Supporting Information for “Fast trimer statistics facilitate accurate decoding of large random DNA barcode sets even at large sequencing error rates.”

William H. Press1,*

1 Department of Computer Science and Department of Integrative Biology, The University of Texas at Austin, Austin, TX

* wpress@utexas.edu

S1. Parallel Computation of Needleman-Wunsch and Levenshtein Distances

These distances are conventionally calculated by dynamic programming on a Cartesian tableau, filling in squares from top to bottom and left to right (see Wikipedia, “Needleman-Wunsch algorithm”). Mapping this to the CUDA strided-slice tensor programming model on a GPU is facilitated by tilting the tableau 45°, thus displaying the desired calculation as a parallel calculation on a top-to-bottom directed acyclic graph, shown as the red grid in Figure 1. Pink dots on the blue grid are filled with multiples of the skew penalty. The Boolean tensor \([A_i == B_j]\) is calculated in parallel on the red grid, then mapped to the blue. The calculation then proceeds top to bottom on the blue grid filling in the green dots. Some parallelism is generated by doing each row’s green dots in parallel (shown in the Figure as at most three, but actually as many as \(M - 1\) for \(M\) nt codewords. A much larger parallelism is achieved by “stacking” 10,000 to 50,000 such grids out of the plane, each with the same string A (shown as \(A_1, A_2, A_3\)) but different strings B (shown as \(B_1, B_2, B_3, B_4\)). PyTorch code for this algorithm is included on GitHub.

Approximate Levenshtein Referring to the red tableau in the figure, and to the sub, ins, del arrows, a parallel calculation can be done for substitutions and insertions a full (red) row at a time, but not for deletions, because the value in an earlier column can affect a later one. Alternatively, a fully parallel calculation a column at a time is possible for substitutions and deletions, but not insertions.

That said, we can ask what happens if we ignore this reality and just do the fully parallel calculation? The answer will be wrong only slightly and only when there are two or more consecutive deletions (if processing by rows) or insertions (if processing by columns). Moreover, if we do the parallel deletion (insertion) step twice, literally repeating the same one line of code, then the answer will be wrong only when there are three or more consecutive deletions (insertions), a relatively rare occurrence.
Figure S1. One-to-many parallel Needleman-Wunsch or Levenshtein distance calculations. The red grid is largely conceptual. The main calculation is done in parallel on a stack of many blue grids, from top to bottom in the figure. Green dot values are filled with the minimum among del, sub, and ins, i.e., the shortest path to that dot. The stack of answers appears as the blue dot at the bottom.

This parallel “approximate Levenshtein” calculation is found to be several times faster on a commodity GPU than the parallel exact Levenshtein calculation. Approximate Levenshtein can be used as a “tertiary triage” (in the language of section 2.5), bringing the computational burden of exact Levenshtein down to almost negligible. In fact, in the simulations that we have tried, the use of approximate Levenshtein alone, without any other followup, gives results for recall and precision (section 3.1) that are virtually indistinguishable from those where exact Levenshtein is used. Since the goal is correct decodes, not exact Levenshtein distances, the use of approximate Levenshtein seems justified by the performance gain.

S2. Fitted Polynomial Expression for Levenshtein Distance Distribution of Random Strings

Figure 1 in the main text indicated by dots the values actually obtained by simulation, which have probabilities as small as \( \lesssim 10^{-7} \). For smaller probabilities, we need to extrapolate. Rather than fit each value \( M \) separately, which would allow extrapolation on Levenshtein distance \( L \), but not interpolation on codelengths \( M \), we fit a bivariate polynomial,

\[
\log_{10} p(L|M) \approx \sum_{i=0}^{I} \sum_{j=0}^{J} c_{ij} M^i L^j
\]

Here \( p(L|M) \) is the probability that two random \( M \) nucleotide strings are separated by a Levenshtein distance \( L \). Bivariate fitting also acts to improve the accuracy, because an improbably deviant small sample in the tail of one \( M \) value is mitigated by the other \( M \) values.
The coefficients $c_{ij}$ for the adopted best fit are,

$$\log_{10} p(L|M) =$$

$$[-1.347e+01 + 5.668e+00 L - 4.964e-01 L^2 + 1.969e-03 L^3]$$

$$+ [+9.239e-01 - 4.846e-01 L + 5.215e-02 L^2 - 6.125e-04 L^3] M$$

$$+ [-5.333e-02 + 1.803e-02 L - 1.876e-03 L^2 + 2.902e-05 L^3] M^2$$

$$+ [+6.412e-04 - 2.185e-04 L + 2.239e-05 L^2 - 3.940e-07 L^3] M^3$$

S3. Distribution of Closest Non-Causal Distance to a Set of $N$ Codewords for Other Nucleotide Lengths

See main text Figure 2, which was for the case of $M = 34$-mer codewords. Here are analogous figures for $M = 30, 26, 22,$ and 18.

Figure S2. More Smallest Levenshtein Distances to Random Sets of Codewords
The diagrams illustrate the cumulative probability of the smallest Levenshtein distance to \( N \) random codewords for 26-mer and 22-mer codewords. The plots show different error rates for the binomial cumulative distribution function (CDF) with error rates of 5%, 10%, and 20%. The graphs are labeled with different colors for each error rate, indicating the cumulative probability across various distances. The x-axis represents the smallest Levenshtein distance, while the y-axis shows the probability on a logarithmic scale.
S4. How Often Do Di-, Tri-, and 4-mers Occur?

