Contig Model for Variable-Order de Bruijn Graphs

Diego Díaz-Domínguez
Department of Computer Science, University of Helsinki, Finland

Pierfrancesco Martinello
Department of Computer Science, University of Helsinki, Finland

Taku Onodera
Graduate School of Information Science and Technology, University of Tokyo, Japan

Simon J. Puglisi
Department of Computer Science, University of Helsinki, Finland

Leena Salmela
Department of Computer Science, University of Helsinki, Finland

Abstract

Choosing an order for constructing a *de Bruijn graph* (DBG) is a crucial step in *de novo* assembly, as no single value allows complete genome reconstruction. The *variable-order de Bruijn graph* (voDBG) addresses this limitation by combining DBGs of multiple orders in a single structure connected by contextual relationships. This representation enables new connections to be identified or ambiguities to be resolved during assembly. However, voDBGs currently lack a formal definition of contigs.

In this paper, we give the first formal definition of contigs for voDBGs. We show that, for a frequency range $[\ell, h]$ with $\ell > h/2$, nodes whose labels occur with frequency $f \in [\ell, h]$ in the reads spell sequences of the genome with high probability under uniform sampling assumptions. We call these sequences (ℓ, h)-tigs. We also present an efficient algorithm to enumerate (ℓ, h)-tigs from a voDBG that accounts for homopolymer errors. Experiments on PacBio HiFi data show that our method significantly improves contiguity compared to unitigs in fixed-order DBGs while remaining considerably lighter than full genome assemblers.

2012 ACM Subject Classification Applied computing → Bioinformatics; Applied computing → Molecular sequence analysis; Theory of computation → Data structures design and analysis; Applied computing → Computational genomics

Keywords and phrases genome assembly, de Bruijn Graph, long reads

Digital Object Identifier 10.4230/LIPIcs.WABI.2026.4

Related Version *Preprint Version*: `https://doi.org/10.1101/2022.09.06.506758`

Supplementary Material *Software (Source Code)*: `https://github.com/ddiazdom/ryu`

Funding *Diego Díaz-Domínguez*: supported by EU Horizon Europe grant No 101060011 and Research Council of Finland via grant 323233.
Pierfrancesco Martinello: supported by Research Council of Finland via grant 370538.
Taku Onodera: supported by Research Council of Finland via grant 323233.
Leena Salmela: supported by Research Council of Finland via grants 323233 and 370538.

1 Introduction

Short-read assemblers [32, 2] have historically relied on *de Bruijn graphs* (DBGs) [14, 27]. These methods decompose reads into k-mers, construct the corresponding DBG, and reconstruct the genome by traversing maximal unary paths. The approach is effective because small k yields compact graphs that are cheap to build and, for short reads, recover most of the genome. Nevertheless, finding a value of k (the DBG order) that maximizes assembly quality is not trivial [8]: small k values produce tangled graphs, whereas large k values cause fragmentation when genome complexity or sequencing coverage varies.

Advances in DNA sequencing have produced longer and more accurate reads. PacBio HiFi reads, for example, are typically 10–20 kbp long with error rates below 1% [31]. Standard DBG-based strategies are less suitable in this case because they require large values of k to disentangle the graph, which increases its size and computational cost. Moreover, a large fixed k does not fully exploit the information in HiFi data.

Modern assemblers typically prefer the overlap–layout–consensus (OLC) [21] strategy for long reads. This method builds a graph where the nodes represent the reads and the edges model suffix-prefix overlaps between them. However, OLC requires the costly computation of the overlaps, a task further complicated by homopolymer-length errors, the dominant error type in PacBio HiFi reads [31]. Tools based on the OLC paradigm include Peregrine [9], HiCanu [23], and hifiasm [6]. Although the recent development of fast heuristics for approximate pattern matching has reduced the costs of computing overlaps [3, 25, 18], obtaining OLC graphs from HiFi reads remains expensive. This has motivated a renewed interest in DBG-based approaches that scale more gracefully.

The main challenge is how to combine the information across multiple DBG orders. Some methods progressively increase k, as in IDBA [26] and SPAdes [2], whereas others selectively increase k in complex regions, such as LJA [1]. A recent method, DVOUG [20], takes the opposite approach, constructing an initial assembly with a large k value and then extending contigs by progressively lowering k. Other strategies reduce graph size through sparsification or compression: MBG [29] constructs a sparse DBG from a subset of k-mers, while mdBG [12] performs assembly in minimizer space. More generally, models supporting variable-order contexts have been proposed. Manifold DBGs [19] allow arbitrary substrings as node labels, and variable-order DBGs (voDBGs) [4] represent all DBGs up to maximum order K.

Among these alternatives, voDBGs provide an appealing framework for long-read assembly as they simultaneously encode multiple DBG orders and are closely related to compressed suffix trees [22]. Such structures admit efficient construction [11, 24] and compact representation [5], suggesting that voDBGs can be built and stored efficiently as well.

However, a fundamental theoretical gap remains. In fixed-order DBGs, contigs are defined by node in- and out-degrees, and following an edge advances sequence context. In a voDBG, edges represent context change, but also order change[1], and contig definitions such as unitigs and omnitigs [30] do not account for the latter, nor do they leverage the k-mer frequencies, which voDBGs naturally encode and which can inform the assembly. There is no formal contig formulation in voDBGs that integrates the structure of multiple DBG orders with k-mer frequency.

Our contribution

We present the first formal framework to enumerate contigs from a voDBG. Our method takes as input a range $[\ell, h]$ bounding the frequency of sequenced substrings across the genome and restricts the voDBG to nodes whose frequency in the reads lies within $[\ell, h]$.

We combine the frequency restriction with a traversal that visits DBG order edges, varying the context length k along the walk. Building a contig in one direction, the traversal right-extends – increasing k – while the node frequency (i.e., number of occurrences of the associated k-mer in the reads) stays within $[\ell, h]$. When the frequency drops below ℓ, it left-contracts – decreasing k – until the range is recovered, then resumes extending. The mirror operations (left extension, right contraction) build the contig in the opposite direction. This procedure implicitly traces a walk over the layout of overlapping reads without ever building one, placing our approach between DBGs and OLC.

[1] Moving from a node with context length k to another with context length $k' \neq k$

We show that requiring $\ell > h/2$ forces every node in the restricted voDBG to have at most one incoming and one outgoing extension edge (and likewise for contraction edges). This lets us partition the restricted voDBG into paths whose labels spell genome segments under uniform coverage and an error-free model. We refer to the path labels as (ℓ, h)-tigs.

We analyze how the choice of ℓ and h affects the contiguity and accuracy of the assembly and provide a method for choosing appropriate values of ℓ and h under the same assumptions. We further exploit connections between the voDBG and compressed indexes to develop an efficient, homopolymer-aware algorithm for enumerating (ℓ, h)-tigs. We implement our ideas in a tool called `Ryu` and show that our approach achieves better contiguity relative to unitigs in fixed-order DBGs. Full-featured assemblers (`Flye` and `Hifiasm`) still achieve higher contiguity through orthogonal steps – iterative graph cleaning, repeat resolution, and polishing – layered on top of contig enumeration. `Ryu` does not perform such post-processing, yet achieves comparable accuracy at lower computational cost, suggesting that (ℓ, h)-tigs are a strong foundation for an assembly pipeline.

