Fitting the World's Data into a Shoebox
DNA Simulated Storage (DNASS): A Low-Cost Solution for Improving DNA Storage

CS5099: Dissertation

University of St. Andrews

Department of Computer Science

Komal Kamal Harjani

Supervisor: Dr Ishbel Duncan

August 21,2020

August 23, 2020

Abstract

Information storage has consistently been exponentially growing. Storage of digital information relies heavily on silicon, which is expected to run out in 2040. Current storage technologies also begin to deteriorate within 10 to 20 years, and have serious limitations to climate change. There is a pressing need for an alternative. Deoxyribonucleic acid (DNA) is an attractive solution, and has all the physical properties to supersede conventional technologies. It is estimated that DNA storage would allow for all of the world's current storage to fit inside a shoebox, with no energy. Therefore, moving away from storing data in binary to storing data in DNA's four-bases A, G, C, T, may have the ability to transform data storage as we know it. Although DNA has huge potential as a data storage device, there are multiple issues that need to be overcome before the adoption of this technology. These issues include exorbitant costs and extremely slow reading and writing times, and the production of errors by reading and writing processes of DNA. This study presents DNA Simulated Storage, a low cost software that is built with the purpose of testing encoding and error correcting schemes for DNA storage, there are usually extremely costly. Some success have been achieved with DNASS, in creating a customisable error simulator and in developing in an algorithm that is able to overcome deletion errors at comparable rates to previous works.

Declaration

I, Komal Harjani, hereby verify that this thesis, which is 12494 words in length, has been written by me, and has not been submitted or written for any previous application. Credit is explicitly given to others by citation or acknowledgment. This project was conducted by me at The University of St Andrews from June 2020 to August 2020 towards fulfilment of the requirements of the University of St Andrews for the degree of MSc under the supervision of Dr Ishbel Duncan.

In submitting this dissertation to the University of St Andrews, I give permission for it to be made available for use in accordance with the regulations of the University Library. I also give permission for the title and abstract to be published and for copies of the report to be made and supplied at cost to any bona fide library or research worker, and to be made available on the World Wide Web.

Acknowledgement

I would like to express my deep and sincere gratitude to my supervisor Dr Ishbel Duncan for her invaluable guidance, suggestions and support throughout this project.

I would also like to thank my parents for their undying support, and for giving me the opportunity to pursue my further education.

Contents

Glossary

1. **Complement**: DNA is structured in pairs, in a double helix structure, where pairs are complementary. A is complementary with T and G is complementary with C.

2. **Deoxyribonucleic acid (DNA)**: DNA is a long molecule that is made up of nitrogenous bases and a phosphate group. This is where all the genetic information is stored in our bodies. Artificial DNA is material that holds non-genetic information, such as digital information encoded into the same nitrogenous bases.

3. **Homopolymer**: A homopolymer is a chain of the same molecules. An occurrence of this can lead to errors in reading DNA.

4. **Nucleotide**: A nucleotide is a base, or the building blocks of DNA. It is one of four chemicals: adenine, thymine, guanine, and cytosine.

5. **Primer**: A primer is a short nucleic acid sequence which is used for the replication of DNA. In artificial DNA synthesis, this is used as a delimiter or payload for information.

6. **Synthesis**: Synthesis is the writing process of DNA.

7. **Sequencing**: Sequencing is the reading process of DNA.

1 Introduction

In recent years, there has been an exponential increase in digitally-stored information. According to Hariri et al. [1], approximately 90 percent of all the data in the world was generated in the last two years. This poses the key question of where and how to best store all the world's data. The present solution takes the form of data centers and warehouses at the enterprise level; buildings dedicated to data storage and communication. Current storage technologies which uphold these large scale operations are either magnetic, flash or optical based. The advancements in data storage and processing methods acted as a key enabler for the rise and proliferation of data in a scalable fashion, resulting in the commercial use of big data, for example. While applications of data to businesses and institutions are widespread, the increased availability of data has fundamentally transformed most industrial and commercial operations, where data-driven decision making and information gathering has become the norm. Given the instrumental role of data to the functioning of any market, industry or company, it is crucial that storage technologies are able to match projected data growth. However, according to [1,2] current storage options will be outpaced or become unsustainable due to a lack of supply of key raw materials, low energy efficiency and high waste output in the next 20 years. This study thus investigates a potential alternative; storing data in deoxyribonucleic acid (DNA). This study presents a prototype, DNA Simulated Storage (DNASS), which provides a customisable platform for researchers to simulate and test the conversion of binary data into DNA's four-base form, with components that include mimicking real-life processes of data loss during the conversion process, and an algorithmic solution to data recovery with lowered redundancy.

The paper proceeds as follows: in Chapter 1, the paper is contextualized, the problem is outlined, DNA storage is introduced. This is followed by Chapter 2, which outlines the requirements of a storage device, the project objectives and the contributions of this paper. Chapter 3 outlines and discusses relevant literature. Chapter 4 details the methodology and the software engineering process. Chapter 5 outlines and provides justifications for the design and implementation of the software. This is followed by Chapter 6 which provides a use case test of the software, how it can be used in practice, and what results are derived from the prototype. Lastly, Chapter 7 concludes and discusses future work.

1.1 Background

In order to contextualise this paper, this section provides a taxonomic guide into storage technologies used today.

Digital information storage falls under two broad uses; archival storage and/or frequently accessed data. Archival storage may include data such as financial audits, legal or record keeping statements that do not require frequent access but are nonetheless vital documentation measures. Frequently accessed data, as the name suggests, refers to data accessed on a regular basis. The technology used to store these two types of data are magnetic, flash or optical and can be either volatile (requires power to store information) or non-volatile (can retain information even while powered off).

Magnetic-based drives store information by altering the magnetic properties of a disc. Information is thus stored as microscopic magnetic fragments and is read through a magnetic head that hovers over the metal plate and moves depending on the magnetic attraction or repulsion the metal plates create. The magnetic connection translates to a either a one or zero [3]. Once the data is read, the information is transmitted to a computer and decoded. Current uses of magnetic storage media include magnetic tape, magnetic drives and hard drive disks (HDD). Archival storage typically uses non-volatile technologies such as magnetic tape [4].

Unlike magnetic storage, flash storage does not have any physical moving parts; data is stored electronically rather than magnetically [3]. In flash storage, binary information is physically represented as electronic voltages through transistor gates. This form of memory storage is much faster as it doesn't have to wait for a physical head to read and write data, and therefore has quicker access times and lower latency [3]. This form of storage is used for fast-access storage. Examples of current flash memory include USBs, and solid state drives (SSD).

Optical disc storage encodes binary information using reflective properties. A laser reads a disc and associates binary encodings based on how much light an area reflects. These are often known as 'WORM' (Write Once Read Many) storage devices, as they can be written only once. DVDs and blu-ray discs are some examples [3].

Large-scale storage of information for companies employ data centres; buildings dedicated to storing data. Uses of data centres include hosting real-time media, applications, and data, which subsequently be delivered to clients or computers, typically over the internet. These buildings not only house storage systems such as servers but are also composed of data communication channels, climate control and security devices. The architecture of the underlying storage systems usually consists of racks of hardware, called servers. These servers have tiers of data, segregated into frequently accessed and archived data [3]. Data centers that hold infrequently accessed data typically use magnetic tape or optical discs as their server architecture, as they more durable, but tend to have longer retrieval times.

The exponential growth of these server technologies has been integral to the success of today's computer systems and has formed the basis of fast-paced technological progress in domains such as cloud computing. In particular, these advancements can be largely attributed to well-connected data centres, making it easy for any industry to scale while using cloud and virtual computing, by adopting services from cloud providers such as Function-as-a-Service (FaaS) and Software-as-a-Service (Saas), among others. This provides users with instant data access through the internet and enables functionalities such as video streaming, email, cloud file storage or hosted servers.

1.2 Problem Outline

Although these technologies are critical and have been successful, they do not present a viable long-term, sustainable approach to data storage as they lack durability and energy efficiency. Alarmingly, the rate at which information and data is being generated far exceeds current storage capabilities, even while taking into account projected density improvements in current storage technologies. The problem is crucially both a technological and economic one, as the demand for data storage is expected to outweigh the supply of silicon in 2040 [1,2]. Silicon is an integral component to computer systems. Far-reaching advancements in technology, such as transistors, have been facilitated by silicon, whose versatility as a semiconductor is highly suited to computer hardware. Because the pace at which information is generated far outweighs the rate of density improvements of these storage methods when silicon runs out, there will only be a finite amount of data that can be stored with the available resources demanding alternative methods of storage.

Moreover, most data centres have tens of thousands of servers, which all require large amounts of energy and generate high volumes of electronic waste. This is not only in the form of energy consumption (electricity) but also for Heating, Ventilation and Air Conditioning (HVAC). For example, a large data centre that holds 1 exabyte of data requires approximately 100 megawatts (MW) of power which is enough to power a small town of 80,000 households [5]. Moreover, this energy report discovered global data centres emitted as much carbon dioxide as the global aviation industry in 2018. Although data centres are slowly transitioning to a reliance on cleaner energy, the projected exponential growth of data storage would require a radical change in data center energy operations in order to present an environmentally-friendly alternative [5].

Currently, archival technologies can only survive under ideal conditions for a few years, HDD has a lifetime of around 3-5 years and tape of around 15-30 years [2]. Therefore, in order to keep up with current consumption of data, a significant amount of physical space and millions of units of hardware are still required, which need to be replaced often, leading to large amounts of electronic waste. Tape technology is nonetheless the most dense storage available today, and is able to fit 10GB per millimetre square area [3], it is also non-volatile and the most durable, which is why it is the most common form of archival storage today. Blu-ray discs may be able to store a petabyte of data (1,000,000 gigabytes) [4] which would be a dramatic improvement in the density of conventional storage. However, this limit is only theoretical and is subject to verification with lab experiments [4].

Ultimately, given the aforementioned scarcity of silicon, energy and density inefficiencies coupled with the exponential growth of data storage demands, there needs to be a significant improvement in density and duration of these current technologies, or an alternative solution altogether. Proposed alternatives include DNA computers, quantum computers and optical computers [2].

1.3 DNA Storage: A Potential Solution

Each cell in living organisms, such as human bodies, contain genetic information in their nucleus measuring at only 6 micro-metres [2]. These include genetic instructions for the development, growth and reproduction of all living matter or organisms. DNA is composed of nucleotides, which refers to a combination of three molecules; a nitrogen-containing (nitrogenous) base, a five-carbon sugar (ribose or deoxyribose) and a phosphate group. The four nitrogenous bases in DNA are adenine (A), guanine (G), cytosine (C) and thymine (T). Nitrogenous bases are where genetic information is represented and stored through different combinations of these four bases in DNA. Just as computers encode digital information in base 2 (binary code), DNA naturally contains genetic information in base 4. Since a single DNA strand can be composed of any combination and sequence of the four bases, artificial DNA synthesis is possible without any template DNA, meaning information can be encoded into these bases.

