Digital Preservation with Synthetic DNA

Eugenio Marinelli[1], Eddy Ghabach[1], Yiqing Yan[1],
Thomas Bolbroe[3], Omer Sella[2], Thomas Heinis[2], and Raja Appuswamy[1]

[1] EURECOM, France
eugenio.marinelli@eurecom.fr
eddy.ghabach@eurecom.fr
yiqing.yan@eurecom.fr
raja.appuswamy@eurecom.fr
[2] Imperial College, London, UK
osella@imperial.ac.uk
t.heinis@imperial.ac.uk
[3] Rigsarkivet, Denmark
tbo@sa.dk

Abstract. The growing adoption of AI and data analytics in various sectors has resulted in digital preservation emerging as a cross-sectoral problem that affects everyone from data-driven enterprises to memory institutions alike. As all contemporary storage media suffer from fundamental density and durability limitations, researchers have started investigating new media that can offer high-density, long-term preservation of digital data. Synthetic Deoxyribo Nucleic Acid (DNA) is one such medium that has received a lot of attention recently. In this paper, we provide an overview of the ongoing collaboration between the European Union-funded, Future and Emerging Technologies project OligoArchive and the Danish National Archive in preserving culturally important digital data with synthetic DNA. In doing so, we highlight the challenges involved using DNA for long-term preservation, and present a holistic data storage pipeline that brings together several novel techniques (standardized file storage, motif-based DNA encoding, scalable read consensus to name a few) to provide reliable, passive, obsolescence-free digital preservation using synthetic DNA.

Keywords: DNA storage, long-term archival, preservation, SIARD-DK

1 Introduction

Today, we live in an increasingly digital society. Digital data pervades all disciplines and has established itself as the bed rock that drives our society, from enabling data-driven decisions based on machine learning, to encoding our collective knowledge compactly in a collection of bits. Thus, preservation of digital data has emerged as an important problem that must be addressed by not just memory institutions today, but also by institutions in several other sectors.

In order to preserve digital data, it is necessary to first store the data safely over a long time frame. Historically, this task has been complicated due to several issues associated with digital storage media. All current media technologies suffer from density scaling limitations resulting in storage capacity improving at a much slower rate than the rate of data growth. For instance, Hard Disk Drive (HDD) and magnetic tape capacity is improving only 16-33% annually, which is much lower than the 60% growth rate of data [10]. All current media also suffer from media decay that can cause data loss due to silent data corruption, and have very limited lifetime compared to the requirements of digital preservation. For instance, HDD and tape have a lifetime of 5–20 years. A recent survey by the Storage and Networking Industry Association stated that several enterprises regularly archive data for much longer time frames [19]. Thus, the current solution for preserving data involves constantly migrating data every few years to deal with device failures and technology upgrades. A recent article summarized the financial impact of such media obsolescence on the movie industry [17].

In project OligoArchive [2], we are exploring a radically new storage media that has received a lot of attention recently—Deoxyribo Nucleic Acid (DNA) [6, 13, 9, 16]. DNA is a macro-molecule that is composed of smaller molecules called *nucleotides*(nt). There are four types of nucleotides: Adenine (A), Cytosine (C), Guanine (G), and Thymine (T). DNA used for data storage is typically a single-stranded sequence of these nucleotides, also referred to as an *oligonucleotide* (oligo). DNA possesses several key advantages over current storage media. First, it is an extremely dense three-dimensional storage medium with a capacity of storing 1 Exabyte/mm^3 which is eight orders of magnitude higher than magnetic tape, the densest medium available today [8]. Second, DNA is very durable and can last millennia in a cold, dry, dark environment. A recent project that attempted to resurrect the Woolly Mammoth using DNA extracted from permafrost fossils that are 5000 years old is testament to the durability of DNA even under adverse conditions [18]. Thus, data once stored in DNA can be left untouched without repeated migration to deal with technology upgrades. Third, as long as there is life on earth, we will always have the necessity and ability to sequence and read genomes, be it for assembling the genome of a previously-unknown species, or for sequencing the genome to detect diseases causing variations. As a result, unlike contemporary storage technologies, where the media that stores data and the technology to read data are tightly interlinked, DNA decouples media (biological molecules) from read technology (sequencing), thus reducing media obsolescence issues.

In this work, we provide an overview of the ongoing collaboration between the Danish National Archive and project OligoArchive in demonstrating a holistic solution for long-term preservation of culturally significant data using DNA. We present a motivating use case for long-term digital preservation, outline the challenges involved in using DNA as a digital storage medium, and present the end-to-end pipeline we have put in place to overcome these challenges.

2 Context and Background

2.1 Danish National Archive Use Case

The Danish National Archives is a knowledge center documenting the historical development of the Danish society. The archive collects, preserves and provide access to original data with the purpose of supporting current and future possible needs of the Danish community - public authorities as well as private citizens. A huge part of this work includes the preservation of digitally created and retro-digitized data securely and cost-effectively. Thus, the archive has received and preserved such data since the 1970s.