2 Preliminaries

2.1 DNA strings

A *read* $R[1..n]$ is a sequence over the alphabet $\Sigma = \{\mathtt{a}, \mathtt{c}, \mathtt{g}, \mathtt{t}\}$. Standard notions of suffix, prefix, and substring apply. A *run* is a maximal substring of identical symbols. A run of length greater than one is called a *homopolymer*. The run-length encoding of $R = c_1^{l_1} c_2^{l_2} \cdots c_r^{l_r}$ is $rle(R) = (c_1, l_1), (c_2, l_2), \ldots, (c_r, l_r)$.

The *complement* $c : \Sigma \to \Sigma$ maps $\mathtt{a} \leftrightarrow \mathtt{t}$ and $\mathtt{c} \leftrightarrow \mathtt{g}$; the *reverse complement* of R is denoted $\overline{R}$. A *read collection* is a multiset $\mathcal{R} = \{R_1, R_2, \ldots, R_q\}$ of DNA strings. Its extended version $\mathcal{R}_{ext} = \{R_1, \overline{R_1}, R_2, \overline{R_2}, \ldots, R_q, \overline{R_q}\}$ contains each read together with its reverse complement. We denote by ρ the maximum read length.

2.2 de Bruijn graphs

The node-centric *de Bruijn graph* (DBG) of order k [10] is a labeled graph $G = (V, E)$ in which each node $u \in V$ represents a distinct k-mer $label(u) \in \Sigma^k$. There is an edge $(u, v) \in E$ labeled $label(v)[k]$ iff $label(u)[2..k] = label(v)[1..k-1]$.

Given a read collection $\mathcal{R}$, the induced graph $G^{\mathcal{R}} \subseteq G$ contains only those nodes whose labels occur in $\mathcal{R}$ and those edges (u, v) such that $label(u) \cdot label(v)[k]$ occurs in $\mathcal{R}$. We define the operator $f(u)$ to denote the frequency of $label(u)$ in $\mathcal{R}$.

2.3 Variable-order DBGs

The *variable-order de Bruijn graph* (voDBG) [4] of $\mathcal{R}$ combines the DBGs of all orders $1 \le k \le \rho$, where ρ is the maximum read length. Its node set consists of all distinct substrings of $\mathcal{R}$, where the nodes with a given length k are the nodes of a DBG of order k.

For a node u of order k:

Right-extension edge: (u, v) to a node v of order $k+1$ exists if $label(u) = label(v)[1..k]$. The edge is labeled with $label(v)[k+1]$.

Left-contraction edge: (u, w) to a node w of order $k-1$ exists if $label(u)[2..k] = label(w)$.

Left extension and right contraction edges are defined symmetrically.

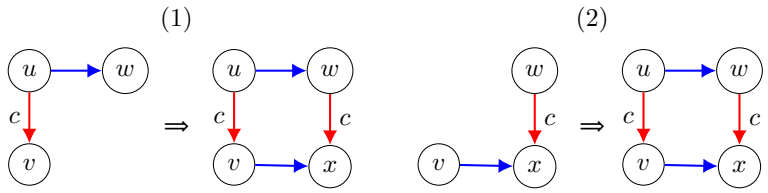

Figure 1 Lemma 3. Red edges are extensions and blue edges are contractions. Extension and contraction edges commute.

3 Our framework

Our assembly method traverses the voDBG of the reads along paths that alternate extension and contraction edges. Extensions refine sequence context and continue while possible; contractions restore context when extensions stop. A contig is formed by the label of the starting node followed by the labels of the visited extension edges.

Extensions decrease frequency and contractions increase it, so we restrict the traversal to nodes whose frequencies lie within the sequencing-coverage interval.

3.1 Description of contigs in the voDBG

In this section, we present our formal concept of contigs in a voDBG $G = (V, E)$ induced from a read collection $\mathcal{R}$.

▶ **Definition 1** (Frequency-restricted voDBG). *Given an interval $[\ell, h]$, a frequency-restricted voDBG $G_{\ell,h} \subseteq G$ retains each node u such that $\ell \le f(u) \le h$ and each edge (u, v) such that $\ell \le f(u) \le h$ and $\ell \le f(v) \le h$.*

▶ **Lemma 2.** *When $h/2 < \ell < h$, each node in $G_{\ell,h}$ has at most one outgoing and at most one incoming extension edge. Similarly, each node has at most one outgoing and at most one incoming contraction edge.*

Proof. Suppose a node $v \in V$ of order $k+1$ has two incoming extension edges (u_1, v) and (u_2, v). Both u_1 and u_2 are the prefixes $label(v)[1..k] = label(u_1) = label(u_2)$ and therefore $u_1 = u_2$. Thus, v has one incoming extension edge.

Now suppose a node $u \in V$ of order k has two outgoing extension edges (u, v_1) and (u, v_2) with $label(v_1)[k+1] \neq label(v_2)[k+1]$. Since $label(v_1)[1..k] = label(v_2)[1..k] = label(u)$, it follows that $f(u) \ge f(v_1) + f(v_2)$.

Because $v_1, v_2 \in V$, we have $f(v_1), f(v_2) \ge \ell$, and therefore $f(u) \ge 2\ell$. Since $\ell > h/2$, we also obtain $2\ell > h$, and hence $f(u) \ge 2\ell > h$. This inequality contradicts $u \in V$, which requires $f(u) \le h$. Therefore, u has at most one outgoing extension edge.

The statement for contraction edges follows by symmetry. ◀

Notice the condition $\ell > h/2$ is tight: if $\ell \le h/2$, branching nodes may exist. From now on, we will assume $\ell > h/2$ holds in $G_{\ell,h}$.

▶ **Lemma 3.** *Extension and contraction edges in $G_{\ell,h}$ commute: (1) if (u, v) is an extension edge labeled c and (u, w) is a contraction edge, there exists a node x such that (v, x) is a contraction edge and (w, x) is an extension edge labeled c and (2) if (w, x) is an extension edge labeled c and (v, x) is a contraction edge, there exists a node u such that (u, w) is a contraction edge and (u, v) is an extension edge labeled c. See Figure 1.*

Proof. (1) Let w, u and v be nodes with orders $k-1$, k and $k+1$, respectively, such that the extension (u, v) and the contraction (u, w) exist. Then it follows $label(v) = label(u)\cdot c$ and $label(w) = label(u)[2..k]$.

Define a node x with $label(x) = label(w)\cdot c = label(u)[2..k]\cdot c$. This implies that (w, x) is an extension edge labeled c. Furthermore, since $label(v) = label(u)\cdot c$, a left contraction yields $label(u)[2..k]\cdot c = label(x)$, so (v, x) is a contraction edge.

An extension decreases frequency and a contraction increases it, so we have $f(v) \le f(x) \le f(w)$. Because $u, v, w \in V$ and their frequencies lie in $[\ell, h]$, it follows that $f(x) \in [\ell, h]$, hence $x \in V$.

(2) follows by symmetry. ◀

Because contraction edges are non-branching (Lemma 2), they partition $G_{\ell,h}$ into maximal directed paths, which we call *contraction paths*.

