MetaGraph-MLA: Label-guided alignment to variable-order De Bruijn graphs

Harun Mustafa[1,2,3][0000−0002−2125−6086], Mikhail Karasikov[1,2,3][0000−0001−6200−5972], Gunnar Rätsch[1,2,3,4][0000−0001−5486−8532], and André Kahles[1,2,3,*][0000−0002−3411−0692]

[1] ETH Zurich, Rämistrasse 101, 8092 Zurich, Switzerland
[2] University Hospital Zurich, Rämistrasse 100, 8091 Zurich, Switzerland
[3] Swiss Institute of Bioinformatics, Zurich, Switzerland
[4] ETH AI Center, Binzmühlestrasse 13, 8092 Zurich, Switzerland
{harun.mustafa,mikhaika,gunnar.raetsch,andre.kahles}@inf.ethz.ch
[*] to whom correspondence should be addressed

Abstract. The amount of data stored in genomic sequence databases is growing exponentially, far exceeding traditional indexing strategies' processing capabilities. Many recent indexing methods organize sequence data into a sequence graph to succinctly represent large genomic data sets from reference genome and sequencing read set databases. These methods typically use De Bruijn graphs as the graph model or the underlying index model, with auxiliary graph annotation data structures to associate graph nodes with various metadata. Examples of metadata can include a node's source samples (called *labels*), genomic coordinates, expression levels, etc.

An important property of these graphs is that the set of sequences spelled by graph walks is a superset of the set of input sequences. Thus, when aligning to graphs indexing samples derived from low-coverage sequencing sets, sequence information present in many target samples can compensate for missing information resulting from a lack of sequence coverage. Aligning a query against an entire sequence graph (as in traditional sequence-to-graph alignment) using state-of-the-art algorithms can be computationally intractable for graphs constructed from thousands of samples, potentially searching through many non-biological combinations of samples before converging on the best alignment.

To address this problem, we propose a novel alignment strategy called *multi-label alignment* (MLA) and an algorithm implementing this strategy using annotated De Bruijn graphs within the MetaGraph framework, called *MetaGraph-MLA*. MLA extends current sequence alignment scoring models with additional *label change* operations for incorporating mixtures of samples into an alignment, penalizing mixtures that are dissimilar in their sequence content. To overcome disconnects in the graph that result from a lack of sequencing coverage, we further extend our graph index to utilize a variable-order De Bruijn graph and introduce *node length change* as an operation. In this model, traversal between nodes that share a suffix of length $< k - 1$ acts as a proxy for inserting nodes into the graph.

MetaGraph-MLA constructs an MLA of a query by chaining single-label alignments using sparse dynamic programming. We evaluate *MetaGraph-MLA* on simulated data against state-of-the-art sequence-to-graph aligners. We demonstrate increases in alignment lengths for simulated viral Illumina-type (by 6.5%), PacBio CLR-type (by 6.2%), and PacBio CCS-type (by 6.7%) sequencing reads, respectively, and show that the graph walks incorporated into our MLAs originate predominantly from samples of the same strain as the reads' ground-truth samples. We envision *MetaGraph-MLA* as a step towards establishing sequence graph tools for sequence search against a wide variety of target sequence types.

Keywords: Sequence-to-graph alignment · Metagenomics · Alignment scoring models.

1 Introduction

Owing to the high computational and human labor burden of genome assembly, a large fraction of sequencing data has yet to be assembled [34]. For example, as of February 2022, the doubling time of assembled sequences in the European Nucleotide Archive is 59.7 months, compared to 18.4 months for sequencing runs [22]. Thus, this wealth of data is, in practice, inaccessible to most researchers, and standard tools need to be established for indexing these data. *Sequence graphs* are an increasingly prominent model for representing and indexing large collections of sequencing data [43,18], enabling improvements in both the scale and accuracy of many biological analysis tasks (e.g., genotyping [57,25], variant calling [36,13], sequence search [32,57]).

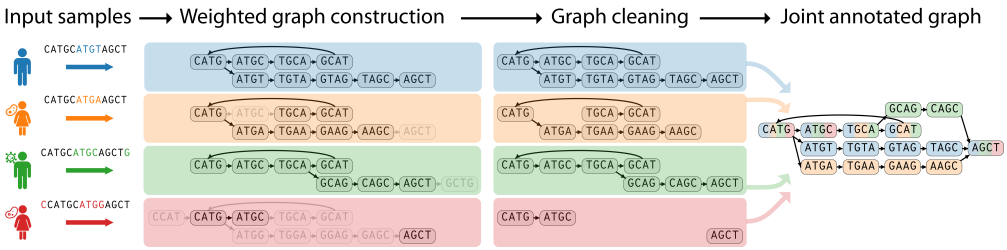

Fig. 1. Construction of an annotated De Bruijn graph from input sequence samples. We construct a separate De Bruijn graph for each input sample, with nodes weighted by k-mer abundance. We then clean each sample graph into an assembly graph by removing insufficiently-abundant k-mers (depicted as shaded graph nodes). Finally, we construct a joint annotated De Bruijn graph from the contigs of the assembly graphs, labeling each node with their samples of origin (depicted as node colors).

Any sequence in the input can be represented by a walk in the sequence graph, where the corresponding sequence of a walk is called its *spelling*. Sequence graphs of nucleotides are also called *genome graphs*.

While a sequence graph by itself can be used to check for the presence or absence of a query sequence within a set, it cannot *classify* or *profile* the query without additional metadata, called graph *annotations*. Graph annotations are a *key-value* store associating each graph node with a number of annotations, where annotations can include the *labels* of the indexed *samples* [32,49,28,26], node abundances [33,44], genomic coordinates [33,1,20], geographic coordinates [32], etc. [59]. Examples of annotated sequence graphs include numerous types of annotated De Bruijn graphs [28,11,35,8,7,6] and annotated variation graphs [56,20]. A *De Bruijn graph* is a graph where each node v_i is a substring of length k (called a *k-mer*) that has been observed in at least one input sequence (Section 2.1 for the full definition). For jointly indexing unassembled read sets, annotated De Bruijn graphs typically scale better than variation graphs in representation size due to the collapse of shared k-mers between samples onto single graph nodes [32]. Before joint graph construction, many indexing tools [32,33,49,58,55,28,59,28] perform error correction (*cleaning*) on each sample to remove *uncertain k-mers* [62]. Thus, these joint graphs can represent the samples' respective *assembly graphs* [32].

One of the inherent limitations of De Bruijn graphs is the intimate connection between the choice of k and the graph topology. Setting k too small leads to a graph topology where the majority of walks will spell sequences that are not present in the underlying genome and do not represent biologically-viable recombination [57] (called *spurious walks* [33]), while setting a large value will reduce the potential for shared substrings to collapse onto shared nodes, and hence, lead to lower coverage and more sequence content being removed during cleaning. *Variable-order De Bruijn graphs* [7] of *maximum order k* overcome this limitation by representing all observed k'-mers, where $1 < k' \leq k$, including additional edges for transitioning between nodes of different k' values as long as they share a common suffix. A number of recent assembly and cleaning strategies feature variable-order De Bruijn graphs, using them to adjust the value of k depending on coverage [5] or to provide more robust generalizations of contigs during genome assembly [15].

1.1 Sequence-to-graph alignment

Sequence graphs are becoming increasingly relevant for sequence analysis. In particular, their ability to more scalably represent diverse collections of sequences partially addresses the reference bias issues common in current standard analysis protocols [10]. To allow for sensitive sequence search in this new domain, many tools generalize pairwise sequence-to-sequence alignment (i.e., computing the minimum edit distance or the maximum similarity *score* between a *query* and a *target* sequence) algorithms to align queries against sequence graphs [1,2,54,33,29,16,37,51,52,16]. These *sequence-to-graph* alignment tools perform a search on the graph to align a query against the spellings of one or more walks. Depending on their choice of scoring model, these tools can be categorized into those that use the graph as a compact representation of disjoint reference sequences for sequence-to-sequence alignment [1,2,33], those that align to species pangenomes [20,54,50,41], or those that treat the entire graph as a single sample [30,24,40,16,29,51,52,16]. Throughout this work, we use the convention that alignment scores measure sequence similarity, and hence, higher values are better.

