PNAS Nexus, 2022, **1**, 1–9
https://doi.org/10.1093/pnasnexus/pgac252
Advance access publication date: 4 November 2022
Research Report

Fast trimer statistics facilitate accurate decoding of large random DNA barcode sets even at large sequencing error rates

William H. Press[*]

Department of Computer Science and Department of Integrative Biology, The University of Texas at Austin, Austin, TX 78712, USA
*To whom correspondence should be addressed: Email: wpress@cs.utexas.edu
Edited By: Shibu Yooseph

Abstract

Predefined sets of short DNA sequences are commonly used as barcodes to identify individual biomolecules in pooled populations. Such use requires either sufficiently small DNA error rates, or else an error-correction methodology. Most existing DNA error-correcting codes (ECCs) correct only one or two errors per barcode in sets of typically $\lesssim 10^4$ barcodes. We here consider the use of random barcodes of sufficient length that they remain accurately decodable even with $\gtrsim 6$ errors and even at $\sim$ 10% or 20% nucleotide error rates. We show that length ~34 nt is sufficient even with $\gtrsim 10^6$ barcodes. The obvious objection to this scheme is that it requires comparing every read to every possible barcode by a slow Levenshtein or Needleman-Wunsch comparison. We show that several orders of magnitude speedup can be achieved by (i) a fast triage method that compares only trimer (three consecutive nucleotide) occurence statistics, precomputed in linear time for both reads and barcodes, and (ii) the massive parallelism available on today's even commodity-grade Graphics Processing Units (GPUs). With 10^6 barcodes of length 34 and 10% DNA errors (substitutions and indels), we achieve in simulation 99.9% precision (decode accuracy) with 98.8% recall (read acceptance rate). Similarly high precision with somewhat smaller recall is achievable even with 20% DNA errors. The amortized computation cost on a commodity workstation with two GPUs (2022 capability and price) is estimated as between US$ 0.15 and US$ 0.60 per million decoded reads.

Significance Statement:

DNA barcodes—known, short, unique DNA segments—are often used by experimenters to identify individual biomolecules in pooled populations. In experiments with large DNA synthesis or DNA sequencing error rates, such segments sometimes use an error-correcting code. But such codes typically can correct only one or two errors per barcode and may limit barcode set sizes to $\sim 10^4$. Here we describe a method for creating and decoding barcodes that gives accurate results even with six or more errors per barcode, can have $>10^6$ barcodes, and is useful even at >10% nucleotide error rates.

1 Introduction

The use of DNA barcode libraries to identify tagged individual biomolecules in pooled populations has become an essential tool for today's massively parallel biomedical experiments. Barcodes find use in gene synthesis (1,2), antibody screens (3,4), drug discovery via tagged chemical libraries (5–7), and many other applications (8–14), including their potential use in schemes for engineered DNA data storage (15,16). For some applications, barcodes must function robustly in experimental situations subject to significant error rates (that is, the unintended occurrence of nucleotide substitutions, insertions, and deletions). Errors may be introduced during barcode synthesis, the processes of the experiment, the final sequencing, or all of these (15). Errors in barcode synthesis ("wrong barcodes") are particularly troublesome because they create errors that persist at any depth of final sequencing.

"Next-generation sequencing" (NGS), as exemplified in Illumna technology (17), has relatively short read lengths (200 to 300 nt), but also relatively small error rates (10^{-3} to10^{-4} per nt). In this regime, barcodes need to be short ($\lesssim$20 nt), but they need only modest (if any) error-correction capability. In other words, barcodes as ultimately sequenced can be assumed to have at most one or two errors, allowing the use of repurposed mathematical error-correcting codes (ECCs) (18,19), sometimes (20–24), but not always (25,26), with the necessary extensions to account for insertion and deletion errors ("indels"). By a "mathematical" ECC, we mean a set of codewords and also an algorithm for recovering an original codeword from its garbled version, specifically without needing to compare every potentially garbled read to every known possible codeword in the library. (We use the terms "barcode" and "codeword" almost interchangeably, the former being the physical manifestation in DNA of the mathematical latter.)

Competing Interests: The author declares no competing interest.
Received: July 4, 2022. **Accepted:** October 31, 2022

Useful mathematical ECCs for use with NGS have in practice been limited to libraries with no more than tens of thousands of unique codewords. While any code should ideally signal a reject ("erasure") rather than return a wrong identification if the garbled word has more errors than the ECC can handle, this in general cannot be mathematically assured if the number of errors is not strictly bounded (19).

Hawkins et al.'s "FREE" barcodes (27) overcame some of these limitations with a direct approach: Libraries of pairwise dissimilar codewords were constructed by comparing each proposed new codeword to all previously accepted ones, a slow, but one-time, process. A novel similarity measure was designed to be tolerant of indels that could produce garbled barcodes with unknown, altered lengths. Advantageously, codewords could be constrained to have balanced GC content, minimal homopolymer runs, reduced hairpin propensity, or any other experimentally motivated constraints. In the FREE scheme, garbled codewords are decoded by table lookup into a very large table containing not only the codewords but also all of their possible single- or double-error garbles. This is very fast, but requires very large computer memory. Practically, this scheme achieves single-error–correcting codes of 16-nt length with 1.6×10^6 barcodes or double-error–correcting codes of 17-nt length with 23,000 codes (27).

In recent years, third-generation sequencing (also known as TGS or long-read sequencing), in variants developed by Pacific Biosciences and Oxford Nanopore (28), has added new capabilities. TGS is capable of very long reads, $>10^4$ nt, so barcode length is of small consequence. However, read error rates may be as high as $\sim$10% (29). Such error rates render virtually useless single- and double-error-correcting barcode libraries of useful size. In the most favorable case of independent random errors, three or more errors can occur frequently; burst errors such as stuttering or repeated deletions only make things worse.

This paper explores a possible solution via the use of random barcodes ("randomers") (30), that is, barcode libraries of any desired size ($\gtrsim 10^6$, for example), whose codewords are approximately uniformly random, as generated by computer, with constraints of GC content, homopolymers, etc., easily imposed by rejecting random draws that fail to meet those constraints (never a significant fraction). Like "designed" barcodes, random barcodes would be synthesized in defined oligo pools, but with the difference that the number of pools could be as large as desired. There are two obvious, immediate objections to this scheme that must be overcome: (1). How can we avoid too-similar pairs of codewords in the library so that the garbles of one are not mistakenly decoded as the other? (2). How can we avoid the impractical all-to-all in-silico comparison of every read to every codeword in the library?

