to ATCG conversion described in Chapter 6. In this stage, the compressed and encoded ATCG sequence is passed through various kinds of compression techniques. Figure 7.1 shows the comparison chart of compression ratios of those compression techniques for few files.



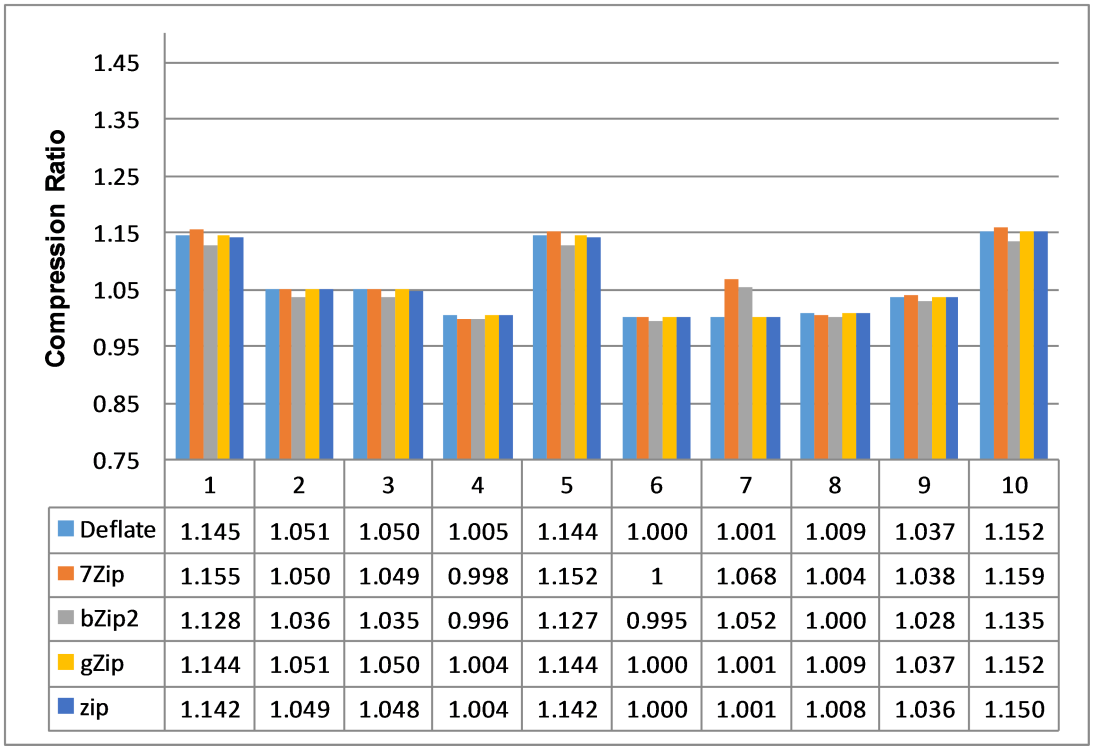

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| Deflate | 1.122 | 1.035 | 1.034 | 1.002 | 1.123 | 0.999 | 1.000 | 1.004 | 1.032 | 1.132 |
| 7zip | 1.121 | 1.029 | 1.028 | 1.000 | 1.120 | 0.999 | 1.040 | 0.999 | 1.030 | 1.128 |
| bzip2 | 1.099 | 1.016 | 1.015 | 0.993 | 1.100 | 0.994 | 1.018 | 0.994 | 1.022 | 1.108 |
| gzip | 1.122 | 1.035 | 1.034 | 1.002 | 1.123 | 0.999 | 1.000 | 1.004 | 1.032 | 1.132 |
| zip | 1.120 | 1.033 | 1.032 | 1.001 | 1.121 | 0.999 | 1 | 1.003 | 1.031 | 1.13 |
| GenCompress | 1.134 | 1.044 | 1.043 | 1.004 | 1.135 | 0.999 | 1.080 | 1.009 | 1.033 | 1.144 |

Figure 7.1: Comparison of compression ratio
The chart is shown for only 10 files. This shows the comparison of compression ratios of various compression techniques Binary to ATCG conversion

Among all of these techniques we find that GenCompress shows the best performance. The output of these processes are binary. As we have to store every files in the genome sequence, we need to convert these binary outputs to ATCG sequence through the process described in Chapter 6.

Figure 7.2 shows the comparison of compression ratios of the algorithms used in above description (except GenCompress) after J2K lossless and before Binary to ATCG conversion. All of these techniques take binary file as input and output binary files. Now, these output files are much more compressed than those that we get after using compression techniques on ATCG sequences.

Figure 7.3 shows the comparison of compression ratios for 30 DICOM files of all the compression methods we have trialed. We find that 7zip compression applied directly after J2K lossless compression gives the best of all of the others.




| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| Deflate | 1.145 | 1.051 | 1.050 | 1.005 | 1.144 | 1.000 | 1.001 | 1.009 | 1.037 | 1.152 |
| 7Zip | 1.155 | 1.050 | 1.049 | 0.998 | 1.152 | 1 | 1.068 | 1.004 | 1.038 | 1.159 |
| bZip2 | 1.128 | 1.036 | 1.035 | 0.996 | 1.127 | 0.995 | 1.052 | 1.000 | 1.028 | 1.135 |
| gZip | 1.144 | 1.051 | 1.050 | 1.004 | 1.144 | 1.000 | 1.001 | 1.009 | 1.037 | 1.152 |
| zip | 1.142 | 1.049 | 1.048 | 1.004 | 1.142 | 1.000 | 1.001 | 1.008 | 1.036 | 1.150 |

Figure 7.2: Comparison of compression ratio
The chart is shown for only 10 files. This shows the comparison of compression ratios of various compression techniques right after J2K lossless compression.

