A k-mer-based maximum likelihood method for estimating distances of reads to genomes enables genome-wide phylogenetic placement.

Ali Osman Berk Şapcı[1] and Siavash Mirarab[1,2*]

[1] *Bioinformatics and Systems Biology Graduate Program, UC San Diego, San Diego, CA 92093, USA*
[2] *Department of Electrical and Computer Engineering, UC San Diego, San Diego, CA 92093, USA*
[*] *Correspondence: smirarab@ucsd.edu*

Abstract

Consider comparing a sequencing read of unknown origin to a set of reference genomes. This problem underlines many applications, including metagenomic analyses. The exact genome generating the read is not in the reference set, but may be evolutionarily related to some references. Ideally, we need not just the identity of the closest references to the read but also their distance to the read. The distances can help us identify the read at the right taxonomic level and, more ambitiously, place it on a reference phylogeny. Aligning reads to reference genomes, the only available approach for computing such distances, becomes impractical for very large reference sets. It is also not effective at higher distances when used with efficient indexes (e.g., Bowtie2). While k-mers can create scalable indexes, existing k-mer-based methods are incapable of distance calculation. Thus, estimating distances between short reads and large, diverse reference sets remains challenging and seldom used. We introduce a method called krepp that combines four ideas to solve this challenge and to further enable placing reads on a reference phylogeny. We use *i*) locality-sensitive hashing to find inexact k-mer matches, *ii*) a phylogeny-guided colored k-mer index to map each k-mer to all references containing it, *iii*) a maximum likelihood framework to estimate read-genome distances using k-mer matches, and *iv*) an extension of distances to clades of the reference tree, which enables placement using a likelihood ratio test. We show that krepp matches true distances using a fraction of time compared to alignment, extends to higher distances, and accurately places short reads coming from any part of the genome (not just marker genes) on the reference phylogeny. We demonstrate that krepp easily extends to databases with tens of thousands of reference genomes and performs well in characterizing real microbial samples.

Availability: The tool is available at `github.com/bo1929/krepp`. All results, auxiliary data, and scripts used in the analyses can be found at `github.com/bo1929/shared.krepp`.

Keywords: k-mers, Average nucleotide identity, Phylogenetic placement, Metagenomics

1 Introduction

Comparing short reads of an unknown origin against an evolutionary diverse set of reference genomes is needed in several applications, such as metagenomics [1] and contamination detection [2]. The references themselves are related evolutionarily, and taxonomic or phylogenetic trees can be used to model those relationships. Since a read is never sampled from a genome present in the reference set (even if the same species or subspecies is present), it is often insufficient to assign a read to a reference genome; instead, we need to characterize it relative to all references. However accomplished, this goal can be thought of as having two components: Quantifying how close a read is to each reference and using the result to place the read in the taxonomic or phylogenetic context.

The phylogenetic tree always has a higher resolution than the taxonomy. For example, the WoL-v2 [3] phylogeny has 15,246 internal nodes compared to 3,755 in its taxonomy. A phylogeny also provides interpretable branch lengths. The phylogenetic placement of metagenomic reads enables many downstream analyses [4], including sample differentiation [5] and UniFrac calculation [6], and has even outperformed *de novo* phylogeny reconstruction in such applications [7]. Thus, the more ambitious goal is to place reads on a reference phylogeny, and the availability of ultra-large reference trees [e.g., 3, 8] has enabled this goal. However, large-scale phylogenetic analyses tend to be more challenging, requiring not just more computation but also more involved pipelines.

Existing methods for metagenomic identification fall into two categories–those that attempt to identify every read [e.g., 9] and those that focus on reads coming from a limited set of markers [10]. Marker-free methods have been content with taxonomic identification (with one exception [11]) because the phylogenetic placement of any read from anywhere in the genome is difficult. Some marker-based methods address the more ambitious goal of phylogenetic placement, often [12–20] but not always [21] by aligning reads to reference alignments. These methods, however, miss out on the vast majority of reads when data are genome-wide (as opposed to amplification or targeted capture of marker genes). Thus, we can *either* use all reads but only get taxonomic labels *or* obtain phylogenetic placement but only for a small fraction of reads. This dichotomy has been a practical necessity due to the lack of genome-wide read placement methods (the only existing method, App-SpaM [11], does not scale, as we will see). Practitioners are left with *ad hoc* solutions such as mapping reads to genomes and assigning reads to tree leaves with the best matches [22].

The overarching goal of this paper is to break this dichotomy and enable genome-wide read placement. Specifically, we address two related problems:

Alignment-free read-genome distance calculation: *Given is a set of query reads of unknown origin and a large and evolutionary diverse set of n reference genomes $\mathcal{R}$. We seek to compute the distance from each read q to each reference genome that is sufficiently close to q.*

Alignment-free marker-free phylogenetic placement of reads: *We are additionally given a phylogeny T leaf-labeled by $\mathcal{R}$. We seek to add each query read to T independently of others.*

We define the read-genome distance as the Hamming distance (HD) between the genome generating q and the reference genome, restricted to the region of the genome from which q is sampled. If the rate of evolution was fixed across the genome, this would be $1-$ANI, but since rates vary across the genome, we expect the read-genome distances to match $1-$ANI only in expectation. The best existing method to compute the read-genome distance is aligning q to reference genomes. Efficient methods exist for aligning q to many genomes (e.g., bowtie2 [23] and minimap2 [24]), but these do not easily scale to tens of thousands of references and are less effective at high distances ($> 10\%$).

We propose a scalable solution to both problems–a k-mer-based algorithm to compute the distance between a read and relevant reference genomes, and a placement method. The resulting method, krepp ($\underline{k}$-mer-based $\underline{\text{re}}$ad $\underline{\text{p}}$hylogenetic $\underline{\text{p}}$lacement), is scalable to tens of thousands of reference genomes and is accurate both in terms of distances it computes and its placements.

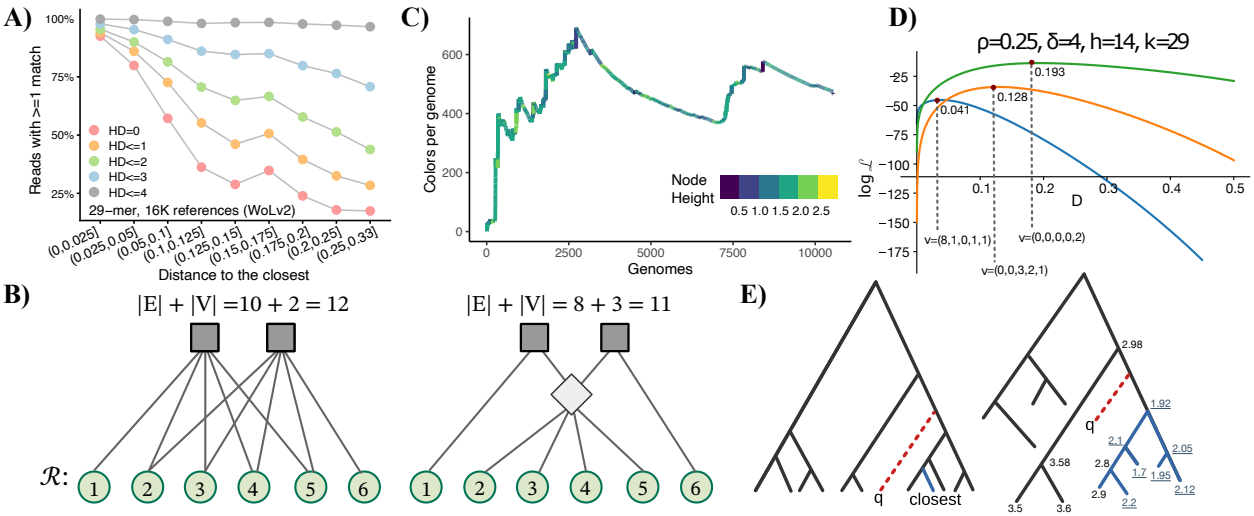

Figure 1: A) As Mash distances to the closest reference (x) increase, few reads have any k-mers that match a reference exactly, but most have some at Hamming distance 4 or less. B) Right: A trivial multitree with two colors (boxes) and six references (circles), forming a bipartite graph. Each color is simply the union of its constituents: $\{1, 2, 3, 4, 5\}$ and $\{2, 3, 4, 5, 6\}$. Left: The multitree could be made smaller (in terms of $|E|+|V|$) by adding a meta-color $\{2, 3, 4, 5, \}$, as shown. C) On WoL-v1 data set ($|\mathcal{R}| = 10,575$), the number of colors per genome as postorder progresses (x-axis). D) The likelihood function and its maxima for three different k-mer match histograms. E) On an ultrametric tree, the query q has the same minimum distances to its sister clade; thus, the choice of the *closest* leaf is arbitrary unless the sister clade is a singleton. On a tree that is sufficiently close to being ultrametric, distances to the sister clade are all similar and may be statistically indistinguishable; placing the query as sister to the largest clade of similarly small distances can find the right placement. Each node is labeled with its χ^2 value according to our likelihood ratio test; all values below 10% significance are indistinguishable (underlined)

$\mathcal{R}$	Set of all references	$\mathcal{R}(x) \subseteq \mathcal{R}$	Genomes that include a k-mer x
T	A tree with $\mathcal{R}$ as leafset	$\mathcal{M}$	All k-mers in $\mathcal{R}$ after minimization
$\delta = 4$	Maximum HD for k-mers	$\mathcal{C} = \{\mathcal{R}(x) : x \in \mathcal{M}\}$	Set of all observed colors
h	Positions sampled for LSH	$\mathbf{A}, \mathbf{C}$	LSH sorted arrays of k-mers and color ids

Table 1: Notations used throughout the paper

2 Methods

The krepp method can 1) compute read-genome distances and 2) place reads on a tree. Distance calculations use three techniques: *i*) A locality-sensitive hashing (LSH) index to enable inexact k-mer matches, which is crucial for computing higher distances (Fig. 1A); *ii*) a new multitree-based k-mer coloring method to map k-mers to $\mathcal{R}$; and *iii*) a maximum likelihood framework for distance calculation from k-mer matching results. Placement uses distances plus three ideas: *i*) An extension of distances to clades of the tree, *ii*) a likelihood-based test of whether two distances are statistically distinguishable given noise, and *iii*) a placement algorithm that finds the largest clade with a distance that is statistically as small as the minimum distance. We next expand on each of these techniques; a summary of all notations introduced below is given in Section 2.

2.1 Locality-sensitive hashing (LSH) index of k-mers

Given a query k-mer, we seek reference k-mers within some HD threshold, denoted by δ. Expanding on the CONSULT family of methods (CONSULT*) [25–27], we use LSH with some changes. We use the bit-sampling LSH index [28] to partition reference k-mers (default: k=29) into subsets. The LSH of a k-mer x, denoted by LSH(x), is computed by sampling $h \ll k$ (default: 14) random but fixed positions of x, providing $[0, 2^{2h})$ buckets indexed by a $2h$-bit integer. For each $r \in \mathcal{R}$, we use minimizers by choosing the k-mer whose encoding has the smallest MurmurHash3 value in a local window (default 35). We save all surviving reference k-mers, denoted by $\mathcal{M}$, in the ascending order of their LSH(x), breaking ties lexicographically. The result is an array $\mathbf{A}$ of size $|\mathcal{M}|$. We build another ordered offset-index array of size 2^{2h}, denoted by $\mathbf{I}$ to note the boundaries of LSH partitions; i.e., $\mathbf{I}[i] = \mathbf{I}[i-1] + |\{x : \text{LSH}(x) = i, x \in \mathcal{M}\}|$ and $\mathbf{I}[-1] = 0$. Thus, unlike CONSULT* methods, which limit the size of LSH buckets by removing k-mers, krepp uses flexible size partitions and keeps all k-mers; this helps (Section 2.3) with defining likelihood. To compute the exact HD for k-mer matches, $\mathbf{A}$ needs to store each k-mer x precisely. Naively, each k-mer requires $2k$ bits; however, the position of x in $\mathbf{A}$, together with offset values in $\mathbf{I}$, already gives the h positions used for LSH(x); we simply store the remaining $2(k-h)$ bits. Thus, $\mathbf{A}$ requires $2(k-h)|\mathcal{M}|$ bits. $\mathbf{I}$ is much smaller and needs $\log_2(|\mathcal{M}|)$ per index and $2^{2h}\log(|\mathcal{M}|)$ bits in total. We adopt a left/right k-mer encoding [25] that enables computing HD with just four instructions (`pop-count`, XOR, OR, shift).

Given a query k-mer x, we only attempt to match it to k-mers with the same LSH value (i.e., $\mathbf{A}[\mathbf{I}[\text{LSH}(x)-1]]$ to $\mathbf{A}[\mathbf{I}[\text{LSH}(x)]]$) by calculating the exact HD. The higher the h is, the smaller these slices tend to become, but with decreased sensitivity (i.e., more false negatives), especially for higher HD. However, since we explicitly calculate the HD, there are no false positives. Assuming independence of positions, two k-mers at HD $= d$ have the same hash with probability $\binom{k-h}{d}/\binom{k}{d}$. This probability is sufficiently high for small enough d (e.g., $d \leq 4$), then drops quickly and diminishes when $d \gg 4$ for appropriate choices of h and k. For a query sequence of length L, the expected number of matches across all $(L-k+1)$ k-mers is sufficiently high for several realistic choices of k and h. The false negatives can be further reduced by using multiple arrays with different LSH functions (randomly sampled h positions), a feature that CONSULT* methods use, but we have not tested for krepp. Finally, note that k-mer matches at very high HD can be spurious (e.g., not orthologous) and will also have many false negatives as a query k-mer and the closest reference match are likely to have different LSH values. Thus, we choose a fixed parameter δ (default: 4) and only keep k-mer matches with HD $\leq \delta$.

2.2 Coloring k-mers using a multitree based on the reference phylogeny

After finding reference k-mers similar to a query k-mer using the LSH index, we need to track *which* references include each matched k-mer x (i.e., $\mathcal{R}(x)$). It is easy to save one id for x, which is what many methods do (e.g., Kraken [9, 29] tracks the lowest common ancestor (LCA) of $\mathcal{R}(x)$ while CONSULT-II stores a soft-LCA [26]). To calculate distances to all relevant references, we need to record all $\mathcal{R}(x)$ ids. Keeping pointers from x to each $r \in \mathcal{R}(x)$ requires too much memory, necessitating a compact representation. Building such maps is the well-studied colored k-mer representation problem [30]. A *color* refers to the set of all references that share a k-mer, and the goal is to represent the set $\mathcal{C}$ of all colors compactly. For simplicity, we assume all singleton colors ($\{r\}$ for $r \in \mathcal{R}$) are in $\mathcal{C}$. The literature focuses on highly similar reference genomes (e.g., pangenomes) in the context of colored de Bruijn graphs [31–33]. In our application, k-mers come from an evolutionary diverse set (e.g., all Prokaryotes), making most colors sparse, while *some* densely sampled clades of

similar genomes also exist. We also have access to the tree T, absent from the classic k-mer coloring setting. Thus, instead of using existing methods, we design a new one with some similarities to an existing method.

