Capacity-approaching DNA storage

Yaniv Erlich[1,2,3,+], Dina Zielinski[1]

[1] New York Genome Center, New York, NY 10013, USA
[2]Department of Computer Science, Fu Foundation School of Engineering, Columbia University, New York, NY, USA.
[3]Center for Computational Biology and Bioinformatics (C2B2), Department of Systems Biology, Columbia University, New York, NY, USA.

[+] To whom correspondence should be addressed (yaniv@cs.columbia.edu)

Abstract

Humanity produces data at exponential rates, creating a growing demand for better storage devices. DNA molecules are an attractive medium to store digital information due to their durability and high information density. Recent studies have made large strides in developing DNA storage schemes by exploiting the advent of massive parallel synthesis of DNA oligos and the high throughput of sequencing platforms. However, most of these experiments reported small gaps and errors in the retrieved information. Here, we report a strategy to store and retrieve DNA information that is robust and approaches the theoretical maximum of information that can be stored per nucleotide. The success of our strategy lies in careful adaption of recent developments in coding theory to the domain specific constrains of DNA storage. To test our strategy, we stored an entire computer operating system, a movie, a gift card, and other computer files with a total of 2.14×10^6 bytes in DNA oligos. We were able to fully retrieve the information without a single error even with a sequencing throughput on the scale of a single tile of an Illumina sequencing flow cell. To further stress our strategy, we created a deep copy of the data by PCR amplifying the oligo pool in a total of nine successive reactions, reflecting one complete path of an exponential process to copy the file 218×10^{12} times. We perfectly retrieved the original data with only five million reads. Taken together, our approach opens the possibility of highly reliable DNA-based storage that approaches the information capacity of DNA molecules and enables virtually unlimited data retrieval.

Main Text

DNA is an excellent medium for storing information. It offers tantalizing information density of petabytes of data per gram, high durability, and over 3 billion years of evolutionary optimization of the machinery to faithfully replicate this information[1,2]. Recently, a series of proof-of-principle experiments have demonstrated the value of DNA as a storage medium[3–9].

To better understand its potential, we explored the Shannon information capacity[10,11] of DNA storage (**Supplementary Material)**. This key measure sets a tight upper bound on the amount of information that can be reliably stored in each nucleotide. In an ideal world, the information capacity of each nucleotide could reach 2bits since there are four possible options. However, DNA encoding faces several practical limitations. First, not all DNA sequences are created equal[12,13]. Biochemical constraints dictate that DNA sequences with high GC content or long homopolymer runs (e.g. `AAAAAA`…) should be avoided as they are difficult to synthesize and prone to sequencing errors. Second, oligo synthesis, PCR amplification, and decay of DNA during storage are all processes that induce uneven representation of the oligos[7,14]. This might result in dropout of a small fraction of oligos that will not be available for decoding. In addition to the biochemical constraints, oligos are sequenced in a pool and necessitate indexing to infer their order, which further limits the real estate for encoding information. Quantitative analysis of these constraints shows that the Shannon information capacity of a DNA storage device is at least 1.83bits per each nucleotide for a range of practical architectures (**Supplementary Material; Supplementary Figures 1-5; Supplementary Tables 1-2**).

Previous studies of DNA storage mostly realized about half of the Shannon information capacity of DNA molecules (**Table 1**). For example, Church et al.[3] used an encoding scheme that maps zeros to either 'A' or 'C' and ones to either 'G' or 'T'. This redundancy prevents homopolymer runs and controls the GC content but reduces the information content to 1bit/nt. In addition, some of the previous schemes addressed oligo dropouts by dividing the original file into overlapping segments so that each input bit is represented by multiple oligos[4,6]. But this repetition coding procedure generates a massive loss of information content since multiple nucleotides carry essentially the same information. Moreover, repetition is not scalable. As files get larger, even a small dropout probability can eventually translate to a near guaranteed corruption of the stored information for any practical fold coverage (**Supplementary Figure 6**). Finally, most previous results reported small gaps in the retrieved information[3,4,6]. Taken together, these results inspired us to seek a robust coding strategy that can better utilize the information capacity of DNA memory devices.

	Church et al.[3]	Goldman et al.[4]	Grass et al.[5]	Bornholt et al.[6]	Blawa et al.[7]	This work
Input data[Mbyte]	0.65	0.75	0.08	0.15	22	2.11
Coding potential[(1)] [bits/nt]	1	1.26	1.78	1.26	1.6	1.98
Robustness to dropouts	No	Yes	No	Yes	Yes	Yes
Redundancy[(2)]	1	4	1	1.5	1.13	1.07
Error correction/detection[(3)]	No	Yes	Yes	No	Yes	Yes
Net density [(4)][bits/nt]	0.83	0.33	1.14	0.88	0.92	1.55
Full recovery[(5)]	No	No	Yes	No	Yes	Yes

Table 1: **Comparison of DNA storage coding schemes and experimental results**. For consistency, the table describes only schemes that were empirically tested with high throughput sequencing data. The schemes are presented chronologically based on date of publication [(1)] Coding potential is the maximal information content of each nucleotide before indexing or error correcting [(2)] Redundancy denotes the excess of synthesized oligos to provide robustness to dropouts [(3)] The presence of error correcting/detection code to handle synthesis and sequencing errors [(4)] The input information in bits divided by the number of overall synthesized bases (excluding adapter annealing sites) [(5)] Whether all information was recovered without any error. See **Supplementary Material** for in-depth discussion.

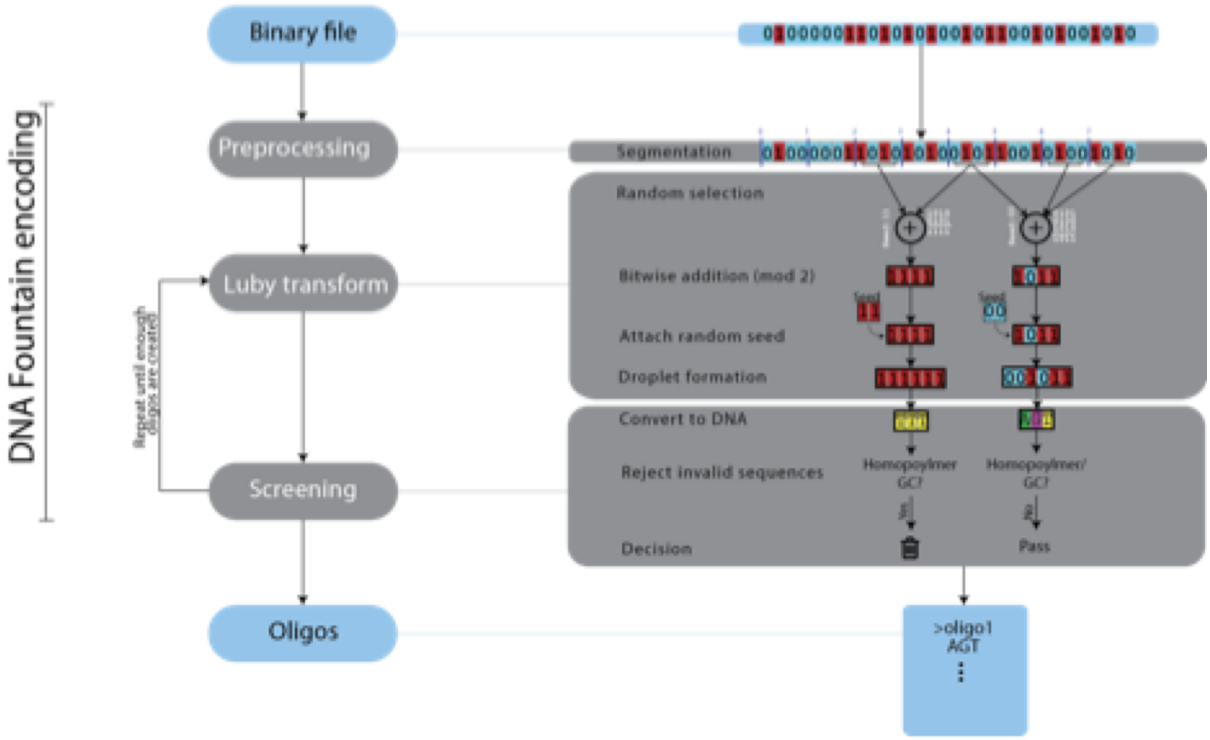


Figure 1: DNA fountain encoding. Left: three main algorithmic steps. Right: an example with a small file of 48bits. For simplicity, we partitioned the file to 8 segments of 4bits. The seeds are represented as 2bit numbers and are presented for display purposes only. Full details of each algorithmic step are given in the **Supplementary Information**.

We devised a new strategy, dubbed DNA Fountain, for DNA storage devices, which approaches the Shannon capacity while providing robustness against data corruption. Our encoder works in three steps (**Figure 1; Supplementary Material**). First, it preprocesses a binary file into a series of non-overlapping segments of a certain length. Next, it iterates over two *computational* steps: Luby Transform and screening. The Luby Transform[15,16] is a relatively recent addition to coding theory and sets the basis for fountain codes, which are used in various communication standards such as mobile TV[17]. Basically, the transform packages data into any desired number of short messages, called droplets, which are transmitted over a noisy channel. A user can recover the file by collecting any subset of droplets as long as the accumulated size of droplets is slightly bigger than the size of the original file. In DNA Fountain, we apply one round of the transform in each iteration to create a single droplet. Briefly, the Luby Transform selects a random subset of segments using a special distribution (**Supplementary Figure 7**) and adds them bitwise together under a binary field. The droplet contains two pieces of information: a data payload part that holds the result of the addition procedure and a short, fixed-length seed. This seed corresponds to