The answers are unexpectedly simple. (1) We use barcodes of length sufficient to make near-collisions statistically unlikely to any desired degree. To implement this, we below investigate the statistics of such near-collisions. (2) Instead of rejecting all-against-all brute force comparison, we embrace it. Below, we will describe a novel, fast computational technique that characterizes codewords by their overlapping trimers (three-nucleotide sequences), both trimer presence versus absence, and the order of those present. We show in particular that these techniques can run with massive parallelism on commodity graphics processing units (GPUs) and that cloud GPU availability (31,32) makes such all-against-all comparisons practical at low cost and with reasonable throughput.

2 Materials and methods

2.1 Distance measures

Given a set of barcode codewords and given a garbled barcode read (possibly because of indels, prefixed, or suffixed by spurious nucleotides), by definition the best decode we can do is to assign the read to its most probable codeword—or to declare it an erasure that cannot be reliably so assigned. "Most probable" implies an accurate statistical characterization of all the processes that produce errors, in practice rarely available (33). So, any practical procedure involves choosing a surrogate, a distance measure between the two strings that at least approximates (a monotonic function of) $P(R|C)$, the probability of a garbled read R given the true codeword C.

A gold standard for such an approximation is the Needleman–Wunsch (34) alignment distance between the strings, with the skew, substitution, insertion, and deletion penalties set to the negative log-probabilities of their respective occurrence in an experimentally validated error model. To the degree that errors are independent, the distance so obtained is the negative log-probability of the most probable single path from codeword to read. Note that even this gold standard is not exact because (i) the implied model of independent and identically distributed (i.i.d.) errors is surely not right in detail and (ii) the probability $P(R|C)$ is actually a sum over all possible paths, not the single most probable path.

Levenshtein distance (also called edit distance) (22) is a kind of silver standard, not as good as Needleman-Wunsch, but also not dependent on knowing error probabilities. Levenshtein distance is identical to Needleman–Wunsch when the skew, substitution, insertion, and deletion penalties all set to the same constant value (without loss of generality, the value 1). In the remainder of this paper, we will use Levenshtein distance exclusively. However, all of the algorithms developed (and all of the implementing computer code) are designed to allow arbitrary penalties, hence the easy generalization to Needleman–Wunsch.

2.2 Levenshtein distance distribution of random strings

If there were no indels, then the Levenshtein distance between two random strings of the same length would be their Hamming distance, with an easily calculated binomial probability distribution (for independent errors). With indels, the distribution of Levenshtein distances between two random strings is a famously unsolved problem, closely related to the better-known unsolved problem of longest-common subsequences (35). While it is known that for asymptotically long strings the mean distance scales as a constant γ_c times string length (hardly a surprise, given that the errors are local), γ_c, termed the Chvátal–Sankoff constant (36), is not known, though it is conjectured to be $2/(1+\sqrt{c})$, where c is the alphabet size (for us, 4). Beyond this mean, virtually nothing is known about the distribution of distances, although there is a conjectured connection to so-called Tracy–Widom distributions (37).

While little is known analytically, simulation is straightforward. Supporting Information S1 (text) describes, and Fig. S1 illustrates, how one-to-many Levenshtein distances can be parallelized on a GPU, allowing the calculation of $>10^8$ distances on a single-headed desktop machine in minutes. Fig. 1 shows the results of such a simulation.

We are concerned about the extreme left-hand tails of the distributions, where a garbled read from one codeword might end up by unlucky chance, close to another in a large set of barcodes. In this regime, direct sampling is impractical, but we can use the polynomial extrapolations (in log-space) shown in Fig. 1. Their

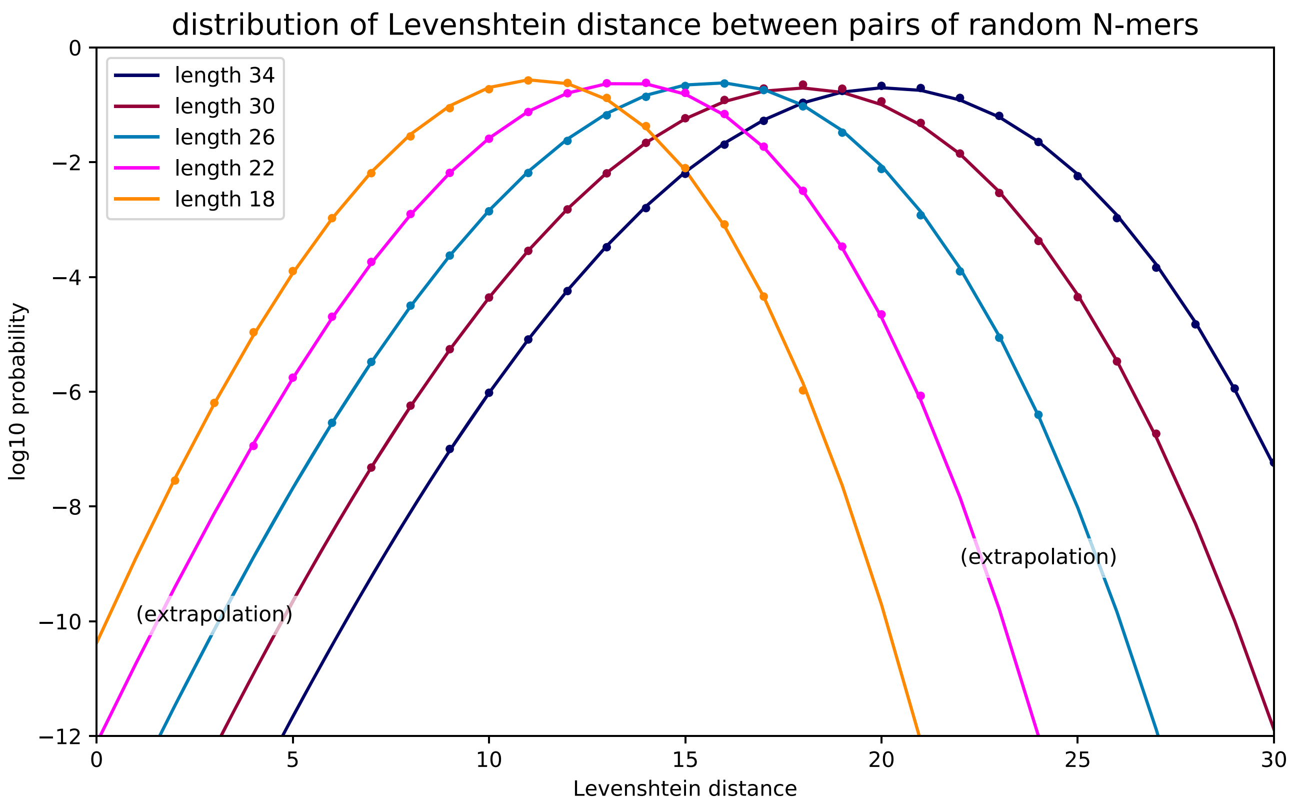

Fig. 1. Probability distribution of Levenshtein distances. Random oligomers of lengths 18, 22, 26, 30, and 34 are generated and random pairwise Levenshtein distances are calculated. Dots show the results. The curves are a bivariate polynomial fit (in log-space) to all the dots simultaneously. The distributions are non-Gaussian in their tail, the curves deviating from parabolas slightly but significantly.

uncertainty at probability 10^{-12} is likely $\lesssim 1$ in Levenshtein distance, as estimated by the robustness of the curves as details of the fitting procedure are varied. Supporting Information S2 gives details of the polynomial fit shown in the figure. Also, we have verified that these results are not significantly altered by choosing otherwise random codewords that satisfy constraints on CG content and homopolymer avoidance (never a large fraction of random draws).

