HEDGES error-correcting code for DNA storage corrects indels and allows sequence constraints

William H. Press[a,b,1], John A. Hawkins[c,d,e], Stephen K. Jones Jr[d,e], Jeffrey M. Schaub[d,e], and Ilya J. Finkelstein[d,e]

[a]Department of Computer Science, The University of Texas at Austin, Austin, TX 78712; [b]Department of Integrative Biology, The University of Texas at Austin, Austin, TX 78712; [c]Oden Institute of Computational Engineering and Sciences, The University of Texas at Austin, Austin, TX 78712; [d]Department of Molecular Biosciences, The University of Texas at Austin, Austin, TX 78712; and [e]Institute for Cellular and Molecular Biology, The University of Texas at Austin, Austin, TX 78712

Contributed by William H. Press, June 6, 2020 (sent for review March 16, 2020; reviewed by Joshua B. Plotkin and H. Vincent Poor)

Synthetic DNA is rapidly emerging as a durable, high-density information storage platform. A major challenge for DNA-based information encoding strategies is the high rate of errors that arise during DNA synthesis and sequencing. Here, we describe the HEDGES (Hash Encoded, Decoded by Greedy Exhaustive Search) error-correcting code that repairs all three basic types of DNA errors: insertions, deletions, and substitutions. HEDGES also converts unresolved or compound errors into substitutions, restoring synchronization for correction via a standard Reed–Solomon outer code that is interleaved across strands. Moreover, HEDGES can incorporate a broad class of user-defined sequence constraints, such as avoiding excess repeats, or too high or too low windowed guanine–cytosine (GC) content. We test our code both via in silico simulations and with synthesized DNA. From its measured performance, we develop a statistical model applicable to much larger datasets. Predicted performance indicates the possibility of error-free recovery of petabyte- and exabyte-scale data from DNA degraded with as much as 10% errors. As the cost of DNA synthesis and sequencing continues to drop, we anticipate that HEDGES will find applications in large-scale error-free information encoding.

DNA | information storage | error-correcting code | indel | Reed–Solomon

DNA is an ideal molecular-scale storage medium for digital information (1–7). An arbitrary digital message can be encoded as a DNA sequence and chemically synthesized as a pool of oligonucleotide strands. These strands can be stored, duplicated, or transported through space and time. DNA sequencing can then be used to recover the digital message, hopefully exactly. Advances in the cost and scale of DNA synthesis and sequencing are increasingly making DNA-based information storage economically feasible. While synthesis today costs $0.001 per nucleotide, some observers project a decrease of orders of magnitude (8). A strand of DNA containing the four natural nucleotides can encode a maximum of 2 bits per DNA character. With this maximum code rate (defined as rate $r = 1.0$), no error correction is possible, because there is no redundancy in the message. However, both DNA synthesis and sequencing introduce errors in the underlying DNA pools, requiring efficient error-correcting codes (ECCs) to extract the underlying information. An ECC reduces the code rate but is necessary to protect against errors when a message is encoded as DNA characters, and, later, when decoding DNA characters back to message bits.

An ECC must correct the three kinds of errors associated with DNA—substitutions of one base by another, as well as spurious insertions or deletions of nucleotides in the DNA strand (indels). Indels represent more than 50% of observed DNA errors (Fig. 1*A*). However, most DNA encoding schemes use ECCs that can only correct substitutions, a standard task in coding theory (9–12). The coding theory literature reports only a few ECCs that correct for deletions, and there are no well-established methods for all three of deletions, insertions, and substitutions (13, 14). Prior DNA storage implementations correct for indels by sequencing to high depth, followed by multiple alignment and consensus base calling (Fig. 1*B*) (1, 3, 6). This approach represents an inefficient "repetition" ECC. Moreover, repetition ECCs only correct errors associated with DNA sequencing. Correcting synthesis errors using this approach also requires pooling multiple synthesis reactions, which is the most costly and time-consuming step in DNA-based information storage (2). Finally, alignment and consensus decoding does not scale well beyond small proof-of-principle experiments. In sum, ECCs that require high-depth repetition in the stored DNA have very small code rates because a large number of stored nucleotides are required per recovered message bit.

Here, we describe an algorithm to achieve high code rates with a minimum requirement for redundancy in the stored DNA. We adapt the coding theory approach of constructing an "inner" code (so termed because it is closest to the physical channel, the DNA) to correct most indel and substitution errors. The inner code translates between a string of $\{A, C, G, T\}$ and an intermediate binary string of $\{0, 1\}$, with no added or dropped bits even in the presence of indels in the DNA string. An efficient "outer" code corrects residual errors with extremely high probability. Our inner code, termed HEDGES (Hash Encoded, Decoded by Greedy Exhaustive Search), is optimized for real-world DNA-based information storage: 1) It finds and corrects indels, or converts them to substitutions (which it also usually corrects). 2) It admits varying code rates, with correspondingly greater tolerance of DNA errors at lower code rates. 3) It is adaptable

Significance

This paper constructs an error-correcting code for the {*A*, *C*, *G*, *T*} alphabet of DNA. By contrast with previous work, the code corrects insertions and deletions directly, in a single strand of DNA, without the need for multiple alignment of strands. This code, when coupled to a standard outer code, can achieve error-free storage of petabyte-scale data even when ∼10% of all nucleotides are erroneous.

Author contributions: W.H.P., J.A.H., S.K.J., and I.J.F. designed research; W.H.P., J.A.H., S.K.J., J.M.S., and I.J.F. performed research; W.H.P., J.A.H., S.K.J., and I.J.F. contributed new reagents/analytic tools; W.H.P., J.A.H., S.K.J., J.M.S., and I.J.F. analyzed data; and W.H.P., J.A.H., S.K.J., J.M.S., and I.J.F. wrote the paper.

Reviewers: J.B.P., University of Pennsylvania; and H.V.P., Princeton University.

Data deposition: The sequenced reads used in testing are available on the Sequence Read Archive (SRA) under accession nos. SAMN14897329–SAMN14897335 (SRA Project PRJNA631961). The computer code used for the generation and testing of the inner HEDGES code and outer RS code is available on GitHub, https://github.com/whpress/hedges.

[1]To whom correspondence may be addressed. Email: wpress@cs.utexas.edu.

This article contains supporting information online at https://www.pnas.org/lookup/suppl/doi:10.1073/pnas.2004821117/-/DCSupplemental.

First published July 16, 2020.