We can represent each non-singleton $C \in \mathcal{C}$ as the union of other colors in $\mathcal{C}$. This defines a DAG $G = (V, E)$ with colors as nodes and edges representing the partition of a set into smaller subsets. G can be stored in an array of size $|E| + |V|$ (by saving the count and indices of children of each node). A trivial DAG can be built by partitioning each non-singleton color into all its constituents (i.e., singletons), which are leaves of the DAG. This bipartite DAG will have $|E| = \sum_{C \in \mathcal{C}} |C|$. However, we can do better. For similar genomes (e.g., pangenomes), we expect to see highly overlapping colors, which helps compression [34]. A main insight, successfully exploited by Campanelli et al. [34] (though not presented as a DAG), is that adding new colors for shared patterns across observed colors can potentially create smaller DAGs (w.r.t. $|E| + |V|$) by describing many colors using these "meta"-colors and thus reducing $|E|$. See an example in Figure 1B. Finding the minimal DAG such that $\mathcal{C} \subseteq V$ appears to be an intractable problem. Campanelli et al. [34] use a clustering heuristic to define meta-colors, while we will use the phylogeny.

With meta-colors, we can reap extra benefits from allowing exactly two children per node. Such a DAG can be stored in an array with $2|V|$ (instead of $3|V|$) elements, each $\log(|V|)$ bits; the array index is the id of the color, and for each index, we store the ids of its children. For each x, we keep the index of $\mathcal{R}(x)$ in an array $\mathbf{C}$ laid out identically to $\mathbf{A}$; this adds $\log(|V|)$ bits to $2(k-h)$ bit k-mer encodings, which is manageable (e.g., $|V| \approx 2^{22}$ for our WoL-v2 reference $|\mathcal{R}| = 15,953$). Additionally, we restrict the children of each node to be disjoint, making our DAG a binary multitree.

We argue that the phylogeny T provides a practical and efficient way to build the multitree. Consider a simple evolutionary model. An ancestral genome evolves down T accumulating random substitutions under an infinite k-mer assumption (similar to infinite sites) where the probability of any k-mer mutating twice is zero. Under this model, each $\mathcal{R}(x)$ becomes a perfect character, meaning that it maps exactly to a clade of T. As a result, all the colors in $\mathcal{C}$ will be a color in T; it is easy to see that the optimal solution w.r.t $|E|$ is simply T after contracting internal nodes without a color associated (i.e., removing branches where no k-mer mutated). While this model is reasonable for relatively short time scales, genome evolution across phylogenetic scales is far more complex and does not produce perfect characters. To handle this complexity, we build a multi-tree instead of a tree.

We start with T as the multitree. We partition each $\mathcal{R}(x)$ into the set of (potentially singleton) clades $C_1, \ldots, C_n \subseteq \mathcal{R}(x)$ of T such that no two clades are sisters (i.e., are maximal). Note that $n = 1$ for a perfect character and $n = |\mathcal{R}(x)|$ in the worst case. Each C_i is already a node of the multitree and thus can be represented with no additional color. For an $\mathcal{R}(x)$ with n maximal clades, $n-1$ internal nodes of T will have at least one clade under both their left and right children. For each of those nodes, we define a (potentially new) color composed of the union of colors of its children, obtaining $\mathcal{R}(x)$ at the LCA of all clades $C_1, \ldots, C_n$. This adds $n-1$ auxiliary colors on internal nodes, but some of these are expected to be shared with other k-mers. This heuristic can be considered effective if a small proportion of the $n-1$ added colors remain unobserved after we process all k-mers. We empirically observe this pattern; for 74% of non-singleton colors, none of their $n-1$ added colors remain unobserved, and for 90% of colors, only 1/3 or less are unobserved (Fig. S1).

Algorithm 1 builds $\mathbf{A}$ and the multitree jointly, moving up the tree T in a post-order traversal (implemented with nested task-parallelism). We find all k-mers shared between children, and for each such k-mer, we create a new color from its existing colors (line 18). On empirical data, this procedure adds ≈500 colors per genome (Fig. 1C). The only difficulty is checking if the union of two colors already exists (Parent), which we implement using an Abelian group (Section A.3).

Algorithm 1 Building the k-mer index (LSH index and multitree), given k-mers $\mathcal{M}(r)$ for references $r \in \mathcal{R}$ and the reference phylogeny T. $\mathcal{M}_i(r)$ is $\{x : x \in \mathcal{M}(r), \mathrm{LSH}(x){=}i\}$. We implement colors and $C_1 \cup C_2$ (PARENT) using integer encodings and an Abelian group hashing (see Section A.3).

procedure BUILDINDEX($t \in T$)
  $\mathbf{I} \leftarrow \texttt{Array}(2^{2h})$, $\mathbf{A} \leftarrow \texttt{Array}(|\mathcal{M}(t)|)$, $\mathbf{C} \leftarrow \texttt{Array}(|\mathcal{M}(t)|)$
  if $t \in \mathcal{R}$ **then**
    $\mathbf{A} \leftarrow \mathrm{sort}_{\mathrm{LSH}}(\mathcal{M}(t))$ ▷ Hashes to buckets and sorts each buckets
    $\mathbf{G} \leftarrow$ A one-leaf tree with one color $C = \{t\}$
    $\forall i \in [0, |\mathcal{M}(t)|) : \mathbf{C}[i] \leftarrow C$
    $\forall i \in [0, 2^{2h}) : \mathbf{I}[i] \leftarrow \mathbf{I}[i-1] + |\mathcal{M}_i(t)|$ ▷ $\mathbf{I}[-1]$=0.
  else
    $\mathbf{A}_1, \mathbf{I}_1, \mathbf{G}_1, \mathbf{C}_1 \leftarrow$ BUILDINDEX(left child of t)
    $\mathbf{A}_2, \mathbf{I}_2, \mathbf{G}_2, \mathbf{C}_2 \leftarrow$ BUILDINDEX(right child of t)
    $\mathbf{G} \leftarrow \mathbf{G}_1 \cup \mathbf{G}_2$ ▷ Simply combines multitrees without merging any nodes
    for all $i \in [0, 2^{2h})$ **do**
      $\mathcal{M}_i(t_1) \leftarrow \mathbf{A}_1[\mathbf{I}_1[i-1] \ldots \mathbf{I}_1[i]]$, $\mathcal{M}_i(t_2) \leftarrow \mathbf{A}_2[\mathbf{I}_2[i-1] \ldots \mathbf{I}_2[i]]$
      $\mathbf{I}[i] \leftarrow |\mathcal{M}_i(t_1) \cup \mathcal{M}_i(t_2)|$
      $\mathbf{A}[\mathbf{I}[i-1] \ldots \mathbf{I}[i]] \leftarrow \mathrm{sort}_{\mathrm{LSH}}(\mathcal{M}_i(t_1) \cup \mathcal{M}_i(t_2)$
      for all $\mathbf{I}[i-1] \leq j < \mathbf{I}[i]$ **do**
        if $\exists k_1, k_2 : \mathbf{A}[j] = \mathbf{A}_1[k_1] = \mathbf{A}_2[k_2] \in \mathcal{M}_i(t_1) \cap \mathcal{M}_i(t_2)$ **then**
          $\mathbf{C}[j] \leftarrow$ PARENT($\mathbf{G}, \mathbf{C}_1[k_1], \mathbf{C}_2[k_2]$) ▷ Finds $\mathbf{C}_1[k_1] \cup \mathbf{C}_2[k_2]$ in $\mathbf{G}$; add it if not present
        else
          $\mathbf{C}[j] \leftarrow \mathbf{C}_1[j]$ **if** $\mathbf{A}[j] \in \mathcal{M}_i(t_1)$ **else** $\mathbf{C}_2[j]$
  return $\mathbf{A}$, $\mathbf{I}$, $\mathbf{G}$, $\mathbf{C}$

In addition to the size of the multitree, we care about the number of edges that we need to follow to reconstruct $\mathcal{R}(x)$ during the query time. This consideration, which can conflict with the need to have the smallest multitree, roughly translates to desiring DAGs with low height. In our algorithm, the height of the multitree is bounded by the phylogenetic tree T, and reference trees tend to be sufficiently balanced. On our empirical dataset with 15,953 genomes, the height of the multitree is often short, never exceeding 4 for 99.8% of k-mers, and above 10 for only 0.02% (Fig. S1B). The average height of the multitree across all k-mers is 0.047, whereas this value increases to 0.068 when a random tree simulated using dual-birth model [35] is used instead of the inferred WoL-v2 tree, and to 0.092 when a random caterpillar tree is used. Using random trees leads to more k-mers labelled with colors that have non-zero heights (Fig. S1C). While the number of observed colors is fixed and cannot be optimized across these trees, the ladder tree and the dual-birth tree lead to 47% and 3% more unobserved colors, respectively.

2.3 Maximum likelihood estimation of the distance

For a query read q, we start with all reference genomes with at least one k-mer match with HD capped at δ, filter out highly distant genomes when low distant ones also exist (see Section A.2), and let remaining reference genomes be $\mathcal{R}'$. We compute the distance of q to each $r \in \mathcal{R}'$. We make two simplifying assumptions: *i*) The k-mer match with the lowest HD is the orthologous one, and therefore, if multiple k-mers in r match a query k-mer, we take the lowest HD and discard others. *ii*) There is no dependency between adjacent k-mers, and thus, we ignore positions of matched k-mers despite the ability to incorporate them. Overlapping k-mers are clearly not independent, and Blanca et al. [36] have modeled the dependencies for exact matches. However, analyzing dependency between inexact k-mer matches is far more challenging. Thus, we adopt this simplifying assumption,

noting that the use of minimizers reduces the number of overlapping k-mer matches and reduces dependence.

Let $v_{r,d}$ be the number of matches of HD d to the reference r and $\mathbf{v}_r = (v_{r,0}, \ldots, v_{r,\delta})$. With our assumptions, $\mathbf{v}_r$ is the sufficient statistics. Let $u_r = L - k + 1 - \sum_{d=0}^{\delta} v_{r,d}$ be the number of unmatched k-mers. Due to the independence, the likelihood function for $D \in [0, 1]$ becomes

$$\mathcal{L}_r(D; k, h, \delta, u_r, \mathbf{v}_r) = (P_{\text{miss}}(D; k, h, \delta))^{u_r} \prod_{d=0}^{\delta} (P_{\text{match}}(D; d, k, h))^{v_{r,d}} \tag{1}$$

where P_{match} is the probability of observing a match of HD d when the read-wide distance is D and P_{miss} is the probability of no such match for HD up to δ. To compute P_{match}, note that if we assume every match is due to homology, observing a match requires three independent events: *i*) The k-mer should be indexed despite using minimizers (or any form of random k-mer subsampling); we precompute this probability and call it ρ_r (see below). *ii*) Observing d mismatches when the underlying distance is D; this happens with probability

$$P_{\text{mutate}}(D; d, k) = D^d (1 - D)^{(k-d)} \binom{k}{d} .$$

iii) The LSH search finds the k-mer match at HD $= d$ with probability

$$P_{\text{collide}}(d, k, h) = \frac{\binom{k-h}{d}}{\binom{k}{d}} .$$

Therefore,

$$P_{\text{match}}(D; d, k, h) = \rho_r P_{\text{mutate}}(D; d, k) P_{\text{collide}}(d, k, h) = \rho_r D^d (1 - D)^{(k-d)} \binom{k}{d} \frac{\binom{k-h}{d}}{\binom{k}{d}} .$$

For ρ_r, note we exclude k-mers only due to use of minimizers. We precompute ρ_r during the indexing, setting it to the ratio of the number of minimizers to distinct k-mers for each r.

We can similarly compute the probability of missing a k-mer match, which can happen due to either of two disjoint events: *i*) The k-mer is not indexed, which happens with probability $1 - \rho_r$. *ii*) The k-mer is indexed but is not found by our method. An indexed k-mer can be missed either because it has HD $> \delta$, and we automatically ignore such matches or LSH does not find it (a false negative) even when HD $\leq \delta$. The probabilities of these two events are respectively $\sum_{d=\delta+1}^{k} P_{\text{mutate}}(D; d, k)$ and $\sum_{d=0}^{\delta} P_{\text{mutate}}(D; d, k)(1 - P_{\text{collide}}(d, k, h))$. Therefore, we end up with

$$P_{\text{miss}}(D; k, h, \delta) = (1 - \rho_r) + \rho_r \left(\sum_{d=0}^{\delta} P_{\text{mutate}}(D; d, k) \left(1 - P_{\text{collide}}(d, k, h)\right) + \sum_{d=\delta+1}^{k} P_{\text{mutate}}(D; d, k) \right) .$$

For each reference r, we maximize its likelihood function to get an estimate for D. Equivalently, we maximize the log-likelihood, which considerably simplifies Equation (1) and P_{match} but does not help with the P_{miss} due to summation. After dropping constant terms (w.r.t D), we get:

$$\hat{D}_r = \arg\max_{D} \; u_r \log \left(P_{\text{miss}}(D; k, h, \delta)\right) + \sum_{d=0}^{\delta} v_{r,d} \left(d \log(D) + (k - d) \log(1 - D)\right) . \tag{2}$$

The log-likelihood function (Fig. 1D) is concave for $D \in (0, 0.5)$ provided that k, h and δ follow restrictions given in Eq. (4), including for the default values (see Section B). We use the simple Brent's method [37] for solving the optimization, which in preliminary analyses was faster than alternatives, such as L-BFGS-B [38]. Brent's method uses quadratic interpolation to locate the global minimum by numerically approximating the derivative of single-variable convex functions.

2.4 Phylogenetic placement by applying likelihood-ratio test

Given distances, we can simply place q as sister to its closest match, but this method is clearly wrong even for an ultrametric tree (Fig. 1E) and never places q as sister to a clade, making T moot. Also, distances calculated from reads have low resolution and high variance; distances to many reference genomes may be statistically indistinguishable, making *the* minimum distance even less appealing. Alternatively, we can use the least squared error minimization approach of APPLES [39]. However, this approach faces a subtle challenge (Section E) related to variable rates of evolution across the genome. We need methods that consider high variability and noise of distances.