2.3 Distribution of closest noncausal distance to a set of N codewords

Given a set of N random codewords $\{C\}$ of length M from which a given read R does not derive, what is the probability $P(L)$ that R's smallest Levenshtein distance to the set is L? We may assume that R is itself (close to) random, because it derives from errors on a random starting point, its true codeword. Given one of the distributions in Fig. 1, which we now denote $p(L|M)$, this is a straightforward calculation in extreme value theory (38): The probability $P(0)$ is the cumulative Poisson probability of one or more zero distances when the mean number is $Np(0|M)$,

$$P(0) = \text{Poisson}\{\geq 1, Np(0|M)\}, \tag{1}$$

Then, recursively,

$$P(i+1) = \left(1 - \sum_{j=0}^{i} P(j)\right) \text{Poisson}\{\geq 1, Np(i+1|M)\}, \tag{2}$$

where the term in parentheses is the remaining probability to be allocated, and the Poisson cumulative distribution function is the probability of allocating it to the value $i+1$. Eq. (2) is easily computed numerically and is shown for the case of $M = 34$ nt in Fig. 2. The values near the peaks seem haphazard due to discreteness effects, but are accurately shown.

The figure also plots the cumulative distribution functions for binomial deviates with parameters 34 (the codeword length) and probabilities 0.05, 0.10, and 0.20. These model, at least crudely, the Levenshtein distances to be expected in the causal case of comparison to the correct codeword. That fact that some orders of magnitude of vertical white space lie between each green curve and at least one other-colored curve points the way forward: by picking an appropriate threshold Levenshtein distance T, calling as decodes all reads with $\leq T$ and as erasures all reads with $>T$, we may hope to achieve both very high accuracy (high precision) on decodes and a very low erasure rate (high recall). The figure demonstrates this in an approximate, but relatively model-independent way. In the results, we will explore a more accurate, detailed model and, importantly, will give a procedure for choosing T based on observed data.

Supporting Information S3 shows figures analogous to Fig. 2 for the cases of other codeword lengths, $M = 30$, 26, 22, and 18 nt.

2.4 Three-parameter Poisson error model for substitutions, insertions, and deletions

An error model for a M-mer barcode set can be described by three parameters, p_{sub}, p_{ins}, and p_{del}, respectively, the probabilities per nucleotide of a substitution, insertion, or deletion error. Formally, we need to be more precise: the different types of errors can interact, and insertion and deletion errors change

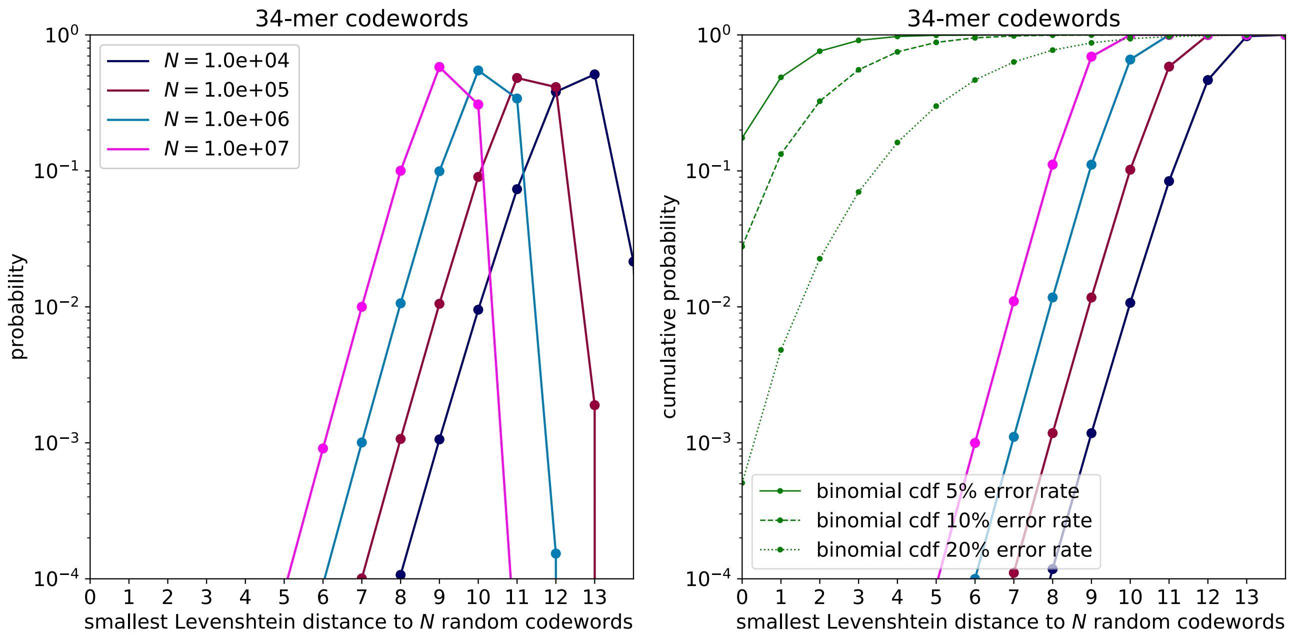

Fig. 2. Probability of the smallest distance to a set of *N*-34 nucleotide random codewords. Left: probability mass function. The larger is *N*, the smaller is the expected distance to a given garbled read by chance. Right: cumulative distribution function. Also shown as thin green lines are the cumulative binomial probabilities for the number of errors in a garbled 34-mer for the large error rates (per nucleotide) 5%, 10%, and 20%.

the length of the string. Among various equally good possibilities, for purposes of this paper we adopt the following error-generation model, the steps to be executed in the order listed.