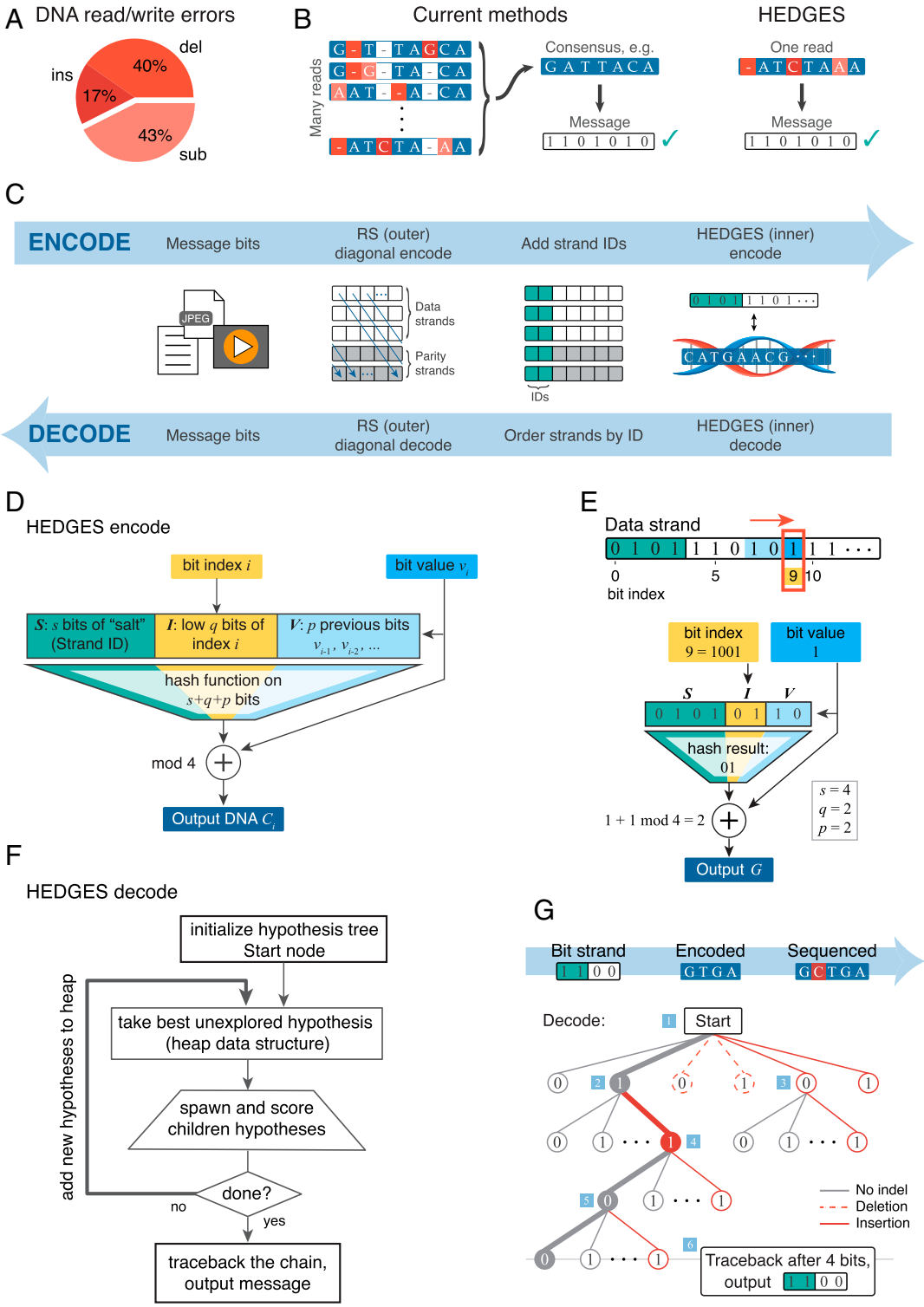

Fig. 1. (*A*) Distribution of insertion and deletion errors (indels) in a typical DNA storage pipeline (Table 1); ins, insertion; del, deletion; sub, substitution. (*B*) (*Left*) Existing DNA-based encoding methods require sequence-level redundancy, strand alignment, and consensus calling to reduce indel errors. (*Right*) HEDGES corrects indel and substitution errors from a single read. (*C*) Overview of the interleaved encoding pipeline used throughout this paper. (*D*) HEDGES encoding algorithm in the simplest case: half-rate code, no sequence constraints. The HEDGES encoding algorithm is a variant of plaintext auto-key, but with redundancy introduced because (in the case of a half-rate code, for example) 1 bit of input generates 2 bits of output. Hashing each bit value with its strand ID, bit index, and a few previous bits "poisons" bad decoding hypotheses, allowing for correction of indels. (*E*) An example HEDGES encode, encoding bit 9 of the shown data strand (red box). As in *D*, half-rate code, no sequence constraints. (*F*) The HEDGES decoding algorithm is a greedy search on an expanding tree of hypotheses. Each hypothesis simultaneously guesses one or more message bits v_i, its bit position index i, and its corresponding DNA character position index k. A "greediness parameter" P_{ok} (see *SI Appendix, Supplementary Text*) limits exponential tree growth: Most spawned nodes are never revisited. (*G*) Illustration of a simplified HEDGES decode. The example bit strand message is encoded and then sequenced with an insertion error. Blue squares give decoding action order: 1, Initialize Start node; 2 to 5, explore best hypothesis at each step; and 6, traceback and output the best hypothesis message. DNA image credit: freepik.com.

to the experimental constraints on DNA synthesis, for example, balanced GC content and the avoidance of homopolymer runs. 4) It has, effectively, zero strand ordering errors, removing a source of large bursts of errors. Although this paper's main contribution is an efficient indel-correcting code, we also develop a specific implementation of the outer Reed–Solomon (RS) code for DNA-based storage. The RS code is applied "diagonally" across multiple DNA strands (Fig. 1*C*) to more evenly distribute synthesis and sequencing errors, which improves error correction performance (15). We test our strategy (both in silico and in vitro) with degraded DNA oligonucleotide pools. Based on these experiments, we use computer simulations to demonstrate that this coding strategy enables error-free exabyte (10^{18})-scale DNA storage.

Results

HEDGES Theoretical Design. Fig. 1 shows the data flow for HEDGES encoding and decoding algorithms. In the terminology of coding theory, HEDGES is an infinite-constraint-length convolutional code (a "tree code") incorporating some features specific to the DNA channel. It is decoded via a stack algorithm that assigns costs to both indels and substitutions. The decoding algorithm succeeds probabilistically, with the ability to signal success or failure. Decoding failures are then regarded as erasures (unknown bits or bytes) and can be corrected in the outer code [i.e., an RS(255,223) code]. Alternatively, the error strand can be discarded and resequenced.

