# Possible structure:

* Introduction
* Related work
* Methodology
  + Mapping and translation
  + Remove screening
  + Altered decoding stage
* Simulation 🡪 Discuss about our simulation procedure
  + ~~Encoding time comparison~~
  + Decoding time comparison
  + File decoding success rate comparison 🡪 Should we show all possible error combination or random error combination
* Experiment 🡪
  + How we got 80 million seq?
  + why second file were reverse complement.
  + Details about synthesis/sequencing technology?
  + Probable Figure:
    - Random sub sample from all the sequences and decoding status
    - Length distribution
    - Number of correct sequence histogram
    - Error distribution graph
    - Ground truth frequency
* Error Analysis:
  + Why we were not able to see 2 sequences at all. Is something wrong with those sequences? Like unbalance GC content, homopolymer run etc?
  + Error count for each sequence(todo)
* Conclusion

# What about the website we made? Should we make it online again? Add dNAM with that probably?

Possible journals to publish:

* European polymer journal
* IEEE transactions on information theory
* Nature chemistry
* Agnewandte Chemie
* Chemical Communications
* Nucleic Acids Research
* IEEE Communications Letters
* Nature Biotechnology \*
* ACS Central Science
* Nanotechnology

# Introduction:

Technological advances in personal computing and computer enabled devices has caused digital technology to become a consistent part of people’s everyday lives. A growing number of us interact with web and mobile applications that track our uses, multiple times a day. Personal and environmental data are collected continuously. This includes personal health data, public government records, Facebook likes, and weather sensor information. Industries are reaching a point where they are trying to keep this ever-expanding amount of information valid and accessible over extended periods of time. Consistent long-term data storage at this volume requires innovation beyond traditional data storage techniques. This paper explores the challenges and possible approaches to storing large datasets using synthetic DNA.

To show the recent growth of data storage, in May of 2013, one researcher estimated that 90% of the world's data was generated in just the previous two years [1]. This makes it clear that the data collection rate and the variety of data being collected will continue to increase. Some types of data need to be accessed often, but information such as scientific, financial, government, historical, genealogical, and genetic records can be stored away for safekeeping for hundreds of years. To combat the excessive use of resources to store vast amounts of data, especially over long periods, researchers are seeking alternative storage techniques. A system in which data could be stored for hundreds or even thousands of years is highly desirable. Deoxyribonucleic acid (DNA) as a data storage material is a promising concept. This organic material is composed of four unique chemical components and is used by organisms to store genetic information. DNA is a durable material that can be stable when stored under the right conditions for hundreds of years [2]. DNA can be stored in water or dry air for long durations as a data storage method. Another advantage of DNA as a storage material is its high data density. Molecules are nanoscale in size, therefore, the physical space needed to store DNA is relatively minuscule. Researchers estimate that today’s global storage needs, about 1022bits, could be stored within a 10x10x10cm3box [2], about the size of a square tissue box. Developing an efficient algorithm to convert arbitrary data to stable strands of quaternary DNA bases, and then back again, can open the door to a new kind of long-term data storage solution.

# Related work

Church et al[3] first introduced DNA data storage idea in larger scale for the first time. They have successfully stored 5.27 megabits of data in DNA. When recovering the data, all blocks were recovered with a total of 10 bit errors out of 5.27 million in the experiment. These errors generally occurred within homopolymer runs at the end of the oligos where there was only single sequence coverage. Their approach represents a promising first step in developing an encoding and decoding algorithm and a significant improvement in the amount of data stored in DNA.

Grass and his colleagues [4] wanted to incorporate a specific error correction technique in an encoding and decoding algorithm. They have implemented two level of Reed Solomon (RS) code which works as an error correction algorithm. The inner code corrected 0.7 nt errors per sequences on average. And the outer code corrected about 0.4% of the sequences.

Random access was first addressed by Bornholt[5] and his team. An unique idea of indexing enabled them to achieve random access. The authors have improved the encoding algorithm that was presented by Goldman and his team[6]. The newly presented algorithm is called XOR encoding that incorporates less redundancy than Goldman’s algorithm.

Organick et al[7] stored 200.2 MB of digital data of varying file sizes in DNA. Which is the highest amount of data that is stored in DNA by far. Essential elements of this algorithm include using an exclusive-or operation, a Reed Solomon parity code and a selective amplification technique to successfully implement random access of the data set. The researchers also designed and validated a library of primers in order to make the experiments possible.