The main text discusses that trimers are preferred because they most occur once, with few occurring two or more times, in a code of length 30. The calculation of how often different mers occur is not straightforward combinatorics, because the mers overlap. However, it is an easy thing to determine by simulation, with results shown in the figure below.

Figure S3. Fraction of di, tri, or 4-mers out of (respectively) 16, 64, and 256 that occur different numbers of times in a random 30-mer code.
S5. Collisions in the Trimer Position Function

We noted above that equation 4 might not define a single-valued equation for the positions of each trimer, because a specific trimer ("cgt" for example) might occur in more than one position. In practice, it is not too bad to pick any one position, randomly. Such a function \( V(R)_i \) returns a (any) position in the codeword for each of 64 trimers \( t_i \), defining the position to be zero if the trimer does not occur in the read \( R \).

Even better results are obtained by combining all positions \( \{i\} \) of a given trimer \( t \) causally into some kind of pseudo-position. This can be done either before or after applying the kernel \( K() \) in equation 5. After some trial and error among alternatives, we replace the components of the vector \( K(V(R)_j) \), that appears in the dot product, by

\[
K(V(R)_j) \implies \sum_{j \text{ s.t. } t=j} K(V(R)_j)
\]

that is to say, we sum the colliding kernels (in our case, cosines) before taking the dot product. The sums can be done in parallel by a scatter-add operation.

S6. Models for Estimating Precision

Here we develop two models that allow a user to estimate, for codewords of length \( M \), as a function of threshold Levenshtein distance \( T \), the precision of decoded garbled barcode reads. The models can then be used to choose a value \( T \) that appropriately trades off precision and recall. We assume that the user has an estimate for the total error rate \( r \) (or chooses some value as an upper bound).

**Model 1.** For every read, there is a distribution of Levenshtein distances \( L_t \) from its true (causal) codeword that we model as a binomial probability \( \text{binom}(L_t|M, r) \); and there is a distribution \( P(L_f) \) of its distances \( L_f \) from the closest false (non-causal) codeword, which was calculated above in Section 2.3. When we have \( L_t \leq T \) and \( L_t < L_f \) we can score a true positive (TP). For ties \( L_t \leq T \) and \( L_t = L_f \), we resolve the tie randomly and score half a true positive. Conversely, when we have \( L_f \leq T \) and \( L_f < L_t \) we can score a false positive (FP), or half a false positive if \( L_f = L_t \). The remaining case is when \( L_t > T \) and \( L_f > T \), which is an erasure.

Parsing these inequalities in the two-dimensional grid of \( L_t \) and \( L_f \), and with the assumed probability distributions, gives the rates for TP and FP,

\[
\text{TP} = \sum_{i=0}^{T} \text{binom}(i|M, r) \sum_{j=i}^{\infty} P(j)
\]

\[
\text{FP} = \sum_{j=0}^{T} P(j) \sum_{i=j}^{\infty} \text{binom}(i|M, r)
\]

Here \( \Sigma' \) denotes a sum with a factor \( 1/2 \) applied to its first term. In terms of these quantities,

\[
\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}
\]

\[
\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FP}}
\]

\[
\text{erasure rate} = 1 - \text{TP} - \text{FP}
\]

**Model 2.** A weakness is Model 1 is its assumption of a binomial distribution for \( L_t \), when we know this is not correct with indels. Also, Model 1 does not make use of the...
experimentally observable distribution of distances $L$, a mixture of the causal and non-causal distributions,

$$P_{\text{tot}}(L) \equiv \alpha P_c(L) + (1 - \alpha)P(L)$$

(4)

(In Figure 4, this mixture was shown as the green and red histograms.) Implicitly, Model 1 estimated $P_c(L)$ by

$$\widehat{P}_c(L) = \text{binom}(L|M, r) \sum_{j=L}^{\infty} P(j) \quad \text{(censored binomial)}$$

$$\widehat{P}_c(L) = \frac{\sum_{L=0}^{\infty} \widehat{P}_c(L)}{\sum_{L=0}^{\infty} \widehat{P}_c(L)} \quad \text{(then renormalized)}$$

where “hat” denotes an estimator.

For Model 2, we first least-squares estimate $\alpha$ in equation 4 by the formula,

$$\hat{\alpha} = \frac{\sum_{L} |P_{\text{tot}}(L) - P(L)|[\widehat{P}_c(L) - P(L)]}{\sum_{L} [\widehat{P}_c(L) - P(L)]^2}$$

(6)

in terms of which the precision is then estimated by

$$\text{precision} = \frac{TP}{(TP + FP)} = \frac{T}{\sum_{L=0}^{T} \hat{\alpha} \widehat{P}_c(L)} / \frac{T}{\sum_{L=0}^{T} [\hat{\alpha} \widehat{P}_c(L) + (1 - \hat{\alpha})P(L)]}$$

(7)

It is not obvious mathematically that Model 2’s precision estimate must be better than Model 1’s, but in simulation we generally find it to be.

S7. Precision and Recall for Other Simulated DNA Error Rates

These figures are analogous to Figure 3, but for the different DNA error rates 10% and 30%.

Figure S3. Precision and Recall Simulations for Additional DNA Error Rates
Levenshtein distance to best codeword

0.00
0.05
0.10
0.15
0.20
0.25

Probability

best is correct (true positive)
best is incorrect (false positive)

max accepted Levenshtein distance

0.0
0.2
0.4
0.6
0.8
1.0

Simulation error rates:
sub=7.5%
ins=7.5%
del=15.0%

Recall (1 erasures)
Precision (actual)
Precision (model 1)
Precision (model 2)