▶ **Lemma 4.** *Let $U = (u_i)_{1\le i\le n}$ and $P = (p_i)_{1\le i\le m}$ be distinct contraction paths in $G_{\ell,h}$. (1) If there is an extension edge (u_i, p_1) with $1 \le i \le n$ labeled with c, then for every $0 \le j \le n-i$, there is also an extension edge (u_{i+j}, p_{1+j}) labeled c. (2) If there is an extension edge (u_n, p_i) with $1 \le i \le m$ labeled with c, then for every $0 \le j < i$, there is also an extension edge (u_{n-j}, p_{i-j}) labeled c. See Figure 2.*

Proof. (1) By assumption when $j = 0$, the extension edge (u_{i+j}, p_{1+j}) labeled c exists. If $1 \le j \le n-i$ and the extension edge (u_{i+j-1}, p_{1+j-1}) labeled c exists, then by Lemma 3 also the extension edge (u_{i+j}, p_{1+j}) labeled c exists. Thus by induction the extension edge (u_{i+j}, p_{1+j}) labeled c exists for all $0 \le j \le n-i$.

(2) follows by symmetry. ◀

Lemma 4 allows us to lift $G_{\ell,h}$ to a meta-graph:

▶ **Definition 5** (voDBG meta-graph). *The meta-graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ of $G_{\ell,h}$ is defined as follows:*

Each maximal contraction path U in $G_{\ell,h}$ corresponds to a node in $\mathcal{V}$. Since contraction strictly decreases the order, U contains a unique node of maximal order. The sequence $label(U)$ is defined as the label of that node.

Two meta-graph nodes $U, P \in \mathcal{V}$ are connected by an edge $(U, P) \in \mathcal{E}$ labeled c iff there exists an extension edge labeled c in $G_{\ell,h}$ between U and P.

By Lemma 4, if a node $u_i \in U$ has an extension edge labeled c to a node in contraction path P, all successive nodes in U also have an extension edge labeled c to a node in contraction path P. By Lemma 2, the outdegree of these nodes is at most one and thus the outdegree of each node in $\mathcal{G}$ is at most one. Similarly, by Lemmas 4 and 2, the indegree of each node in $\mathcal{G}$ is at most one. Consequently, connected components of $\mathcal{G}$ are directed paths or cycles. We use this structure to define (ℓ, h)-tigs analogously to unitigs in fixed-order DBGs:

▶ **Definition 6** ((ℓ, h)-tigs). *Let $\mathcal{P} = (U_1, U_2, \ldots, U_q)$ be the nodes of a connected component of $\mathcal{G}$, ordered so that $(U_i, U_{i+1}) \in \mathcal{E}$ for $i \in [1, q-1]$. If $\mathcal{P}$ is a directed cycle, the choice of U_1 is arbitrary and $(U_q, U_1) \in \mathcal{E}$. If $\mathcal{P}$ is a directed path, U_1 has indegree 0 and U_q has outdegree 0. An (ℓ, h)-tig is the string formed by $label(U_1)$ followed by the labels of the edges (U_i, U_{i+1}) in $\mathcal{P}$.*

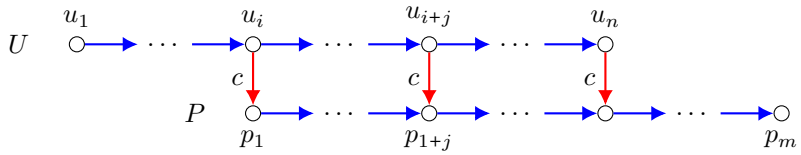

Figure 2 Lemma 4 (1). U and P are distinct contraction paths in $G_{\ell,h}$. If there is an extension edge from some node $u_i \in U$ to p_1, then Lemma 3 implies that all nodes following u_i in U also have an extension edge to a node in P.

3.1.1 Enumerating (ℓ, h)-tigs

We spell (ℓ, h)-tigs by traversing $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. For each $U \in \mathcal{V}$ with indegree 0, we initialize $L = \mathit{label}(U)$ and follow successive extension edges $(U_i, U_{i+1}) \in \mathcal{E}$, appending their labels to L, until reaching a node with outdegree 0. The resulting L is an (ℓ, h)-tig. For components that are directed cycles, we start from an arbitrary node and traverse the cycle once. After visiting all connected components of $\mathcal{G}$, the algorithm outputs one string per component.

3.1.2 Connection with genome assembly

We now describe how our (ℓ, h)-tig enumerating algorithm from Section 3.1.1 relates to genome assembly. Without loss of generality, assume the assembled genome is a single sequence $W[1..w]$, and that the reads in $\mathcal{R}$ come from the same DNA strand.

▶ **Definition 7** (Ground-truth read layout). *For a read $R_i \in \mathcal{R}$, let $[a_i, b_i]$ be the interval from which it originates (i.e., $R_i = W[a_i..b_i]$). The ground-truth read layout is the multiset $\{[a_i, b_i]\}_{i=1}^{q}$ ordered by start position a_i. The layout determines, for each position $W[i]$, the reads that cover it.*

Two reads R_z and R_x, with $a_z \le a_x$ overlap iff $a_x < b_z$, in which case they share $o = b_z - a_x + 1$ symbols and their collapse $R_z[1..n_z]R_x[o+1..n_z]$ equals $W[a_z..b_x]$. Thus, any genome substring is reconstructed by taking the reads that cover it and collapsing their sequences according to their overlaps in the layout. Consequently, any interval of W covered by a chain of pairwise-overlapping reads equals the collapse of that chain. Reconstructing W without knowing the read layout is the assembly problem.

The (ℓ, h)-tig enumeration algorithm performs an approximate traversal of the ground-truth layout. Specifically, walking over the nodes of a component in the meta-graph $\mathcal{G}$ traverses the layout symbol by symbol along W. This traversal is approximate for two reasons. First, each component node U_i identifies all occurrences of a string $\mathit{label}(U_i)$ regardless of their genomic origin, conflating distinct loci such as repeats. Second, the range $[\ell, h]$ constrains k-mer frequency, which only approximates the positional coverage of the layout.

Comparison with other assembly models. Our approach sits between the DBG and OLC models and draws on the strengths of both. From DBGs, it inherits an overlap-free construction: the voDBG is built directly from the k-mers, with no all-against-all overlap computation. From OLC, it inherits the ability to work across multiple DBG orders, extending and contracting the context dynamically as the layout traversal proceeds. This allows the assembly to extend contigs where fixed-order DBGs would fragment them.

3.1.3 Correctness of the reconstruction

Let us consider the conditions under which the (ℓ, h)-tig enumeration algorithm of Section 3.1.1 reconstructs a genome – that is, a single component of $\mathcal{G}$ spells an entire chromosome. We prove the single-chromosome case, assuming $W[1..w]$ is linear and haploid, and write $f(S)$ for the frequency of a string S in $\mathcal{R}$. The case when W is circular is similar.

For the algorithm's approximate layout traversal to reconstruct W, two requirements must hold. First, the occurrences counted by $f(label(U_i))$ must all come from reads overlapping at a single locus; otherwise $label(U_i)$ conflates distinct genomic regions, and its frequency no longer reflects local coverage. Second, every position of W must be covered by some node reachable by the traversal, so that no part of W is skipped. The following theorem makes these requirements precise as frequency conditions on the substrings of W.