The characteristics of DNA make it a naturally suitable storer of information, namely, it's dense and immutable characteristics. The theoretical limit of DNA storage is 1 exabyte per gram of synthetic DNA [10]. As a reference point, Facebook built an entire data centre to store 1 exabyte of cold storage (archival storage), something that could potentially be stored in a single gram of DNA with comparatively negligible energy consumption [6]. It is estimated that all of the world's storage can be stored in a shoebox with current theoretical capacities of DNA storage [2]. Furthermore, DNA is composed of double-stranded molecules, which makes it highly stable under ideal conditions and environments, giving it a half-life of 500 years [10]. Synthetic DNA thus provides a possible solution to many of the problems traditional data storage methods pose with respect to density, durability and maintenance.

There has been rapid progress in synthesising (writing to) and sequencing (reading) DNA with accuracy and sequencing costs are declining rapidly [2]. Furthermore, motivation to improve the technologies that read DNA is eternal, meaning the read process will never become obsolete while the study of genetic DNA in biology exists. Additionally, the growth of the study into artificial gene synthesis is improving write processes of synthetic DNA. These reasons have facilitated the exploration of synthetic DNA as a potential cost-effective solution in the near future.

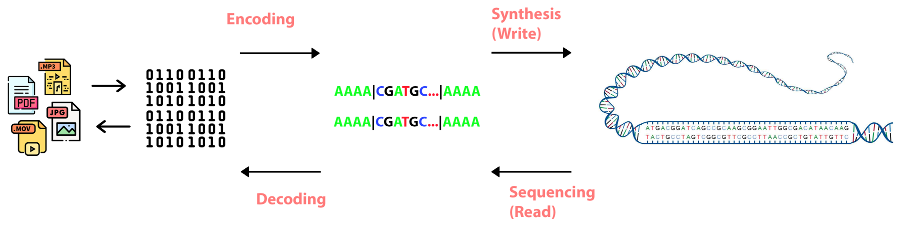


Figure 1: Process of storing digital information in DNA

1.3.1 Process of Encoding DNA

Storing messages in DNA was first demonstrated in 1988. The process of storing DNA is outlined in figure 1. Since all current computer systems are based on binary code, any storage system not using binary code has to be encoded into binary in order to be processed by current systems built using binary or higher level languages, which have underlying binary encodings. The steps needed to encode data into DNA are outlined below.

1. Encoding: the first step is to map binary code to DNA code. The result of encoding is a long string of DNA made of combinations of A, C, G and T. There are different mappings, or encoding methods which can be used to achieve this.

2. Synthesis (Writing DNA): the string of DNA can then be synthesised chemically, by altering the physical state of silicon or any non-genetic material to represent the data structure. This process involves binding each nucleotide to a molecule of the material [6]. After DNA is synthesised, the information is then stored until it is needed to be retrieved.

3. Sequencing (Reading DNA): when the data needs to be accessed, the DNA is sent through a sequencing machine, which pieces the molecules back together as a long string of DNA. Different bases have different physical fluorescent properties, and therefore the sequence is read optically [6].

4. Decoding: the long strings of DNA are then mapped out to their binary encodings for current computer systems to read the data. This step also involves error correcting any lost data.

1.3.2 Current Issues with DNA

DNA storage needs to overcome several pressing challenges before it can become a commercial solution that suits the data needs of today. There are four main challenges that need to be addressed in order for DNA storage to be a viable solution. Namely, cost, synthesis and sequencing errors, digital redundancy, and ethical concerns regarding the difficulty of deleting data.

Firstly, projections of storing 1TB in DNA is still 70 million times more expensive than a modern 1TB hard drive [2]. Unless these costs drastically reduce in the foreseeable future, DNA storage will not be able to compete with current storage technologies. High costs don't only present a barrier for DNA storage becoming a commercial solution, but also present a barrier for research development.

Secondly, despite the extremely long half-life of DNA, reading and writing DNA still produces significant data loss, making it hard to retrieve the data. These usually increase with the size of the DNA string being read or written to. Common errors include insertions, substitutions and deletions of bases in the DNA string requiring error-correcting after sequencing [6]. In order to make DNA storage a success, synthesis and sequencing errors must be overcome. This can be done in two ways. Firstly, by improving synthesis and sequencing machinery to be more accurate so that less data is lost in the process. And secondly, improving error correcting algorithms and encoding schemes in order to retrieve the most data as possible after it has been lost. It is pertinent to devise robust encoding methods that can withstand errors and retrieve the data with accuracy, which is the chosen focus of this project. The error-retrieval process is also integral to conventional storage methods, as information has to be encoded into binary after their respective states of magnetic, electrical or optical properties is read.

Thirdly, reading and writing large amounts of data from storage, such as an exabyte of data, would still require storing a digital copy of the data to be manipulated, processed and read on a computer, which could require conventional storage of the same size as a means of temporary storage. The issue of digital redundancy will exist as long as we are dependent on current computer systems and binary encodings for digital information, or until there are large advances in automating the read and write processes (e.g. mini synthesisers and sequencers that can fit inside a device and read and write to DNA in real time). Nonetheless, current storage capabilities are extremely useful and well-functioning for frequently accessed data, an ideal solution would be to employ DNA storage for archival storage and continue to use and improve traditional storage methods such as HDD and SSD for frequently accessed data.

Lastly, it is extremely hard to delete specific data in DNA; the strand would need to be sequenced and the data needs to be manually deleted before it is synthesised again for storage. Nonetheless, there have been successful attempts in experiments that accommodate for random access in DNA [6,7]. This can help to overcome the issue of deletion and ethics, as random access indexing can be specified as a marker in current sequencing technologies, which avoids sequencing everything else. However, it cannot delete that specific point of the data, or re-write it with a dummy, as DNA is still a WORM device.

1.4 Requirements of A Storage Device

It is pertinent to consider what characteristics underpin a reliable storage device, which also acts as a criterion when evaluating a proposed storage method. Density, durability, access speed, and cost are the most important functional characteristics of a reliable storage device [6]. Although minimising power consumption and achieving energy efficiency of large data centres and storage methods has recently been taken into consideration by large corporations who sell commercial storage, it is often overlooked as an important characteristic, and often sacrificed in favour for operability [8]. Needless to say, it is an important element of building a storage device, as companies must be accountable for their carbon emissions and electronic waste. Accordingly, this study has identified these five characteristics to assess existing or potential storage solutions. This will be used to evaluate DNA storage as potential storage solution.

1. Density; how much data can fit into a unit of physical volume
2. Durability; how long the data can be stored in a device before degrading
3. Access Speed; how long it takes to access the data, considering latency and bandwidth
4. Cost of Storage and Accessing Data; how much it costs to store and access data
5. Power Consumption or Energy Efficiency; how much energy it requires to store information in DNA

1.5 Project Objectives

The focus of this study is to overcome the current limitations of DNA Storage. In particular, this study will focus on reducing the high costs, minimizing data loss, and improving error correcting schemes when storing data in DNA. In order to address these issues and propose a low-cost solution, this study presents DNA Simulated Storage (DNASS), which is a web interface that reconstructs data after data loss, typically caused by the synthesis (writing) and sequencing (reading) processes of DNA. This paper presents a prototype with a three-pronged approach. Firstly, one that is able to provide the functionality of the entire process of storing data in DNA. In particular, a program that mimics errors of substitution, insertion, and deletion without the high costs and lengthy time associated in synthesising and sequencing DNA. Secondly, the paper presents an algorithm that is able to error correct data after an induced data loss of up to 5 percent. Thirdly, the program also aims to provide a platform for experimental testing, which users can employ in experimental studies to examine how different combinations of redundancy factors and error simulations affect data recovery following a specified data loss.

1.5.1 Primary Objectives

1. Create a program that is able to encode data into DNA, and decode it back into its original format.
2. Create an error simulator that can mimic errors in synthesis and sequencing processes, in particular mimic errors of insertion, deletion and substitution. The error simulator should also be able to be customised by the user according to their experimental needs.
3. Create a simple GUI which allows users to:
    - Choose several types of coverage, or redundancy schemes, to compare against each other.
    - Specify which errors to simulate and whether they are continuous errors of a specified length or single character errors.
    - Specify the error rate, which is the amount of data out of the total data the user wants to perform errors on as a percentage.
4. Display a comparison of the results from the chosen simulations. The results should display the size of the input data, the processing time of the script, the encoded DNA and decoded data following data loss.

1.5.2 Secondary Objectives

1. Develop an algorithm to test the simulator with a dummy experiment that attempts to error-correct data loss with the customisable error-recovery combinations.

2. Demonstrate an experiment that could be carried out with DNASS, following a series of customisable data loss simulations with different redundancy schemes.

3. Based on the error-correcting algorithm, display the data retrieval rate post data loss (how much data has been recovered following an error-correcting algorithm) as an addition to the results display in primary objective 4.

4. Allow the program to take in several inputs besides text, such as images, audio files and videos.

1.6 Contribution to Research

The majority of the literature has largely been focused on the possibility of encoding DNA, exploring the storage limits of DNA, and overcoming data losses. Studies have done this by using high levels of coverage or redundancy, approximately between 10 - 3000 copies of the original data [6,9]. Although this has worked for experiments in the literature, it is neither scalable nor efficient to scan through countless redundant copies and manually fix the errors. While the research has been invaluable in confirming the possibility of DNA as a viable storage solution, it is equally important to devise efficient encoding schemes that can improve the density of the data and reconstruct data with low redundancy.

Moreover, both synthesis and sequencing technology is expensive and rare, and requires the encoded DNA to be shipped to an organisation with specific synthesis and sequencing machines before the extracted DNA string can be decoded. Presently, one commercial portable sequencer (Nanopore Sequencer) exists, but has higher error rates than traditional sequencers [10]. Thus, creating a simulator provides a low-cost approach to testing encoding schemes and allows for a thorough investigation on which forms of redundancy have the biggest impact on error-correcting schemes. Moreover, since DNASS is a simulation, it is not constrained by size limits or budgets. Therefore, larger files with higher error rates can also be tested on the platform.

This study will contribute to the research by providing users with customisable redundancy schemes, error types (insertion, deletion or substitution), and error rates to allow users to test which combination of redundancy and simulated errors results in more robust data recovery. This acts as a time-efficient, low-cost pre-cursor to physical DNA storage experiments, which provides assurance that if the error correction methods from the simulation were employed in practical setting, the data will be protected against errors that have been accounted for and tested. This results in smaller need for coverage, as data recovery relies on optimal encoding and error correction.

2 Literature Review

One of the first experiments of storing data in DNA was in 1999, where DNA was employed to hide secret information using steganography. This was achieved through Microdots on paper, which held DNA that was encoded with information [11]. By the early 2010s, there were major breakthroughs in wet lab experiments, with successes in storing data such as images, videos and movies into DNA, paving the way for the exploration of DNA data storage as a potential commercial solution. The largest amount of digital information successfully stored in DNA is 200MB by Organick et al. [7] in 2017. This demonstrates the infancy of DNA storage compared to current storage needs in zettabytes of information, and is ostensibly pending improvements to reach current storage capacity levels.