The simplest case is a half-rate code (1 bit encoded per nucleotide) with no constraints on the output DNA sequence, shown in Fig. 1*D* (see *SI Appendix*, Fig. S1 for full diagram). The basic plan is a variant of a centuries-old "text auto-key encoding" cryptographic technique (16) (see also Wikipedia, "Autokey cipher"). We generate a stream of pseudorandom characters $K_i \in \{0,1,2,3\} \equiv \{A, C, G, T\}$, where each K_i depends deterministically, via a hash function, on a fixed number of previous message bits $\{b_j\}$, on the current bit position index i of the current message bit b_i, and on the strand ID. We then emit a character C_i in $\{A, C, G, T\}$ with $C_i = K_i + b_i$, the addition performed modulo 4. Redundancy for error correction occurs because, at each i, C_i can take on only two out of four values, because $b_i \in \{0,1\}$. The output DNA is always completely (pseudo)random because the hash is pseudorandom. Thus, the DNA is, in this sense, independent of the encoded message. We note that nothing limits this scheme to the four "natural" nucleotides; generalization to an increased number is straightforward.

The decoding algorithm sequentially guesses message bits (Fig. 1*F*). Each guessed bit b_i—along with its guessed position i—allows the algorithm to predict (via the forward encoding algorithm) a nucleotide value C_i. If C_i agrees with the observed value, a reward is assigned to that guess. If C_i disagrees, the guess is penalized as appropriate for a substitution. If it disagrees, but would agree at a different assumed position, plus or minus one, that distinct guess is assigned a penalty appropriate for an insertion or deletion. A heap structure keeps systematic track of all currently viable chains of guesses. To prevent the heap from growing exponentially, only chains of guesses that have close to the best total score are extended preferentially, using a variant of the A^* algorithm (17).

In summary, the algorithm encodes information as a stream of nucleotides such that any single decoding error in either nucleotide identity or nucleotide position will "poison" the downstream predictions. Thus, on decoding, there will be only one good-scoring chain of guesses—the correct one. In the unlikely case that the heap grows larger than a preset size H_{limit} (e.g., 10^6), we declare a decode failure and mark the remaining part of the strand as an erasure. Similarly, by including the strand ID in the hash function input, any strand with incorrectly decoded strand ID will be "poisoned" for the full length of the message and fail to decode. This results in effectively zero strand ordering errors, the most expensive type of simple error in interleaved encoding schemes.

Testing In Silico. For in silico testing, we assumed equal rates for substitutions, insertions, and deletions with total error probability per nucleotide $P_{\text{err}} = P_{\text{sub}} + P_{\text{ins}} + P_{\text{del}}$. Any other distribution can then be conservatively bounded by the choice $P_{\text{err}} = 3\max(P_{\text{sub}}, P_{\text{ins}}, P_{\text{del}})$. For simplicity, we assumed that errors occurred at random positions in the DNA strand. The experimental tests in vitro had no such assumption (see below).

The HEDGES algorithm was implemented in C++ for speed, with a Python-callable interface for encoding/decoding single strands. As an initial validation of programming accuracy and interface design, we used an outer-code concatenated design with packets of 255 strands of length 300 protected by RS(255,223). A detailed description of the encoding design is provided in *SI Appendix*, *Supplementary Text*; all corresponding computer programs are provided in *SI Appendix* and available via GitHub. For every code rate r in $\{0.166, 0.250, 0.333, 0.500, 0.600, 0.750\}$, and each assumed P_{err} in $\{0.01, 0.02, 0.03, 0.05, 0.07, 0.10, 0.15\}$, we encoded 10 packets ($\sim 10^6$ nucleotides), pooled all of the strands, and duplicated them as if sequencing to depth 5, meaning that each strand was duplicated a Poisson random number of times, with mean 5. We verified that the pooled and duplicated strands could be decoded, the packets recovered and ordered, RS applied, and the messages recovered without error—but only up to some maximum tolerable error rate P_{err} that increased with decreasing code rate. That is, we "tested to failure" on P_{err}. For this test, the maximum heap size was set to an intentionally small value, $H_{\text{limit}} = 10^5$, to increase the number of decode failures and thus stress the program. The results of this test were as expected and gave us confidence to proceed with in vitro and larger-scale in silico testing (below).

We next needed to construct a statistical error model that could be extrapolated to the petabyte or exabyte scale. For this model, we needed to know the rate of bit errors and byte errors in HEDGES output (for each code rate r as a function of P_{err}). Because decode errors tended to occur in bursts, the rate of byte errors was less than the 8× expected for independent bit errors. We also needed to know the probability per strand of a decode failure, and the distribution of such failures along the strands. For each pair (r, P_{err}), we simulated 12,000 strands of length 10,000 (about 10^8 nucleotides), now with $H_{\text{limit}} = 10^6$. Decode failures occurred approximately uniformly along the strands, consistent with the hypothesis that the heap decode "loses its way" only on rare, local, random sequences (*SI Appendix*, Fig. S2). Decode failures are thus characterized by a single value, the mean run length to failure in a Poisson model. Fig. 2*A* shows the byte error rate as a function of code rate, while *SI Appendix*, Fig. S2 gives full details on observed bit and byte error rates, and mean runs to decode failures. Byte error rates were typically 3 to 5 (not 8) times the bit error rates.

Using these byte error rates, we then modeled HEDGES in an overall concatenated ECC design. Fig. 2*B* shows the average number of message bytes that could be decoded before encountering an uncorrectable error using the concatenated design previously described (see *Methods* for details). A broad set of code rates (green) are suitable for gigabyte- to exabyte-scale DNA storage. HEDGES decode failures in this region occur every 10^4 to 10^5 nucleotides but get corrected by the RS outer code or, optionally, by using another exemplar of the strand (see *Discussion*). Similar simulations with the imposed output constraints of no homopolymer runs (e.g., GGGG or AAAA) greater than four, and $4 \le \text{CG} \le 8$ in any window of 12 nucleotides, are shown in *SI Appendix*, Fig. S3, and are not substantially different from Fig. 2. We also modeled the effects of sequencing