▶ **Theorem 8** (Full genome reconstruction). *The (ℓ, h)-tig algorithm of Section 3.1.1 reconstructs W from $\mathcal{R}$ if the following conditions hold:*

1. *If $S \in \Sigma^*$ is a repeat in W, then $f(S) > h$.*
2. *The longest prefix $W[1..i_p]$ such that $f(W[1..i_p]) \in [\ell, h]$ exists.*
3. *For each i, $i_p \le i < w$, there exists a $j \le i$ such that*
   a. *$f(W[j..i]) \in [\ell, h]$ and*
   b. *$f(W[j..i+1]) \in [\ell, h]$.*

Proof. Condition 1 ensures that the node labels of $G_{\ell,h}$ are unique in W. Condition 2 ensures that $W[1..i_p]$ is a node of $G_{\ell,h}$: its frequency is at most h, so by Condition 1 it is not a repeat in W. Because $W[1..i_p]$ is not a repeat and is the longest prefix of W whose frequency is within $[\ell, h]$, it labels a node U_1 in $\mathcal{G}$ with indegree 0 from which the algorithm starts extending an (ℓ, h)-tig. Condition 3 ensures that a string $label(U_i)$ with frequency in $[\ell, h]$ extends by one symbol to a string $label(U_{i+1})$ whose frequency is also in $[\ell, h]$; iterating from U_1 yields a node chain $U_1, U_2, \ldots, U_q$ ending in a node U_q of outdegree 0, with $label(U_q)$ a suffix of W. By Lemmas 2, 3, and 4, these nodes are connected by extension edges into a unary path in $\mathcal{G}$ labeled W (i.e., the (ℓ, h)-tig). ◀

A read collection where each repeat of the underlying genome is fully contained in more than h reads fulfills Condition 1. Conditions 2 and 3 require that the coverage of $\mathcal{R}$ is not too low at any position and that there are no positions (except for the first and last positions) on the genome where many reads start or end (i.e., read endpoints are not clustered).

Real data never meets the conditions of Theorem 8 in practice. We nonetheless analyze assembly under this scenario first because it isolates the structural reason (ℓ, h)-tigs reconstruct the genome. In the next section, we will discuss how to choose $[\ell, h]$ from $\mathcal{R}$ to balance contiguity against the risk of misassembly when the assumptions fail.

3.1.4 Modeling misassemblies

In practice, reads are affected by sequencing errors, unsolved repeats, non-uniform coverage, and allele-specific coverage variation. These factors cause (ℓ, h)-tigs to be interrupted (fragmentation) or deviate from true genomic substrings (misassembly).

Fragmentation occurs when the true genomic continuation of a (ℓ, h)-tig is removed from $G_{\ell,h}$ because its coverage drops below ℓ.

Misassemblies arise when the frequency of a repeat of the genome is within the range $[\ell, h]$ and the number of reads supporting the correct extension decreases and matching reads from other contexts keep their frequency within $[\ell, h]$. This problem combines a lack of coverage with spurious overlaps generated by sequencing errors and/or repeats. Figure 3 shows this scenario.

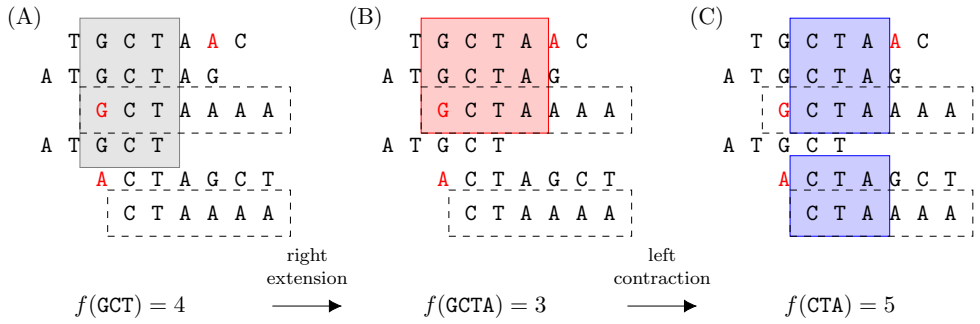

Figure 3 Genome assembly as a sequence of extensions and contractions with $\ell = 3$ and $h = 5$. Overlapping reads represent the ground-truth layout. The dashed reads are misplaced in the layout as they come from other genomic regions but form spurious matches with the correctly placed reads. Red symbols indicate sequencing errors. The colored boxes depict the sequence context the (ℓ, h)-tig algorithm can see at every step. In (A), `GCT` loses one occurrence due to a mismatch in `ACTAGCT`, and gains an occurrence from the spurious overlap with `GCTAAAA`. In (A-B), the right-extension `GCT` → `GCTA` has $f(\texttt{GCTA}) = 3$. Two of its occurrences come from the correct reads in the layout, while the third comes from a spurious overlap. In (C), `CTA` would be incorrectly extended with `A` because of the spurious overlap created by the mismatch in `TGCTAAC`.

When the reads in $\mathcal{R}$ do not satisfy the ideal conditions of Theorem 8, the interval $[\ell, h]$ must balance misassembly and fragmentation. We show how to choose ℓ and h to control the probability of these events.

3.1.5 Fragmentation

Let $\delta = h - \ell + 1$ be the smallest frequency change that pushes a node out of $[\ell, h]$, disrupting the extension of an (ℓ, h)-tig. Connectivity is broken when a contraction or extension changes frequency by at least δ.

Consider a substring $S = W[i..i + k - 1]$. If $\geq \delta$ reads start at i, then contraction from $cS = W[i - 1..i + k - 1]$ yields $f(S) \geq f(cS) + \delta > h$, deleting the edge from cS to S. Symmetrically, if $\geq \delta$ reads end at $i + k - 1$, the extension $Sc = W[i..i + k]$ yields $f(Sc) \leq f(S) - \delta < \ell$, deleting the edge from S to Sc.

Sequencing errors transfer occurrences of S in $\mathcal{R}$ to other strings S', decreasing $f(S)$ and increasing $f(S')$. Given an error rate e, coverage C, and genome size w, the expected number of such transfers is eCw.

Assuming these events – sequencing errors and read endpoints – occur uniformly across the genome, the maximum number of events at a single position bounds the maximum perturbation of a k-mer's frequency. The problem of choosing the values of (ℓ, h) therefore reduces to a balls-into-bins model with $|\mathcal{R}| + eCw$ balls and w bins. By Raab and Steger [28], if

$$\delta \geq 2\frac{\ln w}{\ln \frac{w \ln w}{|\mathcal{R}| + eCw}}, \tag{1}$$

then with high probability[2] no position of W is the start or end of $\geq \delta$ reads. Here ln denotes the natural logarithm.

[2] A high probability here is a probability $1 - o(1)$, i.e., the probability tends to 1 as w grows to infinity.