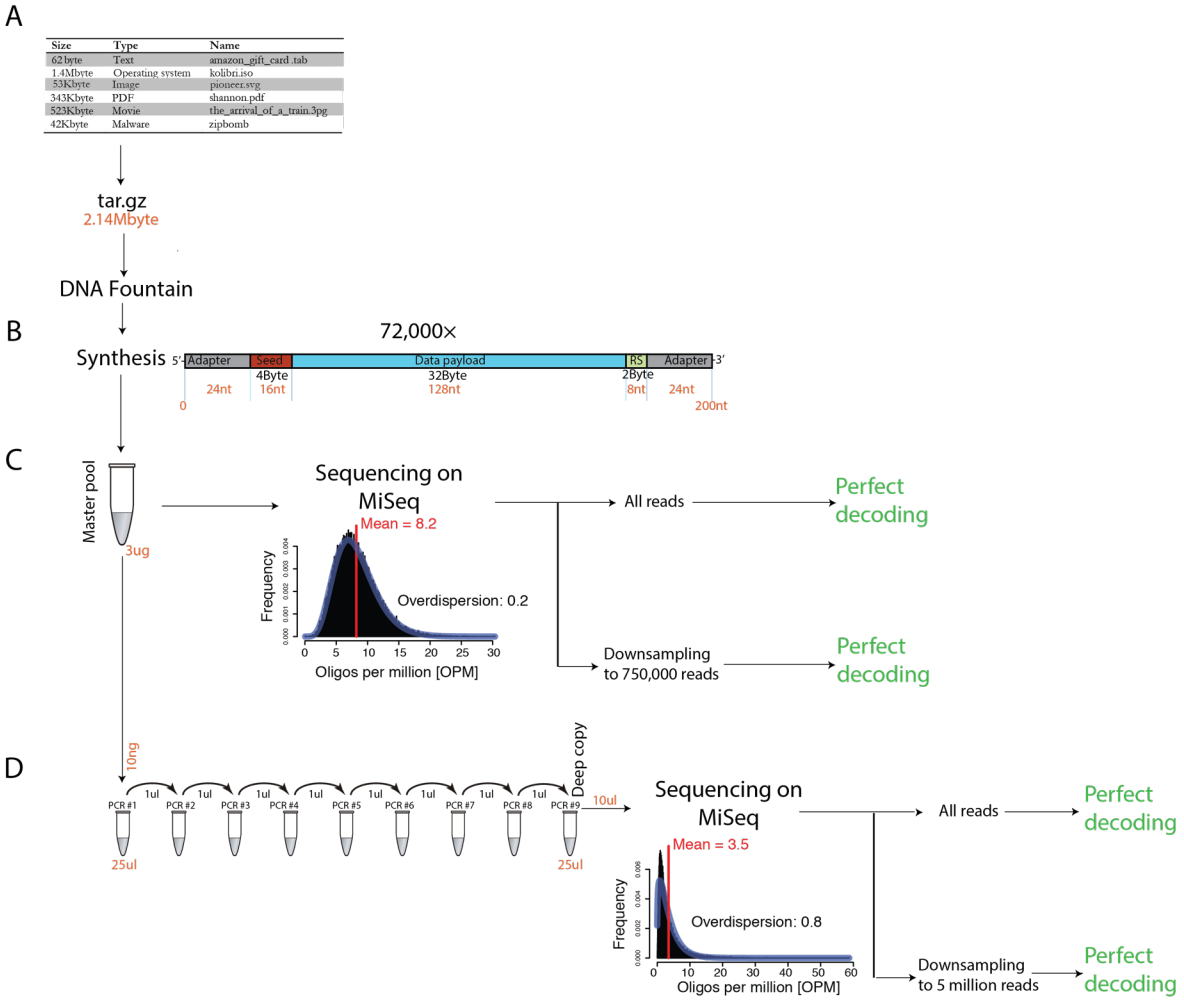


Figure 2: Experimental setting and results for storing data on DNA (A) The input files for encoding, size, and type. The total amount of data was 2.14Mbyte after compression **(B)** The structure of the oligos. Black labels: length in bytes. Red: length in nucleotides. RS: Reed-Solomon error correcting code **(C)** experimental results of the master pool **(C+D)** The histogram displays the frequency of perfect calls per million (pcpm) sequenced reads. Red: mean, blue: negative binomial fitting to the pcpm **(D)** Experimental procedures of deep copying the oligo pool.

the state of the random number generator of the transform during the droplet creation and allows the decoder algorithm to infer the identities of the segments in the droplet. Next, in the droplet screening stage, the algorithm first translates the binary droplet to a DNA sequence by converting {00,01,10,11} to `{A,C,G,T}`, respectively. Then, it screens the sequence for desired biochemical properties such as GC content and homopolymer runs. If the sequence passes the screen, it is considered valid and added to the oligo design file; otherwise, the algorithm simply trashes the droplet. Since the Luby Transform can create any desired number of droplets, we keep iterating over the droplet creation and screening steps until a sufficient number of valid oligos are generated. In practice, we recommend generating at least 5% more oligos than the number of segments for robustness against dropouts and convergence of the decoder (**Supplementary Material**). Searching for valid oligos scales well with the size of the input file and is economical for various oligo lengths within and beyond current synthesis limits (**Supplementary Material; Supplementary Table 3**).

We used DNA Fountain to encode a single compressed file of 2,146,816 bytes in a DNA oligo pool The input data was a tarball that packaged several files, including a complete operating system and a $50 Amazon gift card (**Figure 2A; Supplementary Figure 8; Supplementary Material**). We split the input tarball into 67,088 segments of 32bytes and iterated over the steps of DNA Fountain to create valid oligos. Each droplet was 38bytes (304bits): 4bytes of the random number generator seed, 32bytes for the data payload, and 2bytes for a Reed-Solomon error correcting code, to confer robustness against sequencing errors in low coverage conditions. With this seed length, our strategy supports encoding files of up to 500Mbyte (**Supplementary Material**). The DNA oligos had a length of (304/2)=152nt and were screened for homopolymer runs of ≤3nt and GC content of 45%-55%. We instructed DNA Fountain to generate 72,000 oligos, yielding a redundancy of (72,000/67,088-1)=7%. We selected this number of oligos mainly because the manufacturer offered a flat price for 66,000-72,000 oligos, allowing us to maximize the number of oligos per dollar. Finally, we added upstream and downstream annealing sites for Illumina adapters, making our final oligos 200nt long (**Figure 2B; Supplementary Figure 9**). The entire encoding time took 2.5min on a single CPU of a standard laptop and achieved an information density of 1.55bit/nt, only 15% from the Shannon capacity of DNA storage and 60% more than previous studies with proven robustness for dropouts (**Table 1**).

Sequencing and decoding the oligo pool fully recovered the entire input file with zero errors (**Figure 2C**). To retrieve the information, we PCR-amplified the oligo pool and sequenced the DNA library on one MiSeq flowcell with 150bp pair-end reads, which yielded 32 million sequences. After pre-processing the library to collapse identical reads, we ran the decoder algorithm. This algorithm recovered the droplet by mapping the read to its original binary format, rejected droplets with errors based on the Reed-Solomon code, and employed a message passing algorithm to reverse the Luby Transform and obtain the original data (**Supplementary Material)**. In practice, decoding took approximately 9min using a Python script on a single CPU of a standard laptop (**Supplementary Movie 1**). The decoder recovered the input tar file with 100% accuracy after observing only 69,870 oligos out of the 72,000 in our library (**Supplementary Figure 10**). To further test the robustness of our strategy, we down-sampled the raw Illumina data to 750,000 reads, which are equivalent to one tile of an Illumina MiSeq flow cell. This procedure resulted in 1.3% oligo dropout from the library. Despite these limitations, the decoder was able to perfectly recover the original 2.1Mb in twenty out of twenty random down-sampling experiments. These results indicate that beyond its high information density, DNA Fountain can also reduce the amount of sequencing required for data retrieval, which would be beneficial when storing large-scale information.

DNA Fountain can also perfectly recover the file after creating a deep copy of the sample. One of the caveats of DNA storage is that each retrieval of information consumes an aliquot of the material. Copying the oligo library using PCR is possible, but this procedure introduces noise and induces oligo dropout. To further test the robustness of our strategy, we created a deep copy of the file by propagating the sample through nine serial PCR amplifications (**Figure 2D; Supplementary Material**). The first PCR reaction used 10ng of material out of the 300ng master pool. Each subsequent PCR reaction consumed 1ul of the previous PCR reaction and employed 10 cycles in each 25ul reaction volume. We sequenced the final library using one run on the Illumina MiSeq. Overall, this recursive PCR reflects one full arm of an exponential process that theoretically could generate $30\times25^9\times2$ = 228 trillion copies of the file by repeating the same procedure with each aliquot (**Supplemental Figure 11**). As expected, the quality of the deep copy was substantially worse than the initial experiment with the master pool. The average coverage per oligo dropped from an average of 7.8 perfect calls for each oligo per million reads (pcpm) to 4.5pcpm in the deep copy. In addition, the deep copy showed much higher skewed representation with a negative binomial overdispersion parameter (1/size) of 0.76 compared to 0.15 in the master pool. Despite the lower quality, the DNA Fountain decoder was able to fully recover the file without a single error with the full sequencing data. We also down-sampled the sequencing data to five million reads, which resulted in approximately 1.0% dropout rate. Yet, we were able to perfectly recover the file in ten out of ten trials.

These results suggest that with DNA fountain, DNA storage can be copied virtually an unlimited number of times while preserving the data integrity of the sample.

DNA as a storage medium is gaining increased attention. During the preparation of this manuscript, Microsoft publicly announced the beginning of large scale experiments to store about 100Mb of data on DNA and the United State's IARPA has expressed interest in this domain[18]. The cost reduction of DNA synthesis exceeds Moore's law[19], meaning that large scale DNA storage might be economically feasible in the next few years. Such storage will require coding methods that better realize the information capacity of DNA and enable strong integrity for the stored information.