A prominent strategy for approximating sequence-to-sequence and sequence-to-graph alignment is *seed-and-extend* [3], entailing three main steps: 1) *seed anchoring* or *seeding*, 2) *anchor filtration*, and 3) *anchor*

extension. In the context of sequence-to-graph alignment, a *seed*, generally, is a subsequence of the query that has a match in the graph. Many tools use a more strict definition, where a seed is a substring of the query that exactly matches a substring of a walk spelling. An *anchor* (sometimes called a *match*) is a tuple of the seed and the graph walk whose spelling subsequence matches the seed. Anchor filtration selects a subset of anchors that are likely to lead to a high-scoring alignment. Finally, anchor extension is the alignment of the query to the graph via forward and backward search from the ends of each anchor. A common anchor filtration method is *co-linear chaining* [38,39,2,33,42,9], a dynamic programming algorithm that finds high-scoring anchor chains. A *chain* is a series of anchors that appear in the correct order with respect to the query such that each anchor can reach the subsequent anchor in the chain via graph traversal. A chain is scored more favorably if it contains more anchors and is penalized if the distances between the anchors in the query differ from their corresponding graph traversal distances [39,33,2].

1.2 Challenges when aligning to sets of assembly graphs

A large proportion of SRA samples, in particular large metagenomics projects such as MetaSUB, are sequenced at low coverage, typically sequencing their constituent organisms at or below $1\times$ coverage [14]. A k-mer that is missing from a sample with the label ℓ due to low coverage can manifest itself in an annotated sequence graph in one of two ways: it can either 1) be present in the graph but not be annotated with ℓ, or it can be 2) missing from the graph entirely. To address the first case, sequence-to-graph aligners that do not take label information into account can align to these samples since the k-mer still exists in the graph. However, this approach poses a higher computational burden for alignment due to the greater search space [57,33] and may produce alignments to non-biological walks [57]. In the second case, the entire query may only be alignable if the graph is reconstructed with a smaller k, or if nodes can be inserted into the graph as an allowable edit operation. Sequence-to-graph alignment with graph edits is known to be $\mathcal{NP}$-complete [31,21], so heuristics become an even greater necessity.

1.3 Label-guided sequence-to-graph alignment using MetaGraph

In this work, we wish to address the challenges of aligning to assembly graphs while maintaining competitive alignment speed and accuracy. We propose a novel *multi-label alignment* (MLA) strategy, incorporating sequences from multiple samples in a score-guided manner to compensate for gaps within the assembly graph of any given label. To tractably implement this strategy, we propose a two-step approach (Figure 2): *(i)* a new algorithm *MetaGraph-LA* for aligning query sequences to walks that share at least one common label [41] (called *label-consistent alignments*) and *(ii)* a chaining algorithm *MetaGraph-MLA* for constructing MLAs from label-consistent alignments of the query computed by *MetaGraph-LA*. *MetaGraph-MLA* compensates for coverage gaps in the graph by both using walks from other samples and by allowing for traversal to nodes that share a suffix of length $< k-1$ to compensate for k-mers missing completely from the graph. We introduce a novel scoring model for MLA to add penalties (i.e., to subtract from the score) for both changes in the label and the graph k during the search.

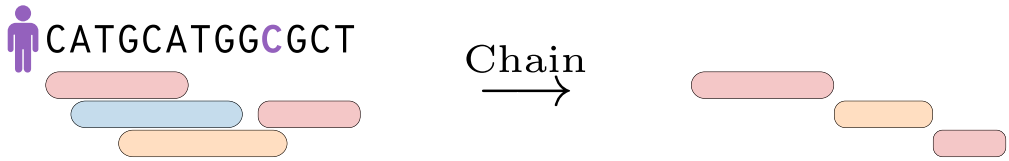

Fig. 2. Computing multi-label alignment (MLA) chains using *MetaGraph-MLA* **from an anchor set of label-consistent alignments computed by** *MetaGraph-LA* **(left).** When incorporating an alignment a into an existing chain ending with alignment a', we trim the suffix of a' and the prefix of a to ensure that they do not overlap before adding a to the chain. See Section 2.3 for more details.

The rest of this manuscript is structured as follows: After briefly introducing our notation (Section 2.1), we outline the alignment strategies and algorithms for label-guided alignment within the *MetaGraph* framework (Sections 2.2-2.3) along with our underlying scoring model (Section 2.4). Afterwards, we outline the methodology for evaluating our alignment strategy (Section 3). Section 4 presents our evaluation of *MetaGraph-LA*

and *MetaGraph-MLA* against other state-of-the-art alignment tools, demonstrating that *MetaGraph-MLA* produces longer alignments with the same median accuracy as other sequence-to-graph alignment methods. Section 5 closes by summarizing our findings and remarking on possible directions for future work.

2 Methods

2.1 Notation and Preliminaries

This section introduces the notation that we use for strings, graphs, and alignments. A *string* is a finite sequence of characters drawn from an alphabet Σ, where Σ^k denotes the set of all strings of length k (called *k-mers*) and Σ^* denotes the set of all strings over the alphabet. Given a string denoted as $s = s^1 s^2 \cdots s^l$ of length l (denoted $|s| = l$), with indices $1 \leq i \leq j \leq l$, the notation $s[i]$ denotes s^i and $s[i:j]$ denotes the substring $s^i \cdots s^j$. Let $S = \{s_1, \ldots, s_n\}$ be a string set. The *k-mer spectrum* of S is the set $\mathcal{K}(S,k) \subseteq \Sigma^k$ of all k-mers drawn from each $s \in S$. More precisely, $\mathcal{K}(S,k) := \bigcup_{s \in S} \bigcup_{i=1}^{|s|-k+1} \{s[i : i+k-1]\}$.

A *graph* is a (node set, edge set) pair, typically denoted by $G = (V, E)$, where $E \subseteq V \times V$. A *walk* is a sequence of nodes $(n_1, \ldots, n_m) \in V^m$, where $m \geq 1$ and for each $1 \leq i < m$, $(n_i, n_{i+1}) \in E$. A *node-centric De Bruijn graph* of order k for the string set S is the graph $DBG(S,k) := (V, E(V))$ with node set $V := \mathcal{K}(S,k)$ and edge set $E(V) := \{(v_1, v_2) \in V^2 : v_1[2:k] = v_2[1:k-1]\}$ [12]. The *spelling* of a walk $(n_1, \ldots, n_m)$ on a sequence graph is the sequence represented by the walk. If the graph is a De Bruijn graph, the spelling is $n_1 n_2[k] \cdots n_m[k]$. An *annotated De Bruijn Graph* has an auxiliary set of labels $\mathcal{L}$ (typically strings) and an annotation $\mathcal{A} : V \to 2^{\mathcal{L}}$ that associates each graph node with a set of labels. A walk $W = (n_1, \ldots, n_d)$ is *label consistent* if $L_W := \bigcap_{i=1}^{d} \mathcal{A}(n_i) \neq \emptyset$.

We use Q to denote a user-provided query sequence in Σ^* of length at least 1. Given a *target sequence* T of length at least 1, an *alignment* a of Q to T is a tuple (Q_a, T_a, E_a), where Q_a is a substring of Q, T_a is a substring of T, and E_a is a sequence of *edit operations* needed to transform Q_a to T_a. These operations are drawn from the set $\{\text{match}, \text{mismatch}, \text{insertion open}, \text{insertion extension}, \text{deletion open}, \text{deletion extension}\}$. Each operation incurs a *score*, denoted by $\Delta_=$, $\Delta_\neq$, Δ_{IO}, Δ_{IE}, Δ_{DO}, and Δ_{DE}, respectively. If a score is negative, we call it a *penalty*. The overall score of the alignment is the sum of all edit operation scores: $\Delta_S(a) := \sum_{\Delta_E \in E_a} \Delta_E$, where a greater score indicates greater similarity. A *sequence-to-graph* alignment has an additional element W, a walk in a graph whose spelling is a superstring of T_a. When the graph is an annotated De Bruijn graph, a is said to be a *label-consistent alignment* if its walk is label consistent. Then, the label set of the alignment is $L_a := L_W$. Given indices $1 \leq i_1 \leq i_2 < |Q|$ and substring lengths $1 < l_1, l_2 \leq |Q|$, and two alignments a_1, a_2 mapping their respective substrings $Q[i_1 : i_1 + l_1 - 1], Q[i_2 : i_2 + l_2 - 1]$ to G, we define $\text{overlap}(a_1, a_2) := \min\{l_2, i_1 + l_1 - i_2\}$. a_1 and a_2 *overlap* if $\text{overlap}(a_1, a_2) > 0$ and are *disjoint* by a *gap* of length $-\text{overlap}(a_1, a_2)$ otherwise. Given a gap length $l \in \mathbb{N}$, we denote the *gap penalty* prescribed by an edit scoring model by $\Delta_G(l)$.