If T is ultrametric, the distance from q to all leaves of its sister clade C is the same and is minimum across the tree (Fig. 1E). Thus, given true distances, a valid approach is to place q as the sister to all references that have the minimum distance to it. With computed distances, this approach needs to account for noise; this can be done by defining the distance of q to a clade (mean or max across the clade) and finding the largest clade with a distance indistinguishable from the minimum distance. This idea remains reasonable when the tree is not fully ultrametric but is sufficiently close to ultrametricity (Fig. 1E). Below, we propose a principled likelihood-based test of statistical distinguishability for distances. With such a test, our goal becomes: *Find the largest clade C of the reference tree T, such that the distance of q to references in C is statistically indistinguishable from the minimum distance of q to any node.* We base our algorithm on this idea with a particular definition of clade distances (see below). Our formulation needs a root, which is often available and can otherwise be easily obtained [40].

Distance to a clade. On a tree with bounded deviation from ultrametricity, q has low mean distances to both left and right children at the correct placement edge or any edge below it; on other nodes, it will have a high distance on one child and a lower distance on the other (Fig. 1E). To capture this intuition, we extend the notion of distance to a clade C parent to clades C_1 and C_2 by first defining $\mathbf{v}_C = {}^1\!/\!_2(\mathbf{v}_{C_1} + \mathbf{v}_{C_2})$. We also define $\rho_C = \max(\rho_{C_1}, \rho_{C_2})$. With this definition, Equations (1) and (2) are extended to clades and are used to compute the read-to-clade distances.

Statistical distinguishability. Small ML distances $d_\emptyset$ and d_A to two clades $C_\emptyset$ and C_A may differ due to random noise or small changes in ρ (a clade can be just a single reference). We can use likelihood to statistically test for this. Let $C_\emptyset$ be the one with the lower distance, and let $l_\emptyset$ and l_A be the log-likelihood (log of Eq. (1)) computed at $d_\emptyset$ and d_A, using $\mathbf{v}_A$ as data. The likelihood ratio test (i.e., χ^2 test with 1 degree of freedom on $2(l_A - l_\emptyset)$) can be used to test if the higher likelihood clade is statistically distinguishable from the null (default $\alpha = 10\%$ significance level).

Finding optimal placement. For every internal node of the tree, we compute the distance and use the indistinguishability test to ask if its distance is statistically tied with the minimum-distance clade. Among all clades where we fail to reject the null hypothesis, we choose the largest, breaking ties by choosing the clade with the smallest distance. We do not place reads if $\sum_{i=0}^{\tau} v_{C,i} \leq 1$ for the chosen clade C (default: τ=2). Furthermore, if the chosen clade is the root, we characterize q as unplaced.

3 Results

3.1 Comparing MLE distances from krepp with genomic distance and alignment

We started by benchmarking krepp's MLE distances in a case with known ground truth. We simulated pairs of genomes at controlled distances using a simple sequence evolution model (see Section C for details). On these simulated data, krepp achieved high accuracy and very little bias in computing the true read to genome distances. As expected, there is noise, and the noise increases for higher distances; nevertheless, the average read distance matched true genome-wide distances, showing very little bias (Fig. S2).

Since our simulations are simple compared to real data, we next used real data to benchmark the method. We used a reference set of 15,953 genomes called WoL-v2 [3] used by Woltka [22]. We selected 500 query genomes not present in WoL-v2 across 36 phyla (267 genera) from RefSeq, spanning a wide range of novelty levels, quantified by the Mash [41] distance to the closest reference genome ($\mathcal{D}^*$). We simulated 33 million 150bp Illumina reads across all genomes using ART [42], and computed read-genome distances (D) using bowtie2 (v2.4.1) and krepp (v0.0.3). To find all alignments possible, we ran bowtie2 with `--very-sensitive` and `--all` options. We used Mash (v2.3 with sketch size of s=100,000 k-mers and k=29) [41] to compute genome-wide distances ($\mathcal{D}$) from each query genome to all reference genomes with read matches.

Compared to bowtie2, krepp performs similarly at low distances and substantially better at higher distances (Fig. 2). For reads where bowtie2 is able to compute the distance, we take its output as ground truth. In these cases, krepp computes similar distances to bowtie2 (Fig. 2A).

An advantage of krepp compared to alignment (e.g., using bowtie2) is that it analyzes many more reads. To show this, we average D from all reads mapped from a query genome to a reference genome and compare it with genome-wide distance computed using Mash. Both bowtie2 and krepp estimate $\mathcal{D} < 0.05$ accurately, with bowtie2 performing slightly better; however, as $\mathcal{D}$ increase to 5–12%, the bowtie2 average severely underestimates the Mash distance (due to unaligned reads) while krepp slightly (1.09× on average) overestimates it (Fig. 2B). Although we opted to use Mash for genome-wide distance estimates, alternative ANI estimation methods such as skani [43] and orthoANI [44] showed similar trends, except for higher distances $\mathcal{D}$ >15% (see Fig. S3). At these levels, krepp is still more accurate in computing mean distance compared to alignment but appears to under-estimate the distance computed using orthoANI and skani. Binning query genomes by their minimum Mash distance $\mathcal{D}^*$, we observe that bowtie2 fails to match more than two-thirds of the reads to *any* reference when $\mathcal{D}^* > 0.1$, and it fails to map a read to more than one reference for more than half the reads across all $\mathcal{D}^*$ (Fig. 2C). In contrast, krepp computes distances to multiple references for >90% of reads regardless of $\mathcal{D}^*$. Examining matches to all references, the krepp distances are accurate on average even for $0.1 < \mathcal{D} < 0.2$, though the portion of reads mapped reduces to 50% or less (Fig. 2D). Overall, except for the most similar genomes with $\mathcal{D} < 0.01$, bowtie2 always maps fewer reads and thus has an underestimation bias while krepp has an overestimation bias. The magnitude of biases is similar for $\mathcal{D} < 8\%$ but becomes smaller for krepp beyond that (Fig. 2E).

Beyond accuracy, krepp also enjoys better scalability than alignment using bowtie2 (Figs. 2 and S4F). As the number of references increases from 2000 to 15,953, bowtie2's running time increases 10× while krepp's increases by less than 2×, leading to ≈ 3× improvement over bowtie2 with the largest input. Both methods scale similarly with the number of reads (Fig. S4A). The total time needed to build the library is also shorter for krepp than bowtie2, whether we consider each of its batches (divided by hash table row ranges) or the total time (Fig. 2F). Both methods use similar levels of memory at query time, while krepp can use less during index building due to its

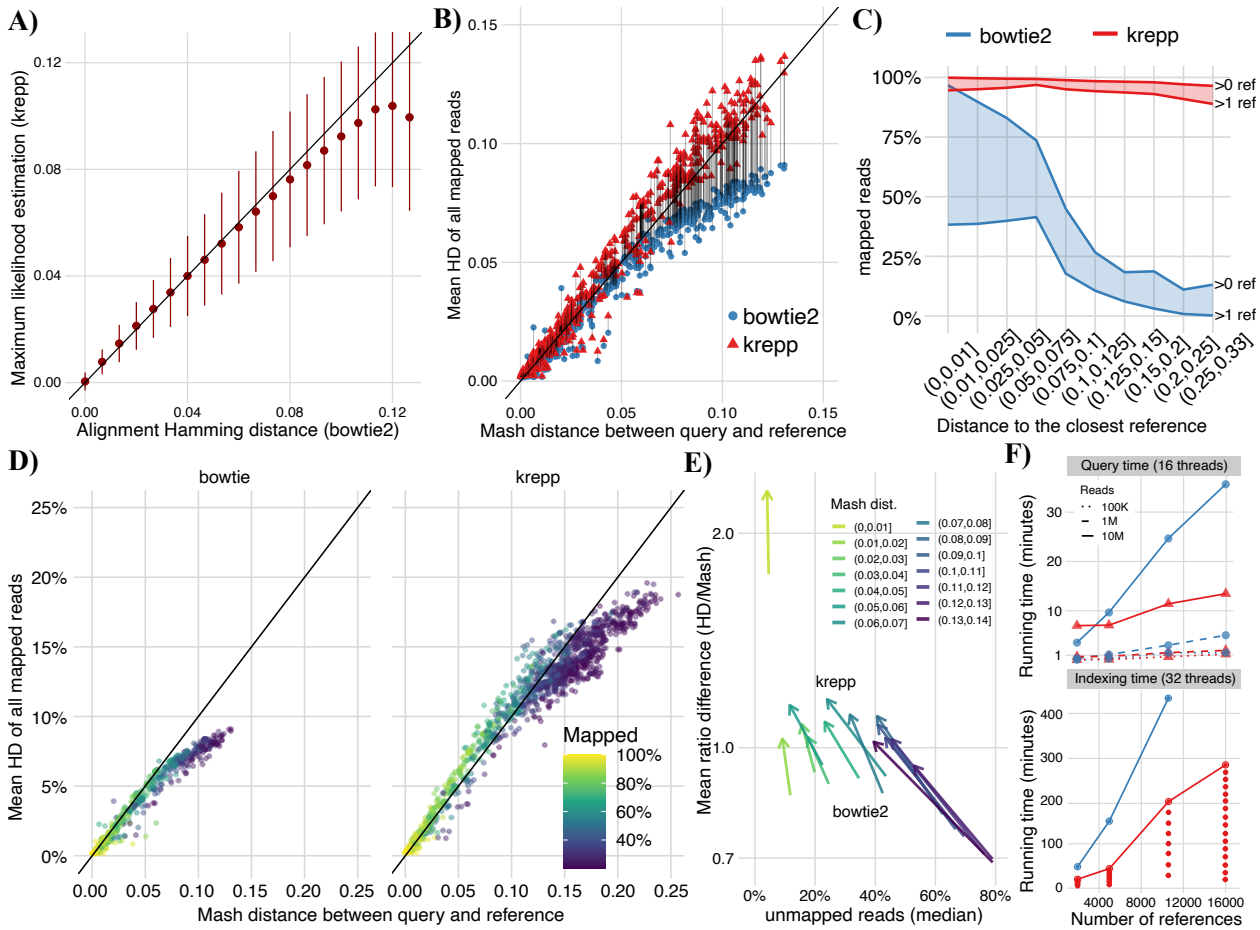

Figure 2: A) ML distance estimate of krepp versus bowtie2 alignment distance, i.e., HD normalized by the read length, (mean, standard deviation). B) Mean distance across reads ($\bar{D}$) versus Mash genome-wide distance $\mathcal{D}$ for query/ref genome pairs with $\geq$20% reads mapped by both methods. C) The portion of reads mapped to at least 1 (tail) or 2 (head) references, binning queries by Mash distance to their closest reference ($\mathcal{D}^*$). D) $\bar{D}$ versus Mash distance $\mathcal{D}$ from each query to all references with at least 20% reads mapped (colors). E) Binning data of panel (D) by Mash distance, change in the relative error ($\bar{D}$ over Mash) from bowtie2 to krepp. E) Running time versus the number of references. krepp builds the index in batches; we show the sum (line), but batches (dots) are run separately. 16K reference with bowtie2 had to be built on a more powerful machine and cannot be compared.

batching strategy (Fig. S4B). Note that bowtie2 solves a more difficult problem (alignment) than krepp; these comparisons are to clarify that if the only goal is to compute the distance of a read to reference genomes, using krepp is preferable to bowtie2.

3.2 Placement accuracy in controlled novelty experiments

We selected 110 query genomes from the WoL-v2 [3] tree using TreeCluster to ensure a range of novelty levels (path length on T to the closest leaf) are represented (Section D). We simulated reads as previous analyses and attempted to place every read on the backbone tree (15,952 leaves) after pruning the query genome from the library. We compared krepp placement to using distances from either krepp or bowtie2 and placing as sister to the closest genome. We measured error by the number of edges between the output of methods and the correct placement before pruning.

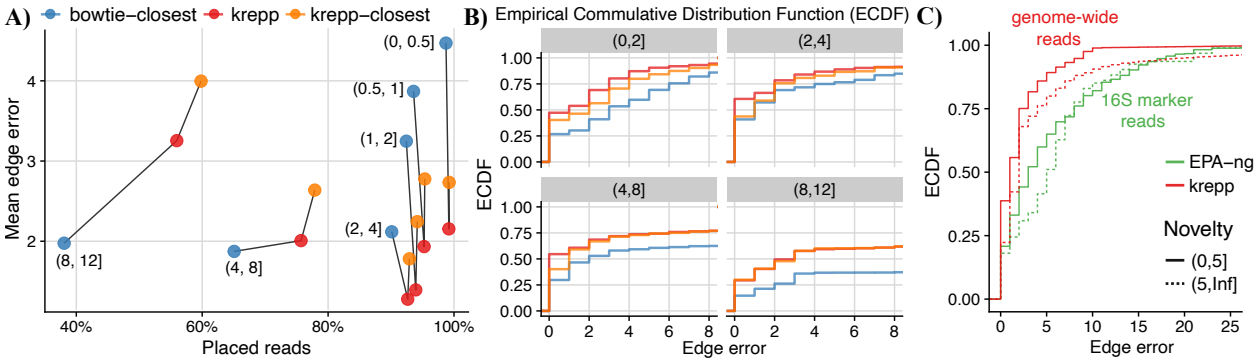

Figure 3: A,B) Placement error and effectiveness for 110 query genomes, binned based on novelty, labels in A) and titles in B), measured as the path length to the closest leaf on the WoL-v2 tree. In (A), unplaced reads are ignored in computing mean error; in (B), shown error distributions treat unplaced reads as infinity error. C) Comparing 16S marker-based placement using maximum likelihood method EPA-ng to genome-wide read placement using krepp on 100 query genomes. We exclude unplaced reads since EPA-ng is limited to 16S rRNA genes and the total number of reads analyzed differs immensely (608 versus 1M); see Fig. S6.

The mean error of krepp is low despite placing most reads (Fig. 3A). Even for the most novel query genomes, the mean (median) placement error is 3.2 edges (1) for the 50% of reads that it places. Simply using the closest krepp match increases the error, especially for more novel genomes. Placing each query sister to the LCA of all leaves that could not be statistically distinguished from the minimum krepp distance also increases error compared to our main algorithm (Fig. S5). On most novel queries, krepp leaves a sizable portion (48%) of reads unplaced, leading to fewer placed reads than krepp-closest. Compared to bowtie-closest, krepp-closest places slightly more reads and has a substantially lower errors, except for most novel queries, where it maps far more reads but has worse accuracy. However, krepp has much better accuracy than bowtie-closest in most cases. In the penultimate level of novelty, accuracies match despite krepp classifying more reads; at the highest level of novelty, krepp places more reads but has a proportionally higher level of mean error.

If we consider unplaced reads as (arbitrarily high) errors, krepp retains a median error of 0–2 in all levels of novelty. It also finds the correct edge for 26–61% of reads, depending on the novelty; bowtie-closest has a median error of 1–4 and does not place 50% of reads for most novel bins. krepp places >85% of all reads within four edges of the correct node, except for (4,8] and (8, 12] bins, for which 69% and 50% of reads still have $\leq$4 errors, respectively. The edge error is a bit higher on the least novel bin than the second least novel bin, perhaps because placing among highly similar genomes (short branches) is challenging.