References

1. Wallance, M. Molecular cybernetics: the next step? *Kybernetes* **7,** 265–268 (1978).
2. Bancroft, C., Bowler, T., Bloom, B. & Clelland, C. T. Long-term storage of information in DNA. *Science* **293,** 1763–1765 (2001).
3. Church, G. M., Gao, Y. & Kosuri, S. Next-generation digital information storage in DNA. *Science* **337,** 1628 (2012).
4. Goldman, N. *et al.* Towards practical, high-capacity, low-maintenance information storage in synthesized DNA. *Nature* **494,** 77–80 (2013).
5. Grass, R. N., Heckel, R., Puddu, M., Paunescu, D. & Stark, W. J. Robust Chemical Preservation of Digital Information on DNA in Silica with Error-Correcting Codes. *Angew. Chem. Int. Ed.* **54,** 2552–2555 (2015).
6. Bornholt, J. *et al.* A DNA-based archival storage system. in *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems* 637–649 (ACM, 2016).
7. Blawat, M. *et al.* Forward Error Correction for DNA Data Storage. *Procedia Comput. Sci.* **80,** 1011–1022 (2016).
8. Yazdi, S. H. T. *et al.* DNA-based storage: Trends and methods. *IEEE Trans. Mol. Biol. Multi-Scale Commun.* **1,** 230–248 (2015).
9. Yazdi, S. H. T., Yuan, Y., Ma, J., Zhao, H. & Milenkovic, O. A rewritable, random-access DNA-based storage system. *Sci. Rep.* **5,** (2015).
10. Shannon, C. E. A mathematical theory of communication. *ACM SIGMOBILE Mob. Comput. Commun. Rev.* **5,** 3–55 (2001).
11. MacKay, D. J. C. *Information Theory, Inference & Learning Algorithms*. (Cambridge University Press, 2002).
12. Schwartz, J. J., Lee, C. & Shendure, J. Accurate gene synthesis with tag-directed retrieval of sequence-verified DNA molecules. *Nat Methods* **9,** 913–915 (2012).
13. Ross, M. G. *et al.* Characterizing and measuring bias in sequence data. *Genome Biol* **14,** R51 (2013).
14. Erlich, Y. *et al.* DNA Sudoku—harnessing high-throughput sequencing for multiplexed specimen analysis. *Genome Res.* **19,** 1243–1253 (2009).
15. Luby, M. LT codes. *Found. Comput. Sci. 2002 Proc. 43rd Annu. IEEE Symp. On* 271–280 (2002). doi:10.1109/SFCS.2002.1181950
16. MacKay, D. J. C. Fountain codes. *IEE Proc. - Commun.* **152,** 1062–1068 (2005).
17. Stockhammer, T., Shokrollahi, A., Watson, M., Luby, M. & Gasiba, T. *Application layer forward error correction for mobile multimedia broadcasting*. (CRC Press, 2008).
18. Extance, A. How DNA could store all the world's data. *Nature* **537,** 22–24 (2016).
19. Kosuri, S. & Church, G. M. Large-scale de novo DNA synthesis: technologies and applications. *Nat. Methods* **11,** 499–507 (2014).

Acknowledgments

Y.E. holds a Career Award at the Scientific Interface from the Burroughs Wellcome Fund. This study was supported by a generous gift from Andria and Paul Heafy to the Erlich Lab. We thank P. Smibert and M. Stoeckius for technical assistance with oligo design, N. Abe and S. Pescatore for sequencing operations, A. Gordon for creating the movie, and S. Zaaijer and N. Sanjana for useful comments and discussions. After posting the original version of this manuscript on bioRxiv, it was brought to our attention that Sir David Mackay also explored the usage of fountain codes for DNA storage before his death. We dedicate this manuscript in his memory.

Supplementary Information for

Capacity-approaching DNA storage

Yaniv Erlich*, Dina Zielinski

*Correspondence to: yaniv@cs.columbia.edu

Contents

List of Figures 2

List of Tables 2

1 The Shannon Information Capacity of DNA Storage 3
- 1.1 Constraints in encoding DNA information . . . 3
- 1.2 Quantitative analysis . . . 4
  - 1.2.1 The homopolymer constraint: . . . 5
  - 1.2.2 The GC content constraint: . . . 7
  - 1.2.3 Putting the biochemical constraints together: . . . 7
  - 1.2.4 Index length: . . . 8
  - 1.2.5 Dropouts: . . . 9

2 Calculating the net density of DNA storage schemes 10

3 The DNA Fountain coding strategy 12
- 3.1 Encoding overview . . . 12
- 3.2 Seed schedule . . . 14
- 3.3 Tuning the Soliton distribution parameters . . . 15
- 3.4 Decoding . . . 16
- 3.5 DNA Fountain overhead . . . 17
  - 3.5.1 Code rate . . . 17
  - 3.5.2 The time complexity and economy of encoding and decoding . . . 18

4 DNA Fountain experiments 19
- 4.1 Computing . . . 19
- 4.2 Software, data, and input files . . . 19
- 4.3 Command line steps to encode the data . . . 20
- 4.4 Molecular procedures . . . 21
- 4.5 Decoding the data . . . 22
- 4.6 Creating a deep copy . . . 25

References 26

List of Figures

1 **Supplementary Figure** 1 3
2 **Supplementary Figure** 2 4
3 **Supplementary Figure** 3 6
4 **Supplementary Figure** 4 7
5 **Supplementary Figure** 5 9
6 **Supplementary Figure** 6 12
7 **Supplementary Figure** 7 15
8 **Supplementary Figure** 8 20
9 **Supplementary Figure** 9 22
10 **Supplementary Figure** 10 24
11 **Supplementary Figure** 11 25

List of Tables

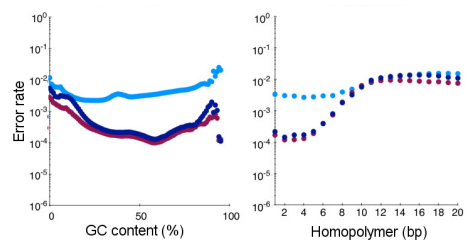
1 **Supplementary Table** 1 4
2 **Supplementary Table** 2 6
3 **Supplementary Table** 3 19

1 The Shannon Information Capacity of DNA Storage

1.1 Constraints in encoding DNA information

DNA storage is basically a communication channel. We transmit information over the channel by synthesizing DNA oligos. We receive information by sequencing the oligos and decoding the sequencing data. The channel is noisy due to various experimental factors, including DNA synthesis imperfections, PCR dropout, stutter noise, degradation of DNA molecules over time, and sequencing errors. Different from classical information theoretic channels (e.g. the binary symmetric channel) where the noise is identical and independently distributed, the error pattern in DNA heavily depends on the input sequence. Previous studies of error patterns identified that homopolymer runs and GC content are major determinants of synthesis and sequencing errors [1, 2, 3, 4, 5, 6, 7]. For example, Schwartz et al. [1] reported that oligos with a GC content above 60% exhibit high dropout rates and that most PCR errors occur in GC rich regions. Ross et al. [2] studied sequencing biases across a large number of genomes. They found that once the homopolymer run is more than 4nt, the insertion and deletion rates start climbing and genomic regions with both high and low GC content are underrepresented in Illumina sequencing (**Supplementary Figure** 1). Ananda et al. [5] studied PCR slippage errors and found a rapid increase in homopolymers greater than 4bp. On the other hand, oligos without these characteristics usually exhibit low rates (1%) of synthesis and sequencing errors[8].

To facilitate quantitative analysis, we model the sequence-specific noise by grouping DNA oligos into two classes: valid and invalid. A sequence will be considered valid if its GC content is within $0.5 \pm c_{gc}$ and its longest homopolymer length is up to m nucleotides. Otherwise, it will be considered invalid and cannot not be transmitted. The coding potential, b, describes the entropy of each nucleotide in valid sequences. Next, valid sequences are exposed to a low δ_v dropout rate. Due to the multiplexing architecture of synthesis reactions and high throughput sequencing, the oligos are completely mixed in a pool. Therefore, we need to index each oligo with a short tag. The length of the total oligo will be denoted by l and the fraction of nucleotides that are used for the index will be denoted by i and we will use K to denote the number of segments needed for decoding in the input file. By selecting realistic values for m, c_{gc}, i, and δ_v, we can approximate the information capacity, which is defined as the upper bound on the number of bits that can be encoded per nucleotide. Our model does not include synthesis and sequencing errors for valid oligos. Previous work involving DNA storage with high throughput sequencing has shown that it is possible to achieve an error-free



Supplementary Figure 1: Error rates of Illumina sequencing as a function of GC content and homopolymer length. Light blue: mismatches; dark blue: deletions; purple: insertions. The figure was modified from Ross et al. [2] with permission according to publisher license CC-BY-2.0

Parameter	Meaning	Possible values	Analysis
m	Maximal homopolymer length	3nt	Sec. 1.2.1
c_{gc}	Maximal deviation of GC content from 0.5	$0.05 - 0.2$	Sec. 1.2.2
b	Coding potential[bits]	1.98	Sec. 1.2.3
i	Index overhead	$0.05 - 0.1$	Sec. 1.2.4
l	Oligo length	$100 - 200$nt	Sec. 1.2.4
δ_v	Dropout rates	$0 - 0.005$	Sec. 1.2.5

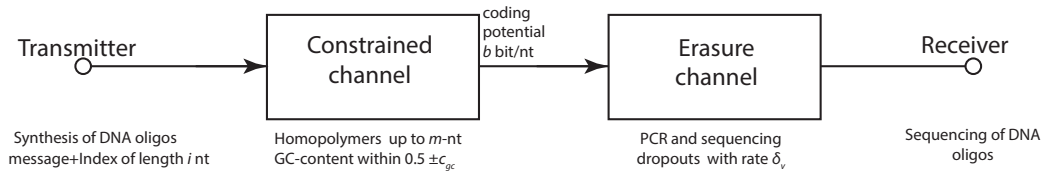
Supplementary Table 1: Key parameters in the channel model for DNA storage

consensus for the original oligos by deep sequencing the oligo pool [8, 9, 10]. We observed a similar pattern in our experiments. When deep sequencing coverage was used (250×), we could reach an error-less recovery without implementing the error correcting code. In a case of low sequencing coverage, our analysis is likely to provide an upper bound on the capacity of DNA storage.

In the next sections, we will show that the information capacity per nucleotide is approximately 1.83bits by setting conservative but realistic parameters.

1.2 Quantitative analysis

Under the model above, a DNA storage device behaves as a constrained channel concatenated to an erasure channel (**Supplementary Figure** 2)



Supplementary Figure 2: Channel model.

Let A_X be the set of all possible transmitted DNA sequences of length l nucleotides, and A_Y be the set of all possible received sequences. X and Y denote random DNA sequences from A_X and A_Y, respectively.

The mutual information of X and Y is defined as:

$$I(X;Y) \equiv H(X) - H(X|Y) \tag{1}$$

where $H(X)$ is the entropy (in bits) of X and $H(X|Y)$ is the conditional entropy of X given Y, or the expected residual uncertainty of the receiver about a transmitted sequence.

The information capacity of each oligo is defined as:

$$C \equiv \max_{p_X} I(X;Y) \tag{2}$$

and the information capacity per nucleotide is:

$$C_{nt} \equiv C/l \tag{3}$$

In a constrained channel, the mutual information is maximized by equiprobable transmission of only valid sequences ([11] pp. 250-251). Thus, $H(X) = \log_2 |A_X|$, where $|\cdot|$ denotes the size of items in a set. Valid sequences are either perfectly received with a probability of $1-\delta_v$ or dropped out with a probability of δ_v. In the former case, the conditional entropy is $H(X|Y) = 0$, whereas in the latter case the conditional entropy is $H(X|Y) = H(X)$. Therefore, overall, $H(X|Y) = \delta_v H(X)$ and the capacity of each nucleotide is:

$$\begin{aligned} C_{nt} &= \frac{H(X) - H(X|Y)}{l} \\ &= \frac{H(X) - \delta_v H(X)}{l} \\ &= \frac{(1-\delta_v)H(X)}{l} \\ &= \frac{(1-\delta_v)\log_2 |A_X|}{l} \end{aligned} \tag{4}$$

1.2.1 The homopolymer constraint:

With the homopolymer constraint, the size of the set of all valid code words is:

$$|A_X^{\mathtt{h}}| = Q(m,l) \cdot 4^l \tag{5}$$

where $|A_X^{\mathtt{h}}|$ denotes the size of the A_X set under the homopolymer constraint and $Q(m,l)$ is the probability to observe up to an m-nt homopolymer run in a random l-nt sequence.