The literature reviewed in this chapter will be split into two sections. Firstly, the literature will be discussed through the lens of the five characteristics that underpin a reliable storage method, as established in Chapter 1, namely, durability, density, access and latency times, cost and power consumption. Secondly, the literature review will provide an overview of the encoding schemes used by the papers which have been most successful in retrieving data. This section will explore different encoding and error-correcting schemes in the literature, which will dictate design choices in the methodology.

2.1 How reliable is DNA as a storage solution?

2.1.1 Durability

Durability represents how long the data can be stored in a device before degrading or how long before hardware needs replacement. As covered in Chapter 1, conventional archival storage methods are only durable for 10-30 years, and have to be replaced often. Contrastingly, DNA is much more durable for two main reasons. Firstly, its stable double helix structure means its half-life is 500 years [10], which is approximately 15 times longer than current storage technologies. Secondly, it requires minimal climate control or energy in order to function. Grass et al. [12] successfully managed to convert and recover 83 kilobytes of stored data, after performing durability and ageing experiments on the DNA. Specifically, they employed stress and thermal tests on four states of synthetic DNA; freeze-dried, on filter card, in polymer and in silica. They found that the DNA was encapsulated in silica and information was recovered error-free even after treating the silicon at

70 degrees Celsius for one week, which is thermally equivalent to storing information on DNA for 2000 years in a hot climate [12]. The experiment concluded that humidity and water can speed up degradation, and thus preservation of DNA was best in when encapsulated in silica, as it has the lowest local water concentration, and can withstand high temperatures. [13] also found that the degradation of DNA was facilitated by hydrolysis or oxidation, but this could be prevented by storing DNA in a dehydrated container or anaerobic environment at low temperatures.

Analogously, the Svalbard Global Seed Vault is a project that can help to visualise what a DNA storage centre may look like in the future. It is a seed bank project that aims to preserve a wide variety of plant seeds, some of which may be lost forever with climate change. Simply, seeds have genetic information which replicate when they are planted. Therefore, their preservation techniques can be applied to DNA storage, as they are essentially preserving DNA. The seeds have limited access to oxygen and the entire facility is enclosed in permafrost to maintain low temperatures if electricity supply fails. Therefore, this study indicates that DNA has an advantage over current storage technologies with respect to maintenance as well as durability.

2.1.2 Density

Density, in this case, refers to how much data can fit into a unit of physical volume. There have been a range of suggestions about the density limits of DNA. While Ceze et al. [10] claim DNA has a theoretical limit of 1 exabyte (1 billion gigabytes) per gram of DNA, Grass [12] suggest that DNA is capable of storing 17 exabytes of data per gram. The information density of DNA is measured against how many conventional bits of data can fit into a nucleotide base. Since all information is currently encoded in binary, this will be the medium to measure density, unless there is a unique mapping of plaintext or digital media directly to DNA code (such as an ASCII for DNA).

Kohll et al. [13] contends that information density (how many bits can be stored per nucleotide) is dependent on a number of factors, namely, redundancy, error detection and correction information. Information density cannot be evaluated independent of these factors as there is a trade-off between reliability and density. For example, while redundancy and parity checks decrease the information density, it increases the data recovery rate.

Similarly, Bornholt et al. [6] emphasises on the trade-off between reliability, storage density and performance. They argue that the way to ensure reliability is to compromise density by physically isolating pools of encoded information in DNA. As a result, coverage or redundancy is increased and random access is provided as long as the pools are isolated in a purposeful fashion, where they can be indexed by location.

2.1.3 Access times

Access times signify the latency and bandwidth of accessing data. Traditional storage methods have faster access times compared to synthesis and sequencing technology. As seen in Table 1, it takes several hours to complete a single request in DNA whereas magnetic tape is much faster and can return all terabytes (bandwidth allowing) in a few hours. In an experiment using the Oxford Nanopore portable sequencer, the sequencing process was 24 hours at a speed of approximately 75 basepairs / second (bp/s). Moreover, accessing and reading DNA is a lengthy process with other sequencers. For example, in order to read data using the Illumina sequencer, the DNA must be shipped to their lab to be sequenced, and has to be preserved in transit.

Technology	Access Time	Cost /MB	Durability	Unit Capacity
Hard Disk Drive (HDD)	7ms	0.00002	∼5 years	2TB
Flash Memory (SSD)	10ns	0.0005	∼5 years	800GB
Magnetic Tape	Minutes	0.0001	∼15 - 30 years	15TB
Synthetic DNA	>10 hours	>$3000	∼500 years	22PB - 17EX

Table 1: A comparison of DNA against current storage technologies [2,3]

2.1.4 Cost of storage and accessing data

Moore's Law is a law of observation that suggests that every 2 years, transistors on a computer microchip will double, while the cost will simultaneously decrease by 50 percent [6]. Comparatively, the cost reduction of sequencing technology in relation to information density in DNA surpasses even Moore's Law [6]. Moreover, there are new technologies such as Nanopore MinION, which is a portable, real-time sequencer and has a fixed cost that ranges from 500 - 4000 USD depending on additional features, used to sequence DNA fragments on-the-go, and has been used in experiments [5,14] in order to sequence digital data. The trend of lowered costs and the advancements of portable technologies provides a positive outlook on the viability of DNA storage in the foreseeable future. The Nanopore device (Figure 2) is the size of a palm, providing hope for standalone data storage, such as DNA computing or DNA fast-access storage in the future, if computer devices can have built-in synthesisers and sequencers.

The costs of accessing data are higher than the costs of storage. There has been a 50,000-fold decrease in the price of sequencing DNA, from 31,250 USD in 2002 per megabase (Mb) to 0.63 USD per megabase in 2016 [14]. However, synthesis costs remain extremely high. Goldman et al. [2, 15] spent 98 percent of their total costs, amounting to 12,660 USD, on synthesis and only 252 USD on sequencing. More recently, Erlich and Zielinski [16] spent a total of 3500 USD to synthesise 2.14MB of digital data into DNA, and 1000 USD to sequence the data [16]. Nonetheless, Goldman et al. [15] estimates DNA-based storage can become practical for archives in less than 50 years if sequencers and synthesisers continue to improve and follow the historical trajectory of lowered costs presenting a path to overcoming the high costs of DNA storage.

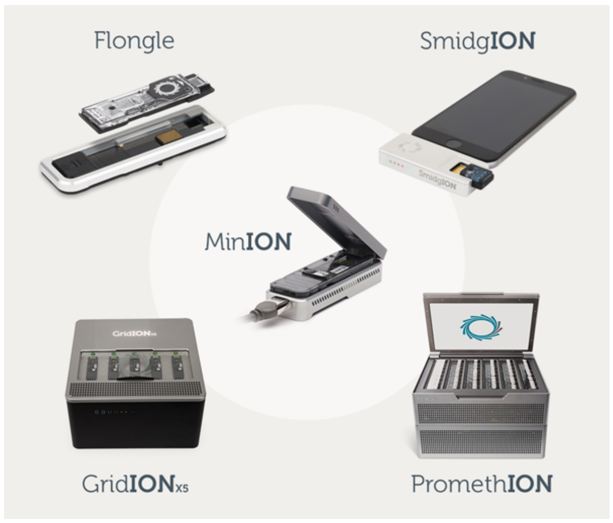


Figure 2: Range of Oxford Nanopore Sequencers

2.1.5 Power consumption and energy efficiency

Grass et al. [12] and Kohll et al. [13] found the best way to preserve DNA is in a dehydrated container or anaerobic environment at low temperatures.

If the theoretical limits of DNA are in the exabyte range, all of the world's information could be stored in a fridge-like device that is cold and dehydrated. This would cut a large portion of traditional storage costs, such as HVAC and electricity supply to servers.

2.2 A Comparison of Encoding and Error Correcting Schemes in DNA Storage Experiments

Most successful attempts at storing digital data in DNA had differing approaches in their encoding and error-correcting schemes. In this section, a range of successful DNA experiments which had the highest levels of storage and retrieval are outlined. In particular, each experiment's encoding, error detection error correcting methods is presented. All the studies reviewed in this chapter are summarised in Table 2. It should be noted that the most popular sequencing technologies, are Illumina HiSeq and the Oxford Nanopore minION.

Goldman et al. [15] were successfully able to encode and recover 739kB in 2012 [15]. The data was encoded into 153,335 strings, each containing 117 bases. The encoding method involved converting the information to base-3 using a Huffman code, which is a form of lossless compression which uses frequency analysis to compress the data. Each fragment was overlapped with other fragments, and alternate fragments were converted to their reverse complement. This was done in order to reduce the chance of data loss, which could lead to unrecoverable data. They also ensured there were no homopolymers (two identical strings) in the data strings. Homopolymers are associated with higher error rates in sequencing technologies and lead to errors. Each segment was also indexed with a parity-check for error-detection.

Although Goldman et al. [15] stored roughly 51 copies of all the encoded DNA, they immediately discarded strings with errors. Their focus was not to create a sophisticated error-correcting scheme, but rather achieve high levels of redundancy and sequencing coverage by all samples until they were able to recover the information from strings that were fully intact. Although this experiment was a successful DNA storage experiment, there was significant coverage overhead, heavy processing and analysis of strings to test whether they were viable for decoding, which could have been avoided with appropriate error correcting schemes.

Study	Data Size	Redundancy / Coverage	Can decode when multiple strings have errors	Random Access	Error Rate / Data Loss	Sequencer
Goldman et al. [15]	739 Kb	51x	No	No	1%, ∼1/500 bases missing	Illumina
Bornholt et al. [6]	151 Kb	40x	No	Yes	1.5%	Illumina
Church et al. [9]	650 Kb	3000x	No	No	0.000002%	Illumina
Erlick and Zielinksi [16]	2.11 MB	10.5x	No	No	1.3%	Illumina
Blawat et al [17]	22 MB	160x	No	No	0.0014%	Illumina
Organick et al. [7]	200 MB	5x	Yes	Yes	12%	Nanopore MinION

Table 2: A comparison of discussed papers [2, 7]

On average, Goldman et al. [15] found there is roughly 1 error per 500 bases, which translates to a 1 percent error rate for their sample size. One of the errors was two gaps, each a run of 25 bases, and this had to be reconstructed using manual intervention of inspecting neighboring regions to reconstruct data, after which the sequence could be decoded. They achieved an information storage density of 2.2PB per gram of DNA.

Bornholt et al. [6] conducted an experiment that built on Goldman's encoding [15], but adapted it to reduce redundancy and introduce primer sequences for random access by mapping a key to a pair of primers. They encoded four image files in 45,652 sequences, each 120 bases long, totaling 151kB of data. As with Goldman, they contended that the likelihood of some errors can be avoided by encoding binary data into base 3 first and then mapping those to DNA. More common ASCII characters are mapped to 5 digit strings, which have compression benefits. They incorporated a simple XOR (Exclusive-OR) encoding for the data by prioritising redundancy for critical data. Important information, such as JPEG headers were bound to other strings and had increased coverage. Whereas less critical information had lowered redundancy. Instead of four-fold redundancy, their DNA strings repeats an average of 1.5 times. Overall, their coverage was 40x. Simulation results found that their encodings were twice as dense as that of Goldman, with equivalent levels of reliability.