To compare krepp to the only alternative method, App-SpaM, and examine error at lower phylogenetic scales, we next analyzed 8 taxa (6 genera and 2 families) sampled from WoL-v1 (Table S1). We selected 40-50 genomes as references and the rest as queries to simulate 150bp Illumina reads. We used the WoL-v1 tree pruned down to the remaining references. Here, krepp is slightly more accurate and far less memory-intensive than App-SpaM (Table 2). App-SpaM requires 84–189GB of memory for these small clades, and we could not build a library for two of the six clades; krepp requires only 1–6GB. krepp is also roughly 2–4X faster, including index construction time. Both methods have similar errors, though on average, krepp has 0.15 fewer errors and places 6% more reads on the correct branch. Substantial errors underscore the difficulty of placing reads on densely sampled groups of very similar genomes. Note that App-SpaM was as accurate as other k-mer-based methods such as RAPPAS [21] and EPIK [19], and APPLES for marker-based placement [11].

Finally, we asked a broader question: Is the marker-free krepp method competitive or better

	% of correct		Avg. node error		Memory (GB)		Running time (s)	
Taxon	krepp	App-SpaM	krepp	App-SpaM	krepp	App-SpaM	krepp	App-SpaM
Piscirickettsiaceae	19	**20**	**3.28**	3.46	**1.58**	132	**213**	732
Bacteroides	**35**	25	**2.73**	3.12	**3.87**	189	**161**	524
Moraxella	**23**	14	2.44	**2.28**	**0.93**	84	**218**	457
Bartonella	**33**	27	**1.77**	1.97	**1.73**	106	**183**	432
Mycobacterium	20	-	3.97	-	6.1	-	347	-
Rhizobiaceae	19	-	3.88	-	5.5	-	231	-

Table 2: Comparison of krepp and App-SpaM on six taxa pruned from the WoL-v1 trees. Measurements were performed on 2.10GHz Intel Xeon Silver 4110 CPUs using 16 threads and 188GB RAM. App-SpaM ran out of memory for *Mycobacterium* and *Rhizobiaceae*. For comparability with App-Spam, we force krepp to place every read (including at root and $\sum_{i=0}^{\tau} v_{C,i} \leq 1$).

than using marker-based placement using aligned reads? To test this, we used the 16S marker gene, selected 100 query taxa (from WoL-v1 with 10,575 ref.) only with limited novelty, generated *error-free* 150bp reads from their 16S marker gene, aligned them using WITCH [45], and placed reads on the species tree using the maximum likelihood method EPA-ng [13]. We also simulated 150bp reads with Illumina error profiles for the same query genomes. All query genomes were removed from the Wol-v1 tree and the references before placement. The placement error of genome-wide reads using krepp is substantially lower than 16S reads using EPA-ng (2.36 versus 5.5 edges on average) (Figs. 3 and S6C). It should be noted that all reads are placed for 16S whereas for genome-wide reads, krepp could place 86.4% of 1M generated reads overall, but >98% for the least novel queries (Fig. S6).

3.3 Human microbiome analysis

To demonstrate application on real metagenomic data, we re-analyzed a subset of The Human Microbiome Project (HMP) [46] data, consisting of 210 samples with 1M sampled reads [22]. These samples represent both male ($n = 138$) and female ($n = 72$) subjects and seven body sites (see Section F for details). We compared krepp's ability to characterize these samples against the OGU approach Zhu et al. [22] (heavily reliant on bowtie2) and Bracken, all using WoL-v2 as the reference. For krepp, we assigned a read to all leaves of the tree with statistically indistinguishable distances and built count-based feature tables using Woltka; we followed Zhu et al. [22] for OGUs and Bracken's species-level profiles. We used these feature tables to compute pairwise similarities between samples based on weighted UniFrac for krepp (Fig. 4B) and Woltka-OGU, and on Bray-Curtis for Bracken. Quantified by PERMANOVA psudeo-F statistics [47], krepp resulted in better separation of both body-sites and host sexes (Fig. 4C). The difference between Bracken and OGU was previously attributed to lower resolution and not incorporating branch lengths [22]. Compared to OGU, krepp resulted in equally strong sample differentiation, with negligible improvements in pseudo-F, but with much reduced running times. Note that the human microbiome is relatively well-studied, making the sensitivity of read requirement less crucial.

4 Discussion

We introduced the first scalable genome-wide phylogenetic read placement method and, in doing so, also designed a method to accurately estimate the distance of a read to a diverse set of genomes. Our method, krepp, provides a practical tool for performing new analyses of modern microbiome data with manageable computational requirements (e.g., the memory needed for a database of 15,953

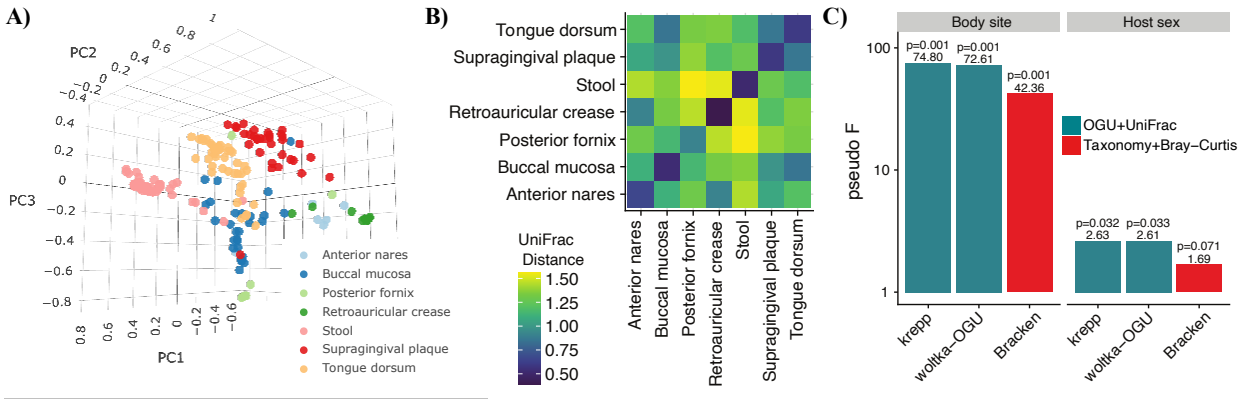

Figure 4: A) PCoA of samples due to krepp, colored by body sites. B) Pairwise weighted UniFrac distance averages between all body sites, estimated by krepp. D) Pseudo-F statistic, indicating the differentiation of community structures by body site and host sex, with corresponding p-values of PERMANOVA test.

diverse genomes across prokaryotes was only 85GB, available on most modern servers). Future work will further integrate krepp with widely used pipelines that currently use alignment to place reads (e.g., Woltka [22]), which is inferior to phylogenetic placement, or limit themselves to marker genes (e.g., TIPP [48]). As the number of references increases, the sublinear scaling of krepp (Fig. 2F) makes it more attractive than alignment. Sublinear scaling is achieved through LSH-based k-mer hashing, which limits the number of k-mers comparisons. To scale to even larger references (e.g., 200,000 genomes in GG2 [49]), running time will not be an issue, but memory could become a limiting factor; for those, we can attempt more aggressive filtering of k-mers, perhaps using unbiased methods such as FracMinHash [50, 51] or the tree-based method KRANK [52].

The framework designed here is general and can be adopted in other applications. The accurate read-genome distances can be used for taxonomic identification of each read and summarization to obtain taxonomic profiles, contamination detection, metagenomic binning, computing rates of evolution across the genome, or perhaps detecting horizontal transfer. Regarding the last application, we placed all reads on a species tree, understanding that gene tree discordance due to HGT [53] can prevent reads from being placed close to the species that contain them. The use of the species tree is a practical necessity and can help downstream applications like sample comparison [49]. We note that our placement errors have a long tail of high errors, some of which have low distances; a likely explanation is horizontal transfers from genomes that on the species tree are far from the query on the species tree. This observation could perhaps be used in the future to design methods that detect HGT or contamination across the genome.

References

[1] Jo Handelsman. Metagenomics: application of genomics to uncultured microorganisms. *Microbiology and molecular biology reviews*, 68(4):669–85, December 2004. ISSN 1092-2172. doi: 10.1128/MMBR.68.4.669-685.2004. URL `http://mmbr.asm.org/cgi/content/abstract/68/4/669`.

[2] Luc Cornet and Denis Baurain. Contamination detection in genomic data: more is not enough. *Genome Biology*, 23(1):60, February 2022. ISSN 1474-760X. doi: 10.1186/s13059-022-02619-9. URL `https://genomebiology.biomedcentral.com/articles/10.1186/s13059-022-02619-9`.

[3] Metin Balaban, Yueyu Jiang, Qiyun Zhu, Daniel McDonald, Rob Knight, and Siavash Mirarab. Generation of accurate, expandable phylogenomic trees with uDance. *Nature Biotechnology*, Online, July 2023. ISSN 1546-1696. doi: 10.1038/s41587-023-01868-8. URL `https://doi.org/10.1038/s41587-023-01868-8`.

[4] Frederick A Matsen. Phylogenetics and the Human Microbiome. *Systematic Biology*, 64(1): e26–e41, January 2015. ISSN 1076-836X. doi: 10.1093/sysbio/syu053. URL `http://arxiv.org/abs/1407.1794`. arXiv: 1407.1794.

[5] Frederick A. Matsen and Steven N. Evans. Edge Principal Components and Squash Clustering: Using the Special Structure of Phylogenetic Placement Data for Sample Comparison. *PLoS ONE*, 8(3):e56859, March 2013. ISSN 1932-6203. doi: 10.1371/journal.pone.0056859. URL `https://dx.plos.org/10.1371/journal.pone.0056859`. arXiv: 1107.5095.

[6] Micah Hamady, Catherine Lozupone, and Rob Knight. Fast UniFrac: facilitating high-throughput phylogenetic analyses of microbial communities including analysis of pyrosequencing and PhyloChip data. *The ISME journal*, 4(1):17–27, 2010. ISSN 1751-7362. Publisher: Nature Publishing Group.

[7] Stefan Janssen, Daniel McDonald, Antonio Gonzalez, Jose A. Navas-Molina, Lingjing Jiang, Zhenjiang Zech Xu, Kevin Winker, Deborah M. Kado, Eric Orwoll, Mark Manary, Siavash Mirarab, and Rob Knight. Phylogenetic Placement of Exact Amplicon Sequences Improves Associations with Clinical Information. *mSystems*, 3(3):e00021–18, April 2018. ISSN 2379-5077. doi: 10.1128/mSystems.00021-18. URL `http://msystems.asm.org/lookup/doi/10.1128/mSystems.00021-18`.

[8] Donovan H Parks, Maria Chuvochina, David W Waite, Christian Rinke, Adam Skarshewski, Pierre-Alain Chaumeil, and Philip Hugenholtz. A standardized bacterial taxonomy based on genome phylogeny substantially revises the tree of life. *Nature Biotechnology*, 36(10):996–1004, November 2018. ISSN 1087-0156. doi: 10.1038/nbt.4229. URL `http://www.nature.com/articles/nbt.4229`.

[9] Derrick E. Wood and Steven L. Salzberg. Kraken: ultrafast metagenomic sequence classification using exact alignments. *Genome Biology*, 15(3):R46, March 2014. ISSN 1474-760X. doi: 10.1186/gb-2014-15-3-r46. URL `https://doi.org/10.1186/gb-2014-15-3-r46`.

[10] Shinichi Sunagawa, Daniel R Mende, Georg Zeller, Fernando Izquierdo-Carrasco, Simon a Berger, Jens Roat Kultima, Luis Pedro Coelho, Manimozhiyan Arumugam, Julien Tap, Henrik Bjørn Nielsen, Simon Rasmussen, Søren Brunak, Oluf Pedersen, Francisco Guarner, Willem M de Vos, Jun Wang, Junhua Li, Joël Doré, S. Dusko Ehrlich, Alexandros Stamatakis, and Peer Bork. Metagenomic species profiling using universal phylogenetic marker genes. *Nature Methods*, 10(12):1196–1199, December 2013. ISSN 1548-7091. doi: 10.1038/nmeth.2693. URL `http://www.ncbi.nlm.nih.gov/pubmed/24141494`.

[11] Matthias Blanke and Burkhard Morgenstern. App-SpaM: phylogenetic placement of short reads without sequence alignment. *Bioinformatics Advances*, 1(1):vbab027, January 2021. ISSN 2635-0041. doi: 10.1093/bioadv/vbab027. URL `https://doi.org/10.1093/bioadv/vbab027`.

[12] Frederick A. Matsen, Robin B. Kodner, and E. Virginia Armbrust. pplacer: linear time maximum-likelihood and Bayesian phylogenetic placement of sequences onto a fixed reference tree. *BMC Bioinformatics*, 11(1):1–16, December 2010. ISSN 1471-2105. doi: 10.

1186/1471-2105-11-538. URL https://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-11-538. Number: 1 Publisher: BioMed Central.

[13] Pierre Barbera, Alexey M Kozlov, Lucas Czech, Benoit Morel, Diego Darriba, Tomáš Flouri, and Alexandros Stamatakis. EPA-ng: Massively Parallel Evolutionary Placement of Genetic Sequences. *Systematic Biology*, 68(2):365–369, March 2019. ISSN 1063-5157. doi: 10.1093/sysbio/syy054. URL https://academic.oup.com/sysbio/article/68/2/365/5079844.

[14] Siavash Mirarab, Nam Nguyen, and Tandy Warnow. SEPP: SATé-Enabled Phylogenetic Placement. In *Pacific Symposium On Biocomputing*, pages 247–58. WORLD SCIENTIFIC, December 2012. ISBN 978-981-4596-37-4. doi: 10.1142/9789814366496_0024. URL http://www.ncbi.nlm.nih.gov/pubmed/22174280.

[15] Daniel G. Brown and Jakub Truszkowski. LSHPlace: Fast phylogenetic placement using locality-sensitive hashing. In *Proceedings of the Pacific Symposium on Biocomputing 2013*, pages 310–319, Kohala Coast, Hawaii, USA, November 2012. World Scientific. ISBN 978-981-4596-36-7 978-981-4447-97-3. doi: 10.1142/9789814447973_0031. URL http://www.worldscientific.com/doi/abs/10.1142/9789814447973_0031.

[16] Nam-phuong Nguyen, Siavash Mirarab, Bo Liu, Mihai Pop, and Tandy Warnow. TIPP: taxonomic identification and phylogenetic profiling. *Bioinformatics*, 30(24):3548–3555, December 2014. ISSN 1460-2059. doi: 10.1093/bioinformatics/btu721. URL http://bioinformatics.oxfordjournals.org/cgi/doi/10.1093/bioinformatics/btu721.