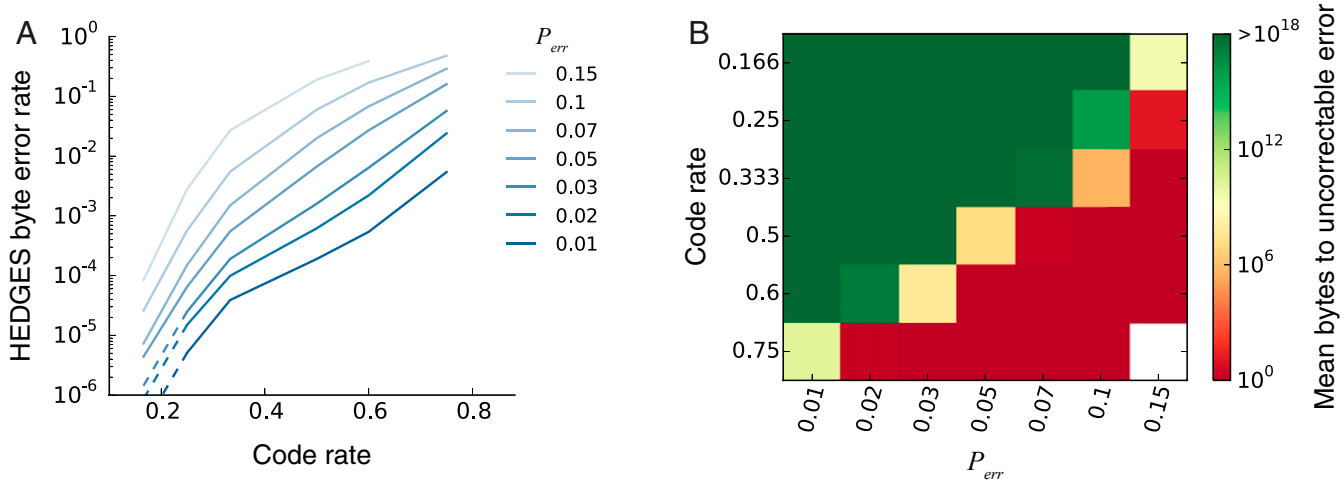

Fig. 2. In silico performance of the HEDGES algorithm. (*A*) The in silico byte error rate for the HEDGES algorithm as a function of code rate, *r*, shown for a range of simulated DNA error rates P_{err}. (*B*) The mean number of bytes to an uncorrectable error, assuming the interleaved RS(255,223) design discussed in the text.

constraints more generally (*SI Appendix*, *Supplementary Text* and Fig. S4), and the combined model and simulation results indicate that the most common sequencing constraints have minimal impact on HEDGES. In sum, in silico simulations indicate that HEDGES is capable of error-free decoding of exabyte-scale messages.

Testing In Vitro. We next tested real-world ECC performance on a pooled sample of 5,865 synthetic 300-base pair DNA strands that were exposed to accelerated aging or enzymatic mutagenesis. Of these, 18 packets of 255 strands were HEDGES inner-encoded (with subsets at each of six code rates) and then RS(255,223) outer-encoded across strands. Five packets, totaling 1,275 strands, were encoded with an unrelated error correction algorithm (18), but also served as a negative control on identifying and sequencing HEDGES strands into packets. Each HEDGES strand consisted of $3'$ and $5'$ primers of length 23 nucleotides (see *Methods*) flanking a 254-nucleotide DNA payload. When decoded into bytes, each payload comprised a 1-byte packet number, a 1-byte sequence number (these "salt-protected" on encryption; see *Methods*), a message payload whose length depended on the code rate, and a 2-byte runout. The sample was PCR amplified and prepared for Illumina-based sequencing. Additionally, we degraded the DNA separately via error-prone PCR mutagenesis or by incubation at high temperature (see *Methods*). Sequencing was done to a mean depth of ∼50.

We performed two kinds of tests of the decoding algorithm, with and without knowledge of the encoded message. "Type A" tests assumed knowledge of the 5,865 strand sequences and could be used to characterize the nature of end-to-end DNA error rates. "Type B" tests were blind decodings of the sequenced data, with knowledge only that the pooled DNA contained HEDGES-encoded data in the specified format.

In our Type A tests, 10 to 15% of sequenced strands could not be uniquely identified with any known input strand, even using quite robust N-gram methods, and even for unmutagenized aliquots. This may be the result of the low concentration, or of contamination at some stage; but it also added to the challenge for the blind type B tests.

For strands whose progenitor sequence could be identified, Table 1 shows measured rates of substitution, insertion, and deletion errors. Notably, only the highest mutagenesis kit protocol produced a substantial increase in DNA errors. Data in ref. 3 estimate DNA degradation over a wide range of timescales and temperatures, suggesting that 50 °C incubation for 8 h should have produced significant mutagenesis. We did not find this, however. So, for further analysis here, we consider only the untreated and high-mutagenesis datasets.

Table 2 shows the results for decoding strands that were identified as belonging to packets of each code rate. Approximately 3% of the strands failed to decode even at small code rates where, in simulation, there were many fewer such failures. Identification of these strands was ambiguous and may stem from PCR mispriming, oligonucleotide misdimerization, and other next generation sequencing (NGS) library preparation artifacts that can vary from batch to batch. Indeed, for lower code rates (where the ECC was relatively unstressed), strand decode failure rates were slightly higher for the untreated case than for the high-mutagenesis case, presumably due to batch-to-batch variation in the number of such artifacts.

For this reason, the values for the mean run to an uncorrectable error in Table 2 are calculated assuming a strategy of rejecting failed decodes, rather than counting them as erasures. We adopted just this rejection strategy in our blind (type B) decoding; the input data were several $\times 10^5$ total reads of the synthesized 5,865 strands (of which 4,590 were message bearing) plus contamination. We shuffled the reads into random order, then attempted HEDGES decoding one read at a time, populating the 18 expected packets of 255 strands with successful decodes and attempting the outer RS error correction when

Table 1. Observed end-to-end DNA error rates, which includes errors introduced during synthesis, sample handling and storage, preparation, and sequencing

		Mutagenesis kit			50°C incubation	
	Untreated	Low	Medium	High	2°h	8°h
Substitution	0.0057	0.0075	0.0178	0.0238	0.0082	0.0085
Deletion	0.0054	0.0045	0.0067	0.0082	0.0040	0.0047
Insertion	0.0023	0.0020	0.0032	0.0039	0.0017	0.0019
Total	0.0134	0.0139	0.0277	0.0359	0.0140	0.0151

The observed total error rates on the order of 1% were approximately doubled and tripled by the medium and high protocols (respectively) of the mutagenesis kit. Incubation at 50 °C for 2 and 8 h had only a small effect and was not further tested.

Table 2. Measured in vitro performance and inferred extrapolation to large datasets

	Code rate					
	0.166	0.250	0.333	0.500	0.600	0.750
Untreated						
Strand decode failure rate	0.033	0.033	0.040	0.045	0.055	0.069
Observed byte error rate	0.00061	0.00110	0.00182	0.00240	0.00248	0.00547
Mean byte errors per RS decode	0.16	0.28	0.46	0.61	0.63	1.39
Mean bytes to uncorrectable	1.8E+28	9.2E+23	2.1E+20	2.2E+18	1.3E+18	3.8E+12
High mutagenesis						
Strand decode failure rate	0.027	0.029	0.029	0.037	0.062	0.322
Observed byte error rate	0.00034	0.00114	0.00137	0.00345	0.00850	0.02888
Mean byte errors per RS decode	0.09	0.29	0.35	0.88	2.17	7.36
Mean bytes to uncorrectable	3.6E+32	5.1E+23	2.4E+22	5.9E+15	4.4E+09	4.9E+02

The upper two values in each box are as experimentally measured in vitro. The bottom values are inferred from the measured quantities for error-free decoding of large datasets under the same experimental conditions. Colors indicate feasibility for large data storage, by the same criteria as Fig. 2.

the number of erasures (missing strands) was small enough (see *Methods* for details).