2.2 MetaGraph-LA: Generating anchors for MLA via label-consistent alignment

Our algorithm for computing label-consistent alignments, called *MetaGraph-LA*, is a seed-and-extend algorithm incorporating methods from *MetaGraph-Align* [32] and the *TCG-Aligner* [33]. Given a seed length $l \leq k$ and a query sequence Q, we extract each l-mer from Q and find all graph nodes with matching l-length suffixes and fetch their corresponding labels. Each query l-mer with a matching node is called a *seed*. Each $(Q[i, i+l-1], n, \ell) \in \Sigma^l \times V \times \mathcal{L}$ tuple is called a seed *anchor*. Note that since a seed can map to multiple nodes, a single seed can produce several anchors. For our initial label and anchor filter, we annotate all query positions in the range $i, \ldots, i+l-1$ with labels fetched from matching nodes. Given a coverage cutoff $0 \leq c \leq 1$, we discard all labels that do not annotate at least $c|Q|$ positions in Q and then discard all anchors with no remaining labels. In this manuscript, we also use the term "anchor" to refer to an alignment of the query to the graph that is to be used as input for a subsequent alignment step.

After seed anchoring, we extend each anchor forward and backward using the same strategy as the *TCG-Aligner* [33], albeit along label-consistent walks. The alignments terminate after reaching a query end, or after satisfying one of the termination criteria described by Karasikov, Mustafa, et al. [33,32]. Like in the *TCG-Aligner*, a single extension from an anchor can produce multiple alignments that differ in their label set by selecting different initial positions in the dynamic programming table from which to backtrack [33].

To further reduce the number of anchors, each anchor extension step is followed by an additional anchor filtering step, operating in an interleaved fashion. We extend the anchors in increasing order with respect to their start index i. Once a given anchor is extended along a label-consistent walk, we check all subsequent anchors. Any subsequent anchor that was traversed during this extension is discarded. By default, the *TCG-Aligner* reports the highest-scoring alignments for each label. To provide a more rich set of alignments for use in *MetaGraph-MLA* and to ensure that a larger number of characters of Q are covered by at least one alignment, we keep all anchor extensions that are constructed.

2.3 MetaGraph-MLA: Generating multi-label alignments via chaining

Given an input set of label-consistent alignments of the query sequence Q generated by *MetaGraph-LA*, we use these alignments as inexactly-matching anchors for *MetaGraph-MLA*. In this step, we compute chains of these anchors through sparse dynamic programming to obtain multi-label alignments (MLAs, illustrated in Figure 2). The process works in three steps: *(i)* computing an optimal chaining score (Algorithm 1), *(ii)* backtracking through the dynamic programming table to construct the chain (Supplementary Algorithm A1), and *(iii)* extending the first and last alignments in the chain with label-change-permitting anchor extension. Step *(iii)* extends the MLA to query regions that were not covered by alignment anchors due to a lack of seeds or extension termination. The MLA assigns a label set to all walk nodes, called a *per-node labeling*.

Multi-label alignment chaining Let $a_1, \ldots, a_n$ be a series of label-consistent alignments of the query sequence Q sorted by their end positions in Q, with each a_i being assigned a label set $L_{a_i} \subseteq \mathcal{L}$. We construct chains by progressively adding new alignments to existing chains constructed from the previous alignments (Figure 2). If the next alignment a being incorporated overlaps with last alignment in a chain a', then the suffix of a' and the prefix of a are trimmed to a'_T and a_T, respectively to discard their overlap before incorporating a_T into the chain.

We now detail this process. Let $f(i, \ell)$ denote the maximum chaining score that can be achieved by a subsequence of $a_1, \ldots, a_i$ ending in a_i, with an assigned chain label $\ell \in L_{a_i}$. More precisely, it is defined recursively for $1 \leq i \leq n$ and $\ell \in L_{a_i}$ as,

$$f(i, \ell) := \Delta_S(a_i) + \max\left\{0, \max_{i' < i,\, \ell' \in L_{a_{i'}}} \left\{ f(i', \ell') + \Delta_{LC}(\ell' \to \ell) + \Delta_A(a_{i'}, a_i) \right\} \right\} \tag{1}$$

where $\Delta_{LC}(\ell' \to \ell) \leq 0$ is the score added for changing from the label ℓ' to the label ℓ (Section 2.4). $\Delta_A(a_{i'}, a_i)$ is the score added by appending a_i to a chain ending with alignment $a_{i'}$,

$$\Delta_A(a_{i'}, a_i) := p_J + \begin{cases} \Delta_O(a_{i'}, a_i) & \text{if overlap}(a_{i'}, a_i) > 0 \\ \Delta_G(-\text{overlap}(a_{i'}, a_i)) & \text{if overlap}(a_{i'}, a_i) \leq 0 \text{ (i.e., they do not overlap)} \end{cases}, \tag{2}$$

where $\Delta_O(a_{i'}, a_i) \leq 0$ is the best score attainable by trimming query characters from $a_{i'}$ and a_i (and subtracting the scores of the discarded alignment regions) until they are non-overlapping. $p_J < 0$ is a fixed penalty for connecting alignments, simulating traversal to lower-k nodes in the variable-order De Bruijn graph (Section 2.4). Alongside computing the highest chaining score, Algorithm 1 also stores the number of query characters trimmed from each alignment and backtracking pointers to assist in chain reconstruction (Supplementary Algorithm A1). After computing the best chaining score, we reconstruct the highest-scoring alignment chain for each label set. To reduce the computational burden of calculating $\Delta_O(a_{i'}, a_i)$ (Supplementary Algorithm A2), we only consider $a_{i'}$ and a_i with $\text{overlap}(a_{i'}, a_i) < k - 1$ (Algorithm 1).

Extending MLAs via label-change-permitting anchor extension The multi-label chains produced by the above procedure are delimited by their respective first and last alignments. Since these constituent alignments are label-consistent, it could be that a multi-label extension from the first and/or last alignment exists in the graph. In particular, this can happen if no seeds are found in these regions or if no anchor extensions are computed due to early termination criteria [32,33]. Thus, we compute these anchor extensions to complete the chain, proceeding similarly to *MetaGraph-LA*. If a label-consistent edge does not exist during search, we consider all other edges, but add label-change scores accordingly (Section 2.4).

Algorithm 1: Computing the best multi-label alignment chain score from label-consistent alignments. See Supplementary Algorithm A2 for the implementation of `Splice`.

Input: Q is a query sequence, $a_1, \ldots, a_n$ are a series of local alignments of Q to the graph sorted by their ending coordinates in Q, with corresponding label sets $L_1, \ldots, L_n$.

Output: $f : \{1, \ldots, n\} \times \mathcal{L} \to \mathbb{Z}$, a sparse score dictionary where $f(i, \ell)$ is the highest chain score that can be computed from a subsequence of $a_1, \ldots, a_i$ that includes a_i and has the chain label $\ell \in L_{a_i}$.
$B : \{1, \ldots, n\} \times \mathcal{L} \to \mathbb{N} \times 2^{\mathcal{L}} \times \mathbb{N} \times \mathbb{N}$, a sparse backtracking dictionary of 3-tuples indicating, for each $f(i, \ell)$, the previous index i' and label ℓ' from which $f(i, \ell)$ was computed, and the corresponding numbers of query characters to be trimmed from the prefix of a_i ($l_{i,\text{prefix}}$)) and the suffix of $a_{i'}$ ($l_{i',\text{suffix}}$).

```
ChainLocalAlignments(Q, [(a_1, L_1), ..., (a_n, L_n)])
f ← Dictionary()
B ← Dictionary()
foreach (i, ℓ) ∈ {1, ... n} × L_i do
    f(i, ℓ) ← Δ_S(a_i) // Initialize the score matrix
    B(i, ℓ) ← (∞, ℓ, 0, 0) // Initialize backtracking helper values
foreach (i', ℓ') from {1, ..., n − 1} × L_i' s.t. f(i', ℓ') has been defined do
    foreach i from i' + 1, ..., n do
        // To reduce computational intensity, only consider alignments with small overlap
        if overlap(a_i', a_i) < k − 1 then
            (Δ_A(a_i', a_i), l_i',suffix, l_i,prefix) ← Splice(a_i', a_i)
            foreach ℓ ∈ L_i do
                s ← f(i', ℓ') + Δ_LC(ℓ' → ℓ) + Δ_A(a_i', a_i)
                if s > f(i, ℓ) then
                    f(i, ℓ) ← s
                    B(i, ℓ) ← (i', ℓ', l_i',suffix, l_i,prefix)
return f, B
```

2.4 Deriving a scoring model for novel alignment operations

We now detail the scoring model for our new alignment operations. These additions to standard alignment scoring models provide a richer arsenal of operations for sequence-to-graph alignment and also allow for standard score-based alignment heuristics (e.g., X-drop alignment [4,63]) to restrict the search space.