3.1.6 Misassemblies

Suppose S has two genomic extensions Sc_1 and Sc_2. Under uniform coverage their read counts follow a binomial distribution with parameter 0.5. A misassembly occurs if one extension has frequency $i \in [\ell, h]$ (kept) while the other $f(S) - i < \ell$ (discarded). The probability of this happening is $P \leq 2\sum_{i=\ell}^{h} \binom{h}{i} 0.5^i 0.5^{h-i} = 2^{1-h} \sum_{i=\ell}^{h} \binom{h}{i}$.

Let $d = \ell - h/2$ denote the deviation from the mean. By the Chernoff bound, $P \leq 2e^{-2d^2/h}$. If a misassembly probability at most ϵ is tolerated, then

$$\ell \geq \frac{h}{2} + \sqrt{\frac{h}{2} \ln\left(\frac{2}{\epsilon}\right)}. \tag{2}$$

This analysis applies to the two-extension case. Combining Equations 1 and 2, we require

$$\frac{h}{2} + \sqrt{\frac{h}{2} \ln\left(\frac{2}{\epsilon}\right)} \leq \ell \leq h - 1 - 2\frac{\ln w}{\ln \frac{w \ln w}{|\mathcal{R}| + eCw}}. \tag{3}$$

We choose the smallest h for which an integer ℓ satisfies Equation 3.

Scope and limitations. It is worth noting that the interval $[\ell, h]$ of Equation 3 neither minimizes fragmentation nor maximizes contiguity. Rather, it gives values of ℓ and h sufficient to keep the per-node misassembly probability below ϵ. The analysis remains idealized, as it assumes frequency fluctuations are distributed uniformly across W; non-uniform and allele-specific coverage break that assumption. Its value is that, even in this simplified setting, sequencing errors are budgeted explicitly into the fluctuation bound δ – unlike the standard fixed-order DBG formulation, where the choice of k does not account for frequency or error rate. In order to compensate for the lack of both uniform total and local coverage in empirical reads, we later define a way to calculate a recommended set of parameters (see Section 5.1).

4 Practical genome assembler

We present a practical long-read assembler based on (ℓ, h)-tigs. Because sequencing samples both DNA strands, the input is the extended collection $\mathcal{R}_{ext}$ containing each read and its reverse complement. We assume suitable values of (ℓ, h) (Section 3.1.6) computed from $\mathcal{R}$ and doubled for $\mathcal{R}_{ext}$.

Long-read technologies often misestimate homopolymer lengths, producing spurious overlaps and potential misassemblies. We incorporate a simple heuristic to mitigate this issue.

4.1 Preparing the data

Similar to previous work [23, 29, 1], our homopolymer-error heuristic keeps each read $R \in \mathcal{R}_{ext}$ in run-length format $rle(R) = (c_1, l_1)(c_2, l_2), \ldots, (c_r, l_r)$. We separate the compressed read into its *symbol sequence* $c_1, c_2, \ldots, c_r$ and *length sequence* $l_1, l_2, \ldots, l_r$. Let $\mathcal{R}_c$ be the collection of symbol sequences.

We index $\mathcal{R}_c$ using a compressed structure I with suffix-tree functionality (see Section 5) that supports navigational queries in the voDBG $G_{\ell,h} = (V, E)$ of $\mathcal{R}_c$. The length sequences are integrated into I so that for each node $u \in V$ we can query

- $hp(u)$: return the list of length sequences corresponding to the uncompressed occurrences of $label(u)$ in $\mathcal{R}_{ext}$.

The index I maps all occurrences of a substring S in $\mathcal{R}_{ext}$ to the same node u in $G_{\ell,h}$ based on their symbol sequence, ignoring homopolymer length differences. This removes spurious overlaps caused by homopolymer errors. However, different strings $S \neq S'$ with the same symbol sequence may also collapse to the same node. The lists returned by $hp(u)$ allow us to detect such cases: homopolymer errors produce small deviations around the true lengths, whereas multiple strings typically generate distinct length distributions.

Because reverse complements are included, the meta-graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ contains two components per chromosome, one for each DNA strand. For a node $U \in \mathcal{V}$ we denote by $\overline{U}$ the node labeled with the reverse complement of $label(U)$. We also define $hp(U) = hp(u_1)$, where u_1 is the highest-order node of U.

4.2 Assembly

The algorithm selects a pair of nodes $U, \overline{U} \in \mathcal{V}$ with indegree and outdegree 0, respectively. These correspond to the starting and ending points of complementary components of $\mathcal{G}$.

It reconstructs an uncompressed sequence for $S = label(U)$ using the list $\mathcal{L} = hp(U)$. For each position i in S, the homopolymer length is estimated as the median of the lengths at position i in $\mathcal{L}$. If the standard deviation exceeds a threshold τ, the length at position i is marked as *fuzzy*.

After processing all positions, if the fraction of fuzzy lengths exceeds a threshold μ, $(U, \overline{U})$ is discarded and the algorithm moves to another component of $\mathcal{G}$. Otherwise, the algorithm spells the (ℓ, h)-tig starting at U. The uncompressed length of each extension label c is estimated in the same way using the median.

Once the (ℓ, h)-tig is completed, the nodes of U and $\overline{U}$ are marked as processed, and the algorithm continues with the next unvisited pair $(U, \overline{U})$.

5 Experiments

5.1 Setup

Implementation details. We implemented our (ℓ, h)-tig assembler (Section 4) in `C++` as `Ryu`[3]. The tool builds a compressed FMD-index [17] over the symbol sequences $\mathcal{R}_c$. The index uses the bidirectional BWT [16], computed with `grlBWT` [11]. Traversing BWT ranges corresponding to suffix tree nodes enumerates voDBG nodes u and their frequencies. Run-length encoding lengths from $\mathcal{R}_{ext}$ are stored in BWT order to support $hp(u)$ queries, enabling homopolymer reconstruction. `Ryu` then visits the connected components of the meta-graph $\mathcal{G}$ via BWT backward search and spells (ℓ, h)-tigs as described in Section 4.

`Ryu` currently supports limited parallelization, using up to four threads. To distribute work among them, the FMD-index is divided into ranges, which are then assigned to threads to compute the (ℓ, h)-tigs in parallel. However, without first processing the FMD-index, it is difficult to ensure that each thread receives a similar workload. Therefore, we adopt a simpler heuristic that partitions the data structure according to the DNA alphabet.

Datasets. We used three PacBio HiFi read collections: *E. Coli* (ECOLI), *S. Cerevisiae* (YEAST), and the human cell line CHM13 (HUMAN). While ECOLI and HUMAN are complete genomes, the YEAST genome is at a contig level. We assembled these datasets with different

[3] `https://github.com/ddiazdom/Ryu`, commit `0eb064a`

methods and evaluated the output using `Quast` [13]. Table 1 lists the SRA accessions of the reads and the reference genomes used for evaluation, alongside the genome sizes, and reads coverage.

Competitor tools. We compared `Ryu` with `Bcalm2`[4] [7], `Flye`[5] [15], and `Hifiasm`[6] [6] for assembly. `Bcalm2` outputs unitigs from a fixed-order DBG, `Flye` constructs A-Bruijn and repeat graphs, and `Hifiasm` follows an OLC strategy based on minimizer-detected overlaps. `Bcalm2` and `Ryu` only prune the graph based on k-mer frequency and spell sequences from the pruned graph, whereas `Hifiasm` and `Flye` perform full genome assembly. The YEAST reference genome has been assembled using `Flye v.2024.0.7`, so the results may be biased for this particular dataset.