As expected based on the results of Table 2, we achieved error-free decodes of all packets, except in the case of two packets with high mutagenesis at the highest code rate 0.750. With no mutagenesis, 24,000 total reads were required for all 18 packets. With high mutagenesis, 22,000 reads were required for 16 packets, while the undecodable two continued to fail indefinitely. In the successful cases, the number of reads corresponded to about depth 3 on message-bearing message strands. This depth was required merely to sufficiently populate the packets for the outer code to operate due to random strand sampling, not because of any property of HEDGES as the inner code.

Discussion

HEDGES is designed to be flexible with respect to DNA strand lengths, DNA sequencing and synthesis technologies, choices of outer code, and interleaving details. The most important feature of HEDGES is that it always either 1) recovers "perfect" synchronization of the individual DNA strand to which it is applied (that is, completely eliminates insertion and deletion errors) or else 2) signals that it is unable to do so by a decode failure. Here "perfect" means that our reported bit and byte error rates, which are small enough to be completely corrected by a standard outer code such as RS, are already inclusive of any residual instances of missynchronization.

In the feasible (green) regions of Fig. 2, HEDGES decode failures occur about every 10^4 to 10^5 nucleotides (bottom cells). Two strategies are possible: 1) We can keep these strands and mark as erasures the bits after the failure point, or 2) we can, instead, use another strand from the pool showing the same strand ID—thus increasing the sequencing depth requirement by a tiny amount. The performance values shown in Fig. 2 use strategy 1; those in Table 2 use strategy 2. Importantly, HEDGES allows constraints on the encoded DNA strands such as reducing homopolymer runs and maintaining a balanced GC content. *SI Appendix*, Fig. S3, when compared to Fig. 2, shows that such constraints impose little penalty on both the code rate and error correction level. Thus, we demonstrate that both are viable strategies for error correction.

We performed both in silico and in vitro experiments to validate HEDGES across a variety of error rates. Such statistical analyses of rare events, based on both experimental data and simulations, should be a required part of all future proposals for DNA data storage. HEDGES performance on real DNA with observed total errors of ∼1% and ∼3% (Tables 1 and 2) was comparable to computer simulation at the same total DNA error rates and to the statistical model we built using simple Poisson random errors (Fig. 2). In both cases, HEDGES demonstrates the feasibility of large-scale error-free recovery at code rates up to 0.6 (1.2 bits per nucleotide) for ∼1% DNA errors; and 0.5 (1 bit per nucleotide) for ∼3% DNA errors. Error-free exabyte-scale storage is feasible at DNA error rates as large as 7 to 10% with a code rate of 0.25 (0.5 bits per nucleotide). Thus, HEDGES paves the way for robust error correction in large-scale but error-prone pooled synthesis of large DNA libraries.

Methods

HEDGES Encoding in the Half-Rate Case. Given a message stream of bits

$$b_i, \quad i = 0, 1, 2, \ldots, M, \quad b_i \in \{0, 1\} \qquad [1]$$

("the message" or "bits"), we want to emit a stream of DNA characters

$$C_i, \quad i = 0, 1, 2, \ldots, N, \quad C_i \in \{A, C, G, T\} \equiv \{0, 1, 2, 3\}$$

("the codestream" or "characters"). We first review the case of a half-rate code, where we emit exactly one C_i (2 bits of output) for each b_i (1 bit of input). Then we will generalize to codes at other rates r (message bits per codestream bit), $0 < r < 1$, so that the streams b_i and C_i are not then in lockstep, and $M \neq N$. One should think of N as being on the order of 10^2 to 10^4, the maximum length of a single DNA strand that can be cheaply synthesized today or in the foreseeable future. We want to be able to decode, without residual errors, a received codestream C' that differs from C by substitutions (errors), insertions, and deletions (collectively "indels"). Indels are silent: Their positions in the codestream C' are not known to the receiver.

We generate a keystream of characters $K_i \in \{0, 1, 2, 3\}$, where each K_i depends pseudorandomly (but deterministically by a hash function) on some number of previous message bits b_j (with $j < i$), and also directly on the bit position index i and the strand ID,

$$K_i = F(S_i, I_i, B_i), \qquad [2]$$

where S_i is s bits of salt (strand ID), I_i is the lower q bits of the bit index, B_i is the previous p bits, and F is a chosen hash function. (See *SI Appendix, Supplementary Text* for initialization.) We then emit a codestream character

$$C_i = K_i + b_i, \qquad [3]$$

the addition performed modulo 4.

The redundancy necessary for error correction comes from the fact that b_i takes on only two values, while K_i and C_i can have four values. This generates (only) 1 bit of redundancy per character, that is, can be acausally valid by chance half of the time. However, the dependence of K_i on many previous message bits ties any given message bit to many future bits of redundancy. Similarly, the dependence of K_i on i ties every bit to its position index, so that insertions can be identified and removed, and deleted values can be restored.

Some further details about the encoding algorithm are given in *SI Appendix, Supplementary Text*.

HEDGES Decoding Algorithm. For simplicity, assume that error rates are "small," so that "most" DNA bases are received as they were intended. (We saw, in *Results*, that DNA character error rates up to ∼5 to 10% are tolerable.) Suppose we have correctly decoded and synchronized the message through bit b_{i-1} and now want to know bit b_i. Guessing the two possibilities, $\{0, 1\}$, we use Eq. **3** to predict two possibilities for the character C_i. In the absence of an error, only one of these is guaranteed to agree with the observed character C'_i. We assign, to a guess that generates disagreement with C'_i, a penalty score equal (conceptually) to the negative log probability of observing a substitution error. In other words, a wrong guess might actually be right, but only if a substitution has occurred. If neither guess produces the correct C_i, then both are assigned the substitution penalty.

We have not yet accounted for the possibility of insertions and deletions, however. In fact, there are more than the above two possible guesses. We must guess not just $b_i \in \{0, 1\}$, but also a "skew" $\Delta \in \{\ldots, -1, 0, 1, \ldots\}$ that tells us whether, in comparing C to C', we should skip characters ($\Delta > 0$) because of insertions, or posit missing characters ($\Delta < 0$) because of deletions (in which case, there is no comparison to be done). As a practical simplification, we consider only $\Delta \in \{-1, 0, 1\}$, requiring multiple indels to resolve as concatenated single indels. Then there are six guesses for $(b_i, \Delta) \in \{0, 1\} \otimes \{-1, 0, 1\}$. Each can be scored by an appropriate log probability penalty for any implied substitution, insertion, or deletion.