The archival material used for this work consists of selected hand-drawings made by the Danish king Christian IV (1577-1648). Although his reign was marked by military defeat and economic decline, Christian IV stands out as one of the most prominent, popular and admired characters in the line of Danish kings. The hand-drawings date to the period 1583-1591 where the king was 6–14 years old. The material is a part of a larger archival unit consisting of numerous documents and records [4]. The specific image [5] used for this experiment presents a naval battle between several warships. Besides emphasizing the young king's admiration for warfare and naval tactics, the material further indicates his high level of cultural education as well as his talent for drawing. At Danish National Archive, the material is thus ranked as having "Enestående National Betydning" (meaning unique national significance).

2.2 DNA storage challenges

Using DNA as a digital storage medium requires mapping digital data from its binary form into a sequence of nucleotides using an encoding algorithm. Once encoded, the nucleotide sequence is used to *synthesize* DNA using a chemical process that assembles the DNA one nucleotide at a time. Data stored in DNA is read back by *sequencing* the DNA molecules and decoding the information back to the original digital data.

A simple way to convert bits into nucleotides is to adopt a direct mapping that converts 2 bits into a nucleotide, for instance 00 to A, 01 to T, 10 to G, and 11 to C. This way a binary sequence is translated to an arbitrary sequence of nucleotides. However, such a simple approach is not feasible due to several biological limitations imposed by DNA synthesis and sequencing steps. First, DNA synthesis limits the size of an oligo between hundred to few thousands of nucleotides. Therefore, data must be divided into several pieces, with each piece being stored in an oligo. However, unlike current storage devices, oligos do not have logical addressing. Hence, indexing information that can help to identify the order in which the oligos, and hence the corresponding data bits, must be reassembled back during recovery must be stored together with the data bits and integrated in each oligo.

[4] https://www.flickr.com/photos/statensarkiver

[5] https://www.flickr.com/photos/statensarkiver/28273082238

Second, oligos with repeat sequences (like ACACAC), or long consecutive repeats of the same nucleotide (like AAAAA), and oligos with extreme GC content, where the ratio of Gs and Cs in the oligo is less than 30% or more than 70%, are known to be difficult to synthesize, sequence, and process correctly. Thus, when constructing oligos, constraints need to be enforced to minimize homopolymer repeats and balance GC content. Further, care must be taken to minimize similarity across oligos as having too many oligos with positionally-similar nucleotide sequences can exacerbate sequencing errors and make it difficult to identify the original oligo.

Third, sequencing and synthesis are not error free even for well-formed oligos, as they introduce substitution errors, where a wrong nucleotide is reported, or indel errors, where spurious nucleotides are inserted or deleted. Both sequencing and synthesis also introduce bias. Some oligos are copied multiple times during synthesis, while others are not. Similarly, some oligos are read thousands of times during sequencing while others are not sequenced at all. Thus, it is important to use error correction codes in order to recover data back despite these errors.

In addition to the aforementioned media-level challenges in using DNA as a digital storage medium, there are also other problems associated with digital preservation that DNA does not solve. Any digital file stored on DNA is an encoded stream of bits whose interpretation makes sense only in the context of the application used to render, manipulate, and interact with that file format. While DNA might be able to store data for millennia, the associated applications and file formats might become obsolete. Thus, in addition to preserving data, it is also necessary to preserve the meaning of data by ensuring that data is stored in a preservation-friendly, non-proprietary format. Digital data can also be altered due to a variety of reasons and additional data-integrity techniques should be put in place to ensure that data retrieved from DNA can be trusted to be the same as the original source. The digital preservation community has long pioneered file formats, information systems, and operational methodologies for solving such format obsolescence issues [1]. Thus, a holistic DNA-based preservation solution should build on such techniques to solve both media and format obsolescence issues.

3 Design

In this section, we will describe the end-to-end pipeline we have put in place to overcome the aforementioned challenges.

3.1 Overcoming format obsolescence with SIARD-DK

In Denmark, all public institutions and organizations that produce data worthy of persevering are legally bound to submit them to a public archive. As the vast majority of data in the Danish public sector are organized as databases with or without files in various formats, the focus has been on archiving these data in a standardized, system-independent and cost efficient manner. As a result, the archive has implemented a Danish version of the SIARD format (Software Independent Archiving of Relational Databases) [7] named SIARD-DK for storing of such data. SIARD is an open format, designed for archiving relational databases in a vendor-neutral form and is used in the CEF building block "eArchiving".

The first step in preserving data is extracting it and creating an SIARD-DK Archival information Package (AIP). In the creation of this particular AIP, the digitized material was converted to TIFF format. Information relevant to the images such as the preservation format of the files, their title, creator and original size, descriptive information, etc., was extracted and packaged together with relevant documentation in the AIP-format. This was done using proprietary tools developed at the Danish National Archive. The usage of SIARD for storing the files guarantee that the material is preserved in a rich format with relevant metadata stored in a standardized, system and vendor independent way. The resulting AIP is a single ZIP64 file that internally contains the TIFF images, in addition to XML and XSD files that store the schema of the archive and metadata information. This allows for strict validation of the AIP. Further, an MD5 value of each file is stored inside the archive and serves as the fixity to verify data integrity on retrieval.

3.2 DNA data storage pipeline

The end-to-end DNA media storage pipeline is presented in Figure 1. In the rest of this section, we will provide an overview of both the write path that takes as input the SIARD zip file and stores it in DNA, and the read path that restores back the zip file from DNA.