Experimental setup. For all the compared tools, a set of common parameters have been used for all the datasets, while for `Ryu` some particular dataset-specific parameters have been added. For `Ryu`, we varied ℓ $(9-24)$ and h $(16-32)$. While assembling in parallel, in order to avoid contention in fragmented assemblies, we omit (ℓ, h)-tigs with length smaller than 1000 base pairs using `-r 1000` (see discussion in Section 5.2.3) hence renouncing less than 1% of the total length of the (ℓ, h)-tigs assembly. For `Bcalm2`, k was varied in the range between 2^4 and 2^8, and the abundance threshold (`-abundance-min`) from 7 to 15, for a total of 75 different parameters combinations. `Flye` and `Hifiasm` were run with default parameters. The parameters common to each dataset are shown in detail in Table A1.

For each dataset, we calculated the theoretical optimal values of (ℓ, h) for each dataset (Section 3.1.6) with confidence rate 0.95, and the recommended range $[\ell, h]$, on which h is set to half of the coverage and $\ell = \frac{2}{3}h$. The coverage has been calculated by our tool `Ryu` as the rate between the number of symbols in the index I and the expected size given as input. In our experiments, setting the interval $[\ell, h]$ this way produced good assemblies for all datasets. The experiments were executed with 24 threads to allow a comparison of memory usage and time between the tools. The only exception was `Ryu`, which used up to 4 threads due to implementation limitations.

Figure 4 illustrates the tradeoffs between assembly quality and the choice of the values (ℓ, h) (exact values are reported in Table A1). Assembly statistics for the compared tools are summarized in Table 2, while peak memory usage and runtime are listed in Table 3. For `Bcalm2`, we report the configuration with the fewest misassemblies in each dataset. For `Ryu`, we highlight three configurations: *best assembly* (`Ryu` ★), which achieves the highest N50 and genome coverage, *theoretical best* (`Ryu`▲), with $[\ell, h]$ calculated according to Equation 3, and *best performance* (`Ryu` ◆), which completes the assembly of reads in the shortest time. `Ryu` ◆ coincides with the recommended parameters.

Machine specifications. We performed the experiments in a cluster environment running AlmaLinux 8.10 with kernel Linux 4.18.0, using a single node per execution: the node uses an Intel(R) Xeon(R) Gold 6248 CPU, with 80 threads and a maximum frequency of 3.9 GHz, with 380GB RAM. All tools were compiled with `GCC 14.2.0` according to the authors' specifications. We compiled `Ryu` using `-O3`. To achieve speed up on larger values of k, `Bcalm2` has been compile by providing a custom `-DKSIZE_LIST`.

[4] `https://github.com/GATB/bcalm`, commit `fe8a8c5`
[5] `https://github.com/mikolmogorov/Flye`, commit `886b8c1`
[6] `https://github.com/chhylp123/hifiasm`, commit `ec9a8b2`

Table 1 Species and reference genomes used in the experiments. The reference genomes were downloaded from the NCBI database: `https://www.ncbi.nlm.nih.gov/datasets/genome`.

Scientific name	Dataset	SRA accession	Ref. genome GenBank acc.	Genome size (Mbp)	Seq. coverage
E. Coli	ECOLI	SRR10971019	GCA_000005845.2	4.64	300
S. Cerevisiae	YEAST	SRR34579673	GCA_051942865.1	11.36	676
H. Sapiens	HUMAN	SRR11292121	GCA_009914755.1	3,117.28	9.57

5.2 Results and discussion

5.2.1 Effect of ℓ and h on the assembly

Figure 4 shows how different choices of ℓ and h affect the results of the assembly. These experiments reveal a tradeoff between contiguity and correctness controlled by the assembly parameters. Additionally, the different coverage and increasing repeat complexity of the datasets introduces further challenges that influence this tradeoff.

For ECOLI (first row of Figure 4), small values of ℓ and h produce a larger number of (ℓ, h)-tigs, whereas larger values reduce their number. As a consequence, assemblies generated with small ℓ, h values are more fragmented and yield smaller N50 values. In contrast, the number of misassemblies follows the opposite trend: smaller values of ℓ and h produce fewer misassemblies, while larger values increase them.

This behavior can be explained by the way the range $[\ell, h]$ is used. When ℓ and h are small, the first node u from which the extension and an (ℓ, h)-tig can be extended typically has a long label $label(u)$. This happens because the assembly algorithm requires several extensions before reaching a node u such that $f(u) \in [\ell, h]$. At this stage, no contractions have yet been performed, so the assembly has not introduced misassemblies and the constructed prefix of the (ℓ, h)-tig is correct. However, the subsequent chain of contractions and extensions starting from u tends to terminate quickly: because the interval $[\ell, h]$ is narrow for small values of (ℓ, h) (by Lemma 2), the frequency of the extending (ℓ, h)-tig soon falls outside the allowed range, which limits further extension and results in shorter contigs.

The situation in YEAST is somewhat different (second row of Figure 4). As in ECOLI, increasing (ℓ, h) reduces the number of reconstructed sequences. However, the number of misassemblies is the smallest for the recommended values of (ℓ, h). Otherwise the number of misassemblies is minimized for the intermediate choice $\ell = 17, h = 31$. This more complex behaviour likely reflects the higher structural complexity of the dataset, where unsolved repeats in the reads and homopolymer compression introduce spurious overlaps.

The coverage of the HUMAN dataset is much lower than that of ECOLI and YEAST datasets and thus the results are different. The results for HUMAN (third row of Figure 4) show that the number of (ℓ, h)-tigs first increases when $[\ell, h]$ increase. However, after ℓ gets larger than the coverage, 9.57x, the number of (ℓ, h)-tigs starts to decrease because most of the node labels do not have higher occurrence frequencies in the reads and thus the assembly captures less and less of the genome. The assemblies obtained with larger (ℓ, h) values also exhibit smaller N50 values, indicating increased fragmentation due to the low coverage.

The number of misassemblies in HUMAN shows an opposite pattern compared to ECOLI. Here, larger values of (ℓ, h) lead to fewer misassemblies. In this regard, the assembler achieves zero misassemblies for $(\ell = 13, h = 22)$, $(\ell = 20, h = 26)$, and $(\ell = 24, h = 32)$, although these assemblies were highly fragmented.