They also segmented the DNA strings into blocks, and tagged those with primers in order to perform random access and isolate strings. The payload (information about the data) is broken down into blocks. The payload in each DNA string contains addressing information that can identify the primer targets for sequencing. A parity nucleotide is added for basic error detection. The primers work through selective amplification (the sequencer only sequences strands with a specified primer sequence). This helps achieve random access, as the entire pool does not have to be sequenced to recover some data. This also speeds up sequencing and is cheaper, as only the desired data is recovered.

From their experiment, [6] found that synthesis errors grow with the length of the strand, and nucleotides towards the end of the strand are more likely to be incorrect. Therefore, reverse complements or reverse strands are stored to provide accurate decoding. Out of 20.8 million reads from the sequencer, 8.6 million were error free reads, but could be corrected with redundant copies. Three out of four files were recovered without manual intervention, and the other incurred a one-byte substitution error in the JPEG header.

Church et al. [9] converted a book of 53,426 words, 11 JPEG images and 1 javascript program into 54,898 DNA strands, each 159 nucleotides each, where each strand represented a 96-bit data block. The raw data totaled a size of 650 KB without coverage. In order to reduce errors from sequencing 100 bases in each string were overlapped. The data was stored with approximately 3000-fold coverage. All the data was recovered with a total of 10 bit errors out of 5.27 million bits, which were due to homopolymers at the end of a string where there was only single sequence coverage. One bit was used per based (A or C for 0, G or T for 1). This is not a very dense approach as the number of strands could be reduced by half by assigning two bits to a single nucleotide (for example, A for 00, C for 01, T for 10, and G for 11). However, the study's chosen method was advantageous in that it helped to avoid repeats.

Erlich and Zielinski [16] managed to successfully store 2MB of data which included a full computer operating system and a gift card. They achieved a Shannon information density (glossary) of 215 PB per gram of DNA. They employed fountain codes to encode the information into DNA, in particular Luby transform codes. A fountain code works by taking data as an input and encoding it into an unlimited number of blocks, called droplets. Given a subset that is larger than the original data size, the original file can be retrieved. Each droplet contains a data payload and a seed. The seed is a number randomly generated during the creation of the droplet and allows the decoder to infer what the data represents in relation to all the other data. The droplet was converted to a DNA sequence by mapping {00,01,10,11} to {A,C,G,T}, respectively. It also then went through another check that ensures there are no homopolymers, and then it was synthesised. Each droplet represented 38 bytes of data. The data payload in each droplet had a Reed-Solomon code (appendix), which informed the algorithm whether to use the droplet or not. Only fully intact droplets were used to decode the information. Overall, 97 percent of strings were used to decode the data, and 1.3 percent of information was lost during the sequencing process. However, their algorithm was able to recover this using the fountain codes and redundant information.

Blawat et al [17] was successfully able to decode 22MB worth of information, the second highest that has been achieved after Organick et al. [7]. Their encoding was a forward error correction scheme, which can handle errors of insertion, deletion and substitution, based on the encoding by Church et al. [9]. They mapped the first six bits of a byte into a single value by mapping

$\{00,01,10,11\}$ to $\{A,C,G,T\}$, respectively, and then encoded the last two bits into a pair of nucleotides (AA to 00, AC to 01, AG to 10, and AT to 11). This was to ensure that the first three nucleotides and last two nucleotides should not be the same, which would help in creating a set of "valid bytes", which were useful in identifying corrupt bytes.

They synthesised 900 million stands, each 230 nucleotides long. They also employed Reed-Solomon codes for error detection, and they were able to reconstruct their data with a 160-fold coverage. They found that the residual error probability in sequencing and synthesis is similar to the error rates in hard disk drives (HDDs). They found that substitution error rates occurred between 0.0006 - 0.0014 percent of the time, while insertion error rates are 0.0001 percent and deletion error rates are 0.0005 percent.

Organick et al. [7] conducted the most successful study to date as they were able to store and decode the largest amount of data, totaling 200MB of compressed data from 35 files. They had 15 percent of added logical redundancy for error correction, which resulted in an additional 32.2 MB of encoded DNA. There were over 13 million unique DNA strings, where each was between 150-154 bases. Similar to previous work, they employed Reed-Solomon codes for error detection and adds redundant information for error correction. All DNA sequences were appended with primers to allow random access indexing for files, and by using selective amplification for reads, as with Bornholt et al. [6]. Unlike Goldman [15], they did not discard reads with errors, and instead used an algorithm they called trace reconstruction, which decodes the information by comparing it to the redundant DNA sequences. They found that substitutions are the most prominent type of error, occurring 40 percent of the time, twice as likely as deletions, which occur 20 percent of the time, and ten times as likely than insertions, which occur 4 percent of the time. Moreover, they found that first and last 20 positions do not have many errors, which differs from Bornholt et al. [6], who found that errors tend to be concentrated towards the end of the DNA string.

2.3 Summary

From the first section, it is clear that DNA is dense, durable and energy efficient storage method. However its current high costs and slow access speeds are far worse than any current technology which prevents DNA storage from meeting the full requirements of a reliable storage device (Section

1.4). Nonetheless, previous trends of cost deductions and improvements in synthesis and sequencing technology demonstrate that both cost, access times and latency will improve for DNA in the future, implying that there is great potential and that it is valuable to explore despite the stated barriers that this nascent technology will need to overcome.

The second section can broadly be summarised into three relevant sections; error characterisation, encoding schemes, and error-correcting methods. Errors present themselves as either insertions, deletions or substitutions. These are often dispersed along the DNA string, but can also be concentrated at the end of the DNA strings. It is further observable that higher error rates can be caused by homopolymers; repetitions of each string, and this type of error tends to increase as the length of the strands increase.

Previous successful efforts to conduct experiments on DNA storage have heavily relied on large amounts of redundant information to ensure data recovery needs are met. This involves a lengthy process of scanning through redundant copies and comparing them to the original to retrieve data. Despite the heavy time investment, the literature states this is offset by the gains in density as DNA is more space-efficient. Nonetheless, storing up to 3000 copies of redundant information with DNA storage is an expensive undertaking and most strings with errors are discarded. Furthermore, it is neither cost-effective nor computationally efficient to scan through multiple copies of data to identify intact copies of DNA strings. While these studies lay the foundation for how to successfully store and retrieve data using DNA, they display a simultaneous and pressing need to generate more efficient error-correcting algorithms as redundancy and error-correcting schemes on real sequencers and synthesizers is not cost-effective.

Taking into consideration the above issues of cost and the lack of primary focus on error-correcting schemes in this field, this study presents DNASS, a customisable application for testing error correcting schemes against simulated errors of substitution, insertions and deletions. This works towards the objective of creating a platform where encoding schemes can be tested and simulated without the high costs and time associated with DNA storage. Moreover, to demonstrate this simulator, this study will develop an error-correction algorithm, which will attempt to test which combination of errors and redundancy provides the most robust error recovery scheme for given hypothesis. This is in the hopes that future experiments of DNA storage can implement encoding schemes with optimal redundancy that protects them against errors before conducting a physical experiment.

3 Methodology and System Requirements

This chapter presents the requirements of the program as well as outlines the software development process. The methodology aims to reflect the findings from the literature review and project objectives (Section 1.5).

3.1 Requirements Specification

This section presents the functional and non-functional requirements of DNASS. This is organised into four main components: the encoder, the error simulator, the error correction, and the decoder.

3.1.1 Functional Requirements

Encoder

1. The system must be able to convert data into binary and map the data into a DNA string that consists only of {A,G,C,T}.
2. All additional information, such as length, parity bits, or redundant information, must be converted into DNA before going through the error simulator.
3. The application should take in any text input and be able to encode it.

Error Simulator

1. The system must mimic all errors that are found in a typical synthesis and sequencing process, namely, substitution, insertion and/or deletion errors.
2. The system must mimic errors outlined by previous studies, which are summarised as follows:
   - Errors occurring only the end of the string
   - Long runs of errors.
3. This component must simulate errors on the same error combination as the original DNA string on all types of redundant copies, which are summarised as follows:
   - Original: This is original copies of the DNA string.

- Complement: This is the complement of the DNA string.
- Reverse: This is the original DNA string, stored in reverse.
- Reverse Complement: This is a copy a complement DNA string, which is also reversed.
- Fixed Length Strings: This is a DNA string that has been delimited by a primer.

4. The error simulator component must be isolated from previous components.

Error Correction

1. This component must be isolated from previous components, it must employ only the result from the error simulator, which is a DNA string. In the functions that decode or error correct, it will not be able to access variables or functions before the error simulator or from the error simulator. For example, the decoder function should not have information about how many errors there are, as a sequencer cannot identify this either.
2. This component should compare all redundant copies to output a result that has attempted to correct the data that has been identified as corrupt.

Error Correction

1. This component should decode the results using the exact reverse mapping of the encoding scheme.
2. This component should be able to convert the data back to binary and then to its original data representation.
3. This component should display a comparison between the original text input and the version(s) returned from the error correction component.
4. This component must generate and display the data retrieval rate, which is percentage value of how similar the recovered text is to the original text.
5. This component must also display how big the size input is in kilobytes.

3.1.2 Non-Functional Requirements

1. The application will be accessible from different browsers and platforms
2. The system will be performing optimally in order to produce consistent results.

3.2 Software Development Process

This section outlines how the software development process unfolded. With little biological knowledge of DNA and a lack of clear direction due to ever-changing constraints due to covid-19 disruptions, an iterative software development process was adopted, to continually ensure the software could adapt to changes in the requirements due to constraints.

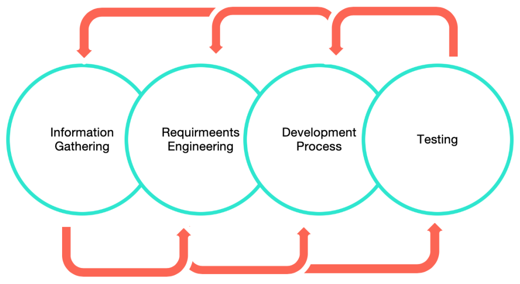


Figure 3: Software Development Process

1. Information Gathering; since DNASS is a simulation, it heavily relies on the literature to dictate how the program should function, so that it can mimic real experiments of DNA storage as accurately as possible. This research laid the foundation of engineering requirements, as a gathering insights was essential to gauging what functionality DNASS should have.
2. Requirements Engineering; the requirements were collated based on the research and filtered based on what could be achieved in time, and a select number of functionality was prioritised to be implemented
3. Development Process; the implementation of the software was done with constant validation against the literature, to ensure that the error

simulator was accurate. Once a component was tested, the sequential component would be built. Each change in direction of algorithmic strategy was recorded in GitHub for version control.

4. Testing; informal testing occurred throughout, to validate each component, by running the program numerous times with different test cases to test for consistency. At the very end, use case testing and formal testing was carried out.

This process then continued with constant feedback, as shown in Figure 3. The development approach was more exploratory than rigid, but it allowed for flexible changes throughout the project.