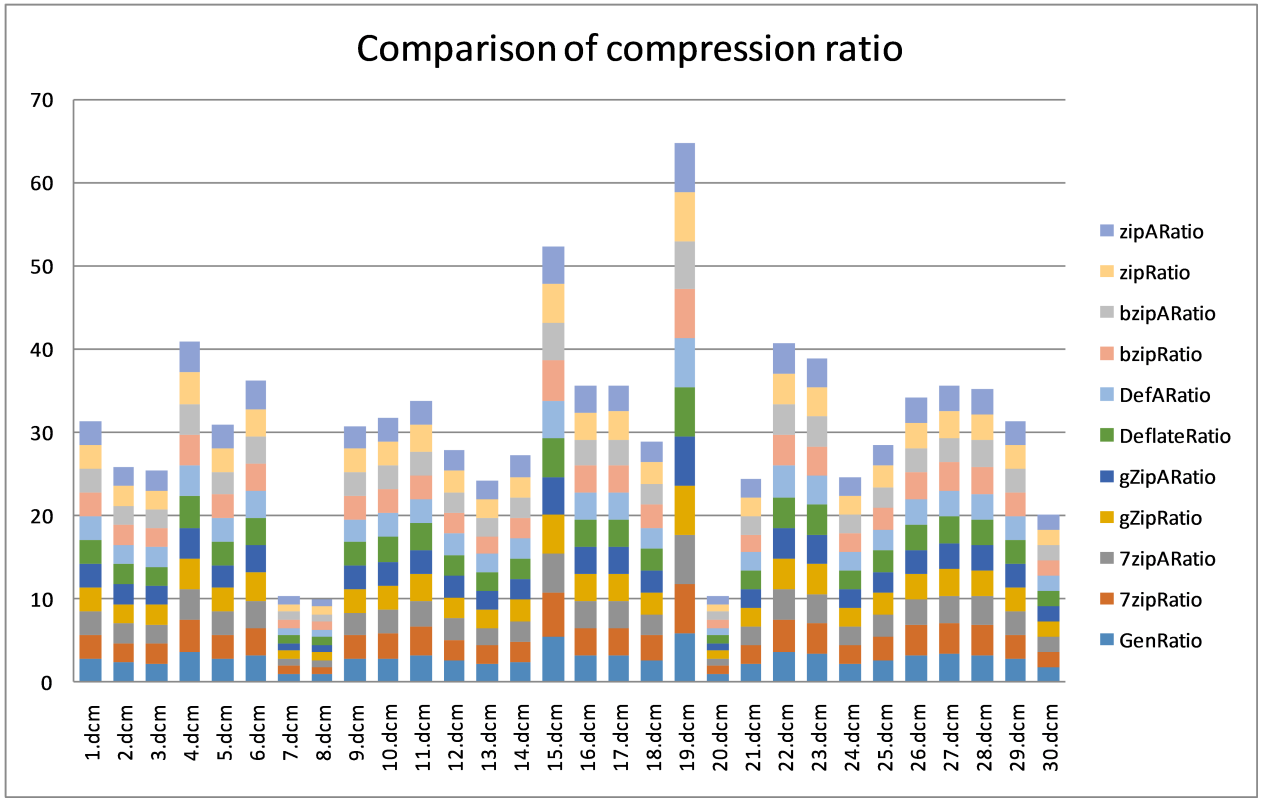



Figure 7.3: Comparison of compression ratio
The chart is shown for only 30 files. GenCompress and all other compression methods are done after Binary to ATCG conversion; all others except GenCompress are also applied before Binary to ATCG conversion. All of these are then compared to find out best compression ratio.

# Chapter 8

# File Management System

We have converted EHR files to base sequence using different stages of the pipeline so far. Next task is to store the acquired sequence data in genome. In this chapter, we discuss how we maintain this incorporation of external file data in genome in our system. For simplicity, we have handled a few basic file management operations in our implementation: insertion, deletion, retrieval. In this chapter, we explore these functionalities and different approaches and entities utilized in the system to facilitate the functionalities, in detail.

## 8.1 Abstract Continuous Space

Junk DNA locations in each chromosome are scattered in chunks of varying length in between coding DNA regions. When we manipulate base strings in order to store external information, which in our case are EHR components, bit stream of an EHR file is first transformed into nucleotide base stream using appropriate procedure, as discussed in previous chapters. In present scenario, we replace an available portion of junk DNA sequence with our generated sequence data from input files. For lossless retrieval of stored files, it is essential to keep track of where and how far nucleotide stream of a particular file has been placed. While storing a file, it can span several junk chunks of a chromosome fully or partially and occupy junk chunk from several chromosomes. Besides, as we will see later in this chapter, deletion of files can make situations further complicated. To maintain data location in genome for a file with separate chromosome numbers and associated chunk positions and lengths for each chromosome is a cumbersome process. To make the maintenance reasonably simpler and ease modeling of the functions of file management, we assemble junk locations from all chromosomes of a genome to form an abstract continuous space (Figure 8.1), which can completely be used for EHR storage.

Our system takes a whole human genome sequence in FASTA format, length of each chromosome in terms of base count and position and size of coding DNA regions as input. The time

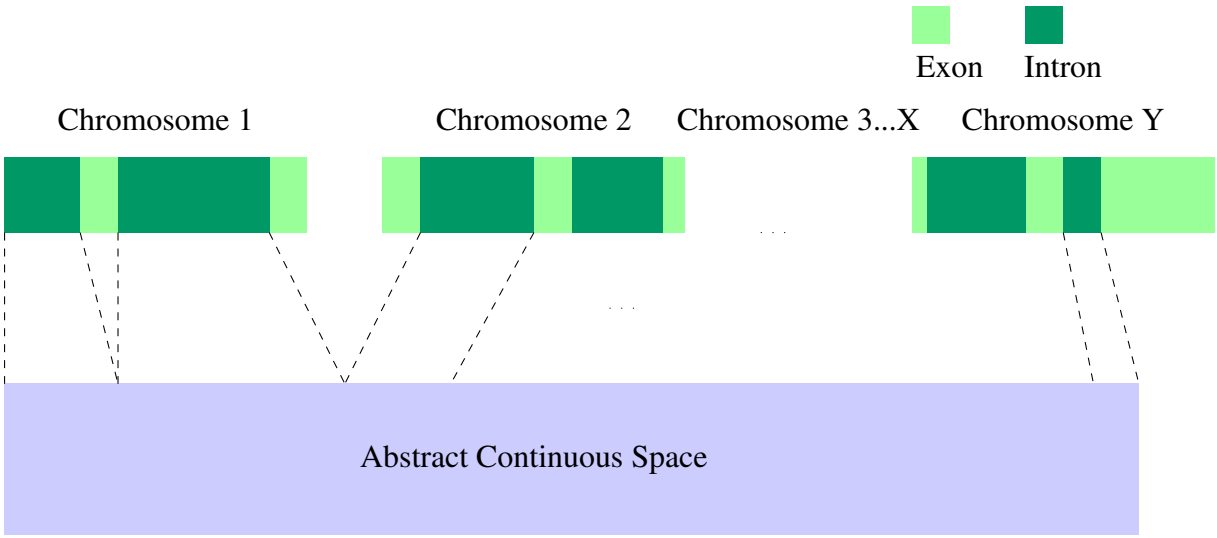



Figure 8.1: Abstract Continuous Space

span after introduction of a valid input set in the system throughout which the system remains actively open for that input set is referred to as a *user session* in this book. At the beginning of every user session, our system constructs the corresponding abstract continuous storage space and maps locations of abstract storage to junk regions of chromosomes. In this way, the complexity is handled in the beginning of a user session and throughout the session dealing with file operations is much relaxed instead of jumping into complicated calculations for every operation.