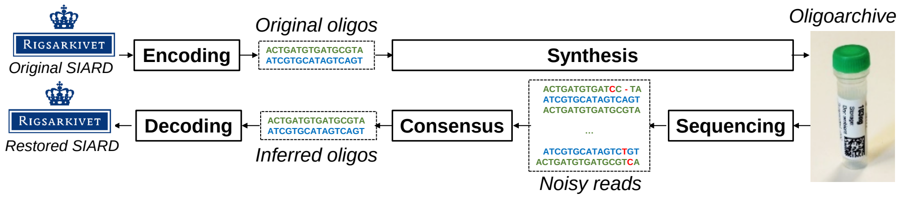

Fig. 1. DNA Storage Pipeline

Write Path In order to store the archive on synthetic DNA, the zip file is first encoded from binary into a quaternary sequence of oligonucleotides, and then synthesized to generate synthetic DNA. The steps for encoding the SIARD archive file into oligos is presented in Figure 2. During encoding, the file is read as a stream of bits and pseudo randomized. In other encoding methods, randomization is used as a way to limit the number of homopolymer repeats in

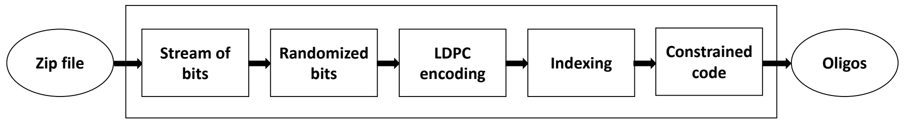

Fig. 2. Encoding bits into oligos

each oligo. In our encoding, homopolymer repeats are handled by an inner constrained code that we explain later. Thus, we do not need randomization for avoiding homopolymer repeats. We use randomization primarily to improve the accuracy of our clustering and consensus methods in the data decoding stage. As mentioned before, data stored in DNA is read back by sequencing the DNA to produce *reads*, which are noisy copies of the original oligos that can contain insertion, deletion, or substitution errors. Our read clustering and consensus methods rely on the fact that the original oligos are well separated in terms of edit distance so that the distance between a noisy read and its corresponding oligo is much smaller than the distance between two oligos. This assumption makes it possible to cluster similar reads and infer the original oligo with a high accuracy. A long sequence of zero or one bits can violate this assumption as they can lead to multiple oligos being similar, or even identical, to each other. A long sequence of bits can also lead to oligos with repetitive sequences (example: ACACACAC). This can pose problems during data decoding, especially if paired-end sequencing is used, where the DNA is partially read from either direction (5" to 3" and 3" to 5") with overlap. In such a case, the two reads corresponding to each orientation must be merged into a single representative read. Repetitive sequences can create issues during this merging process. While we can develop more advanced solutions to perform merging, randomization provides a simple solution that eliminates such issues while ensuring that similarity across oligos is also minimized.

After randomization, error correction encoding is applied to protect the data against errors. We use large-block length Low-Density Parity Check (LDPC) codes [11] with a block size of 256,000 bits as the error correction code, as it has been shown to be able to recover data in the presence of intra-oligo errors, or even if entire oligos are missing [5]. We configure LDPC to add 10% redundancy to convert each sequence of 256,000 bits into 281,600 bits with data and parity. Each 281,600 bit sequence is then used to generate a set of 300-bit sequences, where each 300-bits is composed of 281 data bits and a 19-bit index that is used to order the sequences. Each 300-bit sequence is then passed to a constrained code that converts it into an oligonucleotide sequence.

The constrained code essentially views each oligo as a concatenation of several shorter oligonucleotide sequences, that we henceforth refer to as *motifs*. In our current configuration, the constrained code breaks up each 300-bit sequence into a series of ten 30-bit integers. Each 30-bit integer is fed as input to a *motif generator* that takes the 30-bit value and produces a valid 16nt (nucleotides) motif as output. The motif generator does this mapping by pre-constructing an associative array where the 30-bit value is the key and a 16nt motif is the value. This array is built by first enumerating all possible motifs of length 16nt. Then, all motifs that fail to meet a given set of biological constraints are eliminated.

Our current motif generator is configured to allow up to two homopolymer repeats (AA,CC,GG, or TT), and admits motifs with a G-C content in the range 0.25 to 0.75. With these constraints, using 16nt motifs, out of 4^{16} possible motifs, we end up with only 1,405,798,178 unique, valid motifs with which we can encode any possible 30-bits of data (as the number of all possible 30-bits values is $2^{30} = 1,073,741,824$ which is lower than 1,405,798,178. Thus, we use the billion motifs as values in the array corresponding to keys in the range 0 to 2^{30}. The 30-bit value is thus used as the key into this associative array to produce the corresponding 16nt motif. Thus, at the motif level, the encoding density is 1.875 bits/nt. The reason we limited ourselves to 16nt and 30bits is the fact that this associative array occupies around 100GB of memory, which we can easily meet using our current hardware. The motif generator can be extended to larger motif sizes and more relaxed biological constraints which can lead to higher bit densities. But as this would require the use of external storage, we leave this open to future work.

Using the associative array, each 300-bit sequence is encoded as concatenation of ten motifs, each with a length of 16nt, leading to an oligo that is 160 nucleotides long. We would like to explicitly point out here that the length of an oligo is a configurable parameter. Thus, while we use 160nts in our current system due to favorable pricing provided by our synthesis provider, our encoder can generate shorter or longer oligos if necessary, and automatically adjust various aspects (like the 19-bit index and 281-bit data size) based on desired oligo length.