We will start by analyzing a simpler case of the probability of *not* observing a run of m or more successes in l Bernoulli trials with a success probability p and failure probability of $q = 1-p$, denoted by $q_m(p,l)$. Feller [12] proposed (pp.301-303) a tight approximation for $q_m(p,l)$:

$$q_m(p,l) \approx \frac{\beta}{x^{l+1}} \tag{6}$$

where x is:

$$x = 1 + qp^m + (m+1)(qp^m)^2 \tag{7}$$

and β is:

$$\beta = \frac{1 - px}{(m + 1 - mx)q} \tag{8}$$

Previous studies have analyzed the general case of the probability distribution function for runs of a collection of symbols from a finite alphabet. The formulae of these distributions are typically defined using recursion or require spectral analysis of matrices with complex patterns (e.g. [13, 14, 15]) and resist analytic analysis. For practical purposes, we approximate the distribution of observing up to m-nt homopolymer runs as the product of four independent events:

$$Q_{(m,l)} \approx \Big[q_{m+1}(p = 0.25, l)\Big]^4 \tag{9}$$

Supplementary Table 2 presents the output of Eq. 9 versus the expected rate of homopolymers. Overall the approximation is quite consistent with the observed rate for relevant oligo lengths and homopolymer constraints.

l	m	Obs. $Q_{(m,l)}$	Est. $Q_{(m,l)}$
100	3	0.31	0.31
250	3	0.05	0.05
100	4	0.75	0.75
250	4	0.48	0.48
100	5	0.94	0.93
250	5	0.83	0.84

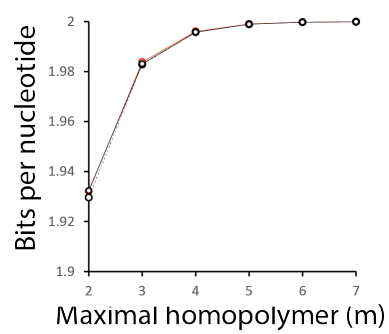
Supplementary Table 2: The observed rate of homopolymer mismatches versus the estimated rates from Eq. 9

Combining Eq. 5, Eq. 6, and Eq. 9:

$$\begin{aligned}\frac{\log_2 |A_X^{\mathtt{h}}|}{l} &= \frac{\log_2[4^l \cdot Q(m,l)]}{l} \\ &= 2 + \frac{4\log_2(\frac{\beta}{x^{l+1}})}{l} \\ &= (2 - 4\log_2 x) + \frac{4[\log_2 \beta - \log_2 x]}{l}\end{aligned} \tag{10}$$

For any $m \geq 3$ and $l \geq 50$, we can further approximate:

$$\begin{aligned}\frac{\log_2 |A_X^{\mathtt{h}}|}{l} &= (2 - 4\log_2 x) + \frac{4[\log_2 \beta - \log_2 x]}{l} \\ &\approx 2 - 4\log_2 x \\ &\approx 2 - 4\Big[qp^{m+1} + (m+2)q^2p^{2m+2}\Big]\log_2 e \\ &\approx 2 - \frac{3\log_2 e}{4^{m+1}}\end{aligned} \tag{11}$$



Supplementary Figure 3: Bits per nucleotide as a function of m and l. Black: $l = 50$nt, red: $l = 10^{11}$nt, broken line: approximating the bits per nucleotide using Eq. 11. The three curves almost entirely overlap with each other, illustrating the agreement between Eq. 10 and the approximation of Eq. 11.

Interestingly, the information capacity per nucleotide under the homopolymer constraint does not depend on the

length of the DNA oligos, only on only the maximal length of homopolymers (**Supplementary Figure** 3).

1.2.2 The GC content constraint:

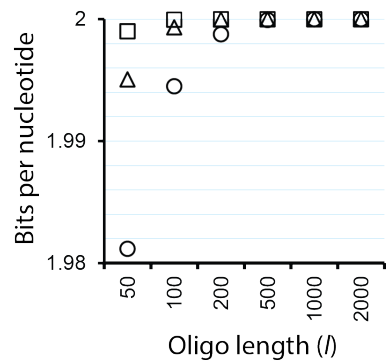
Let p_{gc} be the probability that a sequence of l nucleotides is within $0.5 \pm c_{gc}$. Without any other constraint, the $p_{gc}(x)$ is:

$$p_{gc} = 2\Phi(2\sqrt{l}c_{gc}) - 1 \tag{12}$$

where $\Phi_{(}\cdot)$ is the cumulative function of a standard normal distribution. The number of bits per nucleotide that can be transmitted under this constraint is:

$$\begin{aligned} \frac{\log_2 |A_X^{\mathtt{gc}}|}{l} &= \frac{\log_2[4^l p_{gc}]}{l} \\ &= 2 + \frac{\log_2[2\Phi(2\sqrt{l}c_{gc}) - 1]}{l} \end{aligned} \tag{13}$$

where $|A_X^{\mathtt{gc}}|$ is the size of the A_X set under the GC content constraint.



Supplementary Figure 4: Bits per nucleotide as a function of l and c_{gc}. Circles, triangles, and squares denote $c_{gc} = (0.05, 0.1, 0.15)$, respectively.

For any reasonable allowable range of GC content such as $c_{gc} \geq 0.05$ and minimal oligo length of $l \geq 50$, the GC constraint plays a negligible role in reducing the information content of each nucleotide. For example, with $c_{gc} = 0.05$ and $l = 100$bp, the information content of each nucleotide is 1.992bits, only 0.4% from the theoretical maximum. **Supplementary Figure** 4 presents the channel capacity as a function of the oligo length under various levels of GC content constraints.

1.2.3 Putting the biochemical constraints together:

The homopolymer constraint and the GC content constraint define the output of the constrained channel and b, the coding potential per nucleotide.

$$\begin{aligned} b &= \frac{\log_2 |A_X|}{l} \\ &= \frac{\log_2 |A_X^{\mathtt{h}} \cap A_X^{\mathtt{gc}}|}{l} \\ &\approx 2 - \frac{3\log_2 e}{4^{m+1}} - \frac{\log_2[2\Phi(2\sqrt{l}c_{gc}) - 1]}{l} \end{aligned} \tag{14}$$

To estimate b, we selected a conservative yet practical set of constraints with $c_{gc} = 0.05$ and $m = 3$, and an oligo length of $l = 150$nt. We selected $m = 3$ following the work of [10, 5, 2] that studied the rates of homopolymer errors in synthesis, PCR amplification, and sequencing, respectively. For the GC content, we decided to use a conservative value of $c_{gc} = 0.05$ since this constraint hardly changes the results. $l = 150$ was set to match our experiment and is close to the maximal oligo lengths of major manufacturers before adding two 20nt annealing sites for PCR primers. In this setting, the coding potential is:

$$b \underset{\substack{m=3 \\ c_{gc}=0.05 \\ l=150}}{=} 2 - 0.017 - 0.003 = 1.98\text{bits/nt} \tag{15}$$

1.2.4 Index length:

Each oligo should be indexed since they are not received in any particular order. This requirement means that some of the nucleotides in the oligo cannot be used for encoding DNA information. Let l‘ denote the remaining nucleotides available for encoding data. Then,

$$\begin{aligned} l\text{‘} &= l - \lceil \log_{2^b} K \rceil \\ &= l - \left\lceil \frac{\log_2 K}{b} \right\rceil \end{aligned} \tag{16}$$

The information capacity of each nucleotide after adding the index is reduced to:

$$bl\text{‘}/l \approx b - \frac{\log_2 K}{l} \tag{17}$$

When the oligo length is fixed, it is easy to see that the information capacity goes to zero when the input file becomes larger since indexing would eventually occupy all real estate available on the oligo. However, this issue can be solved by first splitting the file into large blocks of information and encoding each block using a distinct oligo pool that is physically isolated from the other pools, similar to physical hard-drives in an external storage device. For example, we can envision an extreme architecture of a DNA storage architecture that consists of an array of 16Tbytes per oligo pool (comparable to the largest single hard-drive available today). With 150nt oligos, indexing would take only 20nt. This translates to a 13% reduction of the effective oligo length, meaning an information capacity of 1.77bits/nt. In a more practical architecture of an array of oligo pools of 1Mbyte to 1Gbyte of data, the indexing cost goes down to 8 to 13 nucleotides, respectively. This would translate to approximately 7% reduction in the coding capacity to $C_{nt} = 1.84$bits/nt assuming no dropouts.

1.2.5 Dropouts:

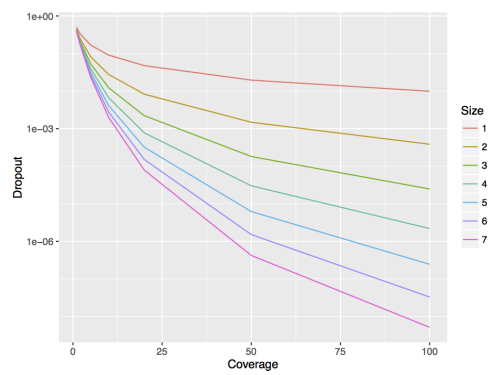
Finally, we have to consider the probability of oligo dropouts on the channel capacity. Previous studies have found that sequencing coverage follows a negative binomial distribution (NB) with average of μ and a size parameter r. The average coverage μ is largely determined by the capacity of the sequencer, whereas r corresponds to the library preparation technology. When r goes to infinity, then the distribution behaves as Poisson. In practice, however, r is usually between 2 and 7. For example, Sampson et al. [16] found that $r = 2$ in an exome sequencing dataset and $r = 4$ for whole genome sequencing using Illumina. In our experiments, we observed $r = 6.4$ for the master pool that used a relatively small number of PCR rounds and $r = 1.3$ for the deep copy that underwent 90 PCR cycles. In the main text, we report the overdispresion, which is defined as $1/r$, because it it more intuitive to understand that larger values are more overdispresed.