From the whole genome sequence, only the part represented by the abstract space is exploited for EHR storage. So from perspective of EHR storage, the abstract space is the only part we are functioning on and using as model for explaining various phenomena. In following discussions we use the broader terms *genome* or *genome space* to refer to this abstract continuous space. Also, whenever we mention about location or position in genome space from now on, it refers to the position according to abstract continuous linear space. While performing file operations, these abstract locations are mapped to actual chromosome positions inside the functions.

## 8.2 Dictionary

We maintain positions of sequence dataand different other information for each file stored in the genome, to ensure proper retrieval. Before we explain how the information are maintained, we notice that as a stored file can be removed afterwards, it can make space free for storage in between already stored file data on both sides. So while storing a new file, it can reside completely in a single contagious chunk or in a number of disjoint chunks, depending on the size of the new file and size of the available free chunks.

Chunk refers to a continuous portion of any size in the genome space. Along that, we also use chunk to refer to a simple representation of the corresponding continuous genome portion.

Each chunk is represented by two positive integers $n1$ and $n2$; $n1$ is the starting position of the chunk in abstract space or genome space and $n2$ is size of the chunk. A sorted chunk list means the order is based on start position $n1$.

A data structure is used to keep track of previously mentioned information for stored files to facilitate insertion, retrieval and deletion. This data structure is referred as *dictionary*. PWe maintain a single dictionary per genome.

### 8.2.1 Entry

For each file kept in the genome, an *entry* is kept in dictionary. Each entry consists of a number of attributes, related to the file it represents:

- **Name** - name of the file being saved
- **Type** (dicom or non-dicom) - to select decompression procedure depending on type of file being dealt with
- **Transfer Syntax** (implicit or explicit, relevant only for dicom) - lossless dicom file decompression using j2k method requires transfer syntax
- **Location** - a sorted list of all the chunks which contain sequence data of this particular file

### 8.2.2 Storing Dictionary

The dictionary of a genome itself is stored in the genome along with the actual files so that when a genome comes as input, our system can track the files, if any, which are already saved in this genome. To keep the dictionary in genome, we need a process to convert a dictionary to base sequence and to be able to reverse map the dictionary structure from the converted base sequence so when system has a new genome it can construct the initial dictionary corresponding to already saved files.

#### Dictionary To Base Sequence

We have designed the dictionary to base sequence conversion procedure carefully to achieve some subtle but useful advantages. Every time a insertion occurs, new entry is appended to dictionary. If the base sequence corresponding to an entry is independent of the other entries and additive i.e., sequences for the entries can be computed individually and concatenated to get the base sequence corresponding to the complete dictionary, then we need to do very little

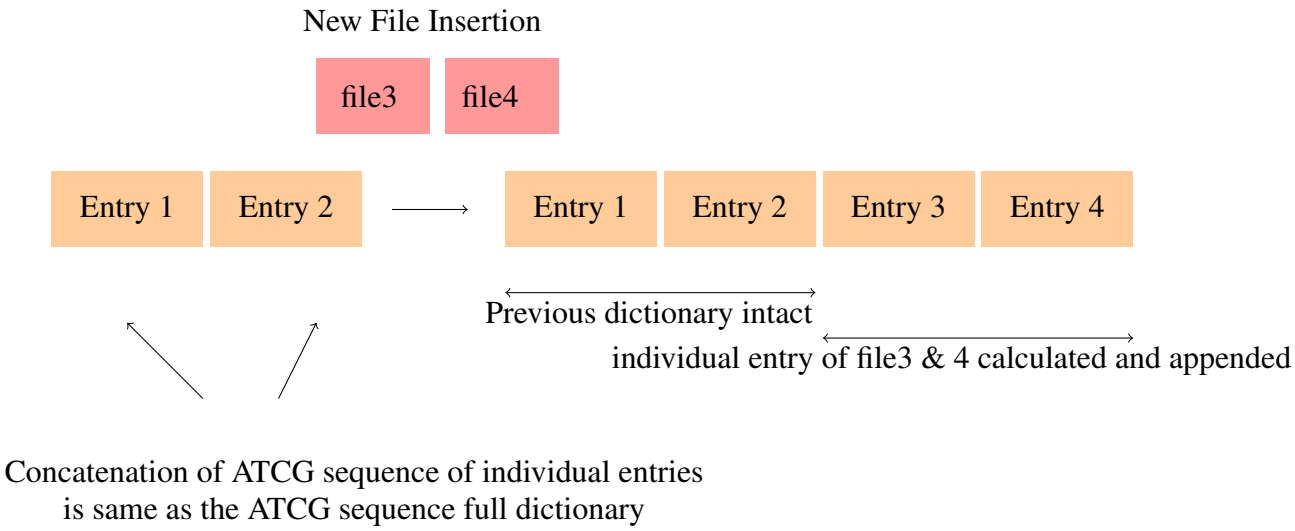



Figure 8.2: Independence and concatenation of dictionary entries in base sequence encoding

calculation with dictionary update during file insertion. We can just determine the sequence for new entries and append it to the end of already existing dictionary data in genome (Figure 8.2). The conversion procedure is described in 8.2.2.

#### Placement in genome

How many files can be stored in a genome is dependent on size, type and properties of files to be stored because the compression and conversion pipeline, discussed in previous chapters, produces varying output to input ratio for different files of same size. So if we want to keep dictionary alongside actual EHR files in an interleaved manner from the same end of genome, two possibilities to keep track of order of dictionary chunks are:

- **Pointer** - In each chunk of dictionary data, store a pointer to next chunk
- **Hierarchy** - Maintain a 2nd level dictionary to track chunks of this dictionary

Another issue in storing files and dictionary from same end is deciding over size of dictionary data chunks. Two options can be -

- Allocating just as much space as necessary for current insertion and creating new dictionary chunk for every insertion.
- When a new dictionary chunk is created, preserving certain or varying amount of extra space than currently needed.

Both of above approaches increase number of chunks in the genome space. As dictionary entries produce extremely shorter base sequence than actual files, file deletion makes small size

chunks free for storing next files and as we will discuss shortly, the more chunks a file uses to accommodate it's sequence data, the larger is the size of base sequence of it's dictionary entry, which in turn reduces file storage capacity unnecessarily. As capacity of genome sequence is relatively fixed, we prefer to avoid this problem.