Read Path To retrieve back the SIARD archive, the DNA is sequenced in order to retrieve back the nucleotide sequence of oligos. As mentioned before, sequencing produces reads, which are noisy copies of the original oligos. Thus, we need a consensus procedure to infer the original oligos from the reads. In prior work, we structured this process as sequence of three algorithms, as depicted in Figure 3. First, we identify all pairs of strings that are similar to each other. As modern sequencers produce hundreds of millions of reads, this first task is extremely computationally intensive due to use of the edit distance as a metric for comparing strings. Thus, we have developed an efficient similarity join algorithm, called OneJoin [15], that exploits the fact that due to randomization during encoding, reads corresponding to the same original oligo are “close” to each other despite some errors and “far” from the reads related to other oligos. The results obtained from the join algorithm are then used to quickly identify clusters of strings that are similar to each other. Each cluster thus groups all reads belonging the same oligo. Finally, we apply a position-wise consensus procedure that uses multiple reads to infer the original oligo in each cluster.

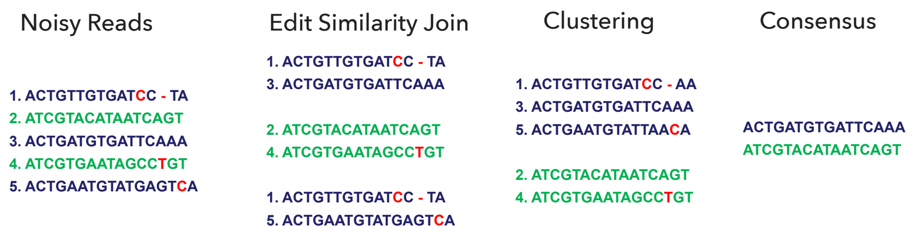

Fig. 3. Various steps in the OneJoin consensus procedure.

While our prior approach was able to infer original oligos with a high accuracy, there were two problems. First, we found that under some datasets, particularly for high coverage reads, OneJoin's memory and computational usage were too high. As OneJoin is a string similarity join, it produces as output all possible pairs of reads that are similar to each other. Thus, given a coverage N, the computational and memory requirements of OneJoin were $O(N^2)$. Second, the use of a general purpose string similarity join led to functionality repetition at multiple places in the read path. For instance, OneJoin internally uses an edit distance check to filter out strings that are not similar. Later in the read pipeline, we had to repeat the edit distance computation in the alignment stage once to get position-wise consensus. This repetition led to needless overhead. To solve these problems, we have developed a new consensus procedure that we refer to as *OneConsensus*.

In the following sections, we describe the key stages of OneConsensus algorithm. In order to efficiently identify and group all the similar reads, our algorithm relies on two well-known algorithmic tools that allow to drastically cut down the computational time: CGK-Embedding and Locality Sensitive Hashing (LSH).

CGK Embedding. As we mentioned in the previous section, the similarity metric used in OneConsensus is the edit distance. Given two strings x and y, the edit distance is defined as the minimum number of edit operations i.e. insertions, deletions and substitutions, necessary to transform x in y. Another metric, commonly used to compare strings is the Hamming distance. However, the latter takes into account only the number of mismatches between the two strings, or in other words the number of substitutions to transform x in y. For example, given the two strings $ACACT$ and $GACAC$, their Hamming distance is 5 since there are no matches, but the edit distance is 2 since it suffices to add G and remove T

From these definitions, we can make the observation that the edit distance takes into account information about the ordering of characters and captures the best alignment between two strings. However, while Hamming distance has complexity that is linear with the string length, edit distance has complexity that is quadratic. While several dynamic programming optimizations exist for accelerating edit distance computations [20], they often rely on pre-specified distance thresholds and are unable to provide performance competitive with fine-tuned Hamming distance computations in practice. Given the complexity of this metric, we rely on randomized embedding techniques to minimize the overhead of the edit distance computations.

Randomized embedding refers to a set of methods that map a complex metric space into a simpler one. `CGK-embedding` algorithm, recently proposed by Chakraborthy et al. [4] is one such algorithm that can map problems from an edit space into a Hamming space. Given two strings x, y of length N taken from an alphabet $\sum$ such that $d_E(x, y)$, the edit distance between x and y, is less than K, CGK-embedding is a function $f\colon \sum^N \rightarrow \sum^{3N}$ that maps strings x and y into $f(x)$ and $f(y)$ such that, with probability at least 0.99, the Hamming distance of $d_H(f(x), f(y))$ is bounded by K^2 when $d_E(x, y) < K$. This implies that the distortion D, defined as the ratio $D(x, y) = \frac{d_H(f(x),f(y))}{d_E(x,y)}$, is at most K. Thus, as long as the edit distance is small, the distortion of embedding is small. This implies that the Hamming distance of embedded strings will accurately track the edit distance of the original strings, thereby making it possible to replace the expensive edit distance computation with cheap Hamming distance computation [22].