Blawat et al[8] incorporated three level of error correction in their encoding/decoding algorithm. The first kind of error correction was BCH codes used in the address portion of a given sequence. 16 bit Cyclic Redundancy Code (CRC) was used to provide the error correction over the strand's data section. Reed Solomon is used as a final level of error correction that checks the overall strand's integrity. By integrating fountain code Erlich and his team[9] were able to achieve near Shannon Information capacity. They were able to store 2.15MB of digital data, including an operating system and a short movie, and recover the information entirely with no errors. The encoding part of the algorithm comprises three main stages: preprocessing, the Luby Transform, and screening.

In the preprocessing steps the data is compressed first to minimize the data and remove the repeating bits for more randomized data. Next data is split into non-overlapping segment. In Luby Transform stage the segments are converted into droplets using bitwise XOR operation. Reed Solomon code is attached in each droplet for error correction. An unique seed value is also attached to the droplet to identify the segments that are presents in the droplet. The segments for a droplets are chosen using robust soliton probability distribution. The droplets are converted into DNA sequences at the final stage using the following mapping: 00,01,10,11 to A,C,G,T. Sequences that do not conform to specific properties(e.g. equal GC content, homopolymer run less or equal 3) are discarded and not used to generate DNA strands. In the decoding steps only the correct length sequences are used. Using the same mapping scheme that was used during encoding the DNA sequences are converted back into binary data. Reed Solomon code check the validity of the binary data. If the binary data is valid then seed are extracted and using that the decoding algorithm XOR back the droplet and get the file segments. At the last steps all the file segments are combined to form the original file. The fountain code and the screening stage of this algorithm make it more robust than previously described algorithms.

# Methodology:

A new mapping scheme that will convert binary data into quaternary molecules of DNA (A, T, G, C) and vice-versa is presented in this paper. We chose to integrate our hex-to-codon mapping scheme and translation stage into the DNA fountain algorithm [9]. The Fountain Codes algorithm balances a variable amount of parity and redundancy that can be fine-tuned by the user. Another helpful element of this design is the XOR operation which can minimize repeating patterns within the data. Since the indexing seed values represent droplets instead of sequences, this algorithm can be configured to be scalable for large amounts of data. The Fountain Codes algorithm is also ideal because the mapping scheme is entirely separate from the rest of the encoding and decoding processes. This allows us to easily include our mapping and translation stage, which takes more biological constraints into account.

## Mapping and Translation:

The mapping and translation stage begin with the development of the mapping scheme. On the digital side, we chose hexadecimal characters which consists of 4 bits in base 16. On the other side, each hexadecimal value is converted to a codon, or a sequence of three nucleotides. This mapping scheme draws inspiration from nature where ribonucleic acid (RNA) strands are decoded by ribosomes three nucleotides at a time. The mapping and translation stage accounts for four different biological constraints. These include the exclusion of start codons, homopolymer runs of three or more, longer repeating sequences of 9 or more and retaining 45-55% GC content.

We have in total 64 available codons for our hex-to-codons mapping scheme. Due to the constraints mentioned above, certain sets of codons are excluded. For example, homopolymer runs, or repeating nucleotides in a row, are disallowed by removing codons with three repeating nucleotides (AAA, TTT, CCC, and GGG) from the mapping. This ensures that resulting sequences will not contain four or more repeating nucleotides in a row. Along with those 4 codons, 12 other codons, known as start codons, were discarded from the map (See Supplementary Table 1).

Now, 48 codons remain to be included in the mapping scheme. However, we do not want any two codons selected for a sequence to contain any of the 16 removed sequences in their concatenation overlap. To avoid this situation, we carefully constructed the mapping by looking at all possible codon options for a given hexadecimal character that can be followed by all 16 hexadecimal characters. That way, any two consecutive hexadecimal characters can be mapped to two codons in which the encoded sequence of those two codons does not contain a bad codon sequence. For that reason, 9 more codons were excluded from the mapping scheme (See Supplementary Table 2).

While these codons could not be used in the mapping, they are allowed in the final encoded sequences. Unlike the first 16 codons which were removed, the set of 9 do not represent sequences that would contradict the biological constraints. Therefore, the final mapping scheme includes 39 codons. This means that each hexadecimal character has either two or three codons' options to choose from during the data encoding process. Table 1 shows the final hexadecimal-to-codon mapping scheme used in our algorithm.

|  |  |
| --- | --- |
| Hexadecimal | Codons |
| 0 | AAC, GAC |
| 1 | AAG, GAG |
| 2 | AGG, GTC, TCT |
| 3 | TCG, CGA |
| 4 | ACT, GCT, TCC |
| 5 | ACC, GCC, CGT |
| 6 | ACG, GCG |
| 7 | AGA, GGA |
| 8 | AGT, GGT |
| 9 | AGC, GGC, CCG |
| a | GAA, CGG |
| b | TAA, CAA |
| c | TAC, CCT, ATC |
| d | TAG, CGC |
| e | TTA, CTA, GTT |
| f | TTC, CTC, GTA |

Table Hexadecimal to Codons Mapping Scheme

Each hexadecimal character's ability to convert to one out of a set of codon options is a distinct technique for our encoding algorithm. This key feature allows the flexibility to account for more constraints during the translation phase of the encoding scheme. Since there are two or three options of codons to choose from during translation, the system can construct optimal sequences.