### Our Approach To Dictionary Placement

We start storing dictionary data from the tail end of genome, approaching towards the front end and file data from front end, towards last end direction. So files and dictionary are stored from two opposite extremes, dictionary data is in a continuous stream at the tail end of genome, which reduces unnecessary chunk. Although situation may arise where a large amount of file data were stored and then some file from the middle is deleted, then dictionary data has to propagate into interleaved chunks with file chunks. But as dictionary base sequence generally is of very short length compared to sequence data of files, it's safe to assume that practically the above mentioned scenario will occur when storage of genome is almost filled up and at that point there will not be much scope to waste genome's storage capacity.

Let, dictionary has $n$ entries $e_1, e_2, \cdots, e_n$. As discussed in 8.2.2, we design a base encoding of dictionary that emulates keeping entries side by side sequentially. We keep the 1$^{st}$ entry $e_1$ at the last end of genome, 2$^{nd}$ entry $e_2$ to the left of $e_1$ , 3$^{rd}$ entry $e_3$ to the left of $e_2$ and so on.

### Base encoding of Dictionary

This procedure actually converts a number of dictionary entries to base sequence. For entries $e_1$, $e_2$, $e_3$, $\cdots$ the base string of entry $e_k$ is generated and appended to the already generated base stream of previous $k-1$ entries $e_1$, $e_2$, $\cdots$, $e_{k-1}$.

For an entry $e$, consisting of file name $f$, file type $t$, transfer syntax $s$, and location $l$, where $l$ is a sorted chunk list $\{c_1, c_2, \cdots, c_i\}$, first $l$ and then $f$ is encoded into nucleotide base symbols and $t$ and $s$ are inserted in special ways in between them.

To encode $l$, the chunks $c_k \in l$ are encoded in last to first order, i.e, $c_i$ is encoded first, then $c_{i-1}$, $\cdots$ , and $c_1$ at last of all. This ensures that when we reverse map $l$ from it's encoded sequence, the resultant chunk list is sorted as original $l$. And preserving the sorted order of $l$ is essential for lossless file retrieval.

As mentioned earlier this section, each chunk $c_k$ contains two positive numbers $n1$ and $n2$, so $n1$ and $n2$ contains no other character than the digits 0–9. We represent each digit of 0–9 by 4 bit, as shown in Table 8.1.

If a number $n$ has $m$ digits and $n = d_m d_{m-1} d_{m-2} \cdots d_1$, for encoding $n$ in ATCG sequence, we traverse $n$'s digits $d_i$ in least to most significant order, i.e., first $d_1$, then $d_2$, then $d_3$, and so on

| Digit | Binary | Write | Read |
|---|---|---|---|
| 0 | 0000 | AA | AA |
| 1 | 0001 | TA | AT |
| 2 | 0010 | CA | AC |
| 3 | 0011 | GA | AG |
| 4 | 0100 | AT | TA |
| 5 | 0101 | TT | TT |
| 6 | 0110 | CT | TC |
| 7 | 0111 | GT | TG |
| 8 | 1000 | AC | CA |
| 9 | 1001 | TC | CT |

Table 8.1: Mapping between digit and base

and append base symbol pair corresponding to $d_i$ according to column *Write* of Table 8.1 to already formed base string from previous $i-1$ digits $d_1, d_2, \cdots, d_{i-1}$.

Procedure of encoding a number to base string is explained in Algorithm 7. Here *Write*$(d)$ is value in *Write* column of row corresponding to digit $d$ in Table 8.1.

**Algorithm 7** Base Encoding of Number

**NUMBER_TO_BASE_SEQ** $(n)$
**Input :** A $m$ digit positive number $n = d_m d_{m-1} \cdots d_1$.
**Output:** A base sequence $B = b_1 b_2 b_3 \cdots$ where each $b_i \in \{A, T, C, G\}$.
$B = \phi$
**for** $i \leftarrow 1$ **to** $d$ **do**
  append *Write*$(d_i)$ to right of $B$
**end**
**return** $B$

There are $2^4 = 16$ possible bit combinations of length 4, of which only 10 are mapped to digits 0–9, as in Table 8.1. We can exploit the unused 6 combinations for some special purposes. As we want base sequences of different entries to be side by side to form the base sequence of entire dictionary, there needs to be some boundary in between the base sequence of two entries. The two combinations 1010 (CC) and 1011(GC) are unused. We utilize these to mark end of entry $e$ and incorporate file type $t$. If the file of $e$ is dicom, we use CC; CG otherwise.

The 4 combinations 1100, 1101, 1110, 1111 are unused too and they all start with 11. 11 corresponds to G according to numeral encoding of Table 6.1. In previous boundary marker, we had to take two bases ($CC$/$CG$) because two other pairs starting with $C$ ($CA$, $CT$) come from valid digits. But no digit in Table 6.1 has *Write* column value starting with $G$. So we can use only $G$ as boundary between $n1$ and $n2$ of each chunk $c_k$ and as boundary between pair of chunks $c_k$ and $c_{k+1}$ for $1 \leq k < l.size$ in location $l$.

After location $l$, we encode file name $f$. Without loss of generality, we assume each character $ch$ of $f$ is 8 bits or 1 byte. So according to numeral encoding of Table 6.1, $ch$ can be represented

by 4 nucleotide base symbols. We traverse characters $ch$ of $f$ in right to left order, e.g, we traverse *"file.ext"* in the order $t \rightarrow x \rightarrow e \rightarrow . \rightarrow e \rightarrow l \rightarrow i \rightarrow f$, and encode each $ch$.

Now we need another boundary in between base strings of $l$ and $f$. 1st 32 ASCII codes of 256 size ASCII table are not written characters, rather those are control codes which can never occur in between name of a file. These 32 codes (0–31) have 0's in 3 most significant positions of 8 bit binary representation of their ASCII value, which the rest of the ASCII table values don't. Now we add another bit with this 000 to incorporate transfer syntax of dicom files: if syntax is *implicit*, 1 is appended to form marker 0001(AT); otherwise, syntax is *explicit* and 0 is appended to make 0000(AA). In between location $l$ and name $f$, AA or AT is placed as boundary marker.

In this method, length of base stream of an entry depends on no. of characters in file name and no. of digits in $n1$ and $n2$ in chunks in location. Let, in entry $e$, file name $f$ has $l$ characters, location $l$ has $s$ chunks, where for each chunk $c_i, n_1$ and $n_2$ have $d_{i1}$ and $d_{i2}$ digits respectively, total length $L_B$ of encoded base sequence of $e$ is

$$L_B = l \times 4 + \sum_{i=1}^{i=s} 2(d_{i1} + d_{i2}) + s \times 2 + 3$$

**Algorithm 8** Base Encoding of Multiple Sequential Entries of Dictionary

**Input :** Dictionary entry indices $p$ and $q$, where $p \leq q$.
**Output:** A base sequence $B = b_1 b_2 b_3 \cdots$ where each $b_i \in \{A, T, C, G\}$.
**ENTRIES_TO_BASE_SEQ** *$(p, q)$*
  Base sequence $B \leftarrow \phi$
  **for** $i \leftarrow p$ **to** $q$ **do**
    $B.appendToRight$(ENTRY_TO_BASE_SEQ($i$th entry of dictionary))
  **end**
**return** $B$

**Algorithm 9** Base Encoding of a Single Entry of Dictionary

**Input :** A dictionary entry $e$.
**Output:** A base sequence $B = b_1b_2b_3\cdots$ where each $b_i \in \{A, T, C, G\}$.