Log probability penalties accumulate additively along any chain of guesses. In the causal case of a chain of all-correct guesses, we accumulate penalties only in the (relatively rare) case of actual errors. However, because of the way that the key K_i (Eq. **3**) is constructed, a single wrong guess for either b_i, i, or Δ throws us into the acausal case where 3/4 of subsequent comparisons of computed C (at some bit position index i) to observed C' (at some index k) will not agree—thus penalties will accumulate rapidly. The decoding problem, conceptually a maximum likelihood search, thus reduces to a shortest-path search in a tree with branching factor 6, but with the saving grace that the correct path will be much shorter than any deviation from it.

The rate of decode errors rises in the last several bytes of message, because some incorrect chains don't have time to accumulate bad scores. To counter this, we pad each strand with (typically) two "runout bytes" of message zeros at encode, and ignore them at decode. The need for runout bytes makes the HEDGES algorithm inefficient (and thus unsuitable) for an application needing very short DNA strands (e.g., tens rather than hundreds of nucleotides).

Further details about the decoding algorithm are given in *SI Appendix, Supplementary Text*.

Use of Salt to Protect Critical Message. In Eq. **3** and Fig. 1*D*, we allowed for some number of bits of known salt S_i when message bit b_i is encoded. The use of salt is optional, but is desirable in an overall interleaved design where sequenced strands from a pool need to be correctly ordered at decode time (*SI Appendix, Supplementary Text*). This is generally the case when the outer code is interleaved across strands. To the outer code decoder, each incorrectly ordered strand is equivalent to a full strand length of random errors, so it is very important to protect strand ID message bits that determine the strand ordering for outer decoding. Here is how salt is enabling of extra protection: Suppose we want to protect an initial s message bits. Then define, recursively, the salt by

$$\begin{aligned} &S_0 = b_0 \\ &S_i = S_{i-1} b_i, \quad i = 1, \ldots, s-1 \quad \text{(denoting concatenation)}. \\ &S_i = S_{i-1}, \quad i \geq s \end{aligned} \qquad [4]$$

Most errors in the first s bits will be corrected as usual by the shortest-path heap search. But any residual error that gets through will "poison" the salt for the entire rest of the strand, rendering it undecodable. In effect, we convert an error in the protected bits into an erasure of the whole strand. This may seem drastic, but it is just what we want: A strand with incorrect serial number (and hence incorrect ordering among other strands) would look like a strand of errors to the outer code; an erased strand is equivalent to only half as many errors.

Code Rates Other than One-Half. A simple modification of the encode and decode algorithms described above allows for code rates other than one-half. Take the input bitstream of expression **1** and partition it into a stream of values v_k with variable numbers of bits in the range 0 to 2, according to a repetitive pattern like the ones shown in Table 3. See also *SI Appendix*, Fig. S1.

Here are two examples showing how to interpret the entries in Table 3 (with adjacency denoting 2-bit values in $\mathbb{Z}_4$):

$$\text{Rate 0.750: } v_0 = b_0 b_1, v_1 = b_2, v_2 = b_3 b_4, v_3 = b_5, \ldots$$
$$\text{Rate 0.250: } v_0 = b_0, v_1 = 0, v_2 = b_1, v_3 = 0, \ldots$$

.

Eq. **3** for encoding now becomes

$$C_i = K_i + v_i = F(S_i, I_i, V_i) + v_i \pmod 4, \qquad [5]$$

where V_i is composed of concatenated previous variable bits. Pattern values of 0 provide 1 bit of additional redundancy check relative to the base case of code rate one-half, while pattern values of 2, encoding 2 bits per DNA character, provide 1 bit less (i.e., zero). By construction, the code rate is one-half the average of the integers in the pattern. The column in the table labeled P_{ok} is a "greediness parameter" that mitigates the tendency of the heap to expand exponentially; see *SI Appendix, Supplementary Text* for details of this.

Decoding follows exactly the same pattern. Guessing a 2-bit v_i spawns 12 child hypotheses, while guessing a zero-bit v_i spawns only 3.

HEDGES Parameters. For encoding, the parameter choices are 1) the choice of code rate and variable bit pattern (as in Table 3), the default case being code rate 0.5; 2) the number $q > 0$ of low-order bits of position index in the hash; 3) the number $p > 0$ of previous message bits in the hash; 4) the number $s \geq 0$ of salt bits; and 5) the number $n \geq 0$ of initial message bits to be protected by salt. Aspects of the choices of, and trade-offs among, these parameters are further discussed in *SI Appendix, Supplementary Text*.

It is an important point that choosing the decode runtime parameters, for example, H_{limit} or P_{ok}, is not an irrevocable choice. Given a DNA message, one can make multiple tries, varying the decode parameters adaptively until acceptable performance is achieved. Simply increasing H_{limit} and retrying will often rescue failed decodes (*SI Appendix, Supplementary Text* and Fig. S6). One can evaluate success by running time and by the count of errors needing correction by the outer RS code. The parameter values that we suggest may be viewed as starting points.

Imposing DNA Output Sequence Constraints. DNA synthesis and sequencing platforms have sequence-dependent error profiles. Imbalanced GC content and homopolymer runs are well known to be problematic, for example, leading to indel and substitution errors or even whole strand dropout errors in popular sequencers such as those from Illumina and Oxford Nanopore (19, 20). Thus, many proposed ECCs impose constraints on GC content, homopolymer runs, or both. These typically involve one-off coding designs for each constraint, often reducing significantly the effective code rate (7). An important property of the HEDGES algorithm is that it can accommodate a large class of sequence constraints without additional, one-off, code design. Moreover, constraints may be imposed without decreasing the code rate (a seeming paradox that we explain below).

Consider the class of constraints that can be applied to a nucleotide sequence with only "past" information. That is, from the emitted sequence $[\ldots, C_{i-3}, C_{i-2}, C_{i-1}]$, we can determine whether the choice of C_i is constrained and, if so, what is the set of its allowed values, here denoted $\{C_i^*\}$, whose size we denote $\#C_i^*$. We assume $0 < \#C_i^* \leq 4$, because a zero value would imply a constraint so severe that the strand cannot be continued at

Table 3. Mapping of bits b_i to variable bits v_i for various code rates

Code Rate	Pattern	P_{ok}*
0.750	2, 1, 2, 1, . . .	−0.035
0.600	2, 1, 1, 1, 1, 2, 1, 1, 1, 1, . . .	−0.082
0.500	1, 1, . . .	−0.127
0.333	1, 1, 0, 1, 1, 0, . . .	−0.229
0.250	1, 0, 1, 0, . . .	−0.265
0.166	1, 0, 0, 1, 0, 0, . . .	−0.324

*See *Code Rates Other than One-Half*.

all. (One could adopt the convention of relaxing such constraints if they are sufficiently rare.)