For simplicity, given a label $\ell \in \mathcal{L}$, we abuse notation and use ℓ to also denote the inverse image $\mathcal{A}^{-1}(\ell)$, the set of graph nodes annotated with that label. Thus, we use the notation $|\ell|$ to denote $|\mathcal{A}^{-1}(\ell)|$. Then, we define the probability $\Pr(\ell)$ as the density of ℓ's annotation column, $\Pr(\ell) := \frac{|\ell|}{|V|}$. We derive our scoring model using two baseline generative probabilistic graphical models similar to the sequence-to-sequence alignment models presented by Frith [19]. Given a starting node in the graph, the first generative model (called the *target model*) generates query sequences with probabilities proportional to their similarity to the spellings of walks that originate at the starting node, assuming that the query sequence is generated by a series of edits to a walk spelling. The second model (called the *null model*) generates both query sequences and graph walks with probabilities only proportional to the sequences' respective lengths, assuming that the two tasks occur independently. Each alignment operation E (defined in Section 2.1) corresponds to traversing an edge in the graphical model (called a *transition*), which occurs with a transition probability denoted by $\Pr(E)$ for the target model and $\Pr_0(E)$ for the null model. The graphical model *emits* a query character if $E \in \{=, \neq, IO, IE\}$ and it traverses a graph edge to emit a target character if $E \in \{=, \neq, DO, DE\}$. We define the score E as $\Delta_E := \lambda \log \frac{\Pr(E)}{\Pr_0(E)}$, where λ is a fixed scaling constant. Thus, the sum of all operation scores (i.e., the alignment score Δ_S) corresponds to the log ratio of the probabilities of the corresponding walks in the target and null models.

Deriving and efficiently computing label-change scores Suppose that we traverse along the edge (n_1, n_2) where we assume that n_1 and n_2 share no common labels and, for simplicity, that $\mathcal{A}(n_1) = \{\ell_1\}$ and $\mathcal{A}(n_2) = \{\ell_2\}$. We call this event a *label change* from ℓ_1 to ℓ_2, denoted by $\ell_1 \to \ell_2$, which we score using the

probability of n_2 also having label ℓ_1 conditioned on n_2 having the label ℓ_2. To formulate this precisely, we extend our graphical models so that each graph-traversing alignment operation also emits a label change, with the transition probability $\Pr(\ell_1 \to \ell_2)$ in the target model and $\Pr_0(\ell_1 \to \ell_2)$ in the null model. For the null model, we assume no relationship between the labels of adjacent graph nodes, so $\Pr_0(\ell_1 \to \ell_2) := \Pr(\ell_2)$. Then, the *label-change score* is $\Delta_{LC}(\ell_1 \to \ell_2) := \lambda \log \frac{\Pr(\ell_1 \to \ell_2)}{\Pr_0(\ell_1 \to \ell_2)} = \lambda \log \frac{\Pr(\ell_1 \to \ell_2)}{\Pr(\ell_2)}$.

We now need to define $\Pr(\ell_1 \to \ell_2)$. For this new scoring model to reduce to that of a label-free setting, we require that $\Delta_{LC}(\ell \to \ell) = 0$ (i.e., only add a score if the label changes) and that $\ell_1 \neq \ell_2 \Rightarrow \Delta_{LC}(\ell_1 \to \ell_2) \leq 0$ (i.e., all label changes decrease the score). By simply defining $\Pr(\ell_1 \to \ell_2) := \Pr(\ell_1 \cap \ell_2)$, we satisfy these requirements and arrive at the equation,

$$\Delta_{LC}(\ell_1 \to \ell_2) = \lambda \log \frac{\Pr(\ell_1 \cap \ell_2)}{\Pr(\ell_2)} = \lambda \log \Pr(\ell_1 \mid \ell_2). \tag{3}$$

For large annotated De Bruijn graphs, computing $\Pr(\ell_1 \mid \ell_2)$ precisely can be computationally intractable since it requires enumerating $\ell_1 \cap \ell_2$. However, due to the log factor in the definition of Δ_{LC}, we only require order-of-magnitude accuracy when computing its value. Thus, we approximate $\Pr(\ell_1 \mid \ell_2)$ using HyperLogLog++ counter [23] sketches of ℓ_1 and ℓ_2, respectively, and the precomputed set sizes $|\ell_1|$ and $|\ell_2|$. Given the *approximate set size* and *approximate set union* operations denoted by $|\cdot|_{\text{HLL}}$ and $\cup_{\text{HLL}}$, respectively, we efficiently compute $\Pr(\ell_1 \mid \ell_2) \approx \frac{|\ell_1| + |\ell_2| - |\ell_1 \cup_{\text{HLL}} \ell_2|_{\text{HLL}}}{|\ell_2|}$.

Deriving penalties for node length changes To better facilitate search on graphs with disconnects caused by low sequencing coverage, we switch our graph model from a De Bruijn graph of fixed order k to a variable-order De Bruijn graph of maximum order k. This introduces additional possibilities into the alignment, namely, the *change-of-node-length* operation. More precisely, given a node n of length k, this entails traversing from a node with spelling s to the node with spelling $s[k-l+1:k]$ for some $1 \leq l < k$ (i.e., decreasing the order from k to l). With this strategy, nodes with length l act as proxies for k-mers that are missing from the graph. However, we need to define penalties for searching among nodes of length l. This is because, for every query sequence, there exists a walk in the variable-order De Bruijn graph along nodes of length 1 that spells the query exactly (assuming that every character in Σ has been observed at least once). Therefore, an alignment to this walk would trivially be the optimal one.

To both reduce the search space for alignment and to avoid this degenerate case, our scoring model introduces a further extension to accommodate variable-order De Bruijn graphs. In this new model, we do not add a negative score (i.e., penalize) for traversal along edges that either increase the node length by 1 or maintain the node length at k. This ensures that our model reduces to standard sequence-to-graph alignment when employed on fixed-k De Bruijn graphs. In addition, we penalize traversal from a node of length k to one of any length $1 \leq l < k$ with a score denoted by $\Delta_L(k \to l)$. Finally, we disallow all other traversals to reduce the chances of traversing along spurious walks, in particular the degenerate walk described above. Given a fixed constant $p_J < 0$, the score for traversing from a node of length l_1 to a node of length l_2 is

$$\Delta_L(l_1 \to l_2) := \begin{cases} 0 & \text{if } l_1 = l_2 = k \text{ or } l_1 + 1 = l_2 \\ p_J & \text{if } l_1 = k \text{ and } l_2 < k \\ -\infty & \text{if } l_1 < k \text{ and } l_1 + 1 \neq l_2 \end{cases}. \tag{4}$$

Although the nodes of length l are meant to be proxies for missing k-mers in the graph, this scoring model employs a fixed constant penalty for insertions of any number of nodes. We motivate this design choice by the following observation: If there is a single character mismatch between position $k-l+1$ in node n's spelling and position $k-l+2$ in node n''s spelling that prevents n from being adjacent to n' (e.g., n spells `GATGC` and n' spells `ACGCT` for $k=5$ and $l=3$), then we require l node insertion edits to create a path to n' from a predecessor of n, despite the more likely possibility that only a single edit led to the mismatch (for our example, given appropriate characters $\texttt{X}, \texttt{Y} \in \Sigma$, these new nodes would be n_1 spelling `XYGAC`, n_2 spelling `YGACG`, and n_3 spelling `GACGC`). Thus, making p_J depend on $k-l$ would prescribe unequal scores for each of these $k-1$ equally-likely edit events.

3 Evaluation Methodology

We evaluate *MetaGraph-LA* and *MetaGraph-MLA* against the *MetaGraph-Align* [32], *GraphAligner* [52], and *minigraph* [39] sequence-to-graph alignment tools. We evaluate our methods on graphs constructed from the Virus50k collection [47], using taxonomic profiles derived from the alignment results to evaluate alignment accuracy (see subsection below). This collection consists of 51,283 assembled whole virus genomes drawn from GenBank. We start by dividing the set into 41,026 randomly-selected genomes for the *graph* set and the remaining genomes for the *test* sets. From each genome in the graph set, we simulate a low-coverage (1×) Illumina HiSeq-type read set using *ART* [27] and assemble cleaned contigs from the reads using the graph cleaning procedure in *MetaGraph* [32] on assembly De Bruijn graphs of order $k = 31$. We then construct an annotated De Bruijn graph of order $k = 31$ from the contigs from all samples, labeling nodes by their respective source accession IDs. We use a compacted De Bruijn graph in GFA format, generated using *MetaGraph*, as an interchange format between *MetaGraph* and other tools. For compatibility with *minigraph*, we use *GetBlunted* to derive a variation graph from the De Bruijn graph [17]. As an additional evaluation, we align the test reads to the original virus reference genomes using the *TCG-Aligner* [33] (the basis for *MetaGraph-LA*) to determine reference values for alignment accuracy during our experiments. See Supplementary Table A1 for statistics about these graphs. To generate query sequences, we simulate reads from the remaining 10,257 genomes in the test set. We simulate 1×-coverage read sets for each genome, using *ART* [27] for Illumina HiSeq-type reads and *pbsim* [48] for PacBio CLR- and PacBio CCS-type reads. For each read type, we aggregate the reads from all test genomes and uniformly randomly sample 1000 reads.