**ENTRY_TO_BASE_SEQ** *(e)*

```
Base sequence B = φ
 if e.type == dicom then
 |  B.appendToRight(CC)
else
 |  B.appendToRight(GC)
end
l = e.location
 for i ← l.size downto 1 do
 |  c ← l.chunkAt(i)
 |   B.appendToRight (NUMBER_TO_BASE_SEQ(c.n2))
 |   B.appendToRight(G)
 |   B.appendToRight (NUMBER_TO_BASE_SEQ(c.n1))
 |   if i > 1 then
 |  |  B.appendToRight(G)
end
if e.syntax == implicit then
 |  B.appendToRight(T)
else
 |  B.appendToRight(A)
end
B.appendToRight(A)
 f ← e.filename
 l ← f.length
 for i ← f.length to 1 do
 |  ch ← f.charAt(i) 8 bit binary representation of f.charAt(i)
 |   for j ← 1 to 7 by 2 do
 |  |  m ← ith bit in binary representation of ch
 |  |   n ← (i + 1)th bit in binary representation of ch
 |  |   if m == 0 then
 |  |  |  if n == 0 then
 |  |  |  |  B.appendToRight(A)
 |  |  |  else
 |  |  |  |  B.appendToRight(C)
 |  |  |  end
 |  |  else
 |  |  |  if n == 0 then
 |  |  |  |  B.appendToRight(T)
 |  |  |  else
 |  |  |  |  B.appendToRight(G)
 |  |  |  end
 |  |  end
 |  end
end
return B
```

**Decoding Dictionary from base Sequence**

Traversing the base sequence from right to left, group of every 4 bases is mapped to a character until the first 2 bases of a group is $AA$ or $AT$. The mapped characters form file name $f$ of current entry. If the marker is $AA$, transfer syntax $s$ for current entry is *implicit*; *explicit* otherwise. Then from the base next to marker, we scan pair of bases until a pair starts with $G$, which marks end of a number. Each scanned pair is mapped to a digit and all digits obtained till we get marker $G$ form a number $n1$. Similarly another number $n2$ is decoded until any of markers $G$, $CC$, $GC$ is encountered. A chunk is formed with $n1$ and $n2$. If the last marker was $G$, more chunks are decoded in similar manner; otherwise all chunks of location $l$ have been decoded. If last marker was $CC$, file type $t$ is *dicom*, *non-dicom* otherwise. $f$, $l$, $t$ and $s$ from an entry $e$. If base stream has bases left, we continue scanning for next entries. All entries are listed to form the dictionary.

## 8.3 Free List

Throughout an entire user session, an in-memory structure *free list* is maintained to keep track of currently available free chunks, locations where neither file or dictionary data exists right now, in genome space. At the beginning of a session, after constructing initial dictionary, initial free list is formed. In free list, chunks are maintained to be sorted in ascending order. After every insertion and deletion operation, free list changes accordingly.

## 8.4 File Operations

In a user session, after initial dictionary and free list has been created, system is ready to perform designated operations: storing files provided that genome space has enough capacity, retrieving already stored files, and removing stored files. In current scenario, removing a stored file does not restore original base sequence in the locations where the deleted file's data were placed.

### 8.4.1 Insertion

User can add one or multiple files at a time in genome. System converts input file(s) to base sequence. For each file, traversing the free list from front end, it makes a temporary list of chunks needed to accommodate the file encoded base stream. Now an entry is formed with this location and other file information and converted to base stream and checked if genome has capacity to place both file and entry base sequence. If yes, the entry is inserted to in-memory

dictionary and entry and file base stream is inserted in corresponding locations of genome space. Algorithm 10 presents file insertion procedure.

**Algorithm 10** Inserting a File in a Genome.

**Input :** A dicom or non-dicom EHR file and a whole genome sequence.
**Output:** Stores the file in the genome, if genome has enough space.

```
INSERT (file, genome)
    freeList ← genome.freelsit
     dictionary ← genome.dictionary
     if file is dicom then
        data ← DCM_TO_BASE_SEQ(file)
    else
        data ← NON_DCM_TO_BASE_SEQ(file)
    end
    location ← φ
     m ← data.length
     while (m > 0 && freeList is not empty) do
        chunk ← freeList.nextChunk()
         n ← chunk.size
         if (m < n) then
            split chunk in two chunks inside freeList with first n size and rest m − n size
              portions
            add the n size chunk to location
            break
        else
            add chunk to location
             m ← m − n
        end
    end
    entry ← new entry with file information and location
     meta ← ENTRY_TO_BASE_SEQ(entry)
     if (data.length + meta.length ≥ freeList.capacity) then
        i ← 1
         foreach chunk ∈ location do
            WRITE_BASE_SEQ_(chunk.n1, data.subSequence(i, chunk.n2))
             i ← i + chunk.size
            remove chunk from freeList
        end
        insert entry at tail of dictionary
         place meta in appropriate location of genome
         return SUCCESS
    else
        return FAILURE
    end
```

### 8.4.2 Retrieval

First, corresponding file entry is picked from in-memory dictionary. Then system reads file encoded base stream from corresponding chunks in location of the entry. This base sequence is then decompressed and reverse mapped to original file.

**Algorithm 11** Retrieving a File Stored in Genome.

**Input :** A index from list of files stored in genome.

**Output:** The original EHR file, corresponding to the index, reconstructed from genome sequence.

**RETRIEVE** *(index, genome)*

```
entry ← dictionary.getEntry(index)
location ← entry.location
data ← φ
foreach chunk ∈ location do
    data.appendToRight(READ_BASE_SEQ(chunk.n1, chunk.n2))
end
file ← FILE_FROM_BASE_SEQ(data)
return file
```

### 8.4.3 Removal

We remove the entry from dictionary and entry's base sequence from dictionary base stream in genome. The dictionary entries to the left of the deleted entry gets shifted to the right.

**Algorithm 12** Removing A File Stored in Genome.

**Input :** A set of indices from list of files stored in genome.
**Output:** Corresponding file entries removed from dictionary.