[17] Francesco Asnicar, Andrew Maltez Thomas, Francesco Beghini, Claudia Mengoni, Serena Manara, Paolo Manghi, Qiyun Zhu, Mattia Bolzan, Fabio Cumbo, Uyen May, Jon G Sanders, Moreno Zolfo, Evguenia Kopylova, Edoardo Pasolli, Rob Knight, Siavash Mirarab, Curtis Huttenhower, and Nicola Segata. Precise phylogenetic analysis of microbial isolates and genomes from metagenomes using PhyloPhlAn 3.0. *Nature Communications*, 11(1):2500, December 2020. ISSN 2041-1723. doi: 10.1038/s41467-020-16366-7. URL http://dx.doi.org/10.1038/s41467-020-16366-7. Publisher: Springer US ISBN: 0348556977.

[18] Simon A. Berger, Denis Krompass, and Alexandros Stamatakis. Performance, accuracy, and Web server for evolutionary placement of short sequence reads under maximum likelihood. *Systematic biology*, 60(3):291–302, May 2011. ISSN 1076-836X. doi: 10.1093/sysbio/syr010. URL http://sysbio.oxfordjournals.org/cgi/content/abstract/60/3/291.

[19] Nikolai Romashchenko, Benjamin Linard, Fabio Pardi, and Eric Rivals. EPIK: precise and scalable evolutionary placement with informative k-mers. *Bioinformatics*, 39(12):btad692, December 2023. ISSN 1367-4811. doi: 10.1093/bioinformatics/btad692. URL https://doi.org/10.1093/bioinformatics/btad692.

[20] Ana Paula Appel, Renato L. F. Cunha, Charu C. Aggarwal, and Marcela Megumi Terakado. Using link and content over time for embedding generation in Dynamic Attributed Networks. *arXiv:1807.06560 [cs, stat]*, 11052:3–18, 2019. doi: 10.1007/978-3-030-10928-8_1. URL http://arxiv.org/abs/1807.06560. arXiv: 1807.06560.

[21] Benjamin Linard, Krister Swenson, and Fabio Pardi. Rapid alignment-free phylogenetic identification of metagenomic sequences. *Bioinformatics*, 35(18):3303–3312, September 2019. ISSN 1367-4803. doi: 10.1093/bioinformatics/btz068. URL https://doi.org/10.1093/bioinformatics/btz068.

[22] Qiyun Zhu, Shi Huang, Antonio Gonzalez, Imran McGrath, Daniel McDonald, Niina Haiminen, George Armstrong, Yoshiki Vázquez-Baeza, Julian Yu, Justin Kuczynski, Gregory D. Sepich-Poore, Austin D. Swafford, Promi Das, Justin P. Shaffer, Franck Lejzerowicz, Pedro Belda-Ferre, Aki S. Havulinna, Guillaume Méric, Teemu Niiranen, Leo Lahti, Veikko Salomaa, Ho-Cheol Kim, Mohit Jain, Michael Inouye, Jack A. Gilbert, and Rob Knight. Phylogeny-Aware Analysis of Metagenome Community Ecology Based on Matched Reference Genomes while Bypassing Taxonomy. *mSystems*, 7(2):e0016722, April 2022. ISSN 2379-5077. doi: 10.1128/msystems.00167-22.

[23] Ben Langmead and Steven L. Salzberg. Fast gapped-read alignment with Bowtie 2. *Nature Methods*, 9(4):357–359, April 2012. ISSN 1548-7105. doi: 10.1038/nmeth.1923. URL `https://www.nature.com/articles/nmeth.1923`. Publisher: Nature Publishing Group.

[24] Heng Li. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics*, 34(18):3094–3100, September 2018. ISSN 1367-4803. doi: 10.1093/bioinformatics/bty191. URL `https://doi.org/10.1093/bioinformatics/bty191`.

[25] Eleonora Rachtman, Vineet Bafna, and Siavash Mirarab. CONSULT: accurate contamination removal using locality-sensitive hashing. *NAR Genomics and Bioinformatics*, 3(3):lqab071, June 2021. ISSN 2631-9268. doi: 10.1093/nargab/lqab071. URL `https://academic.oup.com/nargab/article/doi/10.1093/nargab/lqab071/6342218`.

[26] Ali Osman Berk Şapcı, Eleonora Rachtman, and Siavash Mirarab. CONSULT-II: Accurate taxonomic identification and profiling using locality-sensitive hashing. *Bioinformatics*, page btae150, March 2024. ISSN 1367-4811. doi: 10.1093/bioinformatics/btae150. URL `https://doi.org/10.1093/bioinformatics/btae150`.

[27] Ali Osman Berk Şapcı and Siavash Mirarab. Memory-bound k-mer selection for large and evolutionary diverse reference libraries. *Genome Research*, page gr.279339.124, August 2024. ISSN 1088-9051, 1549-5469. doi: 10.1101/gr.279339.124. URL `https://genome.cshlp.org/content/early/2024/08/29/gr.279339.124`. Company: Cold Spring Harbor Laboratory Press Distributor: Cold Spring Harbor Laboratory Press Institution: Cold Spring Harbor Laboratory Press Label: Cold Spring Harbor Laboratory Press Publisher: Cold Spring Harbor Lab.

[28] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, STOC '98, pages 604–613, New York, NY, USA, May 1998. Association for Computing Machinery. ISBN 978-0-89791-962-3. doi: 10.1145/276698.276876. URL `https://dl.acm.org/doi/10.1145/276698.276876`.

[29] Derrick E. Wood, Jennifer Lu, and Ben Langmead. Improved metagenomic analysis with Kraken 2. *Genome Biology*, 20(1):257, November 2019. ISSN 1474-760X. doi: 10.1186/s13059-019-1891-0. URL `https://doi.org/10.1186/s13059-019-1891-0`.

[30] Camille Marchet, Christina Boucher, Simon J. Puglisi, Paul Medvedev, Mikaël Salson, and Rayan Chikhi. Data structures based on k-mers for querying large collections of sequencing data sets. *Genome Research*, 31(1):1–12, January 2021. ISSN 1088-9051, 1549-5469. doi: 10.1101/gr.260604.119. URL `https://genome.cshlp.org/content/31/1/1`. Company: Cold Spring Harbor Laboratory Press Distributor: Cold Spring Harbor Laboratory Press Institution:

Cold Spring Harbor Laboratory Press Label: Cold Spring Harbor Laboratory Press Publisher: Cold Spring Harbor Lab.

[31] Fatemeh Almodaresi, Hirak Sarkar, Avi Srivastava, and Rob Patro. A space and time-efficient index for the compacted colored de Bruijn graph. *Bioinformatics*, 34(13):i169–i177, July 2018. ISSN 1367-4803. doi: 10.1093/bioinformatics/bty292. URL `https://doi.org/10.1093/bioinformatics/bty292`.

[32] Guillaume Holley and Páll Melsted. Bifrost: highly parallel construction and indexing of colored and compacted de Bruijn graphs. *Genome Biology*, 21(1):1–20, December 2020. ISSN 1474-760X. doi: 10.1186/s13059-020-02135-8. URL `https://genomebiology.biomedcentral.com/articles/10.1186/s13059-020-02135-8`. Number: 1 Publisher: BioMed Central.

[33] Jason Fan, Jamshed Khan, Noor Pratap Singh, Giulio Ermanno Pibiri, and Rob Patro. Fulgor: a fast and compact k-mer index for large-scale matching and color queries. *Algorithms for Molecular Biology*, 19(1):3, January 2024. ISSN 1748-7188. doi: 10.1186/s13015-024-00251-9. URL `https://doi.org/10.1186/s13015-024-00251-9`.

[34] Alessio Campanelli, Giulio Ermanno Pibiri, Jason Fan, and Rob Patro. Where the Patterns Are: Repetition-Aware Compression for Colored de Bruijn Graphs. *Journal of Computational Biology*, October 2024. doi: 10.1089/cmb.2024.0714. URL `https://www.liebertpub.com/doi/full/10.1089/cmb.2024.0714`. Publisher: Mary Ann Liebert, Inc., publishers.

[35] Niema Moshiri and Siavash Mirarab. A Two-State Model of Tree Evolution and Its Applications to Alu Retrotransposition. *Systematic Biology*, 67(3):475–489, May 2018. ISSN 1063-5157. doi: 10.1093/sysbio/syx088. URL `https://doi.org/10.1093/sysbio/syx088`.

[36] Antonio Blanca, Robert S. Harris, David Koslicki, and Paul Medvedev. The Statistics of k-mers from a Sequence Undergoing a Simple Mutation Process Without Spurious Matches. *Journal of Computational Biology: A Journal of Computational Molecular Cell Biology*, 29(2):155–168, February 2022. ISSN 1557-8666. doi: 10.1089/cmb.2021.0431.

[37] Richard P. Brent. *Algorithms for Minimization without Derivatives*. Prentice-Hall, Englewood Cliffs, New Jersey, 1st edition, 1973.

[38] Ciyou Zhu, Richard H. Byrd, Peihuang Lu, and Jorge Nocedal. Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization. *ACM Trans. Math. Softw.*, 23(4):550–560, December 1997. ISSN 0098-3500. doi: 10.1145/279232.279236. URL `https://dl.acm.org/doi/10.1145/279232.279236`.

[39] Metin Balaban, Shahab Sarmashghi, and Siavash Mirarab. APPLES: Scalable Distance-Based Phylogenetic Placement with or without Alignments. *Systematic Biology*, 69(3):566–578, May 2020. ISSN 1063-5157. doi: 10.1093/sysbio/syz063. URL `https://doi.org/10.1093/sysbio/syz063`.

[40] Uyen Mai, Erfan Sayyari, and Siavash Mirarab. Minimum variance rooting of phylogenetic trees and implications for species tree reconstruction. *PLOS ONE*, 12(8):e0182238, August 2017. ISSN 1932-6203. doi: 10.1371/journal.pone.0182238. URL `http://dx.plos.org/10.1371/journal.pone.0182238`. Publisher: Public Library of Science ISBN: 1111111111.

[41] Brian D. Ondov, Todd J. Treangen, Páll Melsted, Adam B. Mallonee, Nicholas H. Bergman, Sergey Koren, and Adam M. Phillippy. Mash: fast genome and metagenome distance estimation

using MinHash. *Genome Biology*, 17(1):1–14, December 2016. ISSN 1474-760X. doi: 10.1186/s13059-016-0997-x. URL https://genomebiology.biomedcentral.com/articles/10.1186/s13059-016-0997-x. Number: 1 Publisher: BioMed Central.

[42] Weichun Huang, Leping Li, Jason R. Myers, and Gabor T. Marth. ART: a next-generation sequencing read simulator. *Bioinformatics*, 28(4):593, December 2011. doi: 10.1093/bioinformatics/btr708. URL https://pmc.ncbi.nlm.nih.gov/articles/PMC3278762/.

[43] Jim Shaw and Yun William Yu. Fast and robust metagenomic sequence comparison through sparse chaining with skani. *Nature Methods*, September 2023. ISSN 1548-7091, 1548-7105. doi: 10.1038/s41592-023-02018-3. URL https://www.nature.com/articles/s41592-023-02018-3.

[44] Imchang Lee, Yeong Ouk Kim, Sang-Cheol Park, and Jongsik Chun. OrthoANI: An improved algorithm and software for calculating average nucleotide identity. *International Journal of Systematic and Evolutionary Microbiology*, 66(2):1100–1103, 2016. ISSN 1466-5034. doi: 10.1099/ijsem.0.000760. URL https://www.microbiologyresearch.org/content/journal/ijsem/10.1099/ijsem.0.000760. Publisher: Microbiology Society,.

[45] Chengze Shen, Minhyuk Park, and Tandy Warnow. WITCH: Improved Multiple Sequence Alignment Through Weighted Consensus Hidden Markov Model Alignment. *Journal of Computational Biology*, 29(8):782–801, August 2022. ISSN 1557-8666. doi: 10.1089/cmb.2021.0585. URL https://www.liebertpub.com/doi/10.1089/cmb.2021.0585.

[46] Human Microbiome Project Consortium. Structure, function and diversity of the healthy human microbiome. *Nature*, 486(7402):207–214, June 2012. ISSN 1476-4687. doi: 10.1038/nature11234.

[47] Marti J Anderson. A new method for non-parametric multivariate analysis of variance. *Austral ecology*, 26(1):32–46, 2001. doi: https://doi.org/10.1111/j.1442-9993.2001.01070.pp.x.

[48] Nidhi Shah, Erin K Molloy, Mihai Pop, and Tandy Warnow. TIPP2: metagenomic taxonomic profiling using phylogenetic markers. *Bioinformatics*, 37(13):1839–1845, July 2021. ISSN 1367-4803. doi: 10.1093/bioinformatics/btab023. URL https://academic.oup.com/bioinformatics/article/37/13/1839/6104844.

[49] Daniel McDonald, Yueyu Jiang, Metin Balaban, Kalen Cantrell, Qiyun Zhu, Antonio Gonzalez, James T. Morton, Giorgia Nicolaou, Donovan H. Parks, Søren M. Karst, Mads Albertsen, Philip Hugenholtz, Todd DeSantis, Se Jin Song, Andrew Bartko, Aki S. Havulinna, Pekka Jousilahti, Susan Cheng, Michael Inouye, Teemu Niiranen, Mohit Jain, Veikko Salomaa, Leo Lahti, Siavash Mirarab, and Rob Knight. Greengenes2 unifies microbial data in a single reference tree. *Nature Biotechnology*, Online, July 2023. ISSN 1546-1696. doi: 10.1038/s41587-023-01845-1. URL https://doi.org/10.1038/s41587-023-01845-1.

[50] Luiz Irber, Phillip T. Brooks, Taylor Reiter, N. Tessa Pierce-Ward, Mahmudur Rahman Hera, David Koslicki, and C. Titus Brown. Lightweight compositional analysis of metagenomes with FracMinHash and minimum metagenome covers. preprint, Bioinformatics, January 2022. URL http://biorxiv.org/lookup/doi/10.1101/2022.01.11.475838.

[51] Mahmudur Rahman Hera, N. Tessa Pierce-Ward, and David Koslicki. Deriving confidence intervals for mutation rates across a wide range of evolutionary distances using FracMinHash.

Genome Research, 33(7):1061–1068, July 2023. ISSN 1088-9051, 1549-5469. doi: 10.1101/gr.277651.123. URL https://genome.cshlp.org/content/33/7/1061. Company: Cold Spring Harbor Laboratory Press Distributor: Cold Spring Harbor Laboratory Press Institution: Cold Spring Harbor Laboratory Press Label: Cold Spring Harbor Laboratory Press Publisher: Cold Spring Harbor Lab.