During the translation stage, the algorithm works to convert each hexadecimal character into codons. At the start, a random codon is chosen from the set given in the mapping scheme. Then, for the next hexadecimal character, the algorithm attempts to choose a codon from the corresponding set. The algorithm must choose a codon that, when concatenated on the end of the sequence, conforms to each of the biological constraints. If the algorithm does not find a valid codon to continue the sequence, it must backtrack and pick a new codon for the previous hexadecimal character. Backtracking may be needed in accounting for two other biological constraints. The first constraint is excluding longer repeating sequences from a strand. Since the map uses codons, three codons, or nine nucleotides, is the smallest length of a sequence that we can limit to be seen one time in the entire strand. If any sequence of three codons is seen a second time in the translation stage, the algorithm will backtrack and choose a new codon. This stage also accounts for the ratio of GC codons to AT codons. As was discussed earlier, a given sequence should have close to 50% GC content.

The droplets are generated using he Fountain code algorithm[9]. Each droplet’s sequence of binary data is converted to a sequence of nucleotides using the mapping scheme and translation process detailed earlier. If a droplet's binary information cannot be converted to a valid nucleotide sequence within the allowable number of backtracks, that droplet is discarded and another one is generated.

Besides employing our hex-to-codon mapping scheme, we integrated two other significant updates to the DNA fountain algorithm. These updates include removing the screening stage and altering the decoding algorithm.

## Removed Screening Stage

Unlike Erlich and Zielinski’s DNA Fountain implementation, we did not include a screening stage. Since the hex-to-codon mapping scheme and the translation process accounted for the biological constraints that the screening process looked for, that step was no longer needed. Now, for our implementation, droplets are no longer discarded if their sequences do not pass the two biological constraints. Instead, our translation process attempts to find a valid sequence within a given number of backtracks. By setting a limit for the number of times the system can backtrack, we shorten the running time. Although, by moving the biological constraint satisfaction process from a post-processing screening stage to the translation process, we still observed two main improvement over the DNA Fountain algorithm.

The fact that droplets are not excluded at as high of a rate has a positive effect on the algorithm’s ability to recover encoded data. Our algorithm typically discards about 10% of the droplets generated, while DNA Fountain can exclude up to 88%. Since our algorithm does not exclude a majority of droplets, we are more likely to reach the intended distribution. We are also more likely to keep the single-segment droplets that are imperative to the successful recovery of the data. As an unintended side effect, we also achieved an decrease in running time. Since DNA Fountain has to generate more droplets than are actually accepted, there is a waste in processing time. Another chance to improve the running time was discovered in the decoding algorithm.

## Altered Decoding stage

We made changes to the decoding algorithm in order to speed up the computational running time. More specifically, we changed the way single-segment droplets are propagated throughout other droplets with the same segment. Erlich and Zeilinski’s DNA Fountain implementation used a depth-first approach for following the path of connected droplets. We instead used a breadth-first approach, which showed a significant decrease in running time.

After the initial recovery and translation of each sequence, the key value is used to seed both random number generators, and eventually, each droplet knows how many segments it contains and precisely which segments from the original dataset it contains. Similar to the DNA Fountain decoding algorithm, our implementation also starts by examining droplets with precisely one segment. Now, other recovered droplets may also include that segment and need to remove it. The difference in our decoding approach is the order in which individual recovered segments are removed from the other droplets that also contain that segment. DNA Fountain solves the depth-first approach with a recursive function. The function starts with a recovered segment. It then finds other droplets that have the same segment. It starts at the beginning of that list of other droplets and removes the segment from the first droplet. There is a chance that the first droplet in the list only has two total segments. So, if the initial segment is removed, the process has now recovered one other single-segment. Instead of moving on to the other droplets that contain the initial segment, the function now processes the newly recovered segment. Each time a single-segment is recovered, it is processed immediately. On the other hand, we implemented a breadth-first approach for processing these single-segments. Therefore, when a recovered segment is removed from a droplet with two segments and a new single-segment is discovered, that segment is added to a the first-in-first-out (FIFO) queue instead of being immediately processed. The starting single-segment is first removed from all other droplets that also contain that segment. Any time a new single segment is discovered, it is simply added to the queue. Once the starting single-segment is finished being removed from all other droplets, the algorithm moves on to process the first segment in the queue. This process is repeated until the queue is empty. If all single segment in the queue have been processed and the dataset has not been wholly encoded, more droplets are processed. The results of a breadth-first decoding approach versus a depth-first approach includes fewer calls to the XOR operation in the code. For example, for tests run on a randomly generated one-megabyte file, the depth-first DNA Fountain implementation made 723,706 calls to the XOR operation. On the other hand, our algorithm breadth-first implementation only made 14,636 calls to the XOR operation. The line of code for the XOR operation is one of the most expensive in terms of computational runtime. We believe that the reduction in the number of XOR operations has reduced the computational run time for the overall decoding program significantly. Besides changing this algorithm, other small changes were made to clean up the code and decrease the running time. We got 99.77 percent of decoding time improvement on a randomly generated 50 MB file.