- Start with a codeword string in $\{a, c, g, t\}$ of length M, indexed as $0 \ldots M - 1$.
- Substitutions. Draw a deviate $n_{sub} \sim$ Binomial $(M, \frac{4}{3}\mathrm{p_{sub}})$. The factor 4/3 corrects for the fact that 1/4 of substitutions will substitute an unchanged nucleotide. Draw (with replacement) n_{sub} indices in the uniform distribution $U(0,\ldots, M - 1)$. Draw (with replacement) n_{sub} values uniformly in the nucleotides $\{a, c, g, t\}$. Substitute the values at the indices. (Note that indices may collide, in which case only one of the corresponding values will "win" the substitution, it doesn't matter which).
- Deletions. Draw a deviate $n_{del} \sim$ Binomial(M, p_{del}). Draw (with replacement) n_{del} indices in the uniform distribution $U(0,\ldots, M - 1)$. Delete the nucleotides at those positions. Here, colliding indices delete the same position only once. Call the new string length $M' \geq M - n_{del}$ with equality in the case of no collisions.
- Insertions. Draw a deviate $n_{ins} \sim Binomial(M', p_{ins})$. Draw (with replacement) n_{ins}indices in the uniform distribution $U(0,\ldots, M')$. Draw (with replacement) n_{ins}values uniformly in the nucleotides $\{a, c, g, t\}$. Insert each value before the original index position (or, for index M', after the last character). Here, colliding indices result in more than one insertion before an existing character (order irrelevant). The string length is now $M'' = M' + n_{ins.}$
- Padding or truncation. If $M'' > M$, truncate the string to length M. If $M'' < M$ pad the string to length M with random characters in $\{a, c, g, t\}$. The resulting string of length M is the garbled codeword. This padding/truncation implements the worst-case assumption that we have no independent information about where the true barcode begins or ends, but simply attempt to decode exactly M characters at the codeword's nominal position (e.g. beginning of strand).

2.5 Fast triage of codewords by trimer similarity

We are committed to comparing *each* of Q (possibly many millions of) reads to *each* of N (possibly millions) codewords of length M, so as to find, for each read, that codeword with the smallest Levenshtein distance. Theoretical scalings imply that the number of implied operations is const $\times$ Q $\times$ N $\times$ M^2, where the constant is $\sim$10 and the factor M^2 is the Levenshtein calculation. While feasible on a supercomputer, the implied $\sim 10^{16}$ operation count is not to be recommended. Here, we show how to reduce it to $\sim 10^{13}$ operations that can be done on a commodity GPU with 10^3 to 10^4 parallelism, implying as few as $\sim 10^9$ calculation steps, feasible on a single-head desktop machine.

We will employ a strategy of "triage," that is, comparing each read to every codeword using only an approximate distance metric or similarity score, but one with a very small number of computer operations per comparison. This step will eliminate a large (often very large) fraction of possible identifications. Then, it is feasible to apply a more exact comparison to the small number of possible identifications that remain—either as a secondary triage (another approximate distance measure) or a true Levenshtein calculation. The final step will always be a Levenshtein (or similar) distance comparison, finding the decoding with the smallest true distance.

Primary triage by trimer Hamming popcount

Every codeword of length M has exactly $M - 2$ overlapping, consecutive trimers in $\{a, c, g, t\}^3$ (a set of cardinality $4^3 = 64$). Why focus on trimers? Why not dimers or tetramers? The results of §2.3, especially in Fig. 2 and Fig. S2 (right panels), suggest the need for barcodes of length $\sim$30. On average, the $4^2 = 16$ dimers will appear in a barcode about twice, and $\gtrsim$80% will occur at least once. So there is relatively little information in either their uniqueness of occurrence or uniqueness of position. For tetramers, which number $4^4 = 256$, only $\sim$10% of them will appear in any given barcode, so $\sim$90% of the effort of keeping track of them is wasted. Trimers, each appearing on the order of once per barcode, are a unique sweet spot.

See Supporting Information S4 for more accurate statistics of this kind.

A suitable function $B(C_i)$ maps each codeword C_i to a 64-bit unsigned integer whose bits signify the presence (1) or absence (0) of each trimer in C_i The values $B_i = B(C_i)$ can be precomputed. Now, for each garbled M-mer read R_j, we compute the distance measure

$$S = \text{Popcount}(B(R_j) \oplus B_i), \tag{3}$$

where $\oplus$ denotes bitwise exclusive-or and Popcount returns the number of set bits in a word, here the Hamming distance. Popcount is a single machine-language CUDA instruction on GPUs (39) that can readily be made accessible to PyTorch, or calculated (a few times slower) as a few-line PyTorch (40) function. Importantly, in either case, the calculation of Eq. (3) can be done in parallel across all N of the B_i's simultaneously. We may then eliminate from further consideration those codewords with the largest distances S, between 90% and 99%, depending on the DNA error rate (see further details below).

Secondary triage by trimer position correlation

Conceptually, a secondary triage should need to be calculated for each read only for the list of codeword candidates that survive the primary triage. The output of the secondary triage would be an even shorter list of survivors. In practice, our proposed secondary triage is almost as fast as the above primary triage. That being the case, it is about equally efficient to apply the primary and secondary triages simultaneously to all the codewords, and then combine the triages, as will be described below. This strategy allows us to then jump directly to a Levenshtein comparison of the joint triage survivors.

Our secondary triage is motivated as follows: To be close in distance, a read R_j and codeword C_i should not only be similar by set-comparison of their trimers (popcount test above), but also close in the position indices, $0, \ldots, M-3$, of identical trimers.

Denote individual trimers as $t \in [0, 64)$, and denote the ordered sequence of trimers in a read or codeword as $t_i, i = 0, \ldots, M-2$. Let $V(R)$ be a function returning an integer vector of length 64, defined for a codeword or read R by the 64 components,

$$V(R)_t = \begin{cases} i, & \text{if } t_i \text{ occurs in R} \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

This is not quite a well-posed definition, because we might have $t_i = t_j$ for $i \neq j$, i.e., a collision in $V(R)_t$. Supporting Information S5 discusses how collisions can be resolved in a computationally fast manner.

Now, the dot product $V(R_j) \cdot V(C_i)$, something like an unnormalized correlation of the two position functions, can be taken as a similarity measure. Since $V(R_j)$ and $V(C_i)$ can be precomputed, the dot products over all i and j can all be done in parallel on the GPU, exactly the kind of tensor calculation it is best at.