Algorithm 1 CGK-embedding

```
Input: A string $S \in \{A, C, G, T\}^N$, a random string $R \in \{0, 1\}^{3N}$ and a char for
    padding $P = 0$
Output: The embedded string $S^{'} \in \sum^{3N}$
 1: $i \leftarrow 0$
 2: for $j = 0 \rightarrow 3N - 1$ do
 3:     if $i < N$ then
 4:         $S^{'}_j \leftarrow S_i$
 5:     else
 6:         $S^{'}_j \leftarrow P$
 7:     end if
 8:     $i \leftarrow i + R_j$
 9: end for
10: return $S^{'}$
```

The pseudo-code of embedding algorithm is shown in Algorithm 1. In this case the procedure is applied to all strings of length N that are composed of the characters A, C, G, T, representing the DNA alphabet. Given an input string, the algorithm builds the corresponding embedded representation by appending one character at time taken from the input string. The character appended can be the repetition of the previous character or the next character in the input string according to the value of a binary random string. In other words, the pointer of the current character in the input string increases or remains the same depending on the random string value, that can be 0 or 1. When the pointer to the input string goes beyond the string length, the embedded string is padded with a special character P. In general, P can be any character that is not included in the alphabet of the given dataset S. For sake of simplicity, in Algorithm 1 we use 0 for padding. In essence, what we get as the output of embedding is a string where characters from the input string can be repeated one or more times.

LSH for Hamming Distance. One of the main advantages of moving from the edit distance space to the Hamming space is being able to use some useful algorithms that are valid in the Hamming space only and instead not applicable to the edit distance. One of these algorithms is Locality Sensitive Hashing (LSH) [12].

Definition 1. *We call a family $\mathcal{H}$ of functions (d_1, d_2, p_1, p_2)-*sensitive *for a distance function D if for any p, $q \in U$ (where U is the item universe):*

- *if $D(p,q) \leq d_1$ then $P[h(p) = h(q)] \geq p_1$, that is, if p and q are close, the probability of a hash collision is high;*
- *if $D(p,q) \geq d_2$ then $P[h(p) = h(q)] \leq p_2$, that is, if p and q are far, the probability of a hash collision is low;*

where $h \in_r H$ are hash functions randomly sampled from the family of hash functions $\mathcal{H}$

Considering two bit-string p and q of length N. In the Hamming distance case, the hash function is defined as the i^{th} bit of these strings. Thus, if their Hamming distance is $d_H(p,q)$, that is, the number of bits that differ position wise in the two strings, then the probability that any given bit at a random position is the same in both strings is $1 - \frac{d_H(p,q)}{N}$. Thus, the bit-sampling LSH family for Hamming distance, defined as: $\mathrm{H}_N = \{h_i : h_i(b_1 ... b_N)) = b_i \mid i \in [N]\}$ is $\left(d_1, d_2, 1 - \frac{d_1}{N}, 1 - \frac{d_2}{N}\right)$-sensitive for the two Hamming distances $d_1 < d_2$.

We use Hamming LSH over embedded reads to separate out the reads into different buckets such that with a very high probability, reads within a bucket are similar to each other, and hence correspond to the same reference.

Clustering based on Edit Distance At this stage, we have all reads grouped in hash buckets based on their similarity. However, we still have two problems to solve. (1) LSH can produce false positives, meaning that two dissimilar reads can end up in the same bucket. The main consequences is that if reads are very different, the consensus procedure lead to the wrong result. (2) Reads can have different lengths due to insertions and deletions errors. Thus, we need to adjust the reads in order to make their lengths uniform while taking into account possible insertion/deletion errors. Both these problems can be solved by aligning the reads in each bucket. More specifically, given a bucket, we sort the reads based on length such that reads with length matching the reference oligos are moved to the front of the bucket. Then, starting with the first read in the bucket, we align all the following reads to the first one. The intuition behind sorting the reads is as follows. If a read has the same length as the reference oligos (160nt), either the read has no insertion/deletion errors, or there are an even number of insertion and deletion errors. Since the probability of errors is low with short-read sequencing, the former scenario is more likely. Thus, by picking reads that are of the correct length and aligning the rest of the reads to it, we increase the probability of finding the correct original reference oligo.

The alignment of reads gives two pieces of information: the edit distance and the Compact Idiosyncratic Gapped Alignment Report (CIGAR). The edit distance allows us to identify reads that are actually dissimilar even if they are in the same buckets. We group only similar reads to form a cluster. The CIGAR contains the base-by-base alignment information (the sequence of matches, insertions and deletions) needed to align one read to the other. Using this information, we adjust the reads by adding gaps where there is a deletion error, or deleting nucleotides where there is an insertion error. Once all similar reads are found within the bucket, we save the cluster and remove the reads from the bucket. Any dissimilar reads that were not a part of the cluster are still left in the bucket. In order to deal with false positives by LSH, the remaining reads are then processed again in the same way, with the procedure being repeated until the bucket is empty.

Position-Wise Consensus The result of the previous stages is a set of clusters. All reads within a cluster are noisy copies of the same reference oligo with some errors in random positions. At this point of the algorithm the only missing step is the consensus procedure. The consensus algorithm works on a per position basis. The key idea is that despite some random errors, all reads in a cluster are aligned and adjusted based on the edit distance and CIGAR. Thus, if we consider a specific position across all reads, it is likely that the majority of reads in that position will contain the correct nucleotide while only some of them report the wrong one. This implies that for any given position, it will be enough to take the most-frequent nucleotide as the consensus outcome. Repeating this procedure for each position, we produce one inferred oligo per cluster. We would like to point out here that not all oligos need to be correctly inferred. In fact, as we show later, some original oligos might not appear at all in the inferred set, and other inferred oligos might have errors. We rely on the parity added by LDPC codes at a higher level to recover data despite these errors.