4 Design and Implementation

This chapter provides a brief description of the selected programming language and tools employed, as well as describes in detail the design and implementation choices of the application, organised by the four main components of the artefact: Encoder, Error Simulator, Data Recovery Algorithms, and Decoder. The whole process of this algorithm is outlined in Figure 4.

4.1 Tools and Language

DNASS is built with JavaScript and HTML/CSS, as it is the most suitable for building a web application with a simple GUI. Moreover, it is an extremely useful language for string manipulation. For example, there are many built-in functions such as testing for equality, removing substrings, inserting characters at different indexes, cloning variables easily, as well searching through strings using regular expressions. This is useful as DNA strings need to be manipulated at different indexes in order to accurately mimic data transformation caused by errors in synthesis and sequencing machines. Trello was also used to keep a constant track of the product backlog, and GitHub was employed for version control.

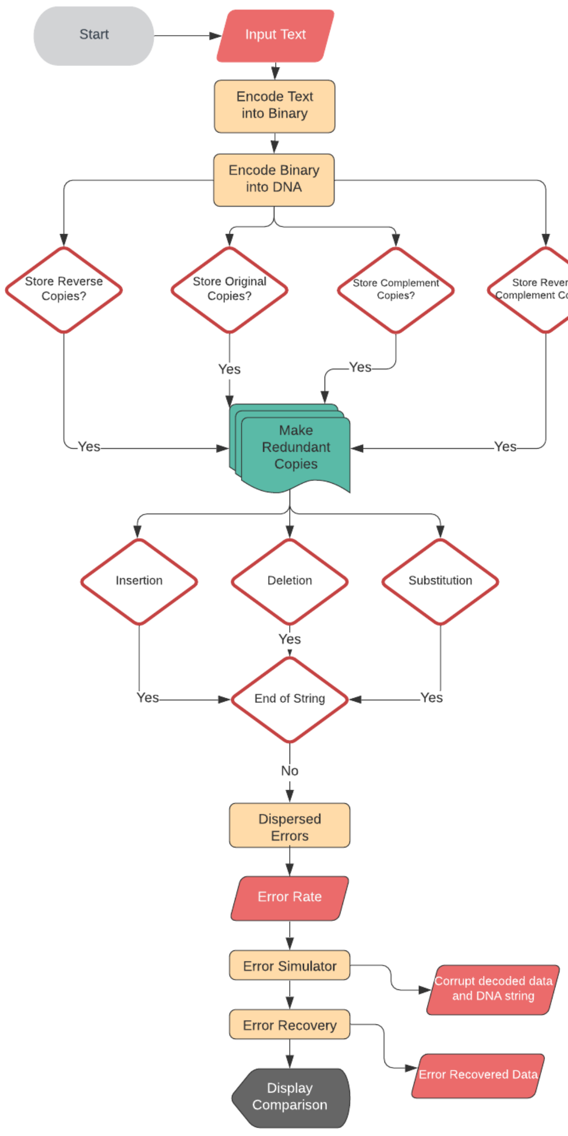


Figure 4: Software Development Process

4.2 Encoder

4.2.1 Encoding into DNA

Firstly, inputted information is converted to unique ASCII character codes, where each character is represented by a byte of information. The bytes of information are extracted from the character codes, delimited by a space. Secondly, the binary representation of the data is mapped to DNA bases. This prototype uses a mapping of {00,01,10,11} to {A,C,G,T} respectively, as this is the most versatile mapping, even though does not result in the most efficient bits per base. For example, mapping different characters to combinations of ACGT could produce a more compressed result, as frequency analysis can be taken into account, however, this sort of mapping would only be limited to text inputs, as it would compress characters only.

4.2.2 Storing the Length

In order to identify errors, the length of the string is also stored at the end, so that the current length can be compared to the original length to find out how many deletion or insertion errors have occurred. This is stored in 4 bases at the end of the string, which are translated to a byte when decoded. If the length is 3560 characters, the decoded length will be 00003650. The additional padding at the beginning has two benefits. Firstly, it helps easily extract the length, by slicing and decoding the last four bases of the DNA string. Secondly, it means lengths of up to 99,999,999 can be stored.

4.2.3 Adding Redundancy

There are five different types of redundancy that the user can select as duplicates for their inputs, namely, original copies of the data, complement copies, reverse copies and reverse complements copies. A combination of these cannot be selected, they can only be selected individually and compared, while controlling for the same error rate and error types. When selected, this component generates redundant copies of the data and transforms them into the specified form. For 'Original', this component creates two additional instances of the original encoded DNA string. For 'Complement', the DNA string is sent through a function that returns the complement string. For example, if a string is 'ACCGTAC', the complement will be 'TGGCATG'. The complement is useful to protect against errors of sequencers or synthesizers which tend to discriminate or corrupt certain bases. Storing the complement ensures some of this data can still be recovered. For the 'Reverse', the string is transposed, where the first element of the array becomes the last element and so on, until the array is reversed. This is useful for errors that tend to occur at the end, so that the end of the DNA string is likely to be preserved in the reverse copy. The 'Reverse Complement' combines the last two processes and results in a string that is the complement of the original and in reverse. The reverse complement is also stored to test whether this increases the chances of recovering end of string errors. Figure 5 displays the implemented GUI, which shows all the customization choices the user has.

The final mechanism is fixed length strings, which is when a DNA string is split into chunks and delimited by a primer. The contents of the primer can be arbitrarily selected, as they have to be encoded in one of the four bases. This function delimits chunks by a string of five T's (TTTTT), as the randomly selected base. Users can also specify the length of the blocks, as seen in Figure 5. The idea behind storing fixed-length strings is to isolate the error, so that it doesn't adversely affect or corrupt the data that follows as each block is error corrected individually, and not in relation to the whole string. For example, if an error occurred in one block and could be identified, the rest could be error corrected and decoded separately. Primers work in conjunction with fixed length strings when used purely for error recovery, as they identify where a string begins and ends implemented. However, these are also used for random access indexing, as primers can have unique values that correspond to keys, allowing to search through large files or long strings of DNA.

Only three copies of the data are used for recovery, as this was the most efficient way to error correct at low redundancy rates. The justification for this is provided in the error recovery component. For each redundancy scheme that has been checked off, the error simulator will receive one copy of the original and two copies of the redundancy scheme that has been selected. If the original is selected, three redundant copies of original will be stored.

4.3 Error Simulator

The error simulator mimics the process of synthesising and sequencing the error. In reality, a synthesiser will arrange the structure of chemical molecules (nucleotide bases) so that they resemble the desired encoded strand. When sequenced, the machine will read the structure of the chemical molecules and provide a reading of the DNA. As aforementioned, this process tends to yield three types of errors: insertion, deletion and substitution. The error simulator in the artefact induces these errors on a DNA string at an error rate specified by the user. For example, the user could select ‘insertion errors that insert two characters next to each other per error, at a 5 percent error rate, where errors occur towards the end of the string’ (as seen in Figure 5).

In order to generate errors, the number of errors to induce on the string is calculated using the error rate specified by the user and the length of the string. For example, if the length is 15000, and the error rate is 1 percent, there will be 150 errors on the DNA string. It should be noted that this variable is not accessible after the error simulator, and the user can only rely on the error correcting algorithm to deduce how many errors and where they have occurred. This is to simulate a real sequencer, where the information can only be extracted from the DNA string, unless there is digital redundancy stored elsewhere. This simulation will assume that there are no digital copies to provide this information.

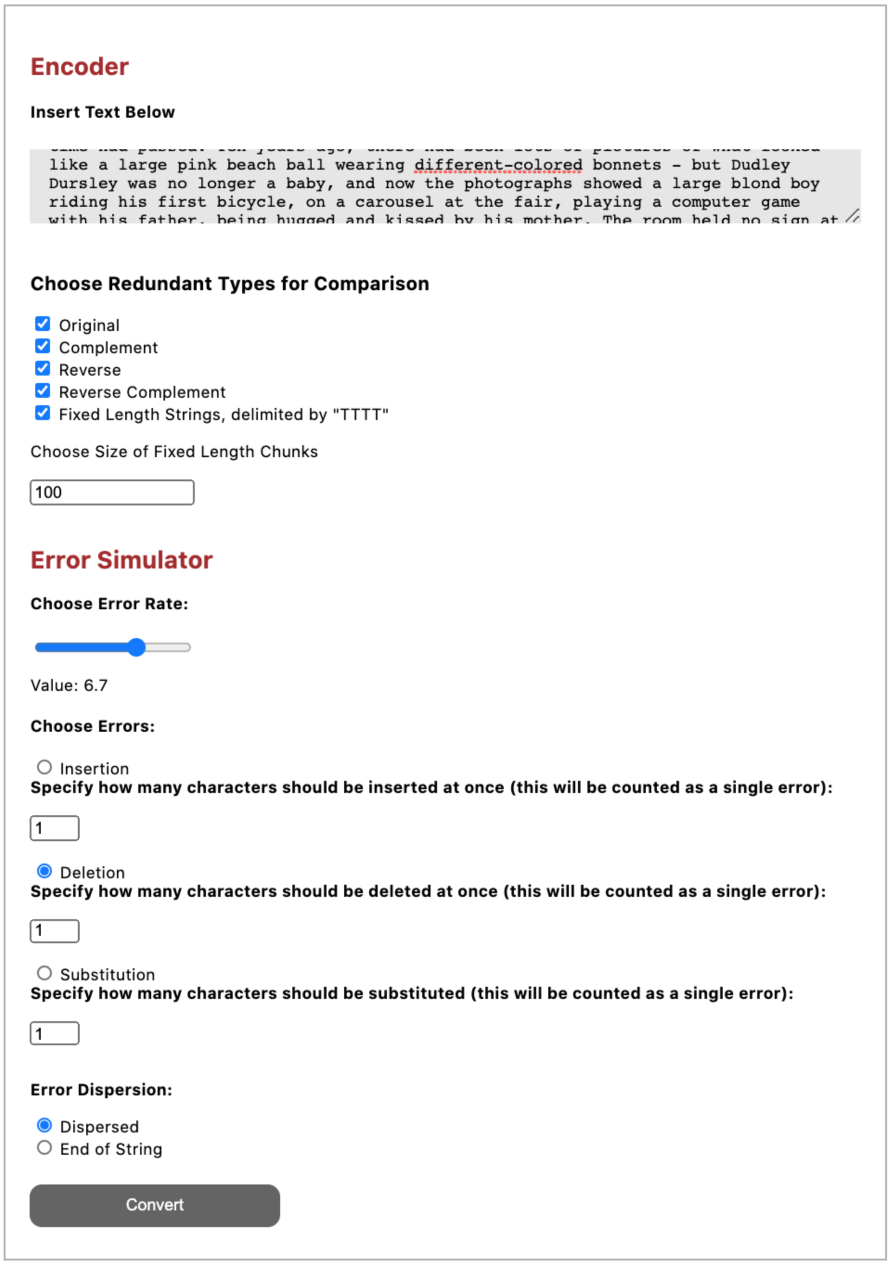
The number of errors also indicates how many iterations of the following procedure to run:

1. Generate a random index between a specified range. For dispersed errors, this is (0, length of DNA string). For errors that occur towards the end of the string, the range is specified as the last 10 percent of the string, based on the literature [6]. This will result in errors that are concentrated towards the end of the string.