But why stop there? For any kernel function K that acts componentwise on a vector, the dot product $K(V(R_j)) \cdot K(V(C_i))$ is also a similarity measure. Multiple K's return different similarity information. We find that kernels of cosine shape,

$$K_n(k) \equiv \cos\left[\pi k n/(M-1)\right], \quad n = 1, 2, \ldots, \quad k = 0, 1, \ldots, M-1, \tag{5}$$

(with small n being the desired higher harmonic of the cosine), each give good results, even better when combined as next described. The intuition here is that values $n > 1$ are more sensitive to the ordering of trimers on finer scales, but only up to some value of n where indels result, on average, in a loss of phase coherence with the cosines. We find that $1 \leq n \leq 4$ works well, with larger values giving little improvement.

Combined triage

Although operations across all pairs of reads and codewords are by definition expensive, we have found it efficient to expend the cost of ranking (i.e. sorting) each read's N distance scores (against every codeword) for the handful of distance measures, Eqs. (3) and (5) (with $n = 1, 2, 3,$ and 4). Let $r(i, n)$ denote the rank of the ith codeword in the nth distance measure, small ranks meaning most similar. Then we define the combined distance measure $r(i)$ as the product

$$r(i) \equiv \prod_n r(i, n).$$

This can be viewed as akin to a naive Bayes estimate, since $r(i, n)/N$ is something like a Bayes evidence factor provided by the distance measure n. Finally, we rank the $r(i)$s.

Fig. 3 shows results for a simulation with $N = 10^6$ codewords of length $M = 34$ whose reads are corrupted (using the error model describe above) with $p_{sub} = p_{ins} = p_{del} = 0.03$, a total error rate of 9%. One sees that, here, the Hamming popcount is doing most of the heavy lifting, but combining with position similarity gives a substantial improvement. In the figure, "cos$n$" denotes the kernel functions in Eq. (5). While these have very similar performances individually, the elimination of any of them decreases the combined performance somewhat.

In this example, triage from 10^6 down to 10^3 codeword possibilities for each read would capture the correct answer almost always ($\gg$99%) so that the exact Levenshtein calculation could be done on only the smaller set, at negligible computational cost.

A somewhat less favorable, but still very feasible, example is for the large error rates $p_{sub} = 0.05$, $p_{ins} = 0.05$, and $p_{del} = 0.10$, as shown in Fig. 4. Here, triage from 10^6 to 10^5 produces negligible loss of recall. We will see in the "Results" section that the parallel computation of Levenshtein on 10^5 codewords per read is also very feasible.

3 Results

Illustrating the use and practicality of the above methods, we here give the results of detailed simulations for the case of one million barcodes ($N = 10^6$) of length $M = 34$ nucleotides in the presence of end-to-end DNA total error rates of 20% per nucleotide (base case) with excursions to smaller (9%) and larger (30%) rates. These rates are intentionally chosen to be all very large as compared to next-generation NGS error rates, and even large or very large as compared to third-generation TGS rates (see the Section 1). For some simulations, we assume equal error rates ($p_{sub} = p_{ins} = p_{del}$), while for others we take $p_{sub} = p_{ins} = 0.5 \times p_{del}$, in deference to experimental results in which p_{del} dominates (27,28,41). We know of no previously proposed barcode sets capable of success with these parameters at plausible computational workloads.

3.1 Precision and recall

It is important to emphasize that the methods of this paper do not give either perfect precision, that is, the correct decoding of *every* garbled read independent of its number of errors, nor perfect recall, that is, no garbled reads rejected as undecodable erasures. Rather, by choice of an integer threshold Levenshtein distance T, the user may set any desired recall between 0% (all erasures) and

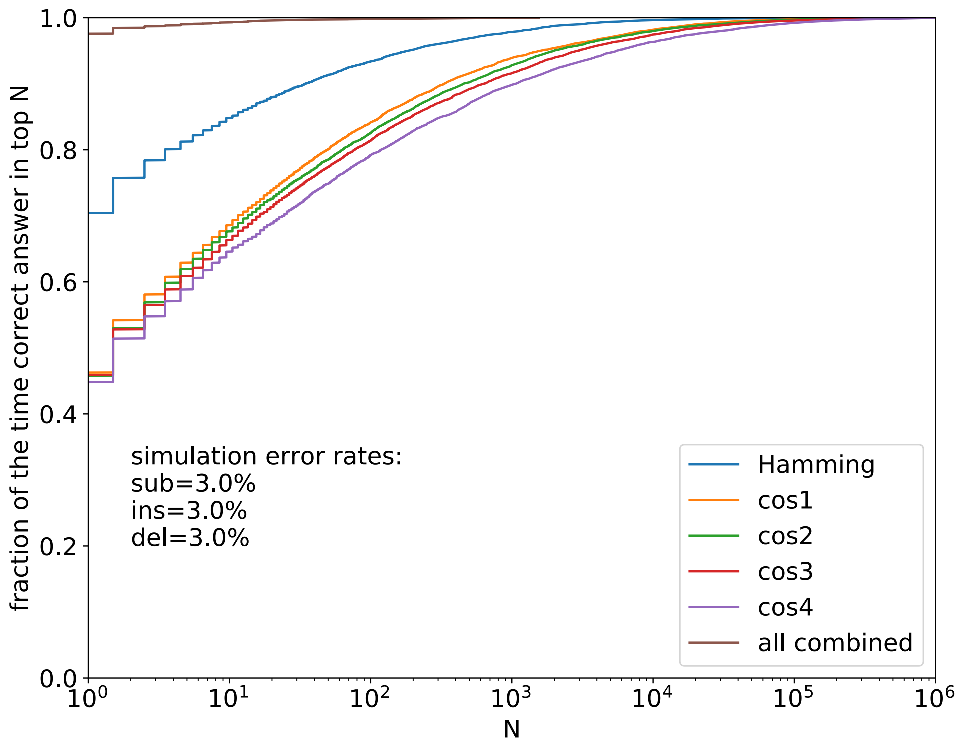

Fig. 3. Triage performance of individual filters and combined. For $N = 10^6$ codewords of length $M = 34$ nt and for an error model with $p_{sub} = p_{ins} = p_{del} = 0.03$, the figure shows the probability of capturing the correct codeword with triage to sets of codeword possibilities much smaller than N. Here, after triage and with negligible loss of recall, exact Levenshtein testing is needed for fewer than 10^3 codewords.