3.1 Evaluation measures for multi-label alignment

Since the labels in our annotated graph represent known organisms, we can evaluate the accuracy of our methods via *taxonomic profiling* using the taxonomic IDs (taxIDs) assigned to each organism. In this direction, we evaluate each tool via two different strategies: 1) measuring the fraction of the query characters that are aligned, and 2) evaluating the accuracy of the alignment's taxonomic profile. Since we simulate reads from known genomes, each read has a *ground-truth profile* assigning its respective genome's taxID as the sole ID in the profile. We evaluate taxonomic profile accuracy using the *weighted EMDUniFrac* (UF) distance between the ground-truth profile and the profiles we compute from the tools' alignment results [45].

We derive a taxonomic profile from an alignment using the per-node labelings of the alignment's walk. These labelings are provided by *MetaGraph-LA*, *Metagraph-MLA*, and *TCG-Aligner* without modification, with a set of labels assigned to each k-mer in the returned target sequence. Since *MetaGraph-Align*, *minigraph*, and *GraphAligner* do not take into account or report annotations for their alignments within our test regime, we determine labels for their alignments by mapping their target sequences (i.e., the sequence to which the query aligned) via exact k-mer matching to the *MetaGraph* index using *MetaGraph-Query* [32].

Given a collection of labels L_i assigned for each k-mer n_i in the target sequence of query sequence Q_i's alignment, we assign a taxID derived from the NCBI Taxonomy [53] that represents the *lowest common ancestor* (LCA) of all labels in L_i using an in-house implementation of the taxonomic profiling algorithm from *Kraken* [61,60]. These LCAs form the *predicted profile* of the query sequence. Then, we use *OPAL* [46] to compute the UF distance d between the ground-truth profile and the predicted profile. The ground-truth profile is simply the root-to-strain path in the taxonomic tree of the strain corresponding to the ground-truth sample accession ID. Since we use a subtree of the taxonomic tree with a total of $h = 9$ levels from the NCBI Taxonomy (superkingdom, kingdom, phylum, class, order, family, genus, species, strain), known as *taxonomic ranks*, our distance values range from 0 (i.e., matching profiles) to $2h$ (i.e., completely different profiles) [46]. For this reason, we use the rank at height $h - \frac{d}{2}$ in the taxonomic tree to infer an alignment's accuracy (e.g., $h - \frac{d}{2} = 0$ indicates strain-level accuracy).

3.2 Experimental setup, code, and data avaialability

We performed all experiments on an Intel(R) Xeon(R) Gold 6140 CPU (3.70GHz) processor from ETH Zurich's shared high-performance compute systems using a single thread and default parameters for each tool. All De Bruijn graphs are of order $k = 31$. For *MetaGraph-LA* and *MetaGraph-MLA*, we use a seed length of $l = 15$. We implemented our methods within the MetaGraph framework [32]. The framework's source code is free (libre) software licensed under the GNU General Public License v3.0 and is hosted at `https://github.com/ratschlab/metagraph`. All experiments were run using the commit `36d6b7a`. The input files and scripts for reproducing our results are available at `https://github.com/ratschlab/mla`.

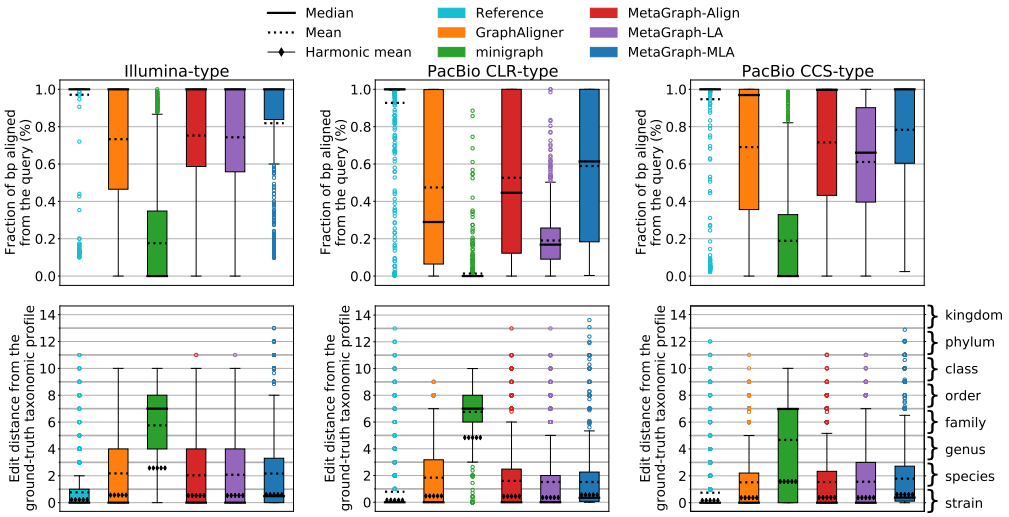

Fig. 3. Distributions of the fractions of base pair fractions of read that are alignable to the Virus40k graph (top) and the distributions of the accuracies of each read's predicted taxonomic profile (bottom). We measure a read's taxonomic profile accuracy as the weighted EMDUniFrac (UF) distance [46,45] between the ground-truth and predicted profiles. We compute reference values for the two measures (in cyan) by aligning the reads to the original reference genomes using the *TCG-Aligner* [33]. The UF distance ranges on the y-axis of the bottom row of figures are annotated with taxonomic rank names indicating degree of similarity (Section 3.1). Due to its reliance on long anchor chains within unipaths, *minigraph* performs significantly worse than the other tools [39].

4 Results and Discussion

In this section, we demonstrate the improved alignment performance of our multi-label alignment approach. We show that MLA produces longer alignments for our query sequences for all read types, while maintaining strain-level taxonomic profiling accuracy. In addition, we demonstrate that our alignment chaining algorithm incorporates alignments predominantly from the correct strain into chains.

4.1 Multi-label alignment produces longer and more accurate alignments

We align our simulated virus reads against the Virus50k graph to evaluate the performance of our *MetaGraph-MLA* approach (Figure 3 and Supplementary Table A2). *MetaGraph-MLA* produces the longest alignments (mean and median) out of all of the methods tested for all read types, while maintaining the same strain-level accuracy as most other tools (median and harmonic mean). In particular, the mean and lower quartile of the Illumina-type read alignment lengths are strictly greater than the mean alignment length of the next best tool (*MetaGraph-Align*). On average, *MetaGraph-MLA* is able to align 6.5% more characters from the Illumina-type reads, 6.2% more from the PacBio CLR-type reads, and 6.7% more from the PacBio CCS-type reads than the next best aligner, *MetaGraph-Align*.

We also plot the UF distances for each of the read types to test the accuracy of our alignments (Figure 3, bottom row). Due to their very skewed distributions for all read types, we consider the mean, median, and harmonic means of the UF distance distributions. By these measures, the *MetaGraph* alignment methods and *GraphAligner* are similarly accurate, with a significantly poorer performance by *minigraph*. The mean UF distances for most methods are between 1.0 and 3.0, indicating species-level profiling accuracy. The median and harmonic mean UF distances for most methods are below 1.0, indicating strain-level accuracy.

We note that due to the disconnected topology of our graph (Supplementary Table A1), *minigraph* is not able to produce comparably long alignments. Its first step finds long co-linear chains *within* graph unitigs [39], which may not exist due to their short average length of ∼80 bp. The other tools do not have this restriction.

Our test data reflects a large collection of genomes that lack repetitive regions. Such repetitive regions would cause a greater number of spurious chains, as it would be more difficult to distinguish between a gap in the graph caused by zero coverage and a gap caused by a genetic duplication event. It is yet to be determined how much this phenomenon will affect the accuracy of multi-label alignment.