To find the expected rate of dropouts, we need to evaluate the probability of getting zero sequencing coverage. Let $P(x; \mu, r)$ be the probability mass function of a negative binomial distribution. In general:

$$P(x; \mu, r) = \binom{x + r - 1}{x} (1 - p)^r p^x \tag{18}$$

where $p = r/(r + \mu)$. Then, the rate of dropouts is

$$\delta_v = P(x = 0; \mu, r) = \left(\frac{\mu}{\mu + r}\right)^r \tag{19}$$



Supplementary Figure 5: The expected dropout rate (δ_v) as a function of the average sequencing coverage (μ) and the size parameter r.

We found an excellent agreement between the model and the empirical number of dropouts. For example, in the downsampling experiments, we reduced the average coverage per oligo to $\mu = 5.86$ (the size parameter is invariant to downsampling and stayed at $r = 6.4$). Eq. 19 predicted a dropout rate of 1.5%, whereas the observed rate of missing oligos was $\delta_v = 1.3\%$. **Supplementary Figure** 5 shows the expected dropout rates for a range of average sequencing coverage and as a function of the size parameter for the relevant range of oligo experiments.

These results show that for a reasonable sequencing coverage of $\mu = 10$ and a relatively mediocre size parameter of $r = 2$, the expected dropout rate is 2.7%. When the size parameter is better with $r = 6$, the dropout rate is approximately 0.25%. We posit that in most storage architectures, the dropout rates should be around $\delta = 0.5\%$. When the size parameter is excellent ($r = 7$), this dropout rate can be achieved with a low coverage of $\mu = 7$. When the size parameter is of low quality ($r = 1.5$), one can achieve this rate with sequencing coverage $\mu = 50$, which is not unreasonable even for large oligo pools. Blawat et al. [17] recently reported a dropout rate of 0.6% in an experiment with one million oligos, very close to the rate in our analysis.

According to Eq. 3, the reduction in capacity per nucleotide is proportional to the dropout rate. In the previous section, we found that the capacity is $C_{nt} = 1.84$bits/nt without any dropout. With a dropout rate of 0.5%, the capacity is $C_{nt} = 1.83$bits/nucleotide.

2 Calculating the net density of DNA storage schemes

1. **Church et al.** [10] used a binary scheme with a coding potential of $b = 1$bit/nt. The scheme has no error correcting code or fold redundancy. The index is of length 19bits. They encoded $658,750$bytes using $54,898$ oligos of length 115nt. The net density is $658,750 \times 8/(54,898 \times 115) = 0.83$bits per nucleotide.

2. **Goldman et al.** [9] use a ternary scheme with $b = 1.25$bit/nt. The scheme has one parity nucleotide for error detection and two nucleotides to detect reverse complement coding and a 4 fold redundancy. The index is of length 14trits, 2trits for file locations and 12 trits for an address, which is equivalent to a 21bit index. They encoded $757,051$bytes using $153,335$ oligos of length 117nt. The net density is $757051 \times 8/(153,335 \times 117) = 0.34$bits per nucleotide.

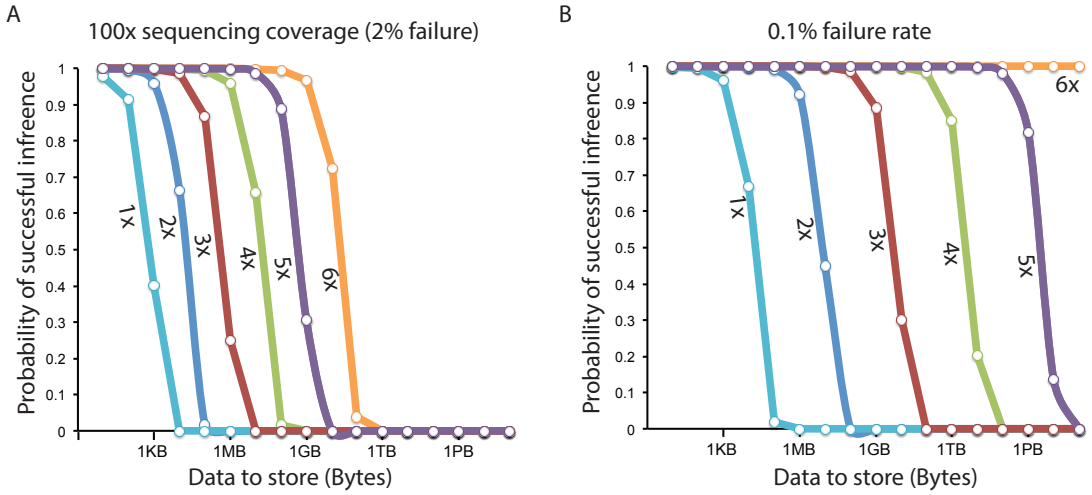
3. **Grass et al.** [18] use a GF(47) scheme that maps every two bytes to nine nucleotides with $b = 1.78$bit/nt. The scheme uses a 3byte index and a two dimensional error correcting code that mapped 30 blocks of data of lengths 594bytes to an array of 713 blocks of size 39bytes including a 3byte index. The manuscript does not describe any correction of dropouts. They encoded $83,000$bytes using $4,991$ oligos of length 120nt. The net density is $83000 \times 8/(4,991 * 120) = 1.14$bits per nucleotide.

4. **Bornholt et al.** [8] used the same scheme as Goldman et al. but with 1.5 fold redundancy against dropouts. The length of the index is not mentioned in the manuscript. They encoded $45,652$bytes and synthesized $151,000$ oligos of length 120, but their experiments included testing the original Goldman et al. scheme for their data. We therefore estimated the net density by reducing the redundancy of Goldman et al. from $4\times$ to $1.5\times$. With this estimate, the net density is $0.34 * 4/1.5 = 0.88$bits per nucleotide.

5. **Blawat et al.** [17] use a scheme that maps each input byte to 5 nucleotides and achieves $b = 1.6$bits/nt. The index is of length 39bits and the scheme uses a two dimensional nested error correcting code. First, the index is mapped to a 63bit vector. Next, the data and the 63bit index are organized into a two dimensional array. The scheme employs a Reed Solomon code that adds 33bits of redundancy to each block of 223bits in one dimension and 16bits of cyclic redundancy check in the other dimension. They encoded 22MByte of data using $1,000,000$ oligos of length 190nt and showed that their scheme can decode dropouts. The net density is $22 \times 10^6 \times 8/(10^6 \times 190) = 0.92$bit per nucleotide.

6. **This work** uses a scheme that screens potential oligos to realize the maximal coding capacity with $b = 1.98$bit/nt. The seed has 4bytes that can encode files of up to 500MByte (see section 3.5). We also add 2byte of Reed-Solomon error correcting code that protects both the seed and the data payload. The level of redundancy against dropouts is determined by the user and we decided on a level of $1.07\times$ because of the proposed cost from the oligo manufacturer. We encoded $2,116,608$bytes of information using $72,000$ oligos of length 152nt. The net density is $2116608 \times 8/(72000 \times 152) = 1.55$bits per nucleotide.

We are fully aware that some of the differences in the net densities can be attributed to different oligo lengths or indexes. We decided to present the reported net density instead of standardizing the schemes to a specific oligo/index length for several reasons: first, certain schemes, such as Blawa et al. [17], employ error correcting codes that are designated for specific input and output lengths. It is not easy to translate the scheme to a different oligo length without completely changing their error correcting strategy. Second, our main focus is to compare schemes that were tested in practice. Imputing the net density for a different architecture without empirical tests can be misleading. For example, Goldman et al. [9] reported a successful filtering of sequencing errors even with low sequencing coverage using a single parity nucleotide when the length of oligos was 117nt. It is not clear whether this error detection scheme can work with 150nt oligos that are more error prone. Third, we found that the standardization does not significantly affect the overall picture. For example, after standardizing the Church et al. scheme to have an index length of 28bits and oligo length of 152nt (similar to our method), the net density goes from 0.83bit/nt to 0.81bit/nt. Similarly, Goldman et al. goes from 0.34bit/nt to 0.28bit/nt after standardization.



Supplementary Figure 6: The probability of perfect retrieval (no gaps) of information as a function of input size and the number of times each input bit is repeated on distinct oligos (colored lines) (A) Dropout rate of 2%. This rate was observed in Goldman et al. [9] after downsampling their sequencing data to reflect a coverage of 100× (B) Dropout rate of 0.1%, six times smaller than the rate observed by Blawat et al. [17]. With this low drop out rate, 1Gbyte storage has 10% chance of data corruption even when each bit is stored on 3 distinct oligos.

3 The DNA Fountain coding strategy

Our encoding algorithm works in three computational steps: (a) preprocessing, (b) Luby Transform, and (c) screening. Its overall aim is to convert input files into a collection of valid DNA oligos that pass the biochemical constraints and be sent to synthesis.

3.1 Encoding overview

1. In the preprocessing step, we start by packaging the files of interest into a single tape-archive (tar) file, which is than compressed using a standard lossless algorithm (e.g. gzip). Besides the obvious advantage of reducing the size of the tar file, compression increases the entropy of each bit of the input file and reduces local correlations, which is important for the screening step. Then, the algorithm logically partitions the compressed file into non-overlapping segments of length L bits, which is a user defined parameter. We used $L = 256$bits (32 bytes) for our experiments, since this number is compatible with standard computing environments and generates oligos of lengths that are within the limit of the manufacturer.

2. The Luby Transform step works as follows:

   (a) We initialize a pseudorandom number generator (PRNG) with a seed, which is selected according to a mathematical rule as explained in Section 3.2.

   (b) The algorithm decides on d, the number of segments to package in the droplet. For this, the algorithm uses the PRNG to draw a random number from a special distribution function, called robust soliton probability distribution. Briefly, the robust soliton distribution function is bi-modal and ensures that most of the droplets are created with either a small number of input segments or a fixed intermediary number of segments. This mathematical property is critical for the decoding process. Section 3.3 presents this distribution in details.

   (c) The algorithm again uses the PRNG to draw d segments without replacement from the collection of segments using a uniform distribution.

   (d) The algorithm performs a bitwise-XOR operation (biwise addition modulo 2) on the segments. For example, consider that the algorithm randomly selected three input fragments: 0100, 1100, 1001; In this case, the droplet is: $0100 \oplus 1100 \oplus 1001 = 0001$.

   (e) The algorithm attaches a fixed-length index that specifies the binary representation of the seed. For example, if the seed is 3 and the fixed index length is 2bits, the output droplet will be 110001. In practice, we used a 32bit (4byte) index for compatibility with standard computing environments.