**REMOVE** *(indices, genome)*

```
dictionary ← genome.dictionary
freeList ← genome.freeList
for i ← 1 to indices.length − 1 do
    shift ← Σ_{j=0}^{i−1} dictionary.getEntry(indices[j]).L_B
    seq ← ENTRIES_TO_BASE_SEQ(indices[i − 1]+1, indices[i]-1)
    pos ← Σ_{j=0}^{indices[i]−1} dictionary.getEntry(j).L_B − shift
    WRITE_BASE_SEQ(pos, seq)
end
i ← indices[indices.length − 1]
j ← dictionary.size − 1
shift ← Σ_{j=0}^{indices.length−1} dictionary.getEntry(j).L_B
pos ← Σ_{j=0}^{dictionary.size−1} dictionary.getEntry(j).L_B − shift
if i < j then
    seq ← ENTRIES_TO_BASE_SEQ(i + 1, j)
    WRITE_BASE_SEQ(pos, seq)
foreach index ∈ indices do
    foreach chunk ∈ dictionary.getEntry(index).location do
        add chunk to freeList preserving ascending order
    end
    dictionary.removeEntry(index)
end
```

# Chapter 9

# Implementation

We have developed a software as a simple demonstration of our concept using proposed pipeline. We used Java and JavaFX for implementation.

## 9.1 Specification

The software takes the whole genome sequence of a person in FASTA format, lengths of all chromosomes, start and end positions of coding regions as inputs before we can use it for archiving DICOM and other files.
The whole genome sequence has to be uploaded to the software in a folder which has seperate files in FASTA format corresponding to each chromosome. Moreover, the lengths of all chromosomes should be in a text file; start and end positions of coding regions should be in a separate text file.
If all inputs are valid, the software displays a list of file(s) already stored in the input genome sequence. User can add one or multiple new file provided that genome has enough capacity left. There are two options for retrieving a stored file - user can select file(s) from displayed list to view; the software reconstructs original file(s) and opens with default viewer of system, or save a file in user device. Finally user can remove one or multiple files, then the files will be removed from displayed file list too.

## 9.2 Outlook


Figure 9.1: Start Page

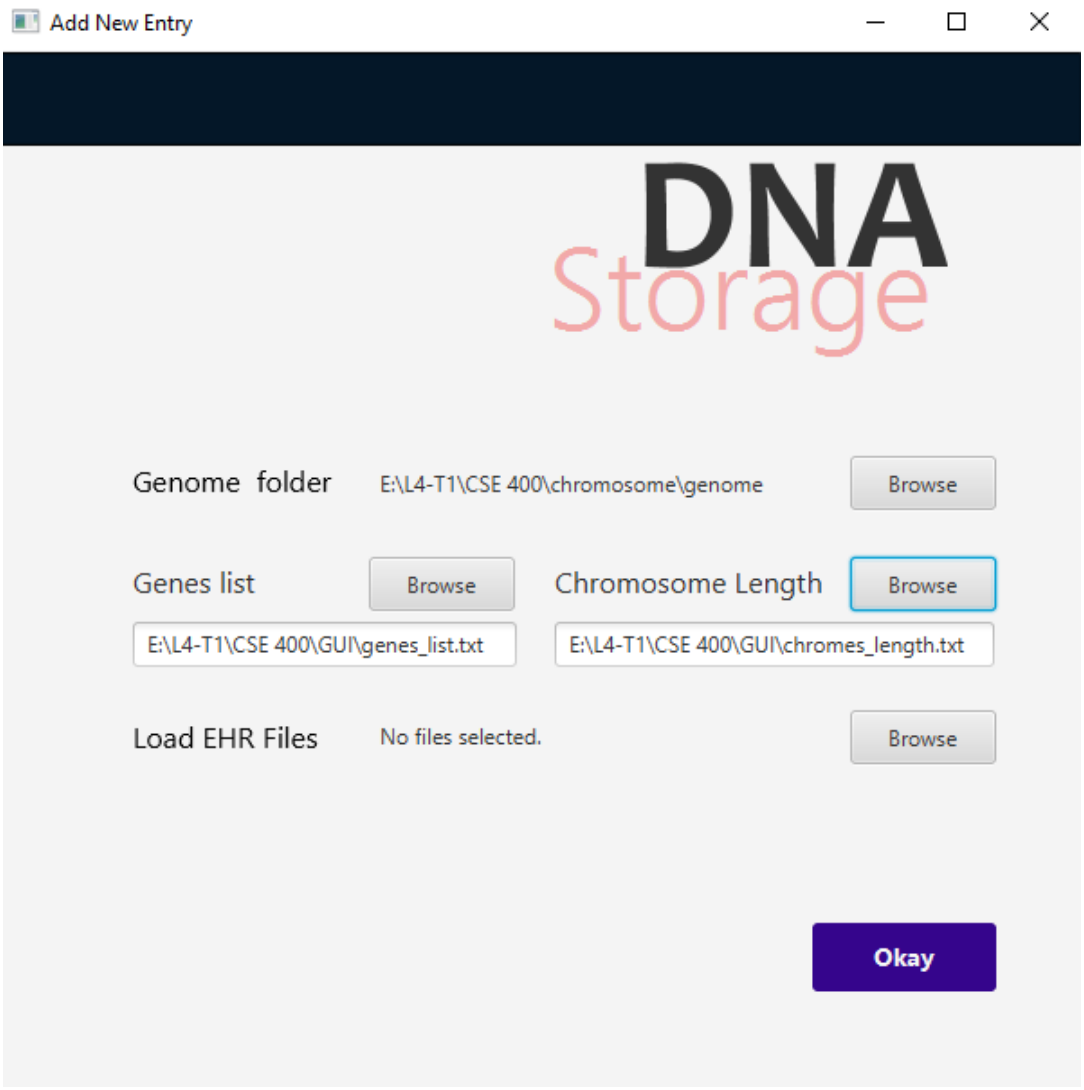



Figure 9.2: Inputs for a new user session.