The inferred oligos are then passed to the decoder which reverses the steps shown in Figure 2. The constrained code is first used to convert each 160nt oligo back into 300-bit sequences by converting each individual 16-nt motif into its corresponding 30-bit value. The index stored in each 300 bits is used to reassemble bits back in the correct order. The LDPC decoder is then used to recover back data even if some bits were wrongly decoded, or some bits were zeroed out as corresponding oligos were missing. The decoded data is then derandomized to obtain a stream of bits that corresponds to the original input.

4 Evaluation

At this stage of our collaboration, we have assembled the entire pipeline. We are in the process of carrying out a real-life, large-scale synthesis experiment. As we do not have results from real experiments yet, we will provide a preliminary, simulation-driven evaluation in this section. Note that we simulate only the synthesis and sequencing steps in Figure 1. We encode/decode the real dataset using our pipeline.

4.1 Experimental Setup

The raw SIARD archive that is fed as input to our pipeline is 12.9MB in size. With redundancy added by LDPC, the resulting binary data to be stored on DNA is 15.19MB in size. We encode the SIARD archive generating 405,212 oligos, each with a length of 160nts. Using these original oligos, we then generate four million reads by using a short-read simulator[6] tool, that adds random errors such as insertion, substitution, and deletion in each read to mimic the actions of an Illumina DNA sequencer. This corresponds to an average coverage of 10×, meaning that each oligo, on average is covered by 10 noisy copies.

4.2 Benchmark with sequence alignment

We begin our analysis by visualizing the coverage across oligos. While the average coverage is 10×, the overall coverage typically follows a negative binomial distribution, with some oligos being covered hundreds of times, and some being not covered at all. To visualize this, we aligned the reads to reference with BWA-MEM v0.7.17 [14], a state-of-the-art short-read aligner. Based on the alignment result from BWA-MEM, we show the histogram of coverage across oligos in Figure 4. The x-axis is the coverage (or number of reads that map to an oligo), and y-axis is the number of oligos with that coverage in log scale. As can be seen, the coverage distribution spans a range from 1 to 26, with majority of oligos being covered 5–15× as expected given that the simulator was configured to produce 10× coverage.

We also use the alignment result to show the histogram of number of errors (indels and substitutions) in Figure 5, where x-axis represents the number of errors, and y-axis represents the number of oligos with that error count in log scale. As can be seen, the error distribution is right skewed, with a majority of reads having fewer than 2 mismatches, and a few reads having as many as 8 mismatches which accounts for a 5% error rate given the oligo length of 160. Finally, we also used the alignment result to verify that each read uniquely maps to an oligo (absence of 'XA' field in the SAM output file produced by BWA-MEM). This shows that oligos are "far" from each other in terms of edit distance due to randomization.

[6] https://sourceforge.net/projects/bbmap/

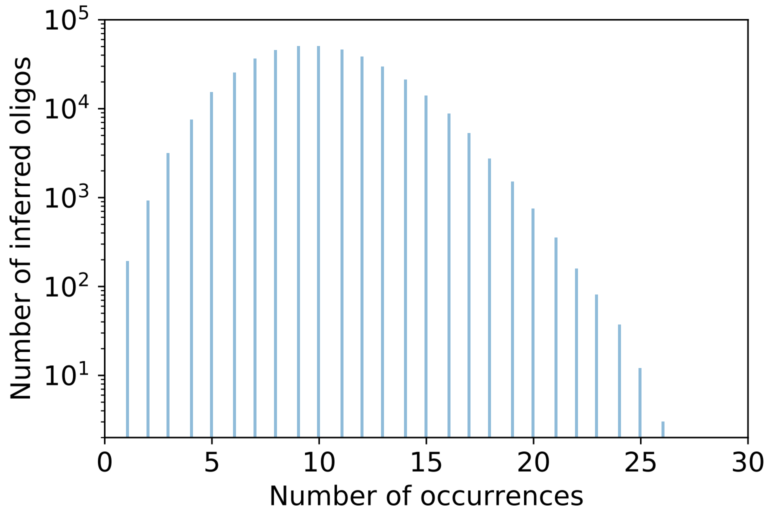

Fig. 4. Histogram of occurrence of inferred oligos

BWA-MEM performs base-by-base, edit-distance-based base-by-base alignment and reports CIGAR and alignment score. As a result, it is computationally very intensive. While such an alignment is required for genomic data analysis in order to determine the exact location of a read in the genome, for the purpose of DNA storage, we found that it is sufficient to simply map each read to an oligo without full alignment. Accel-Align (v1.1.1; [21]) is a short-read aligner that we have developed in the context of project OligoArchive that supports alignment-free mapping-only mode that can quickly map reads to oligos. Thus, we present a comparative analysis of BWA-MEM and Accel-Align here with the goal of presenting it as a open-source tool that can be used by other researchers for both DNA storage and more broadly, for analyzing genomic data.