[52] Ali Osman Berk Sapci and Siavash Mirarab. Memory-bound k -mer selection for large and evolutionary diverse reference libraries. *Genome Research*, 34:1455–1467, August 2024. ISSN 1088-9051. doi: 10.1101/gr.279339.124. URL http://genome.cshlp.org/lookup/doi/10.1101/gr.279339.124.

[53] J. Peter Gogarten, W. Ford Doolittle, and Jeffrey G. Lawrence. Prokaryotic Evolution in Light of Gene Transfer. *Molecular Biology and Evolution*, 19(12):2226–2238, December 2002. ISSN 0737-4038. doi: 10.1093/oxfordjournals.molbev.a004046. URL http://academic.oup.com/mbe/article/19/12/2226/997576.

[54] Evan Bolyen, Jai Ram Rideout, Matthew R. Dillon, Nicholas A. Bokulich, Christian C. Abnet, Gabriel A. Al-Ghalith, Harriet Alexander, Eric J. Alm, Manimozhiyan Arumugam, Francesco Asnicar, Yang Bai, Jordan E. Bisanz, Kyle Bittinger, Asker Brejnrod, Colin J. Brislawn, C. Titus Brown, Benjamin J. Callahan, Andrés Mauricio Caraballo-Rodríguez, John Chase, Emily K. Cope, Ricardo Da Silva, Christian Diener, Pieter C. Dorrestein, Gavin M. Douglas, Daniel M. Durall, Claire Duvallet, Christian F. Edwardson, Madeleine Ernst, Mehrbod Estaki, Jennifer Fouquier, Julia M. Gauglitz, Sean M. Gibbons, Deanna L. Gibson, Antonio Gonzalez, Kestrel Gorlick, Jiarong Guo, Benjamin Hillmann, Susan Holmes, Hannes Holste, Curtis Huttenhower, Gavin A. Huttley, Stefan Janssen, Alan K. Jarmusch, Lingjing Jiang, Benjamin D. Kaehler, Kyo Bin Kang, Christopher R. Keefe, Paul Keim, Scott T. Kelley, Dan Knights, Irina Koester, Tomasz Kosciolek, Jorden Kreps, Morgan G. I. Langille, Joslynn Lee, Ruth Ley, Yong-Xin Liu, Erikka Loftfield, Catherine Lozupone, Massoud Maher, Clarisse Marotz, Bryan D. Martin, Daniel McDonald, Lauren J. McIver, Alexey V. Melnik, Jessica L. Metcalf, Sydney C. Morgan, Jamie T. Morton, Ahmad Turan Naimey, Jose A. Navas-Molina, Louis Felix Nothias, Stephanie B. Orchanian, Talima Pearson, Samuel L. Peoples, Daniel Petras, Mary Lai Preuss, Elmar Pruesse, Lasse Buur Rasmussen, Adam Rivers, Michael S. Robeson, Patrick Rosenthal, Nicola Segata, Michael Shaffer, Arron Shiffer, Rashmi Sinha, Se Jin Song, John R. Spear, Austin D. Swafford, Luke R. Thompson, Pedro J. Torres, Pauline Trinh, Anupriya Tripathi, Peter J. Turnbaugh, Sabah Ul-Hasan, Justin J. J. van der Hooft, Fernando Vargas, Yoshiki Vázquez-Baeza, Emily Vogtmann, Max von Hippel, William Walters, Yunhu Wan, Mingxun Wang, Jonathan Warren, Kyle C. Weber, Charles H. D. Williamson, Amy D. Willis, Zhenjiang Zech Xu, Jesse R. Zaneveld, Yilong Zhang, Qiyun Zhu, Rob Knight, and J. Gregory Caporaso. Reproducible, interactive, scalable and extensible microbiome data science using qiime 2. *Nature Biotechnology*, 37(8):852–857, 2019. ISSN 1546-1696. doi: 10.1038/s41587-019-0209-9. URL https://doi.org/10.1038/s41587-019-0209-9.

Supplementary Methods

Appendix A Details of krepp

A.1 Efficient union of sibling indexes

Algorithm 2 Building the k-mer index (LSH index and multitree).

procedure BUILDINDEX($t \in T$)
  $\mathbf{I} \leftarrow \texttt{Array}(2^{2h})$, $\mathbf{A} \leftarrow \texttt{Array}(|\mathcal{M}(t)|)$, $\mathbf{C} \leftarrow \texttt{Array}(|\mathcal{M}(t)|)$
  if $t \in \mathcal{R}$ **then**
    $\mathbf{A} \leftarrow \text{sort}_{\text{LSH}}(\mathcal{M}(t))$ ▷ Hashes to buckets and sorts each buckets
    $\mathbf{G} \leftarrow$ A one-leaf tree with one color $C = \{t\}$
    $\forall i \in [0, |\mathcal{M}(t)|) : \mathbf{C}[i] \leftarrow C$
    $\forall i \in [0, 2^{2h}) : \mathbf{I}[i] \leftarrow \mathbf{I}[i-1] + |\mathcal{M}_i(t)|$ ▷ $\mathbf{I}[-1]$=0.
  else
    $\mathbf{A}_1, \mathbf{I}_1, \mathbf{G}_1, \mathbf{C}_1 \leftarrow$ BUILDINDEX(left child of t)
    $\mathbf{A}_2, \mathbf{I}_2, \mathbf{G}_2, \mathbf{C}_2 \leftarrow$ BUILDINDEX(right child of t)
    $\mathbf{G} \leftarrow \mathbf{G}_1 \cup \mathbf{G}_2$ ▷ Simply combines multitrees without merging any nodes
    for all $i \in [0, 2^{2h})$ **do**
      $k \leftarrow \mathbf{I}[i-1]$, $j_1 \leftarrow \mathbf{I}_1[i-1]$, $j_2 \leftarrow \mathbf{I}_2[i-1]$
      while $j_1 < \mathbf{I}_1[i]$ and $j_2 < \mathbf{I}_2[i]$ **do**
        if $\mathbf{A}_1[j_1] < \mathbf{A}_2[j_2]$ **then**
          $\mathbf{A}[k] \leftarrow \mathbf{A}_1[j_1]$, $\mathbf{C}[k] \leftarrow \mathbf{C}_1[j_1]$, j_1++
        else if $\mathbf{A}_2[j_2] < \mathbf{A}_1[j_1]$ **then**
          $\mathbf{A}[k] \leftarrow \mathbf{A}_2[j_2]$, $\mathbf{C}[k] \leftarrow \mathbf{C}_2[j_2]$, j_2++
        else
          $\mathbf{A}[k] \leftarrow \mathbf{A}_1[j_1]$, $\mathbf{C}[k] \leftarrow$ PARENT($\mathbf{G}, \mathbf{C}_1[j_1], \mathbf{C}_2[j_2]$)
          j_1++, j_2++
        k++
      while $j_1 < \mathbf{I}_1[i]$ **do**
        $\mathbf{A}[k] \leftarrow \mathbf{A}_1[j_1]$, $\mathbf{C}[k] \leftarrow \mathbf{C}_1[j_1]$, j_1++, k++
      while $j_2 < \mathbf{I}_2[i]$ **do**
        $\mathbf{A}[k] \leftarrow \mathbf{A}_2[j_2]$, $\mathbf{C}[k] \leftarrow \mathbf{C}_2[j_2]$, j_2++, k++
      $\mathbf{I}[i] \leftarrow k$
  return $\mathbf{A}$, $\mathbf{I}$, $\mathbf{G}$, $\mathbf{C}$

A.2 Filtering high distance genomes and placements

We perform the following forms of filtering before computing distances.

- When a read q comes from a region that is shared across many references, the number of matches can be excessively high, especially for high δ (e.g., > 2). For the sake of running time, we ignore references with only high HD matches <u>if low HD matches exist for other references</u>. Specifically, let $d_{\min}$ be the minimum HD across all k-mers for all $r \in \mathcal{R}$. We ignore a reference if the minimum HD across all its matches to q is $2(d_{\min} + 1)$ or higher.

- Optionally, krepp can only report matching references with distances statistically indistinguishable to the ML distance of the closest reference, using the same likelihood ratio test as the placement. This option is disabled by default.

After computing distances and choosing a candidate clade C (the largest one that we fail to reject the null hypothesis for), we avoid placing a read if it has too few k-mer matches ($\sum_{i=0}^{\tau} v_{C,i} \leq 1$, default: τ=2).

A.3 Computing Parent and Abelian group hashing

The encodings of colors in $\mathbf{C}$ plays an important role in efficient computation of $\textsc{Parent}(\mathbf{G}, C_1, C_2)$. A desired encoding should enable fast computation of $C = C_1 \cup C_2$, and we also need to query if C was already added to the multitree, G. We do this by using an Abelian group hashing scheme for sets, i.e., colors.

Using a hash function H with a sufficiently large range, we assign a hash value to each singleton color $H(\{r\})$, $r \in \mathcal{R}$. Then, it is possible to form an Abelian group by setting non-singleton colors hash values to $H(C) = \sum_{r \in C} H(\{r\})$. Since we only compute Parent of disjoint colors, we could obtain $H(C)$ by summing $H(C_1) + H(C_1)$. Thus, $\mathbf{G}$ could be stored in an associative array where we keep $H(C)$ as the key and either $H(C_1)$ or $H(C_2)$ as the value since one can always compute the other by subtraction. When $C \notin \mathbf{G}$, we simply add it, and return $H(C_1) + H(C_2)$. When $C \in \mathbf{G}$, it could be that it is either already seen due to another k-mer or it is a collision. We check collisions by comparing the value associated with $H(C)$ and $H(C_1)$, $H(C_2)$. Despite being extremely rare, collisions are possible. We handle collisions by adding a dummy reference, i.e., a nonce, to the color C. At the beginning, we ensure that there is no collision among colors corresponding to internal nodes (and singletons) by rehashing.

At the end, in order to avoid storing redundantly large hash values, we simply convert colors to integers by enumerating them (using $\log(|\mathcal{C}|)$ bits) and store them in an array where indices correspond to color encodings, instead of an associative array.

Appendix B Derivative of the likelihood function

Given, $u_r, \mathbf{v}_r, \delta, k, h$ and ρ_r, we can compute the derivate of the log-likelihood, Eq. (2), with respect to D as follows:

$$\begin{aligned}
&\frac{\mathrm{d}}{\mathrm{d}D}\left(u_r \log\left(P_{\text{miss}}(D;k,h,\delta)\right) + \sum_{d=0}^{\delta} v_{r,d}\left(d\log(D) + (k-d)\log(1-D)\right)\right) \\
&\quad = \frac{\mathrm{d}}{\mathrm{d}D}\log\left((1-\rho_r) + \rho_r \sum_{d=0}^{k} P_{\text{mutate}}(D;d,k)\left(\mathbb{1}\{d>\delta\} + \mathbb{1}\{d\le\delta\}\left(1 - P_{\text{collide}}(d,k,h)\right)\right)\right) \\
&\quad + \sum_{d=0}^{\delta} v_{r,d}\left(d\frac{1}{D} + (d-k)\frac{1}{1-D}\right).
\end{aligned}$$

For convenience, we define the following function $P_\delta(d)\colon [0,k] \to [0,1]$,

$$P_\delta(d) = \mathbb{1}\{d>\delta\} + \mathbb{1}\{d\le\delta\}\left(1 - \binom{k-h}{d}\Big/\binom{k}{d}\right). \tag{3}$$

where $1 - \binom{k-h}{d}/\binom{k}{d}$ is simply $1 - P_{\text{collide}}(d,k,h)$.

Then, we can substitute $P_{\text{mutate}}(D;d,k)$ with $D^d(1-D)^{(k-d)}\binom{k}{d}$ and compute the derivative of the logarithm,

$$\begin{aligned}
&= \frac{\rho_r u_r\left(\sum_{d=0}^{k} D^d(1-D)^{k-d}\binom{k}{d}\left(\frac{d-k}{1-D} + \frac{d}{D}\right)P_\delta(d)\right)}{1-\rho_r+\rho_r\left(\sum_{d=0}^{k} D^d(1-D)^{k-d}\binom{k}{d}P_\delta(d)\right)} + \sum_{d=0}^{\delta} v_{r,d}\left(d\frac{1}{D} + (d-k)\frac{1}{1-D}\right) \\
&= \frac{\rho_r u_r\left(\sum_{d=0}^{k} D^d(1-D)^{k-d}\binom{k}{d}\left(\frac{d-kD}{D(1-D)}\right)P_\delta(d)\right)}{1-\rho_r+\rho_r\left(\sum_{d=0}^{k} D^d(1-D)^{k-d}\binom{k}{d}P_\delta(d)\right)} + \sum_{d=0}^{\delta} v_{r,d}\left(\frac{d-kD}{D(1-D)}\right).
\end{aligned}$$

The sign of this derivative is positive at $0+\epsilon$, and negative at $1-\epsilon$, which is required for the Brent's method to work and find a local optima.