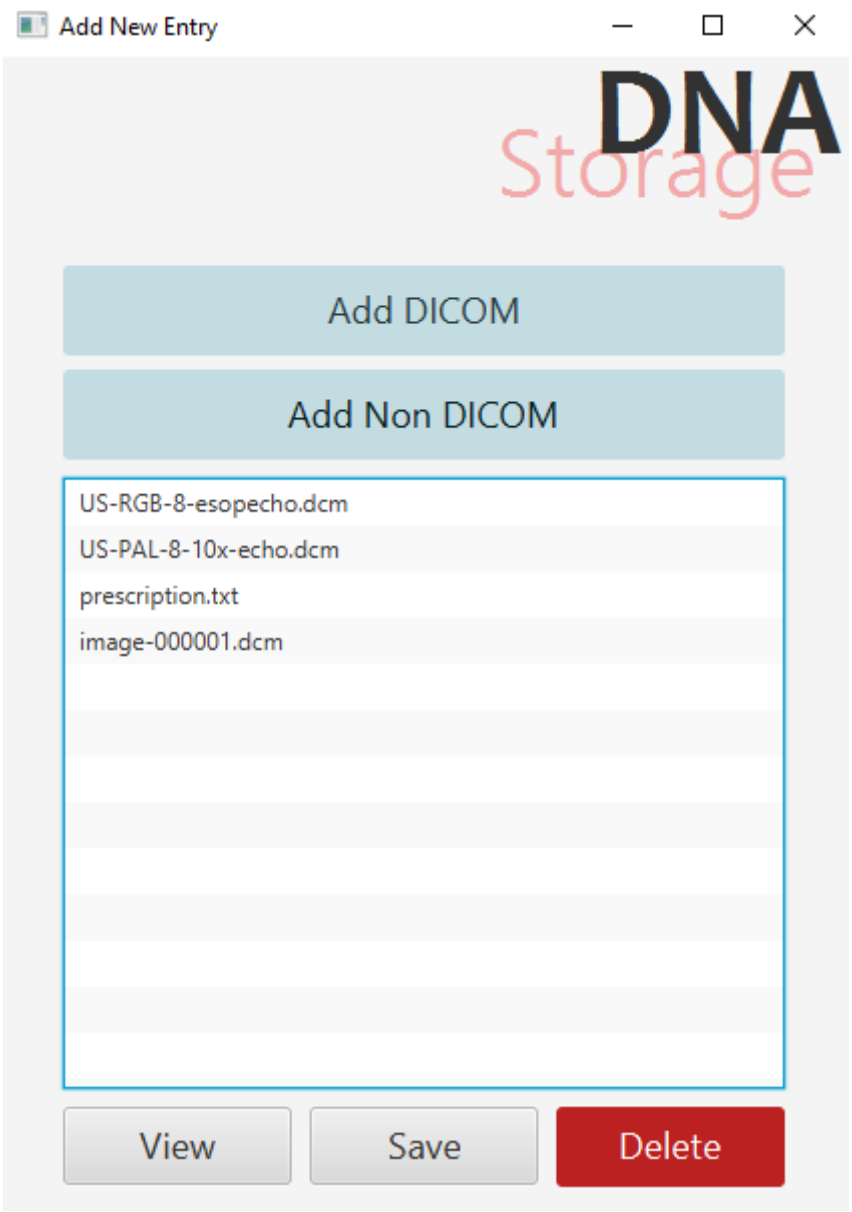



Figure 9.3: Displays currently stored files. Can insert, view, save and remove files.

# Chapter 10

# Concluding Remarks and Discussion

In this thesis, we came up with a proof of concept of storing a patient's lifetime health records inside his genome sequence. We took advantage of the fact of 'Junk DNA' and also took into account the debate on sequence dependent functionalities of introns. So, we made a system where non-functional parts can be customized by the users. The default one uses only the spacer DNA part as it is out of debate. As, we plan to store the entire Electronic Health Record of a patient in his lifetime inside his genome sequence, we introduce compression in every stage of our pipeline. At first we tried to compress the EHR files. We did extensive studies for the DICOM files as it is the most complex part of the EHRs. We tried with the different encoding methods for converting binary stream to nucleotide sequences and finally ended up with a shift based approach that takes advantage of the frequency of long zero sequences in the bitstream of compressed DICOM files. Then we examined with various methods to further compress the the nucleotide sequence. At last we created a file management system using the concept of file management systems of operating systems. We implemented virtual memory concept, dictionary and metadata structures in our system.

There are several immediate extensions to our work. In concern with the huge growth of medical data, steps are being made to store patient data in cloud storage. [41] [42] If the size of EHR exceeds the capacity of non-functional space of the DNA, we propose to store *links* to external cloud storage inside the DNA and if someone clicks at that link it will take him to the corresponding cloud server where the EHR file is located. We also thought of mapping the EHRs into the existing nucleotide sequences of DNA in case of EHR data surpassing available storage either by the growth of EHR or by more and more sequence dependent functions for 'junk DNA's being revealed. We left them for our future work.
With the fast development and the steeply declining cost of error-free sequencing and synthesis technologies, we hope that in near future we may synthesize the whole human genome along with the EHR files inside them and create biological samples that will contain both EHRs and the genetic data for each individual. That will be a remarkable revolution in the medical sector.

In our thesis, we through the vision of getting all the EHR files of a patient's lifetime along with his gene sequences from a sample as small as a blood point. As the tech companies are gradually heeding towards data storage in DNA hard drives [43] and Microsoft has declared that they have taken plan to store their data in DNA medium, we already feel the significance of generating integrated data for storing them in DNA medium. And in our thesis, we generated a process that integrates individuals medical files and genetic information together in efficient way.

# References


[1] Wikipedia, "Arecibo message — Wikipedia, the free encyclopedia." http://en.wikipedia.org/w/index.php?title=Arecibo%20message&oldid=833740338, 2018. [Online; accessed 09-April-2018].

[2] Wikipedia, "DNA digital data storage — Wikipedia, the free encyclopedia." http://en.wikipedia.org/w/index.php?title=DNA%20digital%20data%20storage&oldid=834007898, 2018. [Online; accessed 09-April-2018].

[3] G. M. Church, Y. Gao, and S. Kosuri, "Next-generation digital information storage in dna," *Science*, vol. 337, no. 6102, pp. 1628–1628, 2012.

[4] R. Sikorski and R. Peters, "Genomic medicine: Internet resources for medical genetics," *JAMA*, vol. 278, no. 15, pp. 1212–1213, 1997.

[5] C. Sander, "Genomic medicine and the future of health care," *Science*, vol. 287, no. 5460, pp. 1977–1978, 2000.

[6] A. N. Kho, L. V. Rasmussen, J. J. Connolly, P. L. Peissig, J. Starren, H. Hakonarson, and M. G. Hayes, "Practical challenges in integrating genomic data into the electronic health record," *Genetics in Medicine*, vol. 15, no. 10, p. 772, 2013.

[7] J. Davis, "Microvenus," *Art Journal*, vol. 55, no. 1, pp. 70–74, 1996.

[8] R. J. Hughes, D. M. Alde, P. Dyer, G. G. Luther, G. L. Morgan, and M. Schauer, "Quantum cryptography," *Contemp. Phys.*, vol. 36, p. 149, 1995.