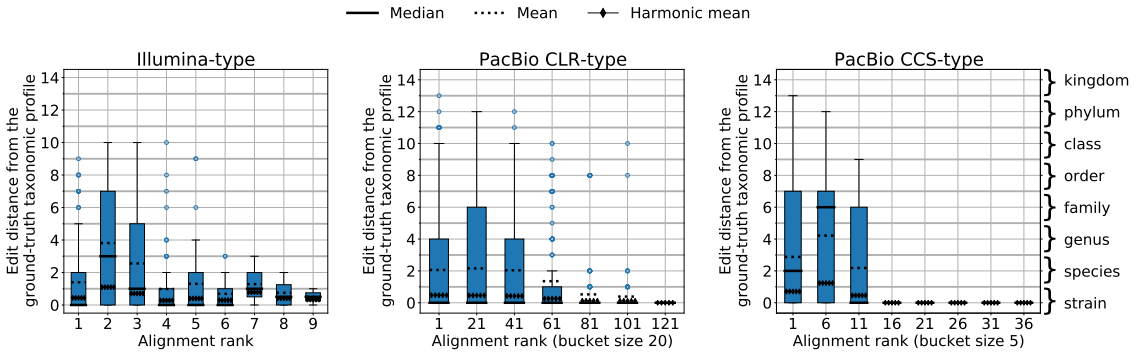

Fig. 4. Accuracies of the taxonomic profiles of all alignments incorporated into chains measured as weighted EMDUniFrac (UF) distances between ground-truth and predicted profiles. Alignments within a chain are ranked by decreasing length since the alignment lengths act as weights when computing the accuracy. The UF distance ranges on the y-axis are annotated with taxonomic rank names indicating degree of similarity (Section 3.1).

4.2 Label-consistent alignments within a mutli-label chain are accurate

Alongside showing that the whole-read profiles produced by *MetaGraph-MLA* are accurate, we also study the accuracies of the taxonomic profiles of the individual alignments incorporated into our alignment chains (Figure 4). For this evaluation, we only consider reads for which an MLA is produced, ignoring reads that are only aligned to a single label-consistent walk. Since the lengths of the constituent alignments act as weights when computing UF distances, for this analysis, we sorted all alignments within a chain by decreasing length. We observe that for all read types, the accuracy of taxonomic profiling tends to increase for the shorter alignments incorporated into the chain. For most alignment ranks, we observe a median and harmonic mean UF distances below 1.0, indicating strain-level accuracy. For all ranks, these measures are below 3.0, indicating species-level accuracy.

5 Conclusions

In this work, we present two aligners, *MetaGraph-LA* and *MetaGraph-MLA*, for label-consistent and multi-label alignment to annotated De Bruijn graphs, respectively. *MetaGraph-LA* allows for aligning DNA sequences to annotated De Bruijn graphs constructed from any sequence type, including unassembled DNA sequencing reads, using the annotation structure to restrict alignments to walks that are consistent with the labels of the initial alignment seed anchors. *MetaGraph-MLA* uses these alignments as inputs to construct multi-label alignments, allowing for sequence information from multiple samples to produce more complete alignments. Using our novel scoring model, we are able to integrate alignments from biologically-relevant combinations of samples, while also overcoming disconnects in the graph due to low coverage in the indexed sequences. We demonstrate on simulated sequencing data from a cohort of virus genomes that we produce longer alignments while predominantly integrating alignments from samples of the correct strain. This leads to strain-level accuracy when taxonomically profiling our test reads based on their alignments.

The chaining algorithm we present does not take into account any genomic coordinate or graph traversal information, as is typically done with *co-linear* chaining algorithms [39,33,2,42,9]. Since coordinates are only defined for the input sequences, applying this technique for coordinates assigned to contigs from low-coverage assembly graphs would produce short alignments, as demonstrated by *minigraph* during our evaluation. One way to alleviate this could be to augment our the chain scoring algorithm to not only incorporate graph traversal distances [39,42], but also to extend the scoring model to include a *contig change* operation with a score penalty. As an alternative to *MetaGraph-MLA*'s heuristic chaining approach we use, one could simply apply node insertion as an allowable edit during *MetaGraph-LA*'s anchor extension phase. However, it is known that doing so optimally is an $\mathcal{NP}$-complete problem [31,21], and it is not yet clear how well this problem can be approximated using known or novel alignment heuristics.

Although we have implemented our algorithms within the *MetaGraph* framework, our approach is readily applicable to other pangenome frameworks and can thus potentially see a more widespread application. We envision the methods developed in *MetaGraph-MLA* to be a step towards making unassembled sequence sets a more powerful resource for bioinformatics research.

Acknowledgements

We would like to thank Nika Mansouri Ghiasi, Ragnar Groot Koerkamp, Maximilian Mordig, Mohammed Alser, Amir Joudaki, Pesho Ivanov, Mario Stanke, and Marek Kokot for their helpful discussions and feedback. H. M. and M. K. were funded as part of Swiss National Research Programme (NRP) 75 "Big Data" by the SNSF grant #407540_167331. H. M., M. K., and A. K. are also partially funded by ETH core funding (to G. R.). We declare no conflicts of interest.

References

1. Almodaresi, F., Sarkar, H., Srivastava, A., Patro, R.: A space and time-efficient index for the compacted colored de Bruijn graph. Bioinformatics **34**(13), i169–i177 (06 2018). https://doi.org/10.1093/bioinformatics/bty292, https://doi.org/10.1093/bioinformatics/bty292
2. Almodaresi, F., Zakeri, M., Patro, R.: PuffAligner: a fast, efficient and accurate aligner based on the Pufferfish index. Bioinformatics **37**(22), 4048–4055 (06 2021). https://doi.org/10.1093/bioinformatics/btab408, https://doi.org/10.1093/bioinformatics/btab408
3. Altschul, S.F., Gish, W., Miller, W., Myers, E.W., Lipman, D.J.: Basic local alignment search tool. Journal of Molecular Biology **215**(3), 403–410 (1990). https://doi.org/https://doi.org/10.1016/S0022-2836(05)80360-2, https://www.sciencedirect.com/science/article/pii/S0022283605803602
4. Altschul, S.F., Madden, T.L., Schäffer, A.A., Zhang, J., Zhang, Z., Miller, W., Lipman, D.J.: Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. Nucleic Acids Research **25**(17), 3389–3402 (09 1997). https://doi.org/10.1093/nar/25.17.3389, https://doi.org/10.1093/nar/25.17.3389
5. Bankevich, A., Bzikadze, A.V., Kolmogorov, M., Antipov, D., Pevzner, P.A.: Multiplex de bruijn graphs enable genome assembly from long, high-fidelity reads. Nature Biotechnology **40**(7), 1075–1081 (07 2022). https://doi.org/10.1038/s41587-022-01220-6, https://doi.org/10.1038/s41587-022-01220-6
6. Belazzougui, D., Cunial, F.: Fully-Functional Bidirectional Burrows-Wheeler Indexes and Infinite-Order De Bruijn Graphs. In: Pisanti, N., Pissis, S.P. (eds.) 30th Annual Symposium on Combinatorial Pattern Matching (CPM 2019). Leibniz International Proceedings in Informatics (LIPIcs), vol. 128, pp. 10:1–10:15. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2019). https://doi.org/10.4230/LIPIcs.CPM.2019.10, http://drops.dagstuhl.de/opus/volltexte/2019/10481
7. Boucher, C., Bowe, A., Gagie, T., Puglisi, S.J., Sadakane, K.: Variable-order de bruijn graphs. In: 2015 Data Compression Conference. pp. 383–392 (2015). https://doi.org/10.1109/DCC.2015.70
8. Bowe, A., Onodera, T., Sadakane, K., Shibuya, T.: Succinct de bruijn graphs. In: Raphael, B., Tang, J. (eds.) Algorithms in Bioinformatics. pp. 225–235. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
9. Chandra, G., Jain, C.: Sequence to graph alignment using gap-sensitive co-linear chaining. bioRxiv (2022). https://doi.org/10.1101/2022.08.29.505691, https://www.biorxiv.org/content/early/2022/09/01/2022.08.29.505691
10. Chen, N.C., Solomon, B., Mun, T., Iyer, S., Langmead, B.: Reference flow: reducing reference bias using multiple population genomes. Genome Biology **22**(1), 8 (01 2021). https://doi.org/10.1186/s13059-020-02229-3, https://doi.org/10.1186/s13059-020-02229-3
11. Chikhi, R., Limasset, A., Medvedev, P.: Compacting de Bruijn graphs from sequencing data quickly and in low memory. Bioinformatics **32**(12), i201–i208 (06 2016). https://doi.org/10.1093/bioinformatics/btw279, https://doi.org/10.1093/bioinformatics/btw279
12. Chikhi, R., Rizk, G.: Space-efficient and exact de bruijn graph representation based on a bloom filter. Algorithms for Molecular Biology **8**(1), 22 (09 2013). https://doi.org/10.1186/1748-7188-8-22, https://doi.org/10.1186/1748-7188-8-22
13. Colquhoun, R.M., Hall, M.B., Lima, L., Roberts, L.W., Malone, K.M., Hunt, M., Letcher, B., Hawkey, J., George, S., Pankhurst, L., Iqbal, Z.: Pandora: nucleotide-resolution bacterial pan-genomics with reference graphs. Genome Biology **22**(1), 267 (09 2021). https://doi.org/10.1186/s13059-021-02473-1, https://doi.org/10.1186/s13059-021-02473-1
14. Danko, D., Bezdan, D., et al.: A global metagenomic map of urban microbiomes and antimicrobial resistance. Cell **184**(13), 3376–3393.e17 (2021). https://doi.org/https://doi.org/10.1016/j.cell.2021.05.002, https://www.sciencedirect.com/science/article/pii/S0092867421005857
15. Díaz-Domínguez, D., Onodera, T., Puglisi, S.J., Salmela, L.: Genome assembly with variable order de bruijn graphs. bioRxiv (2022). https://doi.org/10.1101/2022.09.06.506758, https://www.biorxiv.org/content/early/2022/09/07/2022.09.06.506758
16. Dvorkina, T., Antipov, D., Korobeynikov, A., Nurk, S.: Spaligner: alignment of long diverged molecular sequences to assembly graphs. BMC Bioinformatics **21**(12), 306 (07 2020). https://doi.org/10.1186/s12859-020-03590-7, https://doi.org/10.1186/s12859-020-03590-7