Second derivative is given by

$$\begin{aligned}
&\frac{\mathrm{d}^2}{\mathrm{d}D^2}\left(u_r \log\left(P_{\text{miss}}(D;k,h,\delta)\right) + \sum_{d=0}^{\delta} v_{r,d}\left(d\log(D) + (k-d)\log(1-D)\right)\right) \\
&\quad = \frac{\mathrm{d}}{\mathrm{d}D}\left(\frac{\rho_r u_r\left(\sum_{d=0}^{k} D^d(1-D)^{k-d}\binom{k}{d}\left(\frac{d-kD}{D(1-D)}\right)P_\delta(d)\right)}{1-\rho_r+\rho_r\left(\sum_{d=0}^{k} D^d(1-D)^{k-d}\binom{k}{d}P_\delta(d)\right)}\right) + \sum_{d=0}^{\delta}\frac{\mathrm{d}}{\mathrm{d}D}\left(v_{r,d}\left(\frac{d-kD}{D(1-D)}\right)\right) \\
&\quad = \frac{\mathrm{d}}{\mathrm{d}D}\left(\frac{\rho_r u_r\left(\sum_{d=0}^{k} D^d(1-D)^{k-d}\binom{k}{d}\left(\frac{d-kD}{D(1-D)}\right)P_\delta(d)\right)}{1-\rho_r+\rho_r\left(\sum_{d=0}^{k} D^d(1-D)^{k-d}\binom{k}{d}P_\delta(d)\right)}\right) + \sum_{d=0}^{\delta} v_{r,d}\left(\frac{d(2D-1)-kD^2}{D^2(1-D)^2}\right) \\
&\quad = \frac{g'(D)f(D) - f'(D)g(D)}{\left(1-\rho_r+\rho_r\left(\sum_{d=0}^{k} D^d(1-D)^{k-d}\binom{k}{d}P_\delta(d)\right)\right)^2} + \sum_{d=0}^{\delta} v_{r,d}\left(\frac{d(2D-1)-kD^2}{D^2(1-D)^2}\right),
\end{aligned}$$

where

$$g(D) = \rho_r u_r\left(\sum_{d=0}^{k} D^d(1-D)^{k-d}\binom{k}{d}\left(\frac{d-kD}{D(1-D)}\right)P_\delta(d)\right),$$

and

$$f(D) = 1 - \rho_r + \rho_r \left(\sum_{d=0}^{k} D^d (1-D)^{k-d} \binom{k}{d} P_\delta(d) \right).$$

Here, the denominator is always positive, and the second term coming from P_{match} is non-positive for $D \in (0, 0.5)$. Derivatives of $f(D)$ and $g(D)$ are given by

$$g'(D) = \rho_r u_r \left(\sum_{d=0}^{k} (1-D)^{k-d} D^d \left(\frac{(d^2 + (k-1)kD^2 - d\,(1 + (k-1)2D))}{D^2(1-D)^2} \right) \binom{k}{d} P_\delta(d) \right),$$

and

$$f'(D) = \rho_r \left(\sum_{d=0}^{k} D^d (1-D)^{k-d} \left(\frac{d - kD}{D(1-D)} \right) \binom{k}{d} P_\delta(d) \right).$$

Thus, $f'(D)g(D)$ is always non-negative:

$$u_r \rho_r^2 \left(\sum_{d=0}^{k} D^d (1-D)^{k-d} \left(\frac{d - kD}{D(1-D)} \right) \binom{k}{d} P_\delta(d) \right)^2.$$

It suffices to show $g'(D)f(D) \leq 0$ to prove the concavity of the log-likelihood. Notice that, we can simplify the entire second derivative without changing the sign by multiplying all terms with $D^2(1-D)^2$. Furthermore, since $f(D) > 0$ for $\rho_r \in [0, 1]$ and $D \in (0, 0.5)$, we can focus on the following quantity:

$$\sum_{d=0}^{k} (1-D)^{k-d} D^d \left(d^2 + (k-1)kD^2 - d\,(1 + (k-1)2D) \right) \binom{k}{d} P_\delta(d),$$

which is equal to 0 without $P_\delta(d)$:

$$\sum_{d=0}^{k} (1-D)^{k-d} D^d \left(d^2 + (k-1)kD^2 - d\,(1 + (k-1)2D) \right) \binom{k}{d} = 0.$$

This can be shown by

$$\begin{aligned}
&\sum_{d=0}^{k} (1-D)^{k-d} D^d \left(d^2 + (k-1)kD^2 - d\,(1 + (k-1)2D) \right) \binom{k}{d} \\
&= \sum_{d=0}^{k} \left(d^2 (1-D)^{k-d} D^d \binom{k}{d} \right) - ((2k-2)D + 1) \sum_{d=0}^{k} \left(d (1-D)^{k-d} D^d \binom{k}{d} \right) \\
&+ k(k-1)D^2 \sum_{d=0}^{k} \left((1-D)^{k-d} D^d \binom{k}{d} \right) \\
&= \mathbb{E}(\mathrm{X}^2) + (D(2k-2) + 1)\, \mathbb{E}(\mathrm{X}) + k(k-1)D^2(1) \\
&= \left(k^2 D^2 + (1-D)kD \right) + (D(2k-2) + 1)\,(kD) + k(k-1)D^2 = 0,
\end{aligned}$$

where $X \sim \text{Binomial}(k, D)$.

Finally, to conclude that $g'(D) \leq 0$, we need to show the following inequality holds after including $P_\delta(d)$ back:

$$\sum_{d=0}^{\delta}(1-D)^{k-d}D^d\left(d^2+(k-1)kD^2-d\left(1+(k-1)2D\right)\right)\binom{k-h}{d} \geq 0. \tag{4}$$

This inequality is not true for all choices of δ, h and k, but the default parameter values k=29, h=14, δ=4 satisfy it for $D \in (0, 0.5)$

Appendix C Genome evolution simulations

Starting from a genome X as the base, we added mutations to it at random using the Jukes-Cantor model with Gamma model of rate heterogeneity, with the added caveat that gene boundaries were respected (i.e., stop codons were not disturbed) and mutations fall only on the genes and not the $\approx$5% of the genome that constitutes intergenic regions. We then computed the genomic nucleotide distance (GND) from mutated genome, X', to X by counting the actual number of mutations added during the simulation, divided by the length (including intergenic regions). We mutated X at various nucleotide diversity levels ($D \in \{0\%, 0.05\%, 0.1\%, 0.2\%, \ldots, 1\%, 2\%, 3\%, \ldots, 19\%\}$) to obtain X_0, X_1, X_2, ..., X_{30}. We then compared X with all X_i ($i \in [0, 30]$) genomes. For X, we used 29 bacterial single-cell assembled genomes from the GORG dataset (`https://doi.org/10.1016/j.cell.2019.11.017`), selected to have high levels of completion.

To create rate heterogeneity, we draw a relative rate multiplier r_j for each gene j of X from a Gamma distribution with mean 1 and variance $1/\alpha$ (here, α=5), keeping rates the same across all GND levels. We ensured that each gene in the base genome X is assigned a fixed rate, which is used for all X_is. We then randomly selected $n = D \cdot L$ nucleotide positions (sampling without replacement) to mutate, selecting each position with a probability proportional to the rate r_j of the gene j it belongs to. When placing mutations, we avoid adding or removing both start and stop codons to keep gene boundaries intact. We also avoided interrupting the reading frame. Overlapping areas between genes do get mutated only if they do not change the start and stop codons. The rate multipliers of these overlapping areas are randomly selected among the genes that include them.

Finally, we simulated 7.5 million error-free reads using ART [42] from 899 (29×31) mutated genomes and used coordinates of simulated reads to compute the true Hamming distance.

Appendix D Query selection in WoL-v2 with controlled novelty

We clustered leaves using TreeCluster and set length threshold parameter to the following values: 0.25, 0.1, 0.075, 0.05, 0.04, 0.03, 0.02, 0.01, 0.005, 0.001, 0.0005, and 0.0001. Each of these threshold values result in a clustering of leaves, where some of the leaves remain unclustered as singletons. For all threshold values except 0.25, we selected all singletons that are not singletons with a larger threshold value. Among these, we sampled 10 leaves at random for each level, resulting in 110 query genomes we used for WoL-v2 experiments.

Appendix E Why not APPLES?

Reads come from various parts of the genome, and the rates of evolution and, thus, branch lengths change across the genome. For the least square method APPLES [39] to work well, distances from query to references should be in the same scale as distances among references. This observation was one of the main insights from the original papers. Having consistent estimates of branch length necessitates recomputing distances among references for each read, which is possible with our data structure. However, it will make placement too slow as it would need $O(kn^2)$ computations for k queries and n references, instead of the desired $O(kn)$, provided by krepp. Ignoring changes in rates reduces accuracy and is not a viable option [39].

Appendix F Details for human microbiome analysis

Human microbiome samples analyzed consist of the following seven body sites: stool ($n = 78$), tongue dorsum ($n = 42$), supragingival plaque ($n = 33$), buccal mucosa ($n = 28$), retroauricular crease ($n = 13$), posterior fornix ($n = 10$), and anterior nares ($n = 6$). We used 1M 100-bp paired-end whole-genome sequencing (WGS) reads subsampled in Zhu et al. [22]. krepp processes paired-end reads separately and estimates distances independently.

Feature tables from outputs of all three tools were constructed via `woltka classify` command. For PERMONOVA test and UniFrac computation, we used QIIME2 [54] and its diversity plugin [47], setting sampling depth to 100K, the number of permutations to 999.

Supplementary Figures and Tables

Taxon	Rank	# of reference	Reference size	# of queries	# of query reads
Piscirickettsiaceae	family	40	119 Mbps	12	600K
Bacteroides	genus	40	171 Mbps	13	350K
Moraxella	genus	40	232 Mbps	14	700K
Bartonella	genus	50	93 Mbps	10	560K
Mycobacterium	genus	40	232 Mbps	21	1.05M
Rhizobiaceae	family	50	235 Mbps	10	475K

Table S1: Details of small reference sets consisting of a single taxon, and their corresponding query sets. Query genomes are simply the remaining genomes after excluding randomly selected genomes from WoL-v1.

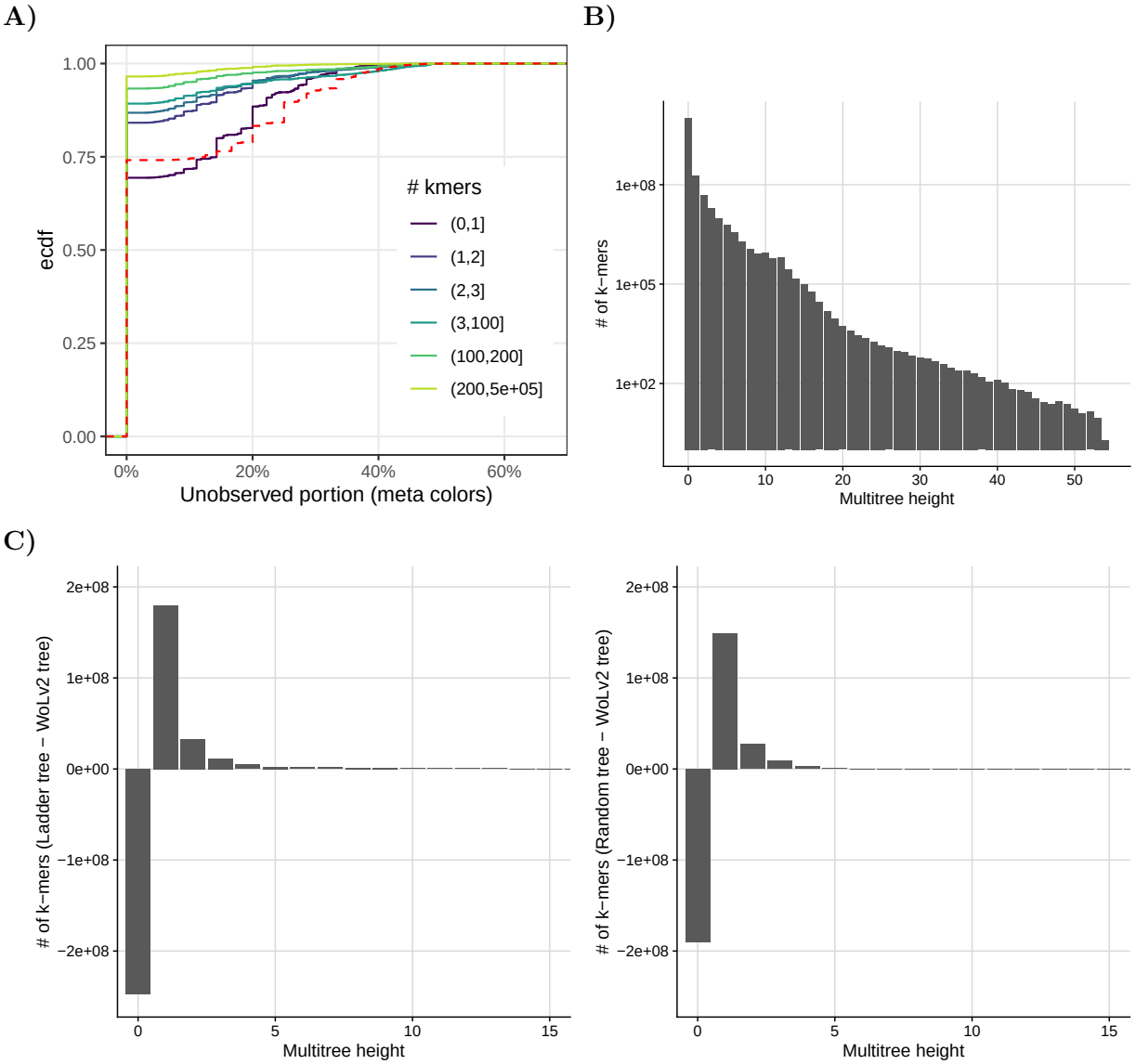

Figure S1: A) For every color in the WoL-v2 reference dataset with 15,953 genomes, we visit the colors below it on the multitree; we then count what percentage of these colors are meta-colors (those not associated with any k-mer). We show the empirical cumulative distribution of portion of colors under each color are unobserved. We divide this based on how many k-mers are labelled by each color, showing the undivided data in the dotted red line. B) The histogram of the number of k-mers with a certain height. Most k-mers belong to colors with height 0, 1, or 2. C) Comparing the number of k-mers with colors with certain multitree heights when different trees are used. The differences of number of k-mers between: a random ladder tree and WoL-v2 tree (left), a random dual-birth model tree ($\lambda_A = 10$, $\lambda_B = 1$) [35] and WoL-v2 tree (right) are shown.