[9] A. Gehani, T. LaBean, and J. Reif, *DNA-based Cryptography*, pp. 167–188. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004.

[10] C. Taylor Clelland, V. Risca, and C. Bancroft, "Hiding messages in dna microdots," vol. 399, pp. 533–4, 07 1999.

[11] A. Leier, C. Richter, W. Banzhaf, and H. Rauhe, "Cryptography with dna binary strands," *Biosystems*, vol. 57, no. 1, pp. 13 – 22, 2000.

[12] M. Ailenberg and O. D. Rotstein, "An improved huffman coding method for archiving text, images, and music characters in dna.," *Biotechniques*, vol. 47, no. 3, p. 747, 2009.

[13] D. G. Gibson, J. I. Glass, C. Lartigue, V. N. Noskov, R.-Y. Chuang, M. A. Algire, G. A. Benders, M. G. Montague, L. Ma, M. M. Moodie, *et al.*, "Creation of a bacterial cell controlled by a chemically synthesized genome," *science*, vol. 329, no. 5987, pp. 52–56, 2010.

[14] G. M. Church, Y. Gao, and S. Kosuri, "Next-generation digital information storage in dna," *Science*, p. 1226355, 2012.

[15] N. Goldman, P. Bertone, S. Chen, C. Dessimoz, E. M. LeProust, B. Sipos, and E. Birney, "Towards practical, high-capacity, low-maintenance information storage in synthesized dna," *Nature*, vol. 494, no. 7435, p. 77, 2013.

[16] S. M. H. T. Yazdi, Y. Yuan, J. Ma, H. Zhao, and O. Milenkovic, "A rewritable, random-access dna-based storage system," *CoRR*, vol. abs/1505.02199, 2015.

[17] E. N. Gilbert, "Synchronization of binary messages," vol. 6, pp. 470 – 477, 10 1960.

[18] J. Bornholt, R. Lopez, D. M. Carmean, L. Ceze, G. Seelig, and K. Strauss, "A dna-based archival storage system," *ACM SIGOPS Operating Systems Review*, vol. 50, no. 2, pp. 637–649, 2016.

[19] R. H. Carlson, *Biology is technology: the promise, peril, and new business of engineering life*. Harvard University Press Cambridge, MA, 2010.

[20] W. Gilbert, "Why genes in pieces?," *Nature*, vol. 271, no. 5645, p. 501, 1978.

[21] J. M. Lackie, *The dictionary of cell & molecular biology*. Academic Press, 2007.

[22] K. Jain, "Personalized medicine.," *Current opinion in molecular therapeutics*, vol. 4, no. 6, pp. 548–558, 2002.

[23] G. S. Ginsburg and J. J. McCarthy, "Personalized medicine: revolutionizing drug discovery and patient care," *TRENDS in Biotechnology*, vol. 19, no. 12, pp. 491–496, 2001.

[24] D. G. Katehakis and M. Tsiknakis, "Electronic health record," *Wiley Encyclopedia of Biomedical Engineering*, 2006.

[25] O. S. Pianykh, *Digital imaging and communications in medicine (DICOM): a practical introduction and survival guide*. Springer Science & Business Media, 2009.

[26] T. Somassoundaram and N. Subramaniam, "High performance angiogram sequence compression using 2d bi-orthogonal multi wavelet and hybrid speck-deflate algorithm," *Biomedical Research*, 2018.

[27] G. J. Sullivan, J. Ohm, W.-J. Han, and T. Wiegand, "Overview of the high efficiency video coding (hevc) standard," *IEEE Transactions on circuits and systems for video technology*, vol. 22, no. 12, pp. 1649–1668, 2012.

[28] V. Sanchez and J. Bartrina-Rapesta, "Lossless compression of medical images based on hevc intra coding," in *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pp. 6622–6626, IEEE, 2014.

[29] S. S. Parikh, D. Ruiz, H. Kalva, G. Fernández-Escribano, and V. Adzic, "High bit-depth medical image compression with hevc," *IEEE journal of biomedical and health informatics*, vol. 22, no. 2, pp. 552–560, 2018.

[30] W. R. Riddle and D. R. Pickens, "Extracting data from a dicom file," *Medical physics*, vol. 32, no. 6Part1, pp. 1537–1541, 2005.

[31] D. A. Huffman, "A method for the construction of minimum-redundancy codes," *Proceedings of the IRE*, vol. 40, pp. 1098–1101, Sept 1952.

[32] M. Burrows and D. J. Wheeler, "A block-sorting lossless data compression algorithm," vol. 1, 07 1995.

[33] I. H. Witten, R. M. Neal, and J. G. Cleary, "Arithmetic coding for data compression," *Commun. ACM*, vol. 30, pp. 520–540, June 1987.

[34] J. Duda, "Asymmetric numeral systems," 02 2009.

[35] T. A. Welch, "A technique for high-performance data compression," *Computer*, vol. 17, pp. 8–19, June 1984.

[36] J. Ziv and A. Lempel, "Compression of individual sequences via variable-rate coding," *IEEE Transactions on Information Theory*, vol. 24, pp. 530–536, September 1978.

[37] S. Grumbach and F. Tahi, "Compression of dna sequences," in *[Proceedings] DCC '93: Data Compression Conference*, pp. 340–350, 1993.

[38] S. Grumbach and F. Tahi, "A new challenge for compression algorithms: Genetic sequences," *Information Processing  Management*, vol. 30, no. 6, pp. 875 – 886, 1994.

[39] K. Daily, P. Rigor, S. Christley, X. Xie, and P. Baldi, "Data structures and compression algorithms for high-throughput sequencing technologies," *BMC Bioinformatics*, vol. 11, p. 514, Oct 2010.

[40] X. Chen, M. Li, B. ma, and J. Tromp, "Dnacompress: Fast and effective dna sequence compression," vol. 18, pp. 1696–8, 01 2003.

[41] M. Bamiah, S. Brohi, S. Chuprat, *et al.*, "A study on significance of adopting cloud computing paradigm in healthcare sector," in *Cloud Computing Technologies, Applications and Management (ICCCTAM), 2012 International Conference on*, pp. 65–68, IEEE, 2012.

[42] A. Bahga and V. K. Madisetti, "A cloud-based approach for interoperable electronic health records (ehrs)," *IEEE Journal of Biomedical and Health Informatics*, vol. 17, no. 5, pp. 894–906, 2013.

[43] E. Strickland, "Tech companies mull archiving data in dna [news]," *IEEE Spectrum*, vol. 53, no. 7, pp. 9–11, 2016.

Generated using Undegraduate Thesis LaTeX Template, Version 1.3. Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka, Bangladesh.

This thesis was generated on Saturday 19th October, 2019 at 2:07pm.