2. The algorithm then checks what errors are selected.

3. If deletion errors are selected, the algorithm will remove the number of characters specified by the user at the random index position generated.

4. For insertion and substitution errors, a random base is also generated out of [A, C, G, T], which is used to insert or substitute the number of errors specified by the user, at the random index position generated.



Encoder

Insert Text Below

```
like a large pink beach ball wearing different-colored bonnets - but Dudley
Dursley was no longer a baby, and now the photographs showed a large blond boy
riding his first bicycle, on a carousel at the fair, playing a computer game
with his father, being hugged and kissed by his mother. The room held no sign at
```

Choose Redundant Types for Comparison

- [✓] Original
- [✓] Complement
- [✓] Reverse
- [✓] Reverse Complement
- [✓] Fixed Length Strings, delimited by "TTTT"

Choose Size of Fixed Length Chunks

100

Error Simulator

Choose Error Rate:

Value: 6.7

Choose Errors:

○ Insertion
Specify how many characters should be inserted at once (this will be counted as a single error):

1

[✓ Deletion]
Specify how many characters should be deleted at once (this will be counted as a single error):

1

○ Substitution
Specify how many characters should be substituted (this will be counted as a single error):

1

Error Dispersion:

[✓ Dispersed]
○ End of String

Figure 5: Screenshot of GUI Interface for User Selection

Based on the literature, the error rate of the synthesis and sequencing process in reality is usually between 0.00002 - 1.3 percent (Table 2). Redundant copies are treated as different reads of the data, so they too pass through an error simulator of a specified rate before being decoded.

4.4 Data Recovery

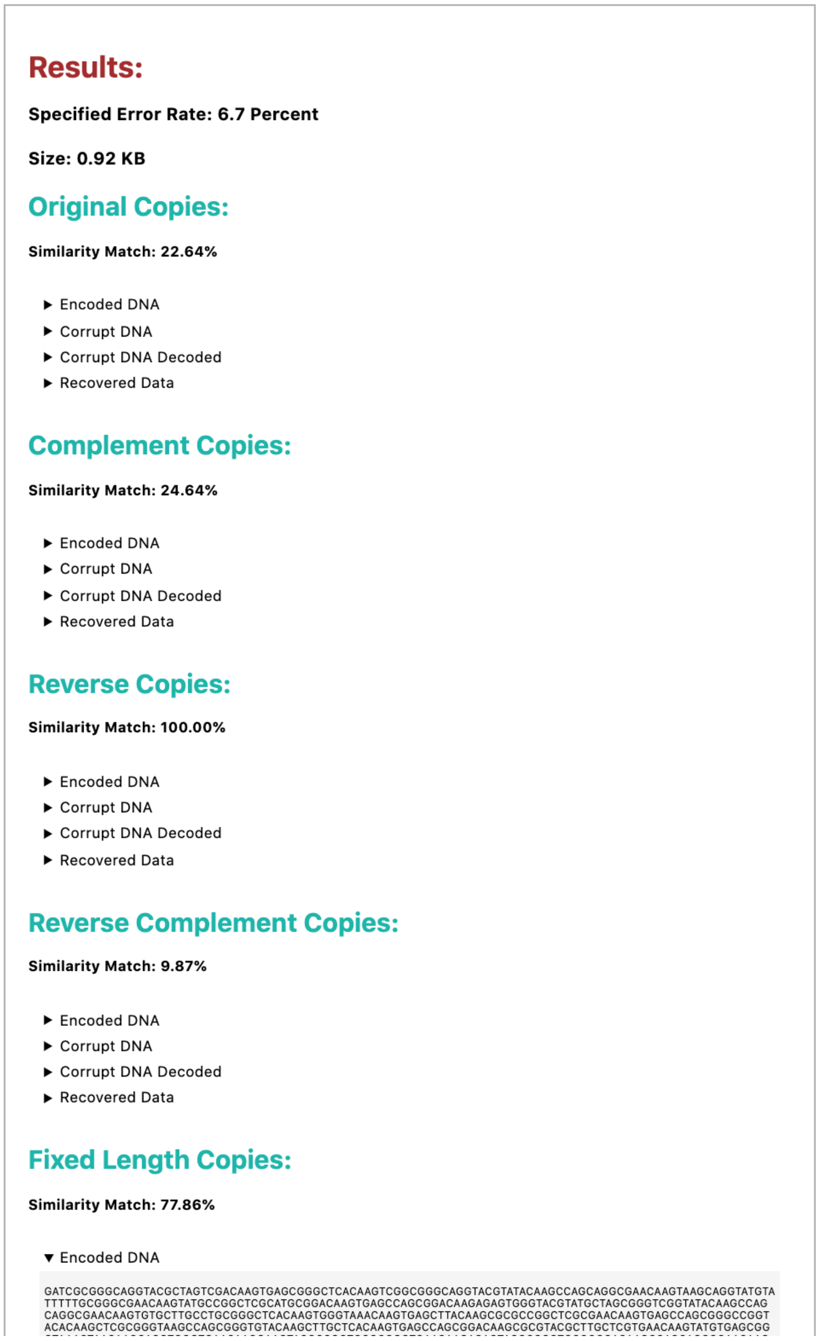
The data recovery algorithm has different variations for each combination of factors. This component can only use information extracted from the DNA string. For redundant information, three copies of the data are used to rebuild any lost information and correct errors sequentially. The error correcting process is a three-step process. Firstly, the length is extracted from the end of the string and passed it through the decoding algorithm to find its binary values and then map it back to ASCII characters. The length is then used to gauge how approximately how many errors have occurred, by comparing the stored length to the current length of the DNA string that has been received by the simulator. If the original length is larger than the extracted DNA length, this indicates there has been deletion errors in the data. If the original length is smaller, this indicates there has been insertion errors. The difference between the two are then used to indicate how many iterations of the following error correcting procedure the algorithm should carry it. For example, if five errors are found and there are three arrays, the algorithm will scan through all arrays fifteen times.

This is so that the algorithm can correct errors in all of the arrays. Each iteration involves scanning through all arrays of DNA and then stopping at an index where the values of all arrays at that point don't match. For example, the loop will stop if any of the arrays have different values, such as {A, G, A}, or {A, G, C}, where each base represents the same index position in three different arrays. In the first scenario, if the next index for all of them have the same value, this indicates that a deletion error has occurred here. In this case, an 'A' will be inserted at this position, as the other two arrays are 'A'.

In the second scenario, all arrays at that index position will be replaced with an 'X', as it is assumed two or more errors have occurred in the same place. This acts as a marker of where the error has occurred, so that it can be fixed later. A post-processing algorithm can test out all combinations of bases at the positions marked 'X' to evaluate whether it can be recovered and compare it to the original string. However, since this study assumes no digital copies of the data is stored, it does not rely on digital copies to check for similarities, and therefore this post-processing algorithm is not implemented. Substitution errors are harder to correct without manual intervention, even if the index position of the substitution error is known, only trial and error of the other bases can help determine what the base was substituted with, as well as comparing to the original string.

Three arrays were chosen as the amount of coverage based on the hopes that there are errors at one position, the other two can correct them. This is based on the assumption that if two arrays have the same value at a given index position, then it provides assurance of what the third element at that array should be. The same process is used for complements, reverse and reverse complement arrays, but are converted back into the original string before this process, so that they can be compared against the original redundant copy in this stage.

For fixed length strings, the length of strings encoded in the length, so if any blocks are shorter, it highlights there is an error in that block. However, the only way to correct for the error, if used on its own, is also by trial and error at different positions. An easier way is to compound fixed length strings with redundant information, so that it can be compared in the same way as above to find the missing or substituted bases.



Results:

Specified Error Rate: 6.7 Percent

Size: 0.92 KB

Original Copies:

Similarity Match: 22.64%

- ▶ Encoded DNA
- ▶ Corrupt DNA
- ▶ Corrupt DNA Decoded
- ▶ Recovered Data

Complement Copies:

Similarity Match: 24.64%

- ▶ Encoded DNA
- ▶ Corrupt DNA
- ▶ Corrupt DNA Decoded
- ▶ Recovered Data

Reverse Copies:

Similarity Match: 100.00%

- ▶ Encoded DNA
- ▶ Corrupt DNA
- ▶ Corrupt DNA Decoded
- ▶ Recovered Data

Reverse Complement Copies:

Similarity Match: 9.87%

- ▶ Encoded DNA
- ▶ Corrupt DNA
- ▶ Corrupt DNA Decoded
- ▶ Recovered Data

Fixed Length Copies:

Similarity Match: 77.86%

- ▼ Encoded DNA

```
GATCGCGGGCAGGTACGCTAGTCGACAAGTGAGCGGGCTCACAAGTCGGCGGGCAGGTACGTATACAAGCCAGCAGGCGAACAAGTAAGCAGGTATGTA
TTTTTGCGGGCGAACAAGTATGCCGGCTCGCATGCGGACAAGTGAGCCAGCGGACAAGAGAGTGGGTACGTATGCTAGCGGGTCGGTATACAAGCCAG
CAGGCGAACAAGTGTGCTTGCCTGCGGGCTCACAAGTGGGTAAACAAGTGAGCTTACAAGCGCGCCGGCTCGCGAACAAGTGAGCCAGCGGGCCGGT
ACACAAGCTCGCGGGTAAGCCAGCGGGTGTACAAGCTTGCTCACAAGTGAGCCAGCGGACAAGCGCGTACGCTTGCTCGTGAACAAGTATGTGAGCGG
```

Figure 6: Screenshot of Results for comparison

4.5 Decoder and Results

Finally, the decoder is able to map the different bases {A,C,G,T} back into their binary representations, and then maps it to ASCII character codes, and then they can be decoded back into their original format. The results are then displayed to the user, which includes three metrics:

1. Size of the input; this presents the size of the input file in megabytes (1byte = 0.001 Kb)

2. Data recovery rate; which is how closely the result matches the original text. This is generated with an algorithm that examines each character and its lengths and examines how closely they match each other. This is presented as a percentage of how many characters match the original text at the right positions.

3. Computational efficiency; to give the user an indication of how computationally efficient the algorithm is.

5 Testing, Evaluation and Limitations

This chapter is divided into three sections. The first section provides a use case test for the software, which includes a demonstration on how DNASS can be employed in an experimental study. The second section presents results from integration testing, which aims to test the robustness of the program as a whole. The last section evaluates the whole project against the project objectives outlined in Chapter 1, and the limitations of the study are discussed. This chapter has the aim of estimating the success of DNASS.

5.1 Use Case Testing

Use case testing is a demonstration of how software responds to user action. This type of testing identifies gaps in the application that can be left undiscovered with individual unit testing. This section provides a demonstration of a hypothetical experiment. The purpose of this demonstration is to highlight how the application could be used and what results could be generated with a specified hypothesis. This is to test and evaluate the robustness of the program from a user-centered point of view.