The only required change in the HEDGES algorithm is to replace Eq. **5** by

$$C_i = K_i + v_i = F(S_i, I_i, V_i) + v_i \ (\mathrm{mod}\ \#C_i^*) \qquad [6]$$

and output the thus-indexed character in the acceptable set $\{C_i^*\}$ (*SI Appendix*, Fig. S1).

Note that Eq. **6** has the same code rate as [**5**]. How is this possible when constraints always act to reduce the number of possible output strings, and hence reduce the channel capacity? The answer is that [**6**] absorbs the reduced capacity completely into the error correction, not into the message. The extreme case is when $\#C_i^* = 1$. Then, the emitted character does not depend on the message bits v_i, which become de facto erasures. At decode time, the missing bits are restored by virtue of the fact that they enter V_i and thus the hash function for later emitted characters.

This is a versatile scheme. When there is a possibility of emitting an unacceptable homopolymer, $\#C_i^*$ will decrease from 4 to 3. When the GC count in a specified window is too large (or small), $\#C_i^*$ will decrease from 4 to 2. More-complex constraints are easily added. For example, Illumina sequencers have been reported to have high error rates following a GGC motif (4). To disallow GGC motifs, one might have imagined building a bespoke code around triplets of characters encoded as members of a large Galois field (3) and removing GGC, NGG, and GCN as possible output triples. Within HEDGES, GGC can be disallowed as an afterthought, simply by adding it to the list of forbidden outputs.

Statistical Model of System Designs with Outer Codes. We take as input, from *SI Appendix*, Fig. S5, the byte error rate P_{byte} and mean run to decode failure L_{fail}. The outer code is characterized by its length $N_T = N_m + N_c$, where N_m is the number of (payload) message bytes, and N_c is the number of correction bytes. For RS(255,223), we have $N_T = 255$ and $N_m = 223$. The number of correctable byte errors is well known to be $N_c/2$, with erasures (bytes known to be missing) counting as half of an error.

We assume that the RS error correction is interleaved, that is, runs across strands. We further assume that, as in our example concatenated design, each RS input byte string samples uniformly along the length of the DNA strands and thus sees uncorrelated errors and erasures with their respective mean rates. We implemented this by applying the RS outer code "diagonally" across strands, so that strand ends (for example) are distributed over multiple RS decodes (Fig. 1*C*). The number of errors in each RS correction is thus a random Poisson variable $\mathcal{N}_{\mathrm{err}}$ with mean $N_T P_{\mathrm{byte}}$.

Erasures can occur for two reasons: 1) If the DNA pool is sequenced to a mean depth D, then a fraction $p_{f1} = \exp(-D)$ strands (and therefore bytes) will occur zero times and be counted as erasures. 2) Along each strand of length S, the probability of a (first) decode failure at position x is $(1/L_{\mathrm{fail}})\exp(-x/L_{\mathrm{fail}})$, in which case $S - x$ DNA positions are left undecoded. The fraction (in either DNA or bytes) of such erasures is thus

$$\begin{aligned} p_{f2} &= \frac{1}{S}\int_0^S \frac{1}{L_{\mathrm{fail}}} e^{-x/L_{\mathrm{fail}}}(S-x)dx \\ &= 1 - \frac{L_{\mathrm{fail}}}{S}\left(1 - e^{-S/L_{\mathrm{fail}}}\right) \approx \tfrac{1}{2}\frac{S}{L_{\mathrm{fail}}} \end{aligned} \qquad [7]$$

The number of erasures is thus a random Poisson variable $\mathcal{N}_{\mathrm{era}}$ with mean $N_T(p_{f1} + p_{f2})$. The mean run (in message bytes) to an uncorrectable error is thus the reciprocal of a tail probability,

$$L_{\mathrm{u.c.}} = 1/\mathrm{Prob}(\mathcal{N}_{\mathrm{err}} + \tfrac{1}{2}\mathcal{N}_{\mathrm{era}} > \tfrac{1}{2}N_c). \qquad [8]$$

While the sum of two Poisson variables is Poisson, the factor one-half in front of $\mathcal{N}_{\mathrm{era}}$ spoils this, so Eq. **8** involves no simple distribution. Replacing the one-half by either zero or 1 does give Poisson distributions, however, that bound the desired result and turn out to be not too different numerically (in their power-of-ten exponents, which is all we care about). As an approximation, we thus may interpolate by setting the factor back to one-half and pretend that it is still Poisson.

Poisson tail probabilities can be written exactly as incomplete gamma functions (21). In our regime of interest, the upper and lower bounds (22),

$$p_n < \mathrm{Prob}(x \geq n|\lambda) < \left(1 - \frac{\lambda}{n+1}\right)^{-1} p_n, \qquad [9]$$

nearly coincide, where $p_n = \mathrm{Prob}(x = n) = \exp(-\lambda)\lambda^n/n!$. We can use the upper bound to get a tight lower bound on $L_{\mathrm{u.c.}}$.

The values in Fig. 2 (red/yellow/green cells) result from performing the above calculations with $N_T = 255$, $N_m = 223$, $N_c = 32$, $S = 300$, and $D = 10$.

Blind Decoding Tests. We eliminated reads in which the flanking 5′ or 3′ primers were not found, or were separated by significantly more or less than the expected payload length of 254. Because any blind test must be ignorant of the message, we did not eliminate the 10 to 15% of strands that (we secretly knew) contained "junk" payloads that corresponded to no intended message.

Taking the reads one at a time, we first attempted decodes at all six designed code rates, but limiting the hypothesis budget to $H_{\mathrm{limit}} = 5{,}000$. Whichever code rate got farthest in the message was declared as the putative actual code rate for that strand. At that code rate, we next attempted to decode 2 bytes (the packet number and strand sequence IDs) plus three runout bytes. If this decode succeeded, we checked by IDs to see whether this strand was already seen and decoded. If not, we performed a full decode with a hypothesis budget of $H_{\mathrm{limit}} = 10^6$. This strategy was designed to avoid unnecessary full decodes.

When the full decode succeeded, we accepted it as authoritative for that strand by packet and sequence ID, and performed no further full decodes for that strand when encountered. When it failed, we discarded the strand. That is, we did not attempt to rescue bytes up to the failure point.