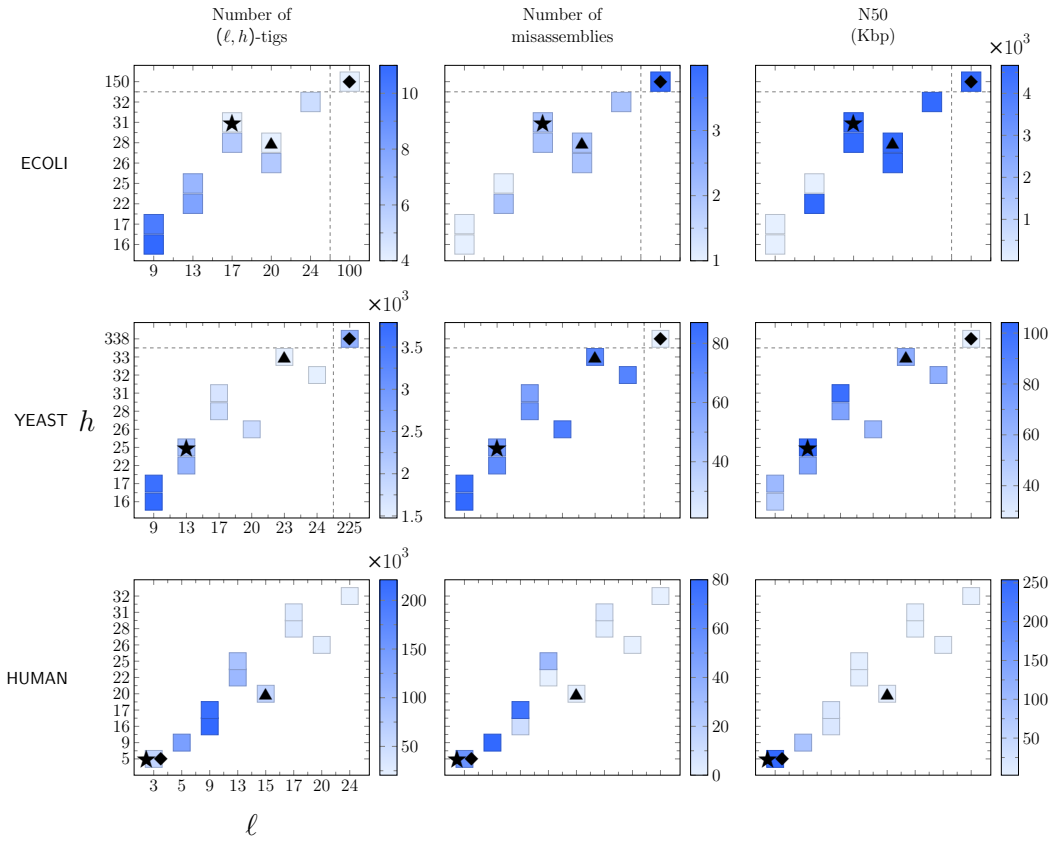

Figure 4 Effect of ℓ and h on the assembly. Each row corresponds to a dataset. The x-axis shows ℓ and the y-axis shows h. Axes are displayed only in the first column of each row because they are identical across panels. ★, ◆, and ▲ indicate the best, fastest, and theoretical optimum assemblies, respectively. The values of ℓ and h (x- and y-axis) do not increase evenly. In particular, the highest ℓ and h values on ECOLI and YEAST deviate a lot from the other values and are thus separated by the dashed lines.

The theoretically optimal values for (ℓ, h) give a guarantee for avoiding misasemblies with the given probability threshold, while at the same time maximizing contiguity. Thus these values attempt to balance these two effects, and we see that the theoretically optimal values for $[\ell, h]$ do not minimize the number of misassemblies or maximize the contiguity on any of the datasets. On the HUMAN dataset both extremes of the theoretically optimal range, $[\ell, h] = [15, 20]$, are larger than the coverage of the dataset indicating that according to the theory, larger coverage would be needed to guarantee a misassembly probability of 0.95. Still, a reasonable assembly is produced with lower values of (ℓ, h), which may be due to the analysis not being tight.

Overall, these results highlight the trade-off inherent in the choice of the pair (ℓ, h). Smaller values favour aggressive extension, producing longer sequences but increasing the risk of misassemblies caused by spurious overlaps. Conversely, larger values restrict extension, which reduces misassemblies but leads to more fragmented assemblies. In practice, choosing h to be half the coverage of the dataset and $\ell = \frac{2}{3}h$ provides a good compromise between contiguity and number of misassemblies. However for datasets with high coverage, a better assembly may be achieved with lower (ℓ, h) pair because `Ryu` can then take advantage of longer contexts and thus bridge repeats better.

Table 2 Assembly statistics for `Ryu`, `Bcalm2`, `Hifiasm`, and `Flye`. The misassemblies column shows the number of misassemblies reported by `QUAST`, divided into relocations **R** (flanking sequences align over 1 kbp apart), translocations **T** (flanking sequences align on different chromosomes), and inversions **I** (flanking sequences align on opposite strands). For `Ryu`, ★, ◆, and ▲ indicate the best, fastest, and theoretical optimum assemblies, respectively. The fastest (◆) assembly coincides with the recommended parameters on all datasets.

Dataset	Tool	Assembly size (Mbp)	N50 (Kbp)	Misassemblies			Indels	Total contigs
				R	**T**	**I**		
ECOLI	`Ryu` ★	4.67	4,665.84	1	0	0	5	4
	`Ryu` ◆	4.65	4,640.70	1	0	2	3	4
	`Ryu` ▲	4.64	4,640.70	1	0	0	3	4
	`Bcalm2`	4.53	2.19	0	0	0	1	2,659
	`Hifiasm`	4.67	4,665.56	1	0	0	4	1
	`Flye`	4.64	1,182.54	1	0	0	3	7
YEAST	`Ryu` ★	19.27	104.14	38	26	0	1,925	2,322
	`Ryu` ◆	12.02	47.16	7	11	0	457	2,522
	`Ryu` ▲	23.61	65.33	47	26	0	1,604	1,476
	`Bcalm2`	10.98	1.43	0	0	0	355	9,753
	`Hifiasm`	33.27	138.67	463	25	0	4,334	890
	`Flye`	11.25	759.17	0	8	0	238	823
HUMAN	`Ryu` ★◆	3,138.71	253.98	46	8	1	302,008	62,016
	`Ryu` ▲	185.98	3.92	0	0	0	2,601	61,922
	`Bcalm2`	1,249.91	1.93	0	0	0	157,855	827,231
	`Hifiasm`	2,853.07	3,967.22	177	63	1	157,855	1,453
	`Flye`	2,789.13	5,953.40	108	128	2	190,702	2,085

5.2.2 Comparison with other methods

The assembly quality of `Ryu` is comparable to that of long reads assemblers (`Hifiasm`, `Flye`) on ECOLI, and substantially better than unitigs in fixed-order DBG assemblers on HUMAN. On YEAST, the assembly quality of `Ryu` is comparable to that of `Hifiasm`, whereas the assembly quality of `Flye` is better. However, since the used reference genome was also produced with `Flye`, the comparison might be biased towards `Flye`. At the same time, the computational efficiency of `Ryu` remains closer to that of DBG-based tools, highlighting its potential as a lightweight alternative to OLC assemblers. We observe minor differences between the best (★), the theoretical best (▲), and the fastest/recommended (◆) `Ryu` configurations.

Table 2 shows that `Bcalm2` produces a much more fragmented assembly than `Ryu` across all datasets, with very low N50 values and a large number of contigs. By leveraging multiple values of k, `Ryu` substantially improves contiguity, producing assemblies with N50 values over 40x higher.

On ECOLI, both `Ryu` and `Hifiasm` reconstruct the genome in a single large contig. `Ryu` produces a few additional short contigs that likely correspond to plasmids or sequencing artifacts, whereas `Flye` splits the genome into several contigs. All methods show similarly low numbers of misassemblies on this dataset.