   (f) the user has the option to use a regular error correcting code computed on the entire droplet. In our experiments, we added two bytes of Reed-Solomon over $GF(256)$ to increase the robustness of our design.

The Luby Transform confers robustness against dropouts. Theoretically, the transform additions can be thought of as representing the input segments as a binary system of linear equations. Each droplet is one equation, where the seed region has one to one correspondence to the 0-1 coefficients of the equation, the payload region is the observation, and the data in the input segments are the unknown variables of the system. To successfully restore the file, the decoder basically needs to solve the linear system. This task can theoretically be done (with a very high probability) by observing any subset of a little more than K droplets. Our encoder exploits this property to create robustness against dropouts. It produces many more oligos than the dropout rates (eg. %5 more oligos) to create an over-determined system. Then, no matter which oligos are dropped, we can solve the system and recover the file as long as we can collect a little more than K. We postpone the formal definition of "a little more" to Section 3.3 that presents the robust solition distribution.

3. In the screening step, the algorithm excludes droplets that violate the required biochemical constraints from the DNA sequence. First, it converts the droplet into a DNA sequence by translating {`00,01,10,11`} to {`A,C,G,T`}, respectively. For example, the droplet "110001" corresponds to "`TAC`". Next, the algorithm screens the sequence for desired properties such as GC content and homopolymer runs. If the sequence passes the screen, it is considered valid and added to the oligo design file; otherwise, the algorithm simply trashes the sequence. Since the compressed input file essentially behaves as a random sequence of bits, screening has no effect on the distribution of the number of source fragments in the valid droplets. For example, a droplet that was created by XOR-ing 10 source fragments has the same chance of passing the screen as a droplet with only 1 source fragment. This asserts that the number of droplets in a valid oligo follows the robust soliton distribution regardless of the screening, which is crucial for the decoder.

We continue the oligo creation and screening until a desired number of oligos are produced. The decoder outputs a FASTA file that can be sent to the oligo manufacturer.

3.2 Seed schedule

The process of droplet creation starts with a seed to initialize the pseudorandom number generator. Typical fountain code implementation starts with a specific seed that is incremented with every round. The problem in our case is that an incremental seed schedule creates bursts of invalid sequences. For example, any seed in interval $[0, 1, ..., 16777215]$ would be mapped to a sequence that starts with an `AAAA` homopolymer when representing the seed as a 32bit number.

We sought a strategy that would go over each number in the interval $[1, \ldots, 2^{32}]$ in a random order to avoid bursts of invalid seeds. For this we used a Galois linear-feedback shift register (LFSR). In this procedure, a hard-coded seed (e.g. 42) is represented as a binary number in the LFSR. Then, we deploy one round of the LFSR, which shifts and performs a XOR operation on specific input bits within the register. This operation corresponds to a polynomial multiplication over a finite field. The new number in the register is used as the seed for the next droplet. By repeating this procedure, we can generate a sequence of seeds in a pseudo-random order. To scan all numbers in the interval, we instructed the LFSR to use the primitive polynomial $x^{32} + x^{30} + x^{26} + x^{25} + 1$ for the multiplication of the number in the register. With this polynomial, the LFSR is guaranteed to examine each seed in the interval $[1, \ldots, 2^{32}]$ without repetition. Other implementations of DNA Fountain might require a different seed space. This can easily be done by switching the polynomial in the LFSR. Public tables such as [19] list LFSR polynomials with a cycle of $2^n - 1$ for a wide range of n.

3.3 Tuning the Soliton distribution parameters

The robust soliton distribution function, $\mu_{K,c,\delta}(d)$ is a key component of the Luby Transform. We will start by describing two probability distribution functions, $\rho(d)$ and $\tau(d)$, that are the building blocks for the robust soliton distribution function.

Let $\rho(d)$ be a probability distribution function with:

$$\rho(d) \equiv \begin{cases} 1/K & \text{if } d = 1, \\ \frac{1}{d(d-1)} & \text{for } d = 2, \ldots, K. \end{cases} \tag{20}$$

Let τ be a probability distribution function with:

$$\tau(d) \equiv \begin{cases} \frac{s}{Kd} & \text{for } d = 1, \ldots, (K/s) - 1 \\ \frac{s \ln(s/\delta)}{K} & \text{for } K/s \\ 0 & \text{for } d > K/s \end{cases} \tag{21}$$

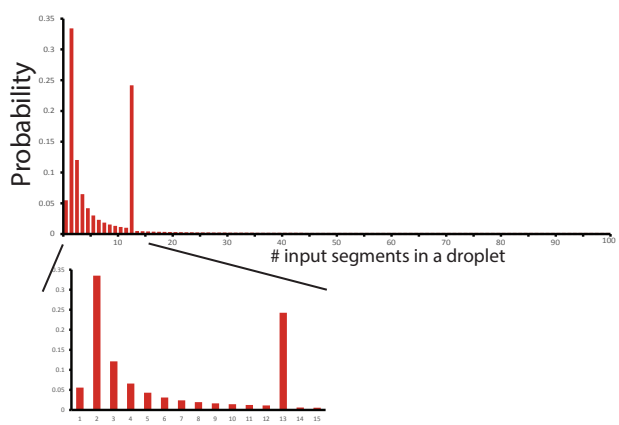
where $s \equiv c\sqrt{K} \ln^2(K/\delta)$.

The robust soliton distribution is defined as:

$$\mu_{K,c,\delta}(d) \equiv \frac{\rho(d) + \tau(d)}{Z} \tag{22}$$

Where Z is a normalization parameter, $Z = \sum_d \rho(d) + \tau(d) = 1$.

Supplementary Figure 7 presents an example of the robust soliton distribution.



Supplementary Figure 7: An example of the Soliton distribution with ($K = 100, c = 0.1, \delta = 0.05$). Most droplets will include 1-3 input segments and a large fraction of the remainder will include 13 segments. The c and δ parameters are controlled by the user.

The two free parameters, c and δ, are specified by the user during the oligo synthesis stage. In general, δ is the upper bound probability of failing to recover the original file after receiving $K \cdot Z$ droplets [20, 21]. Previous mathematical analyses have shown that this upper bound is highly pessimistic and that the probability of a failure is much smaller in practice [21]. The c parameter is a positive number that affects the decoding/encoding performance. On one hand, low values of c reduce the *average* number of oligos that are required for successful decoding. On the other hand, low values of c increase the *variance* around the average [21]. In addition, low values of c increase the average number of segments in each droplet. This translates to deeper recursions of the decoding algorithm and increases the runtime [22].

In our experiments, we selected $\delta = 0.001$ and $c = 0.025$. With these values, $Z = 1.033$, meaning that we needed to generate at least 3% more oligos than the number of segments.

3.4 Decoding

We employed the following steps to decode the data:

1. Preprocessing: first, we stitched the paired-end reads using PEAR [23] and retained only sequences whose length was 152nt. Next, we collapsed identical sequences and stored the collapsed sequence and its number of occurrences in the data. Finally, we sorted the sequences based on their abundance so that more prevalent sequences appear first. This way the decoder observes the highest quality data first and gradually gets sequences with reduced quality. Due to the redundancy of our approach, the decoder usually does not need all oligos to construct the original file and will usually stop before attempting to decode sequences that were observed a small number of times (e.g. singletons), that are more likely to be erroneous.

   From this point the decoder process works sequentially and executes the next two steps on each collapsed sequence until the file is fully resolved:

2. Droplet recovery: the decoder maps the DNA sequence into a binary format by translating $\{\texttt{A,C,G,T}\}$, to $\{0, 1, 2, 3\}$. Next, it parses the sequence read to extract the seed, data payload, and the error correcting code (if it exists). If there is an error correcting code, the decoder checks whether there are any errors. In our experiments, we excluded sequences with an indication of one or more errors and did not attempt to correct them. We found that most errors were due to short insertions and deletions, potentially from the oligo synthesis. Reed-Solomon error correcting code can only handle substitutions and attempting to correct the sequence is more likely to result in erroneous recovery.

3. Segment inference: after validating the integrity of the binary message, the decoder initializes a pseudorandom number generator with the observed seed. This generates a list of input

segment identifiers. Next, it employs one round of a message passing algorithm, which works as follows: first, if the droplet contains segments that were already inferred, the algorithm will XOR these segments from the droplet and remove them from the identity list of the droplet. Second, if the droplet has only one segment left in the list, the algorithm will set the segment to the droplet's data payload. Next, the algorithm will propagate the information about the new inferred segment to all previous droplets and will repeat the same procedure recursively, until no more updates can be made. If the file is not recovered, the decoder will move to the next sequence in the file and execute the droplet recovery and segment inference steps. This process of propagation of information eventually escalates that solves the entire file (**Supplementary Figure** 10).

3.5 DNA Fountain overhead

3.5.1 Code rate

DNA Fountain approaches the channel capacity but still entails a small overhead for a wide range of realistic applications. First, even with zero probability of dropout, we need to synthesize more oligos than the number of input segments. As discussed in section 3.3, the robust soliton distribution asserts convergence of the decoder (with a probability of δ) when $K \cdot Z$ droplets are seen. In general, when K is extremely large, $Z \to 1$, meaning that the code is rateless and entails no overhead. However, for experiments with tens of thousands of oligos, Z is on the order of 3% and empirical results with 10000 segments have observed on average a 5% overhead for successful decoding [24]. Second, our strategy entails a small inflation in the size of the index. Eq. 16 shows that the minimal index length is $\log_{2^b} K = \log_{3.95} K$. In our case, the index space must be larger to accommodate both successful and failed attempts to construct a droplet. This inflation is quite modest and only scales logarithmically with the size of the search space. For example, when screening 150nt oligos for a GC content between 45% and 55% and homopolymer runs of up to 3nt, only 12.5% of the oligos pass these conditions. This requires the index to have an additional $\lceil \log_{3.95}(1/0.125) \rceil = 2$ nucleotides, which reduces the information content by 2/150=1.3% from the theoretical maximum. Thus, we posit that our approach could theoretically reach $100\% - (3\% + 1.3\%) \approx 96\%$ of the channel capacity for experiments with tens of thousands of oligos.

In practice, we realized $1.55/1.83 = 85\%$ of the channel capacity. The difference between the practical code rate and the theoretical rate is explained by three factors. First, our coding strategy included a redundancy level of 7%, about 3% more than the required overhead for successful decoding with 0.5% dropout rate. As explained in the main text, we selected this level mainly because of the price structure of the oligo manufacturer that offered the same price for any number of oligos between 66,000 to 72,000. We consider the flexibility of DNA Fountain to maximize the number of informative oligos within a specific price range as a strong advantage of our technique.