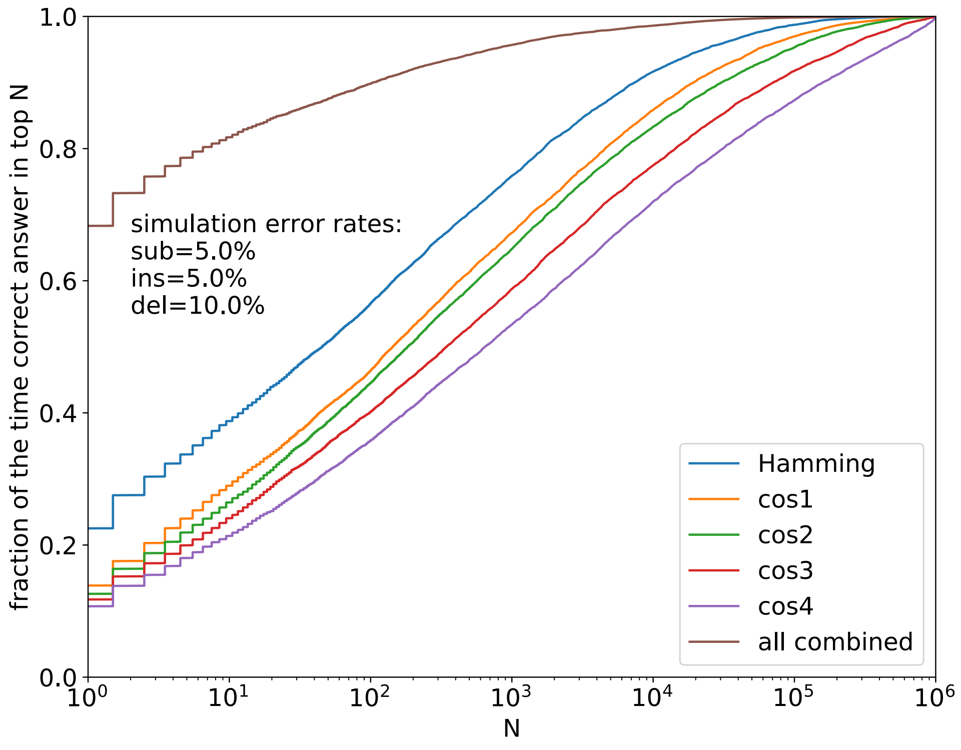

Fig. 4. Same as Fig. 3, but with error rates $p_{sub} = 0.05$, $p_{ins} = 0.05$, and $p_{del} = 0.10$. Here, further testing on about a tenth of codewords is required.

100% (no erasures) and must then accept the implied level of precision.

For these tests, we generated either random sets of codewords, or else otherwise random sets that excluded codewords with homopolymer runs of >3, or CG or AT fraction greater than 0.66. There was no discernible difference in results between same-sized fully random and sequence-constrained codeword sets.

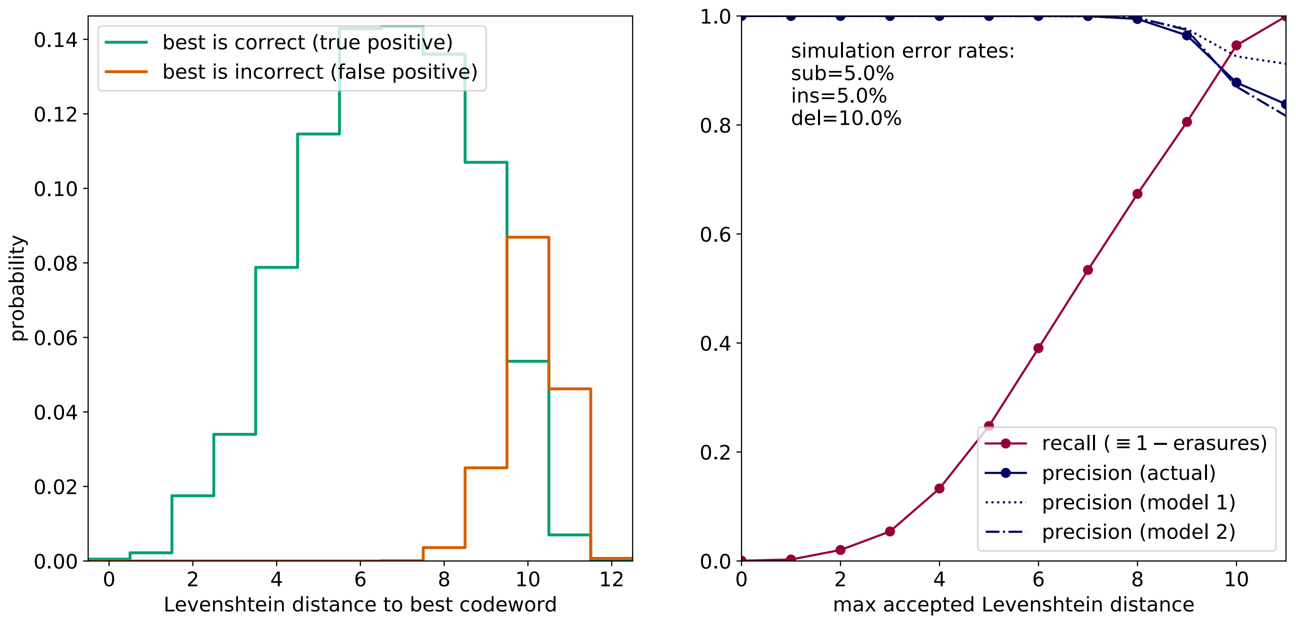

Fig. 5. Simulation with 20% DNA error rate. Left: distribution of distances seen when decoding to the Levenshtein-closest codeword among 10^6 possibilities. The experimenter, not knowing, which decodes are correct, sees the sum of the red and green histograms. Right: with data from the left panel, for each choice of threshold T, recall is the fraction of all events (green plus red) $\leq T$. This is knowable to the experimenter. Precision is, for events $\leq T$, the fraction of green (versus red) events. This is not directly knowable but can be estimated by the models shown (see text).

Garbled reads were assigned to the closest codeword by Levenshtein distance when the distance was $\leq T$, otherwise called as erasures. For the base case of 20% total errors, with $p_{sub} = 0.05$, $p_{ins} = 0.05$, and $p_{del} = 0.10$, Fig. 5 shows results for the full range of choices of T. In a real experiment, the user does not know which decodes are correct, so sees the sum of true and false positives (green and red histograms). That is enough to calculate the recall for each possible value of T, but not the precision, which requires "knowing the answers".

However, the user does know that there is some red histogram whose expected shape was already calculated above in Eqs. (1) and (2) and Fig. 2. The user also knows that the green histogram should be roughly binomial, but "censored" by the red histogram in a computable way. In Supporting Information S6, we show that this is enough information to model the expected precision function either naively (shown in Fig. 5 as Model 1) or, with additional assumptions, somewhat more accurately (shown as Model 2). So, in practice, the user can use these models to choose an appropriate value T. In the figure, a suitable choice based only on the models might be $T = 8$, which in simulation gives 99.6% precision with 67% recall. Whether this is a sufficiently large recall to be useful depends on the design of the experiment, for example, whether a given barcode is expected to be read several or many times, in which case a 33% loss to erasures can be tolerable.