	Exec. time (second)	Correctly aligned (%)
BWA-MEM	25.5	100
Accel-Align	3.8	99.9

Table 1. Performance and accuracy of BWA-MEM and Accel-Align(map mode)

We use the BWA-MEM and Accel-Align to align/map reads to oligos. Using BWA-MEM's alignment as the gold standard, we evaluate Accel-Align's accuracy. Table 1 shows the execution time and the percent of correctly aligned reads. It shows that Accel-Align can performing mapping 6x faster than BWA-MEM at a slight drop in accuracy of 0.004%. On futher analysis, we found this

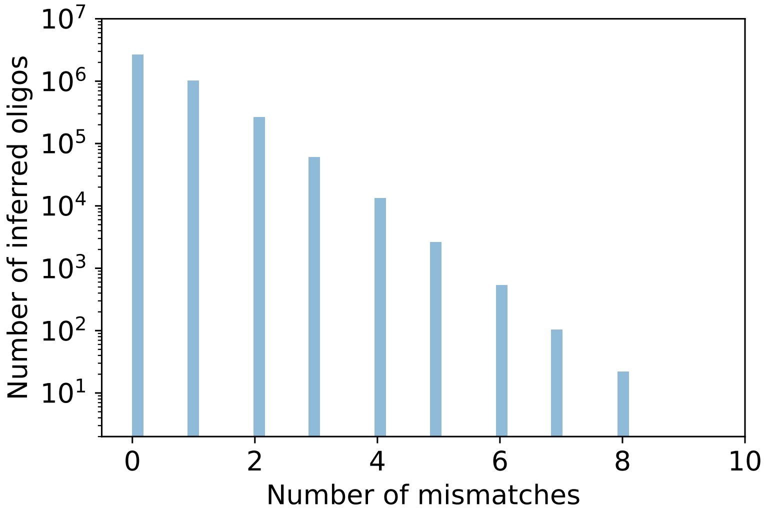

Fig. 5. Histogram of mismatches

drop to be due to Accel-Align's inability to map reads with very high error rate. Accel-Align is specifically built to trade off performance and accuracy for Illumina-based short-read sequencing reads whose error rate is typically much lower than 5%. As our simulated reads offer a more pessimistic error model, Accel-Align experiences a slight drop in accuracy. Despite this, as the coverage histogram with Accel-Align and BWA-MEM are near identical, we have found Accel-Align to be a very useful tool for analyzing both DNA storage reads and more broadly, genomic data [21].

4.3 End-to-end Decoding Results

Having presented an analysis of the reads, we will now present the decoding results. Using OneConsensus procedure described earlier, we obtain the inferred oligos using the simulated dataset. Table 2 shows the error statistics for these oligos. The figures are obtained by comparing the inferred oligo for a certain index with the corresponding original reference oligo. We see that we are able to infer 404,075 oligos that correspond to 99.7% of the original oligos perfectly without errors. In addition, 1010 oligos were inferred with some errors and 127 oligos were completely missing. Note that errors in an oligo does not imply that the entire oligo is different from the original, but differs only with respect to a few motifs. For this reason, we also report the difference between inferred data and original encoded file in terms of number of bits.

We then use the constrained code to convert these inferred oligos into 300-bit sequences, and reassemble them in order based on the 19-bit index. At this stage, we will certainly have situations where an oligo is missing due to sequencing simulation bias, or an oligo could not be converted back into 300-bits due to errors. In both cases, there will be a corresponding index whose data bits cannot be recovered. We insert a sequence of zero bits for such indices and use the reconstructed binary together with the LDPC decoder to restore the original input. Despite the errors reported in Table 2, the LDPC decoder was able to recover back the original archive completely, thanks to the additional parity information added during encoding.

#Original Oligos	405212
#Correctly Inferred Oligos	404075
#Incorrectly Inferred Oligos	1010
#Missing Oligos	127
#Incorrect bits	42678

Table 2. Statistics for decoding of SIARD archive

Finally, in this section we compare OneConsensus with OneJoin-based consensus algorithm. Table 3 compares the two algorithms in terms of the accuracy achieved in inferring the encoding oligos. In terms of the number of oligos correctly inferred in their entirety, meaning an exact match between the inferred oligo and the original reference oligo, we see that OneConsensus slightly underforms OneJoin. But as we mentioned earlier, oligos can also differ by just a few motifs only, making the statistic about the correctly inferred oligos insufficient to determine the actual accuracy of the two algorithms. For this reason, we report also the number of missing oligos and the number of bits wrongly inferred by the two algorithms. The number shows that OneConsensus outperforms OneJoin, as it mistakes only misses 127 oligos, while OneJoin 595 oligos. Overall, OneConsensus leads to 42678 bit errors, compared to the 95935 bit errors produced by the OneJoin-based consensus.

In terms of memory consumption, we observed that OneJoin reaches a peak of 2.5GB, while OneConsensus requires only 1.1GB. While the difference may not seem too striking for this dataset, we would like to point out that while OneConsensus memory consumption grows linearly with the dataset size, OneJoin requires an amount of memory that is quadratic with coverage.