For the seed space, we used 32bits. This seed is about 9bits over what is needed for encoding all successful and failed attempts to create valid oligos for a file of size 2.1Mbyte. We decided to use this length in order to be able to scale the same architecture to store files of 500Mbyte. Finally, we also had an error correcting code of two bytes in each oligo, which reduced the rate by another 5%.

3.5.2 The time complexity and economy of encoding and decoding

Our experiments show that DNA Fountain is feasible for a range of file sizes and oligo lengths. From a computation complexity perspective, our encoding strategy scales log-linearly with the input file size and empirical tests showed that a 50Mbyte file takes about 1 hour on a single CPU of a standard laptop. On the other hand, the complexity scales exponentially with the oligo length because it becomes harder to find sequences without homopolymers and increasing numbers of droplets are created only to be destroyed in the screening process. However, the base of the exponent is very close to 1. For example, with $m = 3$, the complexity scales with $\mathcal{O}(1.01^l)$ and with $m = 4$ the complexity scales with $\mathcal{O}(1.003^l)$. With this low exponent we found that our strategy is still practical for a range of oligo lengths even above the limit of major oligo pool manufacturers of approximately 200-250nt [25]. For example, encoding data on 300nt oligos takes only 3min for 1Mbyte per CPU and encoding 400nt oligos takes 6min for the same conditions (**Supplementary Table** 3).

Our ability to include more information in each nucleotide compensates for the computation price even for long oligos. For example, encoding 1Gbyte of information on 400nt oligos would take 4 CPU days. We can parallelize the encoder on multiple CPUs by letting each encoder thread scan a different region in the seed space. With the current price of \$0.5 for one hour of a 40 CPU server on Amazon Web Services, we can encode the entire input data for \$1.25 in a few hours. The price of the computing time is substantially smaller than the cost reduction in the oligo synthesis costs. Assuming that oligo synthesis is \$640 per 1Mbase (current price for our experiment was \$7000 and synthesized 152×72000 nucleotides without Illumina adapters), synthesizing 1Gbyte with our scheme would cost approximately \$3.27 million. However, when using other methods with lower information content of 0.9bit/nt, the synthesis price is nearly \$5.63 million, rendering the computational costs marginals compared to price. Even if the synthesis price falls by two orders of magnitude, amount saved is still substantial and on the order of tens of thousands of dollars.

	Condition	Run time[sec]	Computing costs [cents]	Oligo synthesis money saved [$]
Input file	1Mbyte	50	0.02	5,000
	2Mbyte	112	0.04	11,000
	4Mbyte	200	0.07	23,000
	8Mbyte	436	0.15	45,000
	16Mbyte	897	0.31	96,000
	32Mbyte	1817	0.63	192,000
	64Mbyte	3699	1.28	377,000
Oligo length	100nt	53	0.02	
	152nt	40	0.01	
	200nt	54	0.02	
	252nt	84	0.03	5,000
	300nt	137	0.05	
	352nt	222	0.07	
	400nt	374	0.13	

Supplementary Table 3: Encoding time and cost saving using DNA Fountain. Run time indicates the empirical time it takes the encoder to convert an input file to oligos for one CPU. Computing costs show an estimate of the price to encode the file using Amazon Cloud with a price of $0.5 per hour for a 40 CPU server. Oligo synthesis money saved reports the estimated difference in budget between our technique and previous techniques with an information capacity of approximately 0.9bit/nt. We assume a constant price for different oligo lengths and sizes of $640 per 1Mbase. Saving was rounded to the closet thousand. The upper part of the table reports the results for different input file sizes and an oligo length of 152nt. The bottom part reports the results for different oligo lengths and a 1Mbyte input file.

With respect to the decoder, the time complexity is $\mathcal{O}(K \log K)$ as in Luby Transform. In principle, we could potentially reduce the decoding complexity to $\mathcal{O}(K)$ by using Raptor codes [26] on top of the Luby Transform. We decided not to pursue this option since Raptor codes are patented and we were concerned that the legal complexities of using this patent could hamper adaptation of our method.

4 DNA Fountain experiments

4.1 Computing

All encoding and decoding experiments were done using a MacBook Air with a 2.2 GHz Intel Core i7 and 8Gbyte of memory. The code was tested with Python 2.7.10.

4.2 Software, data, and input files

Code, data, and input files are available on http://dnafountain.teamerlich.org/.

Supplementary Figure 8 presents snapshots of some of the input files in our library.

Supplementary Figure 8: Input files encoded on DNA (A) The Kolibri operating system (B) The Arrival of a Train (C) The Pioneer plaque (D) Shannon's manuscript on information theory.

4.3 Command line steps to encode the data

For reproducibility, we provide the step-by-step commands:

```
# Packing input files to a tar and compressing:
```

```
tar -b1 -czvvf info_to_code.tar.gz ./info_to_code/

#Zero-padding to make the input be a multiple of 512bytes
truncate -s2116608 ./info_to_code.tar.gz

# Actual encoding of data as DNA:
python encode.py \
--file_in info_to_code.tar.gz \
--size 32 \
-m 3 \
--gc 0.05 \
--rs 2 \
--delta 0.001 \
--c_dist 0.025 \
--out info_to_code.tar.gz.dna \
--stop 72000
# output is a FASTA file named info_to_code.tar.gz.dna

# Adding annealing sites:
cat info_to_code.tar.gz.dna | \
grep -v '>' |
awk '{print "GTTCAGAGTTCTACAGTCCGACGATC"$0"TGGAATTCTCGGGTGCCAAGG"}' \
> info_to_code.tar.gz.dna_order

# Sent info_to_code.tar.gz.dna_order to synthesis company
```

4.4 Molecular procedures

The oligo pool was synthesized by Twist Bioscience. The lyophilized pool consisted of 72,000 oligos of 200nt, which included the 152nt payload flanked by landing sites for sequencing primers:

`GTTCAGAGTTCTACAGTCCGACGATC[N152]TGGAATTCTCGGGTGCCAAGG`

The pool was resuspended in 20uL TE for a final concentration of 150 ng/ul. PCR was performed using Q5 Hot Start High-Fidelity 2X Master Mix (NEB # M0494) and Illumina small RNA primers RP1 and RPl1 (100ng oligos, 2.5ul of each primer (10μM), 25ul Q5 Master Mix in a 50ul reaction).

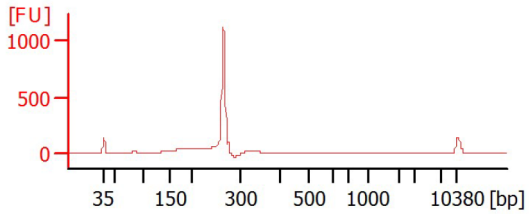
PCR Primer (RP1):

`5' AATGATACGGCGACCACCGAGATCTACACGTTCAGAGTTCTACAGTCCGA`

PCR Primer, Index 1 (RPI1):

```
5' CAAGCAGAAGACGGCATACGAGATCGTGATGTGACTGGAGTTCCTTGGCACCCGAGAATTCCA
```

Thermocycling conditions were as follows: 30s at 98C; 10 cycles of: 10s at 98C, 30s at 60C, 30s at 72C, followed by a 5 min. extension at 72C. The library was then purified in a 1:1 Agencourt AMPure XP (Beckman Coulter # A63880) bead cleanup and eluted in 20ul water. This library was considered the master pool. Running the library on a BioAnalyzer showed an average size of 263nt, close to the expected 265nt expected by the oligo length (152nt before annealing sites) and the two PCR adapters of length 50nt and 63nt.



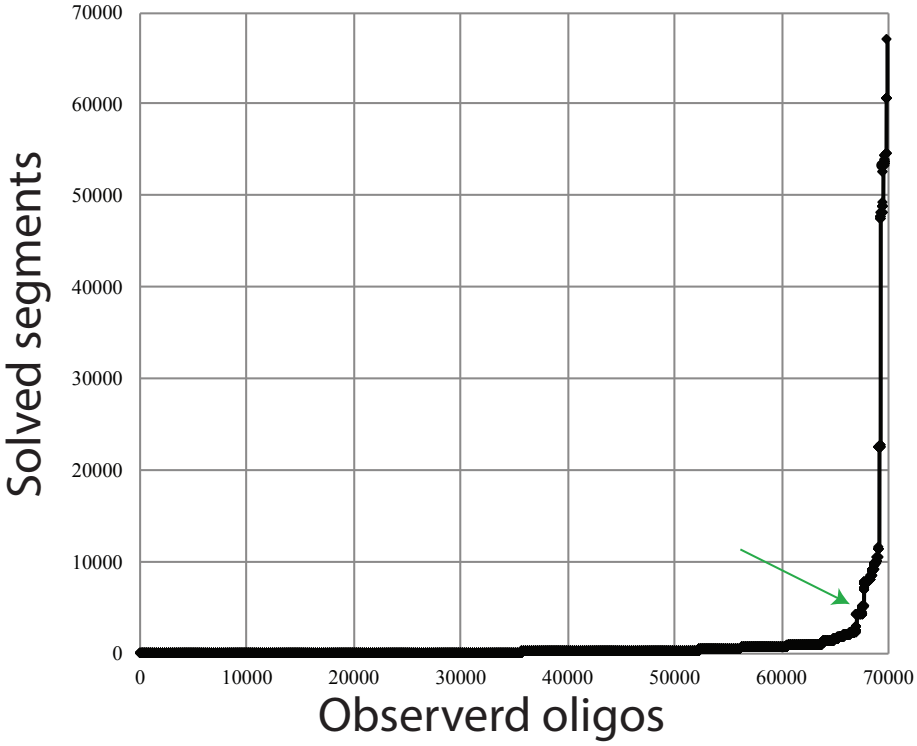
Supplementary Figure 9: BioAnalyzer results for the oligo pool. The average fragment size was 263nt, concordant with an oligo length of 200nt and the two PCR adapters.

We sequenced the oligo pool using on a single flow cell of the Miseq v3 kit (Illumina # MS-102-3001) at the New York Genome Center using a a 150 pair-end read protocol. Raw .bcl files were downloaded from BaseSpace and analyzed using the commands below.