# Result

A screenshot of a cell phone

Description automatically generated

Chart, scatter chart

Description automatically generated

**Chart, scatter chart

Description automatically generated**

# Conclusion/Future work

# Reference:

[1] “Big Data, for better or worse: 90% of world’s data generated over last two years,” *ScienceDaily*. https://www.sciencedaily.com/releases/2013/05/130522085217.htm (accessed Sep. 29, 2020).

[2] V. Zhirnov, R. M. Zadegan, G. S. Sandhu, G. M. Church, and W. L. Hughes, “Nucleic acid memory,” *Nature Mater*, vol. 15, no. 4, pp. 366–370, Apr. 2016, doi: 10.1038/nmat4594.

[3] G. M. Church, Y. Gao, and S. Kosuri, “Next-generation digital information storage in DNA,” *Science*, vol. 337, no. 6102, p. 1628, Sep. 2012, doi: 10.1126/science.1226355.

[4] R. N. Grass, R. Heckel, M. Puddu, D. Paunescu, and W. J. Stark, “Robust Chemical Preservation of Digital Information on DNA in Silica with Error-Correcting Codes,” *Angew. Chem. Int. Ed.*, vol. 54, no. 8, pp. 2552–2555, Feb. 2015, doi: 10.1002/anie.201411378.

[5] J. Bornholt, R. Lopez, D. M. Carmean, L. Ceze, G. Seelig, and K. Strauss, “A DNA-Based Archival Storage System,” in *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems - ASPLOS ’16*, Atlanta, Georgia, USA, 2016, pp. 637–649, doi: 10.1145/2872362.2872397.

[6] N. Goldman *et al.*, “Towards practical, high-capacity, low-maintenance information storage in synthesized DNA,” *Nature*, vol. 494, no. 7435, pp. 77–80, Feb. 2013, doi: 10.1038/nature11875.

[7] L. Organick *et al.*, “Random access in large-scale DNA data storage,” *Nat Biotechnol*, vol. 36, no. 3, pp. 242–248, Mar. 2018, doi: 10.1038/nbt.4079.

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

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

# Supplementary:

|  |
| --- |
| AAT |
| ATA |
| ATT |
| ATG |
| CAC |
| CAT |
| CAG |
| CTT |
| CTG |
| TAT |
| TTG |
| GTG |

Table List of start codons excluded from the map

During the translation process, in which ribosomes create proteins, specific start codons signal the ribosomes to start processing. When generating artificial DNA, our algorithm excludes these start codons from the sequences. This is useful because artificial data DNA comes into contact with any biologically active translation system. The results of this type of interaction are unknown.

|  |
| --- |
| ACA |
| TCA |
| CCA |
| GCA |
| TGA |
| TGT |
| TGC |
| TGG |
| GAT |

Table List of other codons excluded from the mapping scheme

For any given hexadecimal character, it may be mapped to a specific codon. Then it should be possible for another codon to follow the first one for each hexadecimal character. This means that there must be a valid codon option to follow the first one that would not contain a sequence that matches any of the 16 omitted codons. In following that rule, we discovered that specific sequences of characters could not be avoided by including some codons. For example, CAT, CAC, and CAG are all codons in the list of start codons to avoid. So, any codon that ended with CA could only be followed by a codon that started with an A, but not AA. After excluding codons that started with AA and any other codons in the list of excluded codons, only 8 possibilities could follow any particular codon that ended with CA. These 8 possible codons are not enough to give an option for each of the 16 hexadecimal characters in the mapping scheme so the four codons, starting with CA, had to be excluded from the final mapping scheme.