Supporting Information S7 shows the analogous figures for error rates of 10% (with $p_{sub} = 0.033$, $p_{ins} = 0.033$, and $p_{del} = 0.033$) and 30% (with $p_{sub} = 0.075$, $p_{ins} = 0.075$, and $p_{del} = 0.15$). For the former of these, the choice $T = 9$ yields precision 99.9% with recall 98.8%. For the latter, recall must be sacrificed to get good precision. $T = 7$ gives precision 99.8% with recall 20.4%, while $T = 8$ gives precision 98.2% with recall 32.6%.

The user is assumed to know something about the experimental DNA error rate a priori. However, if this is not the case, then the above values can be assumed as lower bounds. Specifically, for any assumed total error rate significantly less than 30%, the value $T = 8$ should give >99% precision along with a recall that will be immediately known from the data, by the number of erasures called.

3.2 Performance and cost

The minimum requirement for using the methods described in this paper is a compute node (or cloud instance) with at least 2 CPU cores and at least 1 commodity- or server-grade GPU having at least 8 GB memory. To use exactly our code (as available on Github), PyTorch (40) and its associated software tool chain is required, but porting to other CUDA tool chains (e.g. TensorFlow (42)) should be straightforward.

We measured actual performance on a standalone workstation with an Intel i9-10900 processor (10 CPU cores and 20 logical processors) and two Nvidia RTX 3090 GPUs, each with 10,496 CUDA cores and 328 tensor cores. The purchase price of this machine (year 2022) was US$12,000. The (year 2022) marginal cost of adding additional comparable GPUs would be about US$1,500 each.

As a typical performance test, we generated 1,000,000 simulated reads of 34-nt random barcodes with 20% error rates. (Performance does not actually depend on error rate.) Reads were divided among processes running concurrently on separate CPU cores. On the above machine, we found that four such processes, two assigned specifically to each GPU, gave the best performance, saturating the two GPUs (and four CPUs) at close to 100% usage. Memory usage per GPU was 7.4 GB. Wallclock execution time was 4943 seconds, implying about 17.5 million reads per 24-hour day. This is likely adequate performance for many applications and will only improve with time as GPU cycles get faster and cheaper.

For applications requiring greater throughput, there are various options: Academic supercomputer centers allocate time (at no cost) competitively to academic users. A current example is the Longhorn computer at the Texas Advanced Computing Center (TACC) (43) with 384 Nvidia V100 GPUs, implying on the order of 7 billion reads per day for the full machine. The "startup" allocation of 100 node hours should process on the order of 150 million reads, and much larger allocations are routinely awarded. Alternatively, commercial cloud instances of GPUs can be stood up by the hour in any desired quantities and thus any desired throughput. Current (2022 (31,32)) prices of about US$ 0.50 per GPU-hour imply a cost of about US$ 1.50 per million reads processed. This can be compared to the 3-year amortized cost of the standalone machine described above implying about US$ 0.60 per million reads; or the amortized marginal cost of each additional GPU, which implies about US$ 0.15 per million reads.

4 Discussion

The main point of this paper is demonstrating by simulations the practicality of all-to-all comparisons for the closest Levenshtein or Needleman–Wunsch match (that is, comparing all reads to all barcodes) with DNA barcodes sets of 10^6 barcodes or larger and for reads numbering many millions or more. The elements that make this possible are (1) the parallel processing capabilities of current commodity GPUs, (2) the use of a novel, very fast parallel triage that, for each read, eliminates from competition all but a small fraction of candidate barcodes, and (3) the ability to parallelize the Levenshtein or Needleman–Wunsch computation to a significant degree, both within a single calculation and across many such.

All-to-all comparison in turn makes practical the use of random barcode sets (defined and fixed for each experiment) that derive error-correcting capability simply by the statistics of their average distances from one another. While the required lengths $\sim$30 nucleotides may be undesirably long for use with short read lengths, they are not a detriment with read lengths of third-generation sequencing. And, in that context, the ability to use of direct, parallel Levenshtein (or an even faster approximation as discussed), allows as many as 6 to 8 errors to be corrected (set above by the threshold value T), along with correctly flagging as undecodable "erasures" reads with more than this number. At 10% errors per nucleotide, considered a large value, we are able to demonstrate precision of 99.9% and recall of 98.8%. Even with 20% errors per nucleotide, we demonstrate 99.6% precision with recall of 67% (meaning that at most 1/3 of reads are wasted). We know of no other existing, practical DNA barcode methodologies that are able to operate in these high-error-rate regimes with $\gtrsim 10^6$ barcodes. In these statistics, errors in barcode synthesis ("wrong barcodes") are as equally correctable as errors created at later stages of an experiment or during final sequencing. High-density DNA microarrays are an example of an application where error rates a high as $\sim$ 15% are reported, due to position-dependent synthesis errors in the array (41).

In contrast to this paper, mathematically constructed error-correcting codes (ECCs) of a given length L are designed to have fewer near-collisions than our random barcodes of the same length. If there existed known mathematical ECCs capable of (i) correcting as many as 6 to 8 errors, and (ii) correcting not just substitution errors, but also insertions and deletions (indels), then these would be superior to random barcodes. But we know of no such ECCs (27).

Acknowledgments

I thank John Hawkins for valuable discussions and thank Anshuman Suri for contributing a useful piece of CUDA coding (providing PyTorch access to the CUDA popcount primitive). Ilya Finkelstein and Jim Rybarski made useful comments on the manuscript.

Supplementary Material

Supplementary material is available at *PNAS Nexus* online.

Funding

This work was supported by a Warren J. and Viola Mae Raymer Fellowship at The University of Texas at Austin.

Author Contributions

WHP designed research, performed research, and wrote the paper.

Data Availability

Python and PyTorch code for all the computations in this paper are available on Github at https://github.com/whpress/RandomBarcodes.