After every 1,000 strands, we looked for packets that were sufficiently populated to possibly allow RS(255,223) outer error correction, counting missing strands as erasures. When the outer correction eventually succeeded, we marked that packet as done. This is a conservative strategy for testing, using only the first observed read of each strand and a near-maximal number of missing strands in each packet for the outer decode. In practice, after sequencing, one would use all available reads and potentially reduce the error rates even further.

DNA Library Prep and Sequencing. A DNA library of 5,865 300-nt oligonucleotides was synthesized by Twist Biosciences. To introduce mutations, the DNA storage library was mutagenized using the Diversify PCR Random Mutagenesis Kit (Takara 630703). Samples were added with 480 μM $MnCl_2$ and 40 μM 2′-deoxyguanosine 5′-triphosphate (dGTP), 640 μM $MnCl_2$ and 200 μM dGTP, or 960 μM $MnCl_2$ and 240 μM dGTP to achieve low, medium, and high levels of mutagenesis, respectively. Samples were then PCR amplified for 12 cycles using primers IF538 and IF539.

To mimic time-dependent aging of DNA, DNA storage library samples were incubated at 50 °C for a period of 2 h or 8 h in water (3). Samples were then PCR amplified for 12 cycles using primers IF538 and IF539.

PCR-amplified samples were purified using a PCR cleanup kit (NEB T1030).

Illumina sequencing libraries were prepared using the NEBNext Multiplex Oligos for Illumina (E7335) primer set. Libraries were sequenced on a paired-end MiSeq chip with 300 base pairs of read length. Final library preparation and sequencing was performed by the University of Texas Genomic Sequencing and Analysis Facility (GSAF).

Data Availability. The sequenced reads used in testing are available on the Sequence Read Archive (SRA) under accession numbers SAMN14897329, SAMN14897330, SAMN14897331, SAMN14897332, SAMN14897333, SAMN14897334, and SAMN14897335 (SRA Project number PRJNA631961) (23).

Oligos used in the study are as follows:

IF538: 5′-TCGAAGTCAGCGTGTATTGTATG-3′.

IF539: 5′-AACACGCTTAATCGCACTCACTA-3′.

The computer code used for the generation and testing of the inner HEDGES code and outer RS code is available at https://github.com/whpress/hedges. This paper utilized two commercial C++ source code libraries: Numerical Recipes (http://www.numerical.recipes) and the Schifra Reed-Solomon Error Correcting Code Library (http://www.schifra.com). The specific routines used in this paper are freely available for noncommercial use and are included in the above GitHub repository.

ACKNOWLEDGMENTS. We thank Twist Biosciences and the GSAF at The University of Texas at Austin for their help, respectively, in synthesizing and sequencing our DNA. We have had useful communication with Dave Forney, Dan Costello, and Felix Pahl. Claire Mirocha assisted with the data analysis. This work was supported by a College of Natural Sciences Catalyst Award, the Welch Foundation (Grant F-1808 to I.J.F.), and the NIH (Grants R01 GM120554 and R01 GM124141 to I.J.F., and Grant F32 AG053051 to S.K.J.). The DNA graphic in Fig. 1 is from freepik.com.

1. G. M. Church, Y. Gao, S. Kosuri, Next-generation digital information storage in DNA. *Science* **337**, 1628 (2012)
2. N. Goldman *et al.*, Towards practical, high-capacity, low-maintenance information storage in synthesized DNA. *Nature* **494**, 77–80 (2013).
3. R. N. Grass, R. Heckel, M. Pudda, D. Paunescu, W. J. Stark, Robust chemical preservation of digital information on DNA in silica with error-correcting codes. *Angew. Chem. Int. Ed.* **54**, 2552–2555 (2015).
4. J. Bornholt *et al.*, A DNA-based archival storage system. *Comput. Architect. News* **44**, 637–649 (2016).
5. Y. Erlich, D. Zielinski, DNA Fountain enables a robust and efficient storage architecture. *Science* **255**, 950–954 (2017).
6. S. M. H. T. Yazdi, R. Gabrys, O. Milenkovic, Portable and error-free DNA-based data storage. *Nat. Sci. Rep.* **7**, 5011 (2017).
7. L. Organick, S. D. Ang, Y.-J. Chen, Random access in large-scale DNA data storage. *Nat. Biotechnol.* **36**, 242–248 (2018).
8. L. Ceze, J. Nivala, K. Strauss, Molecular digital data storage using DNA. *Nat. Rev. Genet.* **20**, 456–466 (2019).
9. F. J. MacWilliams, N. J. A. Sloane, *The Theory of Error-Correcting Codes* (North Holland, 1983).
10. R. M. Roth, *Introduction to Coding Theory* (Cambridge University Press, 2006).
11. T. K. Moon, *Error Correction Coding: Mathematical Methods and Algorithms* (Wiley, 2005).
12. S. Lin, D. J. Costello, *Error Control Coding* (Pearson, ed. 2, 2004).
13. M. Mitzenmacher, A survey of results for deletion channels and related synchronization channels. *Probab. Surv.* **6**, 1–33 (2009).
14. R. Li, New developments in coding against insertions and deletions. Honors thesis, Carnegie Mellon University, Pittsburgh, PA).
15. S. B. Wicker, V. K. Bhargava, *An Introduction to Reed-Solomon Codes* (Wiley-IEEE, 1994).
16. B. de Vigenère, *Traicté des chiffres ou secrètes manières d'escrire* (Abel l'Angelier, Paris 1586), ff. 46r–49v.
17. P. E. Hart, N. J. Nilsson, B. Raphael, A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Sys. Sci. Cyber.* **4**, 100–107 (1968).
18. J. A. Hawkins, S. K. Jones, I. J. Finkelstein, W. H. Press, Indel-correcting DNA barcodes for high-throughput sequencing. *Proc. Natl. Acad. Sci. U.S.A.* **115**, E6217–E6226 (2018).
19. Y.-C. Chen, T. Liu, C.-H. Yu, T.-Y. Chiang, C.-C. Hwang, Effects of GC bias in next-generation-sequencing data on de novo genome assembly. *PloS One* **8**, e62856 (2013).
20. M. Jain *et al.*, Nanopore sequencing and assembly of a human genome with ultra-long reads. *Nat. Biotechnol.* **36**, 338–345 (2018).
21. W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, *Numerical Recipes: The Art of Scientific Computing* (Cambridge University Press, ed. 3, 2007).
22. B. Klar, Bounds on tail probabilities of discrete distributions. *Probab. Eng. Inf. Sci.* **14**, 161–171 (2000).
23. W. H. Press, J. A. Hawkins, S. K. Jones Jr, J. M. Schaub, I. J. Finkelstein, HEDGES error-correcting code for DNA storage corrects indels and allows sequence constraints. NCBI BioProject. https://www.ncbi.nlm.nih.gov/bioproject/631961. Deposited 12 May 2020.