	OneConsensus	OneJoin
#Correctly Inferred Oligos	404075	404104
#Incorrectly Inferred Oligos	1010	513
#Missing Oligos	127	595
#Incorrect bits	42678	95935

Table 3. Statistics for OneConsensus and OneJoin-based consensus

5 Conclusion and Future Work

In this work, we provided an overview of the ongoing collaboration between project OligoArchive and the Danish National Archive in using DNA to preserve culturally significant digital data. Building on prior work on molecular information storage and digital preservation, we presented a holistic, end-to-end pipeline for preserving both data and the meaning of data on DNA, and tested the pipeline using simulation studies. There are several avenues of future work we are pursuing. First, as described earlier, we are in the process of carrying out a large-scale experiment to validate our pipeline using real data. Second, we are investigating various optimizations to both encoding and consensus algorithms to support alternate synthesis and sequencing technologies with potentially higher error rates. Finally, while we addressed the question of preserving data in this work, we left open the question of preserving the decoding algorithm itself. Recent work has investigated the design of nested universal emulators that can be used to preserve and emulate such decoders using analog media like film or archival paper [3]. Thus, we are investigating methods to combine DNA-based digital data storage with analog media-based decoding logic storage.

Acknowledgments

This work was partially funded by the European Union's Horizon 2020 research and innovation programme, project OligoArchive (grant agreement No 863320).

References

1. Digital Preservation Handbook. Digital Preservation Coalition (2015)
2. Appuswamy, R., Lebrigand, K., Barbry, P., Antonini, M., Madderson, O., Freemont, P., MacDonald, J., Heinis, T.: OligoArchive: Using DNA in the DBMS storage hierarchy. In: CIDR (2019)
3. Appuswamy, R., Joguin, V.: Universal layout emulation for long-term database archival. In: CIDR (2021)
4. Chakraborty, D., Goldenberg, E., Kouckỳ, M.: Streaming algorithms for embedding and computing edit distance in the low distance regime. In: Proceedings of the forty-eighth annual ACM symposium on Theory of Computing. pp. 712–725 (2016)
5. Chandak, S., Tatwawadi, K., Lau, B., Mardia, J., Kubit, M., Neu, J., Griffin, P., Wootters, M., Weissman, T., Ji, H.: Improved read/write cost tradeoff in dna-based data storage using ldpc codes. In: 2019 57th Annual Allerton Conference on Communication, Control, and Computing (2019)
6. Church, G.M., Gao, Y., Kosuri, S.: Next-Generation Digital Information Storage in DNA. Science **337**(6102) (2012)
7. of Congress, L.: SIARD (Software Independent Archiving of Relational Databases) Version 1.0. https://www.loc.gov/preservation/digital/formats/fdd/fdd000426.shtml (2015), [Online; accessed 28-May-2021]

8. Corporation, S.R.: 2018 semiconductor synthetic biology roadmap. https://www.src.org/program/grc/semisynbio/ssb-roadmap-2018-1st-edition_e1004.pdf (2018)
9. Erlich, Y., Zielinski, D.: DNA Fountain enables a robust and efficient storage architecture. Science **355**(6328) (2017)
10. Fontana, R.E., Decad, G.M.: Mooreâs law realities for recording systems and memory storage components: Hdd, tape, nand, and optical. AIP Advances **8**(5) (2018)
11. Gallager, R.: Low-density parity-check codes. IRE Transactions on information theory **8**(1), 21–28 (1962)
12. Gionis, A., Indyk, P., Motwani, R.: Similarity search in high dimensions via hashing. In: Proceedings of the 25th International Conference on Very Large Data Bases. p. 518–529. VLDB '99 (1999)
13. Goldman, N., Bertone, P., Chen, S., Dessimoz, C., LeProust, E.M., Sipos, B., Birney, E.: Toward Practical High-capacity Low-maintenance Storage of Digital Information in Synthesised DNA. Nature **494** (2013)
14. Li, H.: Aligning sequence reads, clone sequences and assembly contigs with bwa-mem. arXiv preprint arXiv:1303.3997 (2013)
15. Marinelli, E., Appuswamy, R.: Onejoin: Cross-architecture, scalable edit similarity join for dna data storage using oneapi. In: ADMS (2021)
16. Organick, L., Ang, S.D., Chen, Y.J., Lopez, R., Yekhanin, S., Makarychev, K., Racz, M.Z., Kamath, G., Gopalan, P., Nguyen, B., Takahashi, C.N., Newman, S., Parker, H.Y., Rashtchian, C., Stewart, K., Gupta, G., Carlson, R., Mulligan, J., Carmean, D., Seelig, G., Ceze, L., Strauss, K.: Random access in large-scale DNA data storage. Nature Methods **11**(5) (2014)
17. Perlmutter, M.: The lost picture show. https://tinyurl.com/y9woh4e3 (2017)
18. Shapiro, B.: Mammoth 2.0: will genome engineering resurrect extinct species? Genome Biology (2015)
19. SNIA: 100 year archive requirements survey 10 years later. https://tinyurl.com/yytsbvmb (2017)
20. Ukkonen, E.: Algorithms for approximate string matching. Information and Control **64**(1), 100–118 (1985)
21. Yan, Y., Chaturvedi, N., Appuswamy, R.: Accel-align: A fast sequence mapper and aligner based on the seed-embed-extend method. BMC Bioinformatics (March 2021)
22. Zhang, H., Zhang, Q.: Embedjoin: Efficient edit similarity joins via embeddings. In: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 585–594 (2017)