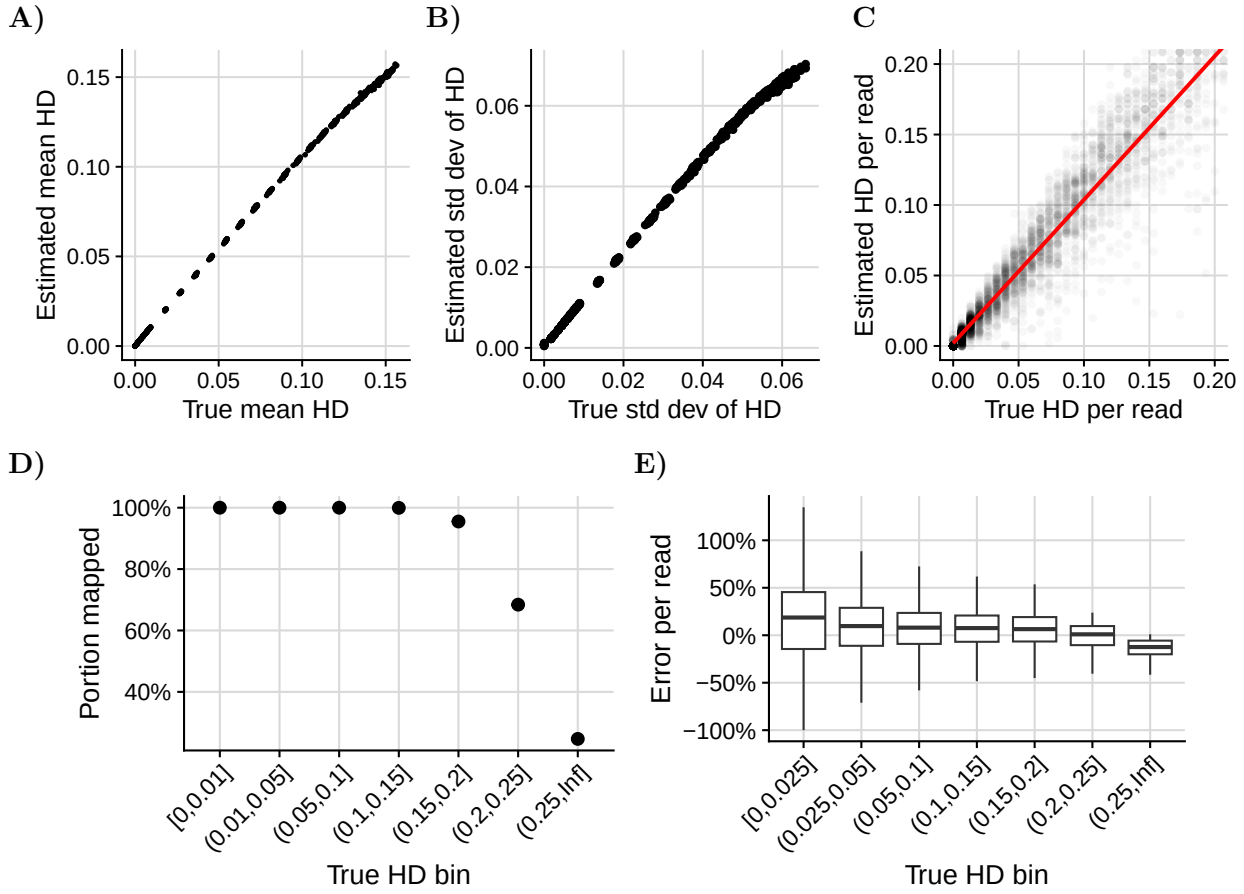

Figure S2: A) Comparing true Hamming distance (normalized by the read length) and the estimated HD for individual 150bp short reads. Each data point is a read and the red line is fit using a linear model. B,C) Each data point is a mutated genome, and the mean and standard deviation values are computed across all reads that krepp is able to map to the corresponding base genome. D) Percentage of reads mapped for mutated genomes in each distance bin on average. E) Read-level percentage error distributions for each true HD bin, demonstrating small overestimation bias, especially in low Hamming distances.

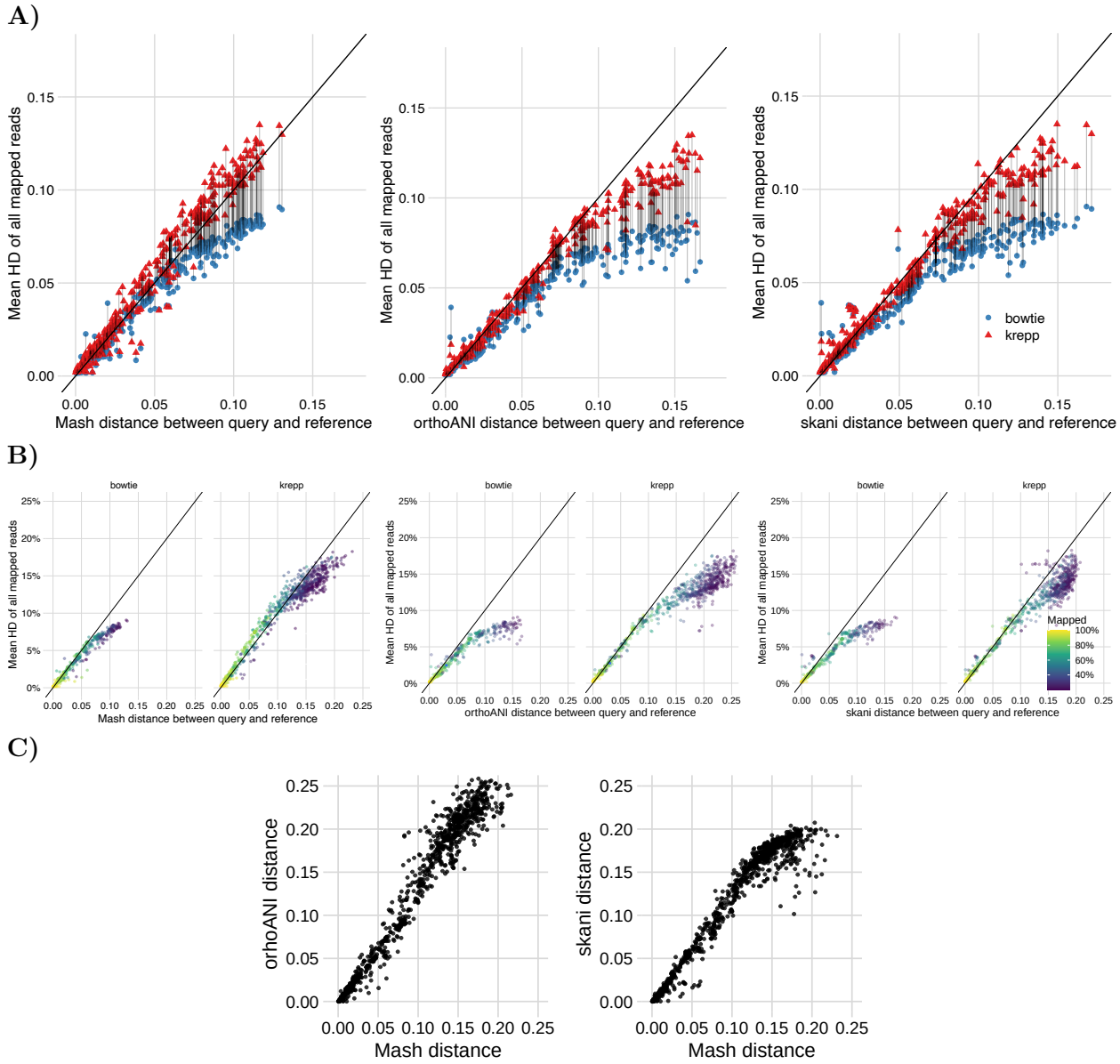

Figure S3: A) Mean distance across reads ($\bar{D}$) versus orthoANI [44] and skani [43] genome-wide distance estimates $\mathcal{D}$ for query/ref genome pairs with $\geq$20% reads mapped by both methods. B) $\bar{D}$ versus orthoANI and skani distance estimates $\mathcal{D}$ from each query to all references with at least 20% reads mapped (colors). C) Comparing genome-wide ANI estimates of Mash with orthoANI and skani. We set `--min-af` parameter of skani to 0 to output distances regardless of the aligned fraction value. All other parameters of orthoANI and skani are set to defaults.

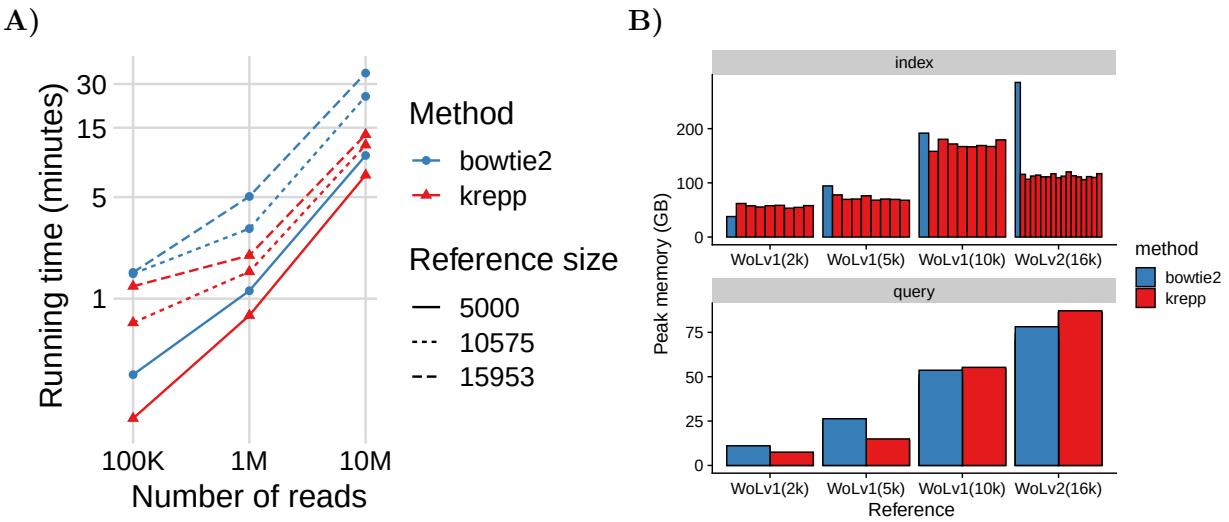

Figure S4: Scaling. A) Running time versus the number of reads. Both methods scale similarly. B) Memory of krepp and bowtie2 for four reference datasets with 2000–15,953 references (size shown parenthetically). For indexing, krepp works in batches; we show batches separately. Note that the largest dataset uses more batches and thus uses less memory per batch.

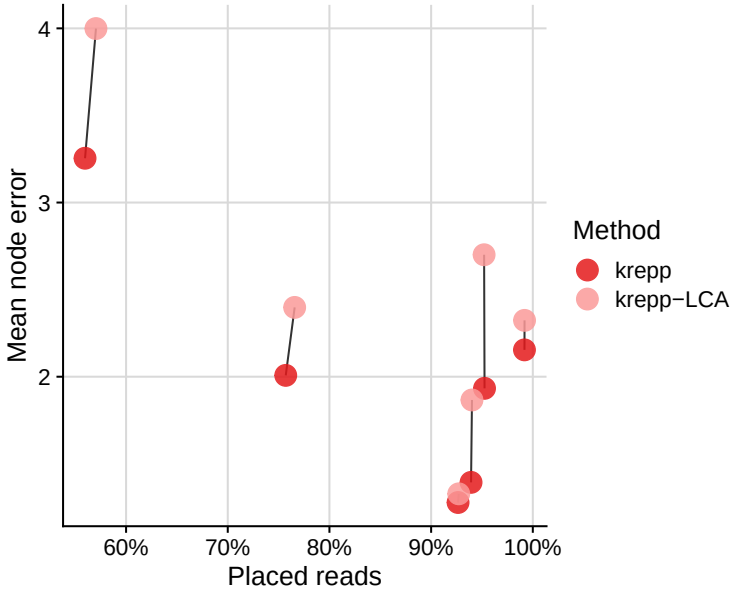

Figure S5: Using krepp distances and statistical test of distinguishability, we can find the set of leaves that are all tied with the closest distance to the query; the read can then be placed as sister to the lowest common ancestor (LCA) of these leaves. This approach, shown here, increases the error compared to our default algorithm.

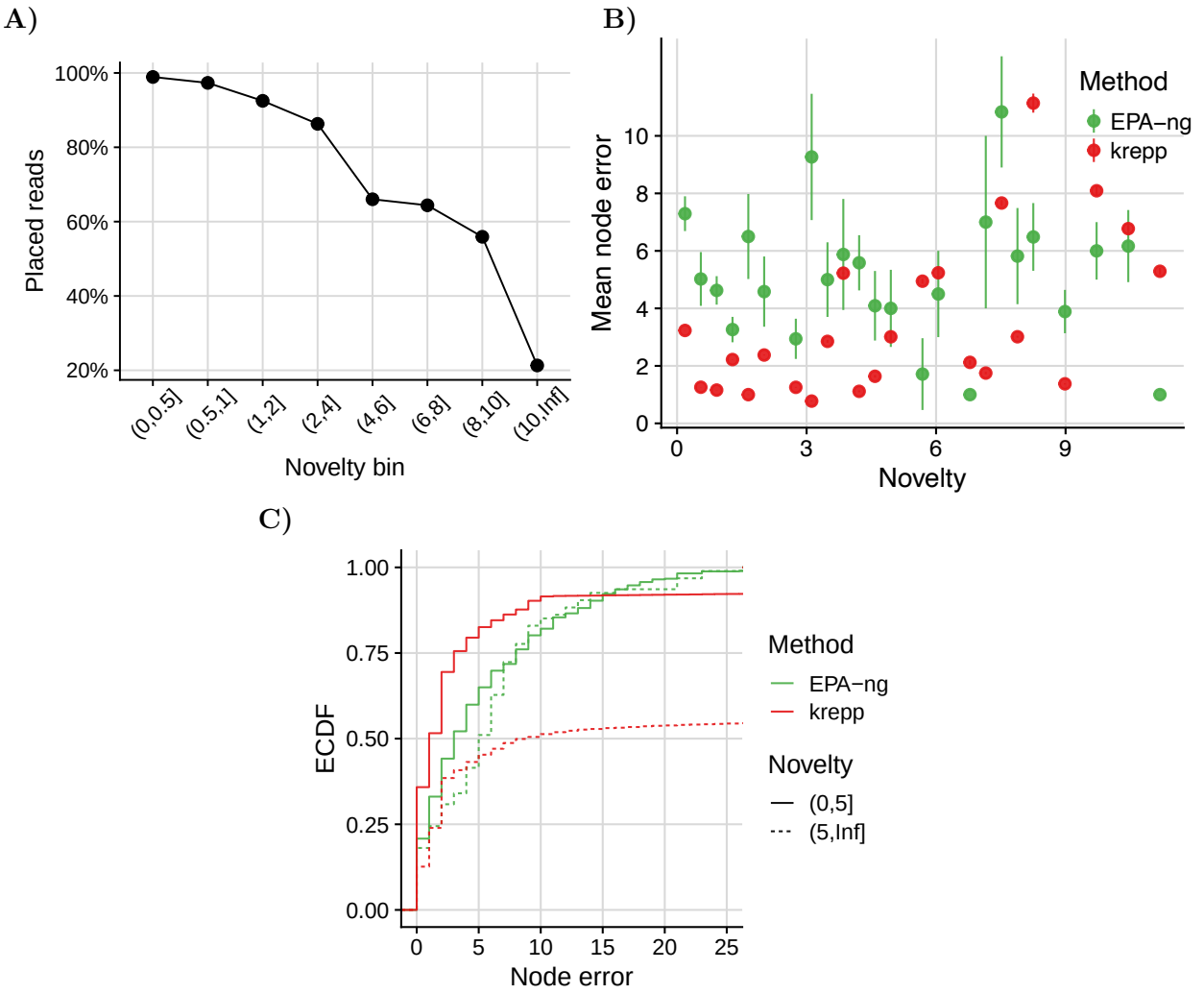

Figure S6: A) Percentage of reads placed for queries selected from WoL-v1 across varying novelty levels. We observed reduced percentages of reads placed compared to using more densely-sampled WoL-v2 as the reference. B) Mean node errors of EPA-ng and krepp across different novelty levels, excluding all unplaced reads (i.e., with distance > 0.2, placed at the root, or has no matches). C) ECDF of node errors in two different novelty bins after replacing errors of all unplaced reads with infinity.