4.5 Decoding the data

```
#Converting bcl to fastq using picard (https://github.com/broadinstitute/picard):
for i in {1101..1119} {2101..2119}; do
   mkdir ~/Downloads/fountaincode/seq_data3/$i/;
done

for i in {1101..1119} {2101..2119}; do
   java -jar ~/Downloads/picard-tools-2.5.0/picard.jar \
   IlluminaBasecallsToFastq \
   BASECALLS_DIR=./raw/19854859/Data/Intensities/BaseCalls/ \
   LANE=1 \
   OUTPUT_PREFIX=./seq_data3/$i/ \
   RUN_BARCODE=19854859 \
   MACHINE_NAME=M00911 \
   READ_STRUCTURE=151T6M151T \
   FIRST_TILE=$i \
   TILE_LIMIT=1 \
   FLOWCELL_BARCODE=AR4JF;
done
```

```
#Read stitching using PEAR (Zhang J et al., Bioinformatics, 2014).
# This step takes the 150nt reads and places them together to get back the full oligo.
for i in {1101..1119} {2101..2119}; do
   pear -f ./$i.1.fastq -r ./$i.2.fastq -o $i.all.fastq;
done


# Retain only fragments with 152nt as the original oligo size:
awk '(NR%4==2 && length($0)==152){print $0}' \
*.all.fastq.assembled.fastq > all.fastq.good

# Sort to prioritize highly abundant reads:
sort -S4G all.fastq.good | uniq -c > all.fastq.good.sorted

gsed -r 's/^\s+//' all.fastq.good.sorted |\
sort -r -n -k1 -S4G > all.fastq.good.sorted.quantity

# Exclude column 1 that specifies the number of times a read was seen and exclude reads
    with N:
cut -f2 -d' ' all.fastq.good.sorted.quantity |\
grep -v 'N' > all.fastq.good.sorted.seq


# Decoding:
python ~/Downloads/fountaincode/receiver.py \
-f ./seq_data3/all.fastq.good.sorted.seq \
--header_size 4 \
--rs 2 \
--delta 0.001 \
--c_dist 0.025 \
-n 67088 \
-m 3 \
--gc 0.05 \
--max_hamming 0 \
--out decoder.out.bin

# Verification:
md5 decoder.out.bin
#output is 8651e90d3a013178b816b63fdbb94b9b
md5 info_to_code.tar.gz
#output is 8651e90d3a013178b816b63fdbb94b9b

#YES!
```

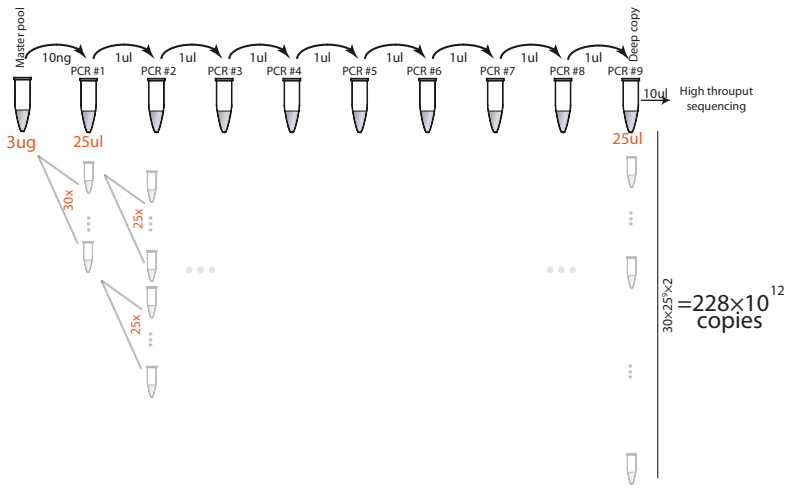


Supplementary Figure 10: The number of solved segments as a function of observed oligos. The figure demonstrates the avalanche process. At first, the observed oligos cannot determine the data in the input segments. When the number of observed oligos is about the same as the number of input segments (green arrow), sufficient information has accumulated to infer the data in the input segments and an avalanche of inference starts with each new oligo. Notice that the decoder needed only to observe 69800 oligos out of the 72000 synthesized oligos to decode the entire file, illustrating the robustness against dropouts.

4.6 Creating a deep copy

For the nine consecutive PCR reactions, we started with the master pool as described above. Then, we performed 9 subsequent rounds of PCR using the same conditions as above. The first round used 10ng of oligo input from the master pool into 25ul PCR reactions. Then, 1ul from each PCR reaction was input into each subsequent round of PCR for a total of 9 reactions without cleanup. The final round was purified in a 1:1 Agencourt AMPure XP bead cleanup and eluted in 20ul deionized water. The final library was run under the same conditions as described.

This procedure can theoretically create $30 \times 25^9 \times 2 = 228$ trillion copies of the file:



Supplementary Figure 11: Exponential amplification process. Each PCR reaction generates enough volume for multiple subsequent reactions. The first PCR reaction used only 10ng from the master pool of 3ug of DNA. This implies that 30 similar reactions can be conducted using the master pool. Assuming a conservative 1ng input, 300 reactions could be performed. Each subsequent PCR used 1ul out of 25ul generated by the previous reaction. Therefore, an exponential process could amplify the number of copies by 25 times in each reaction (gray tubes). Eventually, we consumed about half of the final reaction for sequencing and QC, meaning that the final reaction can be sequenced twice. In total, the nine step amplification process has the potential to create $30 \times 25^9 \times 2 = 228$ trillion copies. Our experiment reflects one end-to-end arm of this copy (black tubes).

The decoding procedure for this library is identical to the one presented for the master copy. Fitting the negative bionomial distribution was done in R using the fitdistr command.

References

[1] J. J. Schwartz, C. Lee, and J. Shendure. Accurate gene synthesis with tag-directed retrieval of sequence-verified DNA molecules. *Nat. Methods*, 9(9):913–915, Sep 2012.

[2] M. G. Ross, C. Russ, M. Costello, A. Hollinger, N. J. Lennon, R. Hegarty, C. Nusbaum, and D. B. Jaffe. Characterizing and measuring bias in sequence data. *Genome Biol.*, 14(5):R51, 2013.

[3] S. Kosuri, N. Eroshenko, E. M. Leproust, M. Super, J. Way, J. B. Li, and G. M. Church. Scalable gene synthesis by selective amplification of DNA pools from high-fidelity microchips. *Nat. Biotechnol.*, 28(12):1295–1299, Dec 2010.

[4] Nikolai Eroshenko, Sriram Kosuri, Adam H Marblestone, Nicholas Conway, and George M Church. Gene assembly from chip-synthesized oligonucleotides. *Current protocols in chemical biology*, pages 1–17, 2012.

[5] Guruprasad Ananda, Erin Walsh, Kimberly D Jacob, Maria Krasilnikova, Kristin A Eckert, Francesca Chiaromonte, and Kateryna D Makova. Distinct mutational behaviors differentiate short tandem repeats from microsatellites in the human genome. *Genome biology and evolution*, 5(3):606–620, 2013.

[6] gblocks® gene fragments frequently asked questions. https://www.idtdna.com/pages/products/genes/gblocks-gene-fragments/gblocks-faqs. Accessed: 2016-09-01.

[7] Brant C Faircloth and Travis C Glenn. Not all sequence tags are created equal: designing and validating sequence identification tags robust to indels. *PloS one*, 7(8):e42543, 2012.

[8] James Bornholt, Randolph Lopez, Douglas M Carmean, Luis Ceze, Georg Seelig, and Karin Strauss. A dna-based archival storage system. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 637–649. ACM, 2016.

[9] N. Goldman, P. Bertone, S. Chen, C. Dessimoz, E. M. LeProust, B. Sipos, and E. Birney. Towards practical, high-capacity, low-maintenance information storage in synthesized DNA. *Nature*, 494(7435):77–80, Feb 2013.

[10] G. M. Church, Y. Gao, and S. Kosuri. Next-generation digital information storage in DNA. *Science*, 337(6102):1628, Sep 2012.

[11] David J. C. MacKay. *Information Theory, Inference & Learning Algorithms*. Cambridge University Press, New York, NY, USA, 2002.

[12] William Feller. An introduction to probability theory and its applications. vol. i. 1950.

[13] Andreas N Philippou, Costas Georghiou, and George N Philippou. A generalized geometric distribution and some of its properties. *Statistics & Probability Letters*, 1(4):171–175, 1983.

[14] K.D. Ling. On geometric distributions of order (k1,..,km). *Statistics & Probability Letters*, 9(2):163 – 171, 1990.

[15] James C. Fu and W. Y. Wendy Lou. Waiting time distributions of simple and compound patterns in a sequence of r-th order markov dependent multi-state trials. *Annals of the Institute of Statistical Mathematics*, 58(2):291–310, 2006.

[16] Joshua Sampson, Kevin Jacobs, Meredith Yeager, Stephen Chanock, and Nilanjan Chatterjee. Efficient study design for next generation sequencing. *Genetic epidemiology*, 35(4):269–277, 2011.

[17] Meinolf Blawat, Klaus Gaedke, Ingo Huetter, Xiao-Ming Chen, Brian Turczyk, Samuel Inverso, Benjamin Pruitt, and George Church. Forward error correction for dna data storage. *Procedia Computer Science*, 80:1011–1022, 2016.

[18] Robert N Grass, Reinhard Heckel, Michela Puddu, Daniela Paunescu, and Wendelin J Stark. Robust chemical preservation of digital information on dna in silica with error-correcting codes. *Angewandte Chemie International Edition*, 54(8):2552–2555, 2015.

[19] Roy William Ward and Timothy Christopher Anthony Molteno. *Table of linear feedback shift registers*. Electronics Group, University of Otago, 2012.

[20] M. Luby. Lt codes. pages 271–280, 2002.

[21] D. J. C. MacKay. Fountain codes. *IEE Proceedings - Communications*, 152(6):1062–1068, Dec 2005.

[22] E. A. Bodine and M. K. Cheng. Characterization of luby transform codes with small message size for low-latency decoding. In *2008 IEEE International Conference on Communications*, pages 1195–1199, May 2008.

[23] Jiajie Zhang, Kassian Kobert, Tomáš Flouri, and Alexandros Stamatakis. Pear: a fast and accurate illumina paired-end read merger. *Bioinformatics*, 30(5):614–620, 2014.

[24] Oliver GH Madge and David JC MacKay. Efficient fountain codes for medium blocklengths. *IEEE TRANSACTIONS ON COMMUNICATIONS*, page 1, 2006.

[25] Sriram Kosuri and George M Church. Large-scale de novo dna synthesis: technologies and applications. *Nature methods*, 11(5):499–507, 2014.

[26] Amin Shokrollahi. Raptor codes. *IEEE transactions on information theory*, 52(6):2551–2567, 2006.