17. Eizenga, J.M., Lorig-Roach, R., Meredith, M.M., Paten, B.: Walk-preserving transformation of overlapped sequence graphs into blunt sequence graphs with getblunted. In: De Mol, L., Weiermann, A., Manea, F., Fernández-Duque, D. (eds.) Connecting with Computability. pp. 169–177. Springer International Publishing, Cham (2021)
18. Eizenga, J.M., Novak, A.M., Sibbesen, J.A., Heumos, S., Ghaffaari, A., Hickey, G., Chang, X., Seaman, J.D., Rounthwaite, R., Ebler, J., Rautiainen, M., Garg, S., Paten, B., Marschall, T., Sirén, J., Garrison, E.: Pangenome graphs. Annual Review of Genomics and Human Genetics **21**(1), 139–162 (2020). https://doi.org/10.1146/annurev-genom-120219-080406, https://doi.org/10.1146/annurev-genom-120219-080406, pMID: 32453966
19. Frith, M.C.: How sequence alignment scores correspond to probability models. Bioinformatics **36**(2), 408–415 (07 2019). https://doi.org/10.1093/bioinformatics/btz576, https://doi.org/10.1093/bioinformatics/btz576
20. Garrison, E., Sirén, J., Novak, A.M., Hickey, G., Eizenga, J.M., Dawson, E.T., Jones, W., Garg, S., Markello, C., Lin, M.F., Paten, B., Durbin, R.: Variation graph toolkit improves read mapping by representing genetic variation in the reference. Nature Biotechnology **36**(9), 875–879 (10 2018). https://doi.org/10.1038/nbt.4227, https://doi.org/10.1038/nbt.4227
21. Gibney, D., Thankachan, S.V., Aluru, S.: The complexity of approximate pattern matching on de bruijn graphs. In: Pe'er, I. (ed.) Research in Computational Molecular Biology. pp. 263–278. Springer International Publishing, Cham (2022)
22. Harrison, P.W., Ahamed, A., Aslam, R., Alako, B.T.F., Burgin, J., Buso, N., Courtot, M., Fan, J., Gupta, D., Haseeb, M., Holt, S., Ibrahim, T., Ivanov, E., Jayathilaka, S., Balavenkataraman Kadhirvelu, V., Kumar, M., Lopez, R., Kay, S., Leinonen, R., Liu, X., O'Cathail, C., Pakseresht, A., Park, Y., Pesant, S., Rahman, N., Rajan, J., Sokolov, A., Vijayaraja, S., Waheed, Z., Zyoud, A., Burdett, T., Cochrane, G.: The European Nucleotide Archive in 2020. Nucleic Acids Research **49**(D1), D82–D85 (11 2020). https://doi.org/10.1093/nar/gkaa1028, https://doi.org/10.1093/nar/gkaa1028
23. Heule, S., Nunkesser, M., Hall, A.: Hyperloglog in practice: Algorithmic engineering of a state of the art cardinality estimation algorithm. In: Proceedings of the 16th International Conference on Extending Database Technology. p. 683–692. EDBT '13, Association for Computing Machinery, New York, NY, USA (2013). https://doi.org/10.1145/2452376.2452456, https://doi.org/10.1145/2452376.2452456
24. Heydari, M., Miclotte, G., Van de Peer, Y., Fostier, J.: Browniealigner: accurate alignment of illumina sequencing data to de bruijn graphs. BMC Bioinformatics **19**(1), 311 (09 2018). https://doi.org/10.1186/s12859-018-2319-7, https://doi.org/10.1186/s12859-018-2319-7
25. Hickey, G., Heller, D., Monlong, J., Sibbesen, J.A., Sirén, J., Eizenga, J., Dawson, E.T., Garrison, E., Novak, A.M., Paten, B.: Genotyping structural variants in pangenome graphs using the vg toolkit. Genome Biology **21**(1), 35 (02 2020). https://doi.org/10.1186/s13059-020-1941-7, https://doi.org/10.1186/s13059-020-1941-7
26. Holley, G., Melsted, P.: Bifrost: highly parallel construction and indexing of colored and compacted de bruijn graphs. Genome Biology **21**(1), 249 (09 2020). https://doi.org/10.1186/s13059-020-02135-8, https://doi.org/10.1186/s13059-020-02135-8
27. Huang, W., Li, L., Myers, J.R., Marth, G.T.: ART: a next-generation sequencing read simulator. Bioinformatics **28**(4), 593–594 (12 2011). https://doi.org/10.1093/bioinformatics/btr708, https://doi.org/10.1093/bioinformatics/btr708
28. Iqbal, Z., Caccamo, M., Turner, I., Flicek, P., McVean, G.: De novo assembly and genotyping of variants using colored de bruijn graphs. Nature Genetics **44**(2), 226–232 (02 2012). https://doi.org/10.1038/ng.1028, https://doi.org/10.1038/ng.1028
29. Ivanov, P., Bichsel, B., Mustafa, H., Kahles, A., Rätsch, G., Vechev, M.: Astarix: Fast and optimal sequence-to-graph alignment. In: Schwartz, R. (ed.) Research in Computational Molecular Biology. pp. 104–119. Springer International Publishing, Cham (2020)
30. Jain, C., Misra, S., Zhang, H., Dilthey, A., Aluru, S.: Accelerating sequence alignment to graphs. In: 2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS). pp. 451–461 (2019). https://doi.org/10.1109/IPDPS.2019.00055
31. Jain, C., Zhang, H., Gao, Y., Aluru, S.: On the complexity of sequence-to-graph alignment. Journal of Computational Biology **27**(4), 640–654 (2020). https://doi.org/10.1089/cmb.2019.0066, https://doi.org/10.1089/cmb.2019.0066
32. Karasikov, M., Mustafa, H., Danciu, D., Zimmermann, M., Barber, C., Rätsch, G., Kahles, A.: Metagraph: Indexing and analysing nucleotide archives at petabase-scale. bioRxiv (2020). https://doi.org/10.1101/2020.10.01.322164, https://www.biorxiv.org/content/early/2020/11/03/2020.10.01.322164
33. Karasikov, M., Mustafa, H., Rätsch, G., Kahles, A.: Lossless indexing with counting de bruijn graphs. Genome Research (2022). https://doi.org/10.1101/gr.276607.122, http://genome.cshlp.org/content/early/2022/05/23/gr.276607.122.abstract