The sample hypotheses are outlined below.

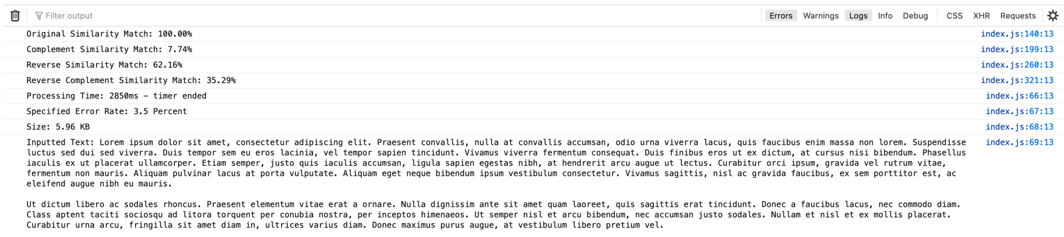
H0: At 1.5 percent error rate, there is no difference between storing reverse copies and reverse complement copies for deletion errors of single characters, which are concentrated at the end of the string.

H1: At 1.5 percent error rate, there is a significant difference between storing reverse copies and reverse complement copies for deletion errors of single characters, which are concentrated at the end of the string.

For the sake of this experiment, the test of the specified controls in the hypothesis is carried out 5 times, on 4 different file sizes. The results displayed in Table 3 are averages of each trial with the same file size, which were natively downloaded from the console (These are attached with the code files). Figure 7 shows the console display following an experiment. The full logs of all 20 trials can be found in Appendix A. Moreover, text inputs were generated using [19], which is a 'Lorem Ipsum' random text generator based on file size. The results are summarised below.

File Size	Average Processing Time (minutes)	Average Data Recovery Rate – Reverse	Average Data Recovery Rate – Reverse Complement
5kB	0.01	100%	100%
50kB	1.59	48.60%	27.70%
100kB	7.88	52.49%	21%
150kB	18.80	15.98%	2.29%

Table 3: Average Results of Demostration Experiment



```
Original Similarity Match: 100.00%                                   index.js:140:13
Complement Similarity Match: 7.74%                                   index.js:199:13
Reverse Similarity Match: 62.16%                                     index.js:260:13
Reverse Complement Similarity Match: 35.29%                          index.js:321:13
Processing Time: 2850ms – timer ended                                 index.js:66:13
Specified Error Rate: 3.5 Percent                                     index.js:67:13
Size: 5.96 KB                                                         index.js:68:13
Inputted Text: Lorem ipsum dolor sit amet, consectetur adipiscing elit. Praesent convallis, nulla at convallis accumsan, odio urna viverra lacus, quis faucibus enim massa non lorem. Suspendisse   index.js:69:13
luctus sed dui sed viverra. Duis tempor sem eu eros lacinia, vel tempor sapien tincidunt. Vivamus viverra fermentum consequat. Duis finibus eros ut ex dictum, at cursus nisi bibendum. Phasellus
iaculis ex ut placerat ullamcorper. Etiam semper, justo quis iaculis accumsan, ligula sapien egestas nibh, at hendrerit arcu augue ut lectus. Curabitur orci ipsum, gravida vel rutrum vitae,
fermentum non mauris. Aliquam pulvinar lacus at porta vulputate. Aliquam eget neque bibendum ipsum vestibulum consectetur. Vivamus sagittis, nisl ac gravida faucibus, ex sem porttitor est, ac
eleifend augue nibh eu mauris.

Ut dictum libero ac sodales rhoncus. Praesent elementum vitae erat a ornare. Nulla dignissim ante sit amet quam laoreet, quis sagittis erat tincidunt. Donec a faucibus lacus, nec commodo diam.
Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Ut semper nisl et arcu bibendum, nec accumsan justo sodales. Nullam et nisl et ex mollis placerat.
Curabitur urna arcu, fringilla sit amet diam in, ultrices varius diam. Donec maximus purus augue, at vestibulum libero pretium vel.
```

Figure 7: Console output of Results

The results in this demonstration find reverse redundancy to be the most effective for deletion errors of single characters, which are concentrated towards the end of the string. In a non-hypothetical experiment, validation tests and significance tests would need to carried out with more data to establish whether a causal relationship can truly be made. However, this study aims to demonstrate use case testing with a sample of results. Based on these preliminary results, one can see that the algorithm becomes less effective as file size increases. Redundancy schemes with original and reverse copies tend to recover data better than redundancy schemes that store original copies and reverse complement copies.

5.2 Integration Testing

This section explains the results of integration testing, which is where all individual units are combined and tested as a group. The purpose of this is to find inconsistencies between integrated units.

No. of Characters Per Error	Error Rate	Dispersion of Errors
1	0.1	Spread Out
1	0.1	Last 10% of String
3	0.1	Spread Out
3	0.1	Last 10% of String
1	1.3	Spread Out
1	1.3	Last 10% of String
3	1.3	Spread Out
3	1.3	Last 10% of String

Table 4: Basic Sampling Method for Integration Testing

In order to carry out integration testing, a sample of combinations were chosen. These were chosen by attempting to incorporate all variables based on endless combinations. Endless because, the three variables; error rates, the number of characters per error and input file sizes are all quantitative variables. This means there are an infinite number of combinations that could be tested on the simulator. Based only on the qualitative variables, there are a total of 8 simulation combinations, which are summarised in Table 4. Based on the literature, error rates commonly average 0.1 to 1.3 percent (Table 2). A total of 24 simulations were completed first, 8 of the simulations in Table 4, per error at 50kB. This was done to test which errors the algorithm was robust at this file size, so for easier comparison. Unfortunately, the program was not robust to substitution errors at all, and the maximum that could be recovered was 22 percent of data with original redundancy at a 0.1 percent error rate with single character errors, which were dispersed. There was only a minor improvement with insertion errors, where the highest data recovery rate was 29.79 percent of the data with original redundancy, at a 0.1 percent error rate with 3 runs of characters inserted. The errors in this case were concentrated towards the end of the string. On the other hand, for deletion errors, the algorithm was able to successfully error-correct data 100 percent of data in four out of eight trials (Appendix A, Table 11), either with original redundancy or reverse redundancy. This was successful for errors that were both dispersed and concentrated towards the back.

5.3 Evaluation

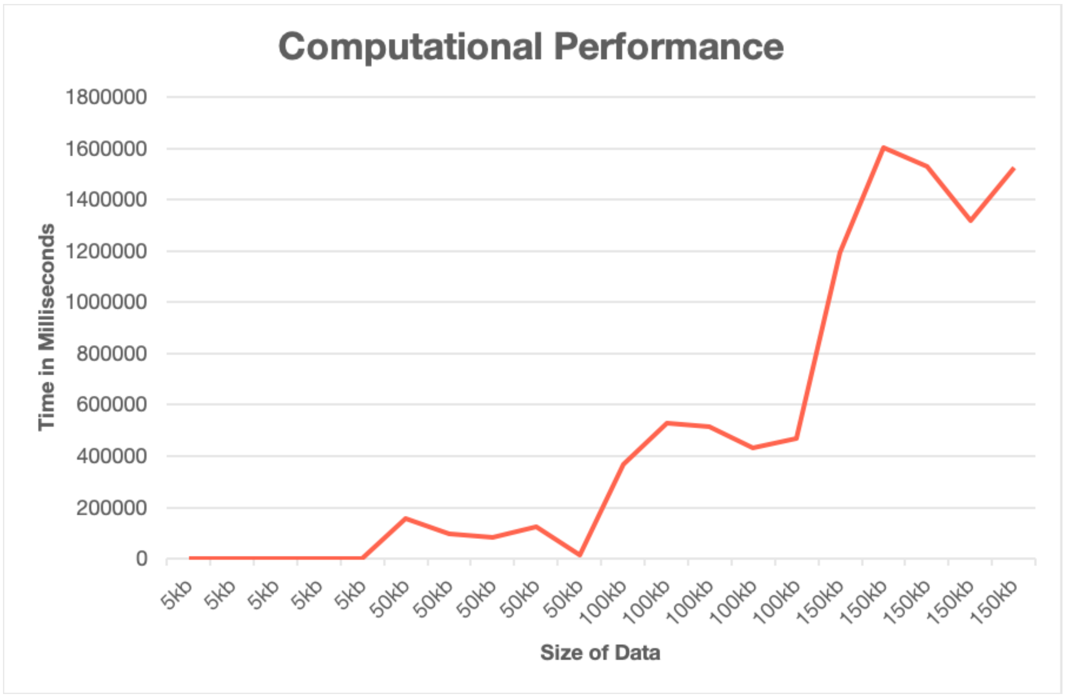


Figure 8: Console output of Results

Based on observations from both tests, the data points for size and processing time was collated into a line graph (Figure 8). This graph shows that the processing time increases almost exponentially as input size increases. The processing time for the algorithm to simulate the DNA storage process on 150kB of data averages 18 minutes, which is extremely long for a file size that small. This highlights the need to identify bottlenecks in the software or finding computational efficiency whereby these errors can be reduced. In the use case testing, the processing time was shorter for data that could be recovered with 100 percent accuracy at 100kB, than 100kB files that were unrecoverable. This indicates that one of the inefficiencies of the program is when the algorithm attempts to recover data that is unrecoverable, but nonetheless keeps checking for matching indexes. Moreover, another issue identified by this test is that the encoder is not able to retain paragraphs after it has converted it into DNA, i.e. the decoder is not able to recover the positions at which new lines occur to correctly represent the data. Moreover, average processing time increases disproportionately to file size, and crashes at 200kb, indicating that this is the upper limit for the software.

The application has been implemented successfully to meet all the primary objectives and most of the secondary objectives of the project. A GUI was built that allows the user to encode a specified text input into DNA, run an error simulator on the DNA string based on user customisations. Error-recovery was also successfully demonstrated with an algorithm that can successfully error correct data from deletion errors at comparable error rates (Table 2) to wet lab experiments found in synthesisers and sequencers. Results are also presented to the user in a format that lets them view the data transformation after the error simulator (and before the error corrector), and the result following the error correcting algorithm. They are also able to view the processing time and the size of the file, and the total data recovery rate, which is all logged to the console which can be natively downloaded for data collection.

5.4 Limitations

The following section highlight the limitations of the system, which are suggestions for further search in the field.

1. Limited Inputs: Due to time and testing constraints, the program is limited to text inputs and was not able to develop an efficient encoding and decoding scheme for mapping other file types to DNA, such as images or videos.

2. Limited Combinations: DNASS should be able to allow users to specify combinations of different errors and redundancy schemes, in terms of amount of redundancy to store and different combinations of errors for a more realistic simulation. Although combined errors was attempted to be implemented (which can be viewed in GitHub version control), there were issues with retaining the error rate based on the specified distribution of errors across more than one type. Moreover, accurate error distributions (such as 20 percent of errors are substitution errors, and 50 percent are deletion) were not be implemented, but is highly recommended for future versions.