On the more complex YEAST and HUMAN datasets, `Ryu` produces assemblies that are more contiguous than those of `Bcalm2` but less contiguous than those of `Hifiasm` and `Flye`. However, `Ryu` introduces fewer misassemblies than `Hifiasm` on both datasets and fewer than

Table 3 Runtime and peak memory usage for the tools used in the experiments. The instance and parameters of `Bcalm2` are the ones yielding the highest N50 value. For `Ryu`, indexing, assembling, and total time are shown for the best (★), fastest (◆), and theoretical optimum (▲) assemblies. The fastest (◆) coincides with the recommended parameters. The values next to `Bcalm2` correspond to k and `-abundance-min`, respectively.

Dataset	Tool			Runtime (sec)	Peak memory (GB)
ECOLI	`Ryu`	★	*indexing*	176	0.63
			assembly	210	0.09
			total	386	0.63
		◆	*assembly*	174	0.09
			total	350	0.63
		▲	*assembly*	243	0.09
			total	419	0.63
	`Bcalm2`		(224, 15)	73	6.37
	`Hifiasm`			888	16.64
	`Flye`			2,569	11.43
YEAST	`Ryu`	★	*indexing*	1,315	2.80
			assembly	1,550	0.42
			total	2,865	2.80
		◆	*assembly*	579	0.39
			total	1,894	2.80
		▲	*assembly*	1,432	0.41
			total	2,747	2.80
	`Bcalm2`		(224, 13)	276	7.03
	`Hifiasm`			4,974	35.62
	`Flye`			21,408	28.95
HUMAN	`Ryu`	★◆	*indexing*	7,666	12.46
			assembly	6,017	12.83
			total	13,683	12.83
		▲	*assembly*	17,909	7.07
			total	25,575	12.46
	`Bcalm2`		(40, 7)	1,580	6.92
	`Hifiasm`			3,261	49.23
	`Flye`			48,641	104.54

`Flye` on HUMAN. `Flye` achieves the most contiguous and accurate assembly on YEAST, although this comparison may be biased because the reference genome was also assembled using `Flye`.

In terms of computational efficiency (Table 3), `Bcalm2` is the fastest method. Nevertheless, `Ryu` is substantially faster than `Hifiasm` and `Flye` on ECOLI and YEAST, while remaining faster than `Flye` on HUMAN. This comparison is conservative for `Ryu`, since the other assemblers use 24 threads, whereas `Ryu` currently uses four. Moreover, a significant fraction of the runtime in `Ryu` corresponds to index construction.

Finally, `Ryu` consistently uses less memory than full assemblers, illustrating the low memory usage of DBG-based approaches. `Bcalm2` uses even less memory than `Ryu` on the largest dataset because it builds a fixed-order DBG, but this comes at the cost of a substantially increased fragmentation.

5.2.3 A note on parallel efficiency

When `Ryu` runs in multi-threaded mode with values of ℓ and h that produce fragmented assemblies, its runtime increases. Parallel execution computes (ℓ, h)-tigs across multiple threads while maintaining shared state to avoid generating both DNA strands of the same (ℓ, h)-tig independently. In highly fragmented assemblies, threads spend little time extending the contig and more time synchronizing the shared information, leading to high contention. As a result, multi-threaded execution can paradoxically become slower than single-threaded mode. In our experiments, we dealt with this situation by filtering (ℓ, h)-tigs shorter than 1000 base pairs: threads discard such contigs and skip the shared-state update. Although this threshold is arbitrary, it had a negligible impact on genome coverage (see Table 2).

6 Conclusions and further work

We presented a theoretical framework for assembling long reads using voDBGs. Our experiments show that `Ryu` achieves a strong compromise between contiguity, correctness, and efficiency, significantly improving over fixed-order DBGs while remaining lighter than long-read assemblers. These results suggest that voDBG-based assemblers can offer a practical alternative to expensive OLC methods in long reads.

Further improvements could make our approach a practical full-fledged *de novo* assembler. Advances in read indexing and misassembly detection may improve both efficiency and accuracy, while ongoing progress in algorithms for constructing and indexing large BWTs is likely to enable more scalable implementations. Additionally, dynamically adjusting the interval $[\ell, h]$ based on local read features could help reduce misassemblies, although identifying the most informative signals remains a challenge. On the other hand, scaffolding strategies that exploit the information in the read index may help reduce fragmentation. Finally, extending the framework to polyploid genomes could enable efficient reconstruction of more complex genomes.

References

1 Anton Bankevich, Andrey Bzikadze, Mikhail Kolmogorov, Dmitry Antipov, and Pavel A. Pevzner. LJA: Assembling long and accurate reads using multiplex de Bruijn graphs. bioRxiv, 2021. `doi:10.1101/2020.12.10.420448`.

2 Anton Bankevich, Sergey Nurk, Dmitry Antipov, Alexey A. Gurevich, Mikhail Dvorkin, Alexander S. Kulikov, Valery M. Lesin, Sergey I. Nikolenko, Son Pham, Andrey D. Prjibelski, et al. SPAdes: A new genome assembly algorithm and its applications to single-cell sequencing. *Journal of Computational Biology*, 19(5):455–477, 2012. `doi:10.1089/cmb.2012.0021`.

3 Konstantin Berlin, Sergey Koren, Chen-Shan Chin, James P Drake, Jane M Landolin, and Adam M Phillippy. Assembling large genomes with single-molecule sequencing and locality-sensitive hashing. *Nature biotechnology*, 33(6):623–630, 2015. URL: `https://www.nature.com/articles/nbt.3238`.

4 Christina Boucher, Alex Bowe, Travis Gagie, Simon J. Puglisi, and Kunihiko Sadakane. Variable-order de Bruijn graphs. In Ali Bilgin, Michael W. Marcellin, Joan Serra-Sagristà, and James A. Storer, editors, *2015 Data Compression Conference, DCC 2015, Snowbird, UT, USA, April 7-9, 2015*, pages 383–392. IEEE, 2015. `doi:10.1109/DCC.2015.70`.

5 Alexander Bowe, Taku Onodera, Kunihiko Sadakane, and Tetsuo Shibuya. Succinct de Bruijn graphs. In Benjamin J. Raphael and Jijun Tang, editors, *Proc. 12th International Workshop on Algorithms in Bioinformatics (WABI)*, volume 7534 of *Lecture Notes in Computer Science*, pages 225–235. Springer, 2012. `doi:10.1007/978-3-642-33122-0_18`.

6 Haoyu Cheng, Gregory T Concepcion, Xiaowen Feng, Haowen Zhang, and Heng Li. Haplotype-resolved de novo assembly using phased assembly graphs with hifiasm. *Nature methods*, 18(2):170–175, 2021. doi:10.1038/s41592-020-01056-5.

7 Rayan Chikhi, Antoine Limasset, and Paul Medvedev. Compacting de bruijn graphs from sequencing data quickly and in low memory. *Bioinform.*, 32(12):201–208, 2016. doi:10.1093/bioinformatics/btw279.

8 Rayan Chikhi and Paul Medvedev. Informed and automated