34. Katz, K., Shutov, O., Lapoint, R., Kimelman, M., Brister, J., O'Sullivan, C.: The Sequence Read Archive: a decade more of explosive growth. Nucleic Acids Research **50**(D1), D387–D390 (11 2021). https://doi.org/10.1093/nar/gkab1053, https://doi.org/10.1093/nar/gkab1053
35. Khan, J., Patro, R.: Cuttlefish: fast, parallel and low-memory compaction of de Bruijn graphs from large-scale genome collections. Bioinformatics **37**(Supplement_1), i177–i186 (07 2021). https://doi.org/10.1093/bioinformatics/btab309, https://doi.org/10.1093/bioinformatics/btab309
36. Krannich, T., White, W.T.J., Niehus, S., Holley, G., Halldórsson, B.V., Kehr, B.: Population-scale detection of non-reference sequence variants using colored de Bruijn graphs. Bioinformatics **38**(3), 604–611 (11 2021). https://doi.org/10.1093/bioinformatics/btab749, https://doi.org/10.1093/bioinformatics/btab749
37. Lee, C., Grasso, C., Sharlow, M.F.: Multiple sequence alignment using partial order graphs . Bioinformatics **18**(3), 452–464 (03 2002). https://doi.org/10.1093/bioinformatics/18.3.452, https://doi.org/10.1093/bioinformatics/18.3.452
38. Li, H.: Minimap2: pairwise alignment for nucleotide sequences. Bioinformatics **34**(18), 3094–3100 (05 2018). https://doi.org/10.1093/bioinformatics/bty191, https://doi.org/10.1093/bioinformatics/bty191
39. Li, H., Feng, X., Chu, C.: The design and construction of reference pangenome graphs with minigraph. Genome Biology **21**(1), 265 (10 2020). https://doi.org/10.1186/s13059-020-02168-z, https://doi.org/10.1186/s13059-020-02168-z
40. Limasset, A., Cazaux, B., Rivals, E., Peterlongo, P.: Read mapping on de bruijn graphs. BMC Bioinformatics **17**(1), 237 (06 2016). https://doi.org/10.1186/s12859-016-1103-9, https://doi.org/10.1186/s12859-016-1103-9
41. Luhmann, N., Holley, G., Achtman, M.: Blastfrost: fast querying of 100,000s of bacterial genomes in bifrost graphs. Genome Biology **22**(1), 30 (01 2021). https://doi.org/10.1186/s13059-020-02237-3, https://doi.org/10.1186/s13059-020-02237-3
42. Ma, J., Cáceres, M., Salmela, L., Mäkinen, V., Tomescu, A.I.: Chaining for accurate alignment of erroneous long reads to acyclic variation graphs. bioRxiv (2022). https://doi.org/10.1101/2022.01.07.475257, https://www.biorxiv.org/content/early/2022/05/19/2022.01.07.475257
43. Marchet, C., Boucher, C., Puglisi, S.J., Medvedev, P., Salson, M., Chikhi, R.: Data structures based on k-mers for querying large collections of sequencing data sets. Genome Research **31**(1), 1–12 (2021). https://doi.org/10.1101/gr.260604.119, http://genome.cshlp.org/content/31/1/1.abstract
44. Marchet, C., Iqbal, Z., Gautheret, D., Salson, M., Chikhi, R.: REINDEER: efficient indexing of k-mer presence and abundance in sequencing datasets. Bioinformatics **36**(Supplement_1), i177–i185 (07 2020). https://doi.org/10.1093/bioinformatics/btaa487, https://doi.org/10.1093/bioinformatics/btaa487
45. McClelland, J., Koslicki, D.: Emdunifrac: exact linear time computation of the unifrac metric and identification of differentially abundant organisms. Journal of Mathematical Biology **77**(4), 935–949 (10 2018). https://doi.org/10.1007/s00285-018-1235-9, https://doi.org/10.1007/s00285-018-1235-9
46. Meyer, F., Bremges, A., Belmann, P., Janssen, S., McHardy, A.C., Koslicki, D.: Assessing taxonomic metagenome profilers with opal. Genome Biology **20**(1), 51 (03 2019). https://doi.org/10.1186/s13059-019-1646-y, https://doi.org/10.1186/s13059-019-1646-y
47. Mustafa, H., Schilken, I., Karasikov, M., Eickhoff, C., Rätsch, G., Kahles, A.: Dynamic compression schemes for graph coloring. Bioinformatics **35**(3), 407–414 (07 2018). https://doi.org/10.1093/bioinformatics/bty632, https://doi.org/10.1093/bioinformatics/bty632
48. Ono, Y., Asai, K., Hamada, M.: PBSIM: PacBio reads simulator—toward accurate genome assembly. Bioinformatics **29**(1), 119–121 (11 2012). https://doi.org/10.1093/bioinformatics/bts649, https://doi.org/10.1093/bioinformatics/bts649
49. Pandey, P., Almodaresi, F., Bender, M.A., Ferdman, M., Johnson, R., Patro, R.: Mantis: A fast, small, and exact large-scale sequence-search index. Cell Systems **7**(2), 201–207.e4 (08 2018). https://doi.org/10.1016/j.cels.2018.05.021, https://doi.org/10.1016/j.cels.2018.05.021
50. Rand, K.D., Grytten, I., Nederbragt, A.J., Storvik, G.O., Glad, I.K., Sandve, G.K.: Coordinates and intervals in graph-based reference genomes. BMC Bioinformatics **18**(1), 263 (05 2017). https://doi.org/10.1186/s12859-017-1678-9, https://doi.org/10.1186/s12859-017-1678-9
51. Rautiainen, M., Mäkinen, V., Marschall, T.: Bit-parallel sequence-to-graph alignment. Bioinformatics **35**(19), 3599–3607 (03 2019). https://doi.org/10.1093/bioinformatics/btz162, https://doi.org/10.1093/bioinformatics/btz162
52. Rautiainen, M., Marschall, T.: Graphaligner: rapid and versatile sequence-to-graph alignment. Genome Biology **21**(1), 253 (09 2020). https://doi.org/10.1186/s13059-020-02157-2, https://doi.org/10.1186/s13059-020-02157-2
53. Schoch, C.L., Ciufo, S., Domrachev, M., Hotton, C.L., Kannan, S., Khovanskaya, R., Leipe, D., Mcveigh, R., O'Neill, K., Robbertse, B., Sharma, S., Soussov, V., Sullivan, J.P., Sun, L., Turner, S., Karsch-Mizrachi, I.: NCBI Taxonomy: a comprehensive update on curation, resources and tools. Database **2020** (08 2020). https://doi.org/10.1093/database/baaa062, https://doi.org/10.1093/database/baaa062, baaa062

54. Schulz, T., Wittler, R., Rahmann, S., Hach, F., Stoye, J.: Detecting high-scoring local alignments in pangenome graphs. Bioinformatics **37**(16), 2266–2274 (02 2021). https://doi.org/10.1093/bioinformatics/btab077, https://doi.org/10.1093/bioinformatics/btab077
55. Sirén, J.: Indexing variation graphs. In: Proceedings of the Meeting on Algorithm Engineering and Experiments (ALENEX). pp. 13–27. Society for Industrial and Applied Mathematics (2017). https://doi.org/10.1137/1.9781611974768.2, https://epubs.siam.org/doi/abs/10.1137/1.9781611974768.2
56. Sirén, J., Välimäki, N., Mäkinen, V.: Indexing graphs for path queries with applications in genome research. IEEE/ACM Trans. Comput. Biol. Bioinformatics **11**(2), 375–388 (03 2014). https://doi.org/10.1109/TCBB.2013.2297101, https://doi.org/10.1109/TCBB.2013.2297101
57. Sirén, J., Monlong, J., Chang, X., Novak, A.M., Eizenga, J.M., Markello, C., Sibbesen, J.A., Hickey, G., Chang, P.C., Carroll, A., Gupta, N., Gabriel, S., Blackwell, T.W., Ratan, A., Taylor, K.D., Rich, S.S., Rotter, J.I., Haussler, D., Garrison, E., Paten, B.: Pangenomics enables genotyping of known structural variants in 5202 diverse genomes. Science **374**(6574), abg8871 (2021). https://doi.org/10.1126/science.abg8871, https://www.science.org/doi/abs/10.1126/science.abg8871
58. Solomon, B., Kingsford, C.: Improved search of large transcriptomic sequencing databases using split sequence bloom trees. Journal of Computational Biology **25**(7), 755–765 (2018). https://doi.org/10.1089/cmb.2017.0265, https://doi.org/10.1089/cmb.2017.0265, pMID: 29641248
59. Turner, I., Garimella, K.V., Iqbal, Z., McVean, G.: Integrating long-range connectivity information into de Bruijn graphs. Bioinformatics **34**(15), 2556–2565 (03 2018). https://doi.org/10.1093/bioinformatics/bty157, https://doi.org/10.1093/bioinformatics/bty157
60. Wood, D.E., Lu, J., Langmead, B.: Improved metagenomic analysis with kraken 2. Genome Biology **20**(1), 257 (11 2019). https://doi.org/10.1186/s13059-019-1891-0, https://doi.org/10.1186/s13059-019-1891-0
61. Wood, D.E., Salzberg, S.L.: Kraken: ultrafast metagenomic sequence classification using exact alignments. Genome Biology **15**(3), R46 (03 2014). https://doi.org/10.1186/gb-2014-15-3-r46, https://doi.org/10.1186/gb-2014-15-3-r46
62. Zerbino, D.R., Birney, E.: Velvet: Algorithms for de novo short read assembly using de bruijn graphs. Genome Research **18**(5), 821–829 (2008). https://doi.org/10.1101/gr.074492.107, http://genome.cshlp.org/content/18/5/821.abstract
63. Zhang, Z., Schwartz, S., Wagner, L., Miller, W.: A greedy algorithm for aligning dna sequences. Journal of Computational Biology **7**(1-2), 203–214 (2000). https://doi.org/10.1089/10665270050081478, https://doi.org/10.1089/10665270050081478, pMID: 10890397