3. Low Thresholds and High Processing Times: Although error-correcting was successful at error rates from the literature, the software was only

able to accept 150kB of data before crashing, and that was taking approximately 18-20 minutes. Therefore, the tests had to be carried out with an even smaller file size of 100kB. DNASS was conceived with the intention of being able to accept large file sizes and higher error rates, something that the algorithm did not allow for. Moreover, fixed length strings could not be tested without the program crashing.

6 Conclusion and Future Work

This chapter provides a summary of DNASS and what it has achieved, as well as future directions that can improve this solution further.

Based on the scope of the literature, there is no such simulation that aims to provide users with a testing platform for DNA storage, especially one that attempts to mimic synthesis and sequencing errors. Most studies in the literature are aimed at attempts of storage in DNA, and testing the density limits of DNA. However, there needs to be a shift away from a reliance in redundancy for these high-cost experiments, and an increase in testing and building more efficient encoding schemes and error-correcting schemes to ensure optimal storage levels in DNA. Although the error-correcting algorithm was not hugely successful in correcting substitution and insertion errors, the software was still able to correct deletion errors at comparable rates to ones generated by synthesis and sequencing processes. DNASS is also able to successfully mimic the basic errors of insertion, substitution and deletions with variations regarding the dispersion of the errors, at any given error rate. This is combined with customisation options of specified character runs and displays the output of the DNA strings and corrupted DNA strings to the user so that they are able to use the data to run other decoding algorithms.

To conclude, the primary aims of the project was successfully achieved by creating an application that can successfully convert data into DNA, simulate errors and attempt to recover data. The secondary objectives were successful for deletion errors, but not for substitution or insertion errors at low or high error rates. Although this study does not provide a solution to DNA storage issues of improving synthesiser and sequencer workflows, it still contributes by providing a low-cost solution to testing DNA errors and simulating them for research.

Due to limited time constraints and limited knowledge of biotechnology, only a select number of customisations could be implemented and tested accordingly for the scope of this project. However, further customisation is crucial in ensuring the error simulator is able to mimic the exact potential error combinations that may be faced from a sequencer or synthesiser. Areas for improvement include further customisation of the error simulator and redundancy schemes combinations, as well as improving computational efficiency of error-correcting algorithms. Another further suggestion is to employ natural language processing algorithms to attempt to error-correct

ASCII combinations of code, which may lead to algorithms that are more effective and computationally efficient.

References

[1] R. Hariri, E. Fredericks and K. Bowers, ”Uncertainty in big data analytics: survey, opportunities, and challenges”,. *Journal of Big Data*, vol. 6, no. 1, 2019. Available: 10.1186/s40537-019-0206-3 [Accessed 18 July 2020].

[2] D. Panda, K. Molla, M. Baig, A. Swain, D. Behera and M. Dash, ”DNA as a digital information storage device: hope or hype?”, *3 Biotech*, vol. 8, no. 5, 2018. Available: 10.1007/s13205-018-1246-7 [Accessed 18 July 2020].

[3] R. Appuswamy, G. Graefe, R. Borovica-Gajic and A. Ailamaki, ”The five-minute rule 30 years later and its impact on the storage hierarchy”, *Communications of the ACM*, vol. 62, no. 11, pp. 114-120, 2019. Available: 10.1145/3318163 [Accessed 18 July 2020].

[4] D. Lomet, ”Cost/performance in modern data stores”, *Proceedings of the 14th International Workshop on Data Management on New Hardware - DAMON* ’18, 2018. Available: 10.1145/3211922.3211927 [Accessed 15 August 2020].

[5] ”How Much Energy Do Data Centers Really Use?”, *Energy Innovation: Policy and Technology*, 2020. [Online]. Available: https://energyinnovation.org/2020/03/17/how-much-energy-do-data-centers-really-use/. [Accessed: 19 Aug- 2020].

[6] J. Bornholt, R. Lopez, D. Carmean, L. Ceze, G. Seelig and K. Strauss, ”A DNA-Based Archival Storage System”, *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems - ASPLOS* ’16, 2016. Available: 10.1145/2872362.2872397 [Accessed 15 August 2020].

[7] L. Organick et al., ”Random access in large-scale DNA data storage”, *Nature Biotechnology*, vol. 36, no. 3, pp. 242-248, 2018. Available: 10.1038/nbt.4079.

[8] Y. Zhang et al., ”Information stored in nanoscale: Encoding data in a single DNA strand with Base64”, *Nano Today*, vol. 33, p. 100871, 2020. Available: 10.1016/j.nantod.2020.100871 [Accessed 15 August 2020].

[9] G. Church, Y. Gao and S. Kosuri, ”Next-Generation Digital Information Storage in DNA”, *Science*, vol. 337, no. 6102, pp. 1628-1628, 2012. Available: 10.1126/science.1226355.

[10] L. Ceze, J. Nivala and K. Strauss, ”Molecular digital data storage using DNA”, *Nature Reviews Genetics*, vol. 20, no. 8, pp. 456-466, 2019. Available: 10.1038/s41576-019-0125-3 [Accessed 15 August 2020].

[11] C. Clelland, V. Risca and C. Bancroft, ”Hiding messages in DNA microdots”, *Nature*, vol. 399, no. 6736, pp. 533-534, 1999. Available: 10.1038/21092.

[12] R. Grass, R. Heckel, M. Puddu, D. Paunescu and W. Stark, ”Robust Chemical Preservation of Digital Information on DNA in Silica with Error-Correcting Codes”, *Angewandte Chemie International Edition*, vol. 54, no. 8, pp. 2552-2555, 2015. Available: 10.1002/anie.201411378 [Accessed 14 August 2020].

[13] A. Kohll et al., ”Stabilizing synthetic DNA for long-term data storage with earth alkaline salts”, *Chemical Communications*, vol. 56, no. 25, pp. 3613-3616, 2020. Available: 10.1039/d0cc00222d [Accessed 15 August 2020].

[14] V. Zhirnov, R. Zadegan, G. Sandhu, G. Church and W. Hughes, ”Nucleic acid memory”, *Nature Materials*, vol. 15, no. 4, pp. 366-370, 2016. Available: 10.1038/nmat4594 [Accessed 15 August 2020].

[15] N. Goldman et al., ”Towards practical, high-capacity, low-maintenance information storage in synthesized DNA”, *Nature, vol.* 494, no. 7435, pp. 77-80, 2013. Available: 10.1038/nature11875 [Accessed 15 August 2020].

[16] Y. Erlich and D. Zielinski, ”DNA Fountain enables a robust and efficient storage architecture”, *Science, vol.* 355, no. 6328, pp. 950-954, 2017. Available: 10.1126/science.aaj2038.

[17] M. Blawat et al., ”Forward Error Correction for DNA Data Storage”, *Procedia Computer Science, vol.* 80, pp. 1011-1022, 2016. Available: 10.1016/j.procs.2016.05.398.

[18] ”Lorem Ipsum - All the facts - Lipsum generator”, *Lipsum.com*, 2020. [Online]. Available: https://www.lipsum.com/feed/html. [Accessed: 15 Aug- 2020].

Appendices

A Use Case Testing

Trial	Processing Time (milliseconds)	Reverse Similarity Rate	Reverse Complement Similarity Rate
1	814	100%	100%
2	828	100%	100%
3	816	100%	100%
4	822	100%	100%
5	828	100%	100%

Table 5: Demo Results for 5kB File

Trial	Processing Time (milliseconds)	Reverse Similarity Rate	Reverse Complement Similarity Rate
1	157723	13.15%	1.90%
2	98468	36.98%	45.97%
3	85059	86.68%	41.49%
4	123376	94.36%	10.49%
5	13607	11.64%	23.87%

Table 6: Demo Results for 50kB File

Trial	Processing Time (milliseconds)	Reverse Similarity Rate	Reverse Complement Similarity Rate
1	368992	60.43%	37.46%
2	528928	100%	5.36%
3	513316	36.41%	1.05%
4	432611	34.48%	36.87%
5	470929	30.74%	24.23%

Table 7: Demo Results for 100kB File

Trial	Processing Time (milliseconds)	Reverse Similarity Rate	Reverse Complement Similarity Rate
1	1192640	39.92%	10.39
2	1602595	2.12%	3.82%
3	1529532	3.63%	5.10%
4	1316378	18.23%	13.86%
5	1523562	9.56%	9.12%

Table 8: Demo Results for 100kB File

B Integration Testing

Trial	No Chars	Error Rate	Dispersion	Original Similarity	Complement Similarity	Reverse Similarity	Reverse Complement Similarity	Processing Time
1	1	0.1	Spread Out	22.12%	16.97%	20.54%	16.93%	201981
2	1	0.1	Last 10%	19.37%	22.44%	14.45%	13.86%	201854
3	3	0.1	Spread Out	16.44%	18.86%	17.48%	14.60%	192474
4	3	0.1	Last 10%	17.90%	19.06%	15.08%	14.98%	209308
5	1	1.3	Spread Out	2.78%	2.62%	2.14%	2.38%	289266
6	1	1.3	Last 10%	3.16%	2.26%	2.55%	3.61%	273240
7	3	1.3	Spread Out	3.28%	2.53%	3.28%	2.09%	265627
8	3	1.3	Last 10%	3.02%	2.89%	2.68%	2.64%	248846

Table 9: Combination Trials for Substitution Errors (50kB File)

Trial	No Chars	Error Rate	Dispersion	Original Similarity	Complement Similarity	Reverse Similarity	Reverse Complement Similarity	Processing Time
1	1	0.1	Spread Out	17.24%	20.29%	20.70%	23.53%	227401
2	1	0.1	Last 10%	16.77%	19.30%	20.74%	24.74%	191736
3	3	0.1	Spread Out	17.06%	16.65%	31.39%	21.91%	184395
4	3	0.1	Last 10%	29.79%	15.52%	18.23%	15.49%	192862
5	1	1.3	Spread Out	2.53%	3.58%	2.85%	2.38%	277504
6	1	1.3	Last 10%	3.30%	3.01%	2.51%	3.61%	288998
7	3	1.3	Spread Out	3.21%	2.95%	2.49%	2.09%	258155
8	3	1.3	Last 10%	2.60%	3.71%	2.12%	2.64	261522

Table 10: Combination Trials for Insertion Errors (50kB File)

Trial	No Chars	Error Rate	Dispersion	Original Similarity	Complement Similarity	Reverse Similarity	Reverse Complement Similarity	Processing Time
1	1	0.1	Spread Out	100%	9.95%	100%	100%	224278
2	1	0.1	Last 10%	100%	10.79%	100%	100%	215268
3	3	0.1	Spread Out	100%	8.76%	100%	7.02%	224269
4	3	0.1	Last 10%	100%	8.56%	100%	12.81%	243268
5	1	1.3	Spread Out	27.51%	1.52%	100%	1.51%	229192
6	1	1.3	Last 10%	14.69%	1.99%	84.92%	2.11%	243268
7	3	1.3	Spread Out	11.67%	2.37%	11.39%	1.55%	273135
8	3	1.3	Last 10%	38.92%	1.56%	100%	1.56%	219521

Table 11: Combination Trials for Deletion Errors (50kB File)