References

1. Eroshenko N, Kosuri S, Marblestone AH, Conway N, Church GM. 2012. Gene assembly from chip-synthesized oligonucleotides. Curr Protoc Chem Biol. https://doi.org/10.1002/9780470559277.ch110190.
2. Plesa C, Sidore AM, Lubock NB, Zhang D, Kosuri S. 2018. Multiplexed gene synthesis in emulsions for exploring protein functional landscapes. Science. 359(6373):343–347. DOI: 10.1126/science.aao5167.
3. Fan R, *et al.* 2008. Integrated barcode chips for rapid, multiplexed analysis of proteins in microliter quantities of blood. Nat Biotechnol. 26:1373–1378.
4. Ma C, *et al.* 2011. A clinical microchip for evaluation of single immune cells reveals high functional heterogeneity in phenotypically similar T cells. Nat Med. 17:738–743.
5. Zimmermann G, Neri D. 2016. DNA-encoded chemical libraries: foundations and applications in lead discovery. Drug Discov Today. 21:1828–1834.
6. Melkko S, Scheuermann J, Dumelin CE, Neri D. 2004. Encoded self-assembling chemical libraries. Nat Biotechnol. 22:568.
7. Satz AL, Brunschweiger A, Flanagan ME, *et al.* 2022. DNA-encoded chemical libraries. Nat Rev Methods Primers. 2:3.
8. Klein AM, *et al.* 2015. Droplet barcoding for single-cell transcriptomics applied to embryonic stem cells. Cell. 161:1187–1201.
9. Macosko EZ, *et al.* 2015. Highly parallel genome-wide expression profiling of individual cells using nanoliter droplets. Cell. 161:1202–1214.
10. Zheng GXY, *et al.* 2016. Haplotyping germline and cancer genomes with high-throughput linked-read sequencing. Nat Biotechnol. 34:303–311.
11. Kitzman JO. 2016. Haplotypes drop by drop. Nat Biotechnol. 34:296–298.
12. Haque A, Engel J, Teichmann SA, Lönnberg T. 2017. A practical guide to single-cell RNA-sequencing for biomedical research and clinical applications. Genome Med. 9:75.
13. Zilionis R, *et al.* 2017. Single-cell barcoding and sequencing using droplet microfluidics. Nat Protoc. 12:44–73.
14. Spies N, *et al.* 2017. Genome-wide reconstruction of complex structural variants using read clouds. Nat Methods. 14:915–920.
15. Press WH, Hawkins JA, Jones SK, Schaub JM, Finkelstein IJ. 2020. HEDGES error-correcting code for DNA storage corrects indels and allows sequence constraints. PNAS. 117(31): 18489–18496.

16. Meiser LC, Nguyen BH, Chen Y, *et al.* 2022. Synthetic DNA applications in information technology. Nat Commun. 13:352.
17. Anonymous. 2017. An introduction to Next-Generation Sequencing Technology. Technical report, Illumina, Inc., at https://www.illumina.com/content/dam/illumina-marketing/documents/products/illuminasequencing introduction.pdf.
18. Peterson WW, Weldon EJ. 1972. Error-correcting Codes. MIT Press. Cambridge, Massachusetts, USA.
19. MacWilliams FJ, Sloane NJA. 1977. The Theory of Error-correcting Codes. Elsevier. New York, NY (USA).
20. Lyons E, Tremmel G, Sheridan P, Miyano S, Sugano S. 2017. Large-scale DNA barcodeg generation for biomolecule identification in high-throughput Screens. Sci Rep. 7:13899.
21. Erlich Y, Zielinski D. 2017. DNA Fountain enables a robust and efficient storage architecture. Science. 355: 950–954.
22. Levenshtein VI. 1966. Binary codes capable of correcting deletions, insertions, and reversals. Soviet Physics Doklady. 10:707–710.
23. Costea PI, Lundeberg J, Akan P. 2013. TagGD: fast and accurate software for DNA tag generation and demultiplexing. PLOS ONE. 8:e57521.
24. Buschmann T, Bystrykh LV. 2013. Levenshtein error-correcting barcodes for multiplexed DNA sequencing. BMC Bioinformatics. 14:272.
25. Bystrykh LV., 2012. Generalized DNA barcode design based on Hamming codes. PLoS ONE. 7(5):e36852.
26. Lyons E, Sheridan P, Tremmel G, *et al.* 2017. Large-scale DNA barcode library generation for biomolecule identification in high-throughput screens. Sci Rep. 7: 13899.
27. Hawkins JA, Jones SK, Finkelstein IJ, Press WH. 2018. Indel-correcting DNA barcodes for high-throughput sequencing. PNAS. 115(27):E6217–E6226.
28. Weirather JL, *et al.* 2017. Comprehensive comparison of pacific biosciences and oxford nanopore technologies and their applications to transcriptome analysis. F1000Research. 6:100.
29. Delahaye C, Nicolas J. 2021. Sequencing DNA with nanopores: troubles and biases. PLoS ONE. 16(10):e0257521.
30. Ezpeleta J, Garcia Labari I, Villanova GV, *et al.* 2022. Robust and scalable barcoding for massively parallel long-read sequencing. Sci Rep. 12:7619.
31. Google, Inc, Cloud GPUs. https://cloud.google.com/gpu. [accessed 2020 Jun 06].
32. Lambda Labs, Inc, Lambda GPU Cloud for Deep Learning. https://lambdalabs.com/service/gpu-cloud. [accessed 2020 Jun 06].
33. Laehnemann D, Borkhardt A, McHardy AC. 2016. Denoising DNA deep sequencing data—high-throughput sequencing errors and their correction. Briefings Bioinf. 17(1):154–179.
34. Needleman SB, Wunsch CD. 1970. A general method applicable to the search for similarities in the amino acid sequence of two proteins. J Mol Biol. 48(3):443–453.
35. Navarro G. 2001. A guided tour to approximate string matching. ACM Computing Surveys. 33(1):31–88.
36. Chvàtal V, Sankof D. 1975. Longest common subsequences of two random sequences. J Appl Probab. 12:306–315.
37. Majumdar SN, Nechae S. 2005. Exact asymptotic results for a model of sequence alignment. Phys Rev E. 72:020901.
38. Castillo E. 2004. Extreme Value and Related Models with Applications in Engineering and Science. Wiley-Interscience(Hoboken, New Jersey, USA)
39. Nvidia, Inc, CUDA Toolkit Documentation: CUDA Math API, Integer Intrinsics. https://docs.nvidia.com/cuda/cuda-math-api/group_CUDA_MATH_INTRINSIC_INT.html. [accessed 2020 Jun 06].
40. PyTorch org. https://pytorch.org/. [accessed 2020 Jun 06].
41. Lietard J, *et al.* 2021. Chemical and photochemical error rates in light-directed synthesis of complex DNA libraries. Nucleic Acids Res 2021. 49(12):6687–6701.
42. TensorFlow org. https://www.tensorflow.org/. [accessed 2020 Jun 06].
43. Texas Advanced Computer Center, LONGHORN: GPU accelerated workloads. https://www.tacc.utexas.edu/systems/longhorn. [accessed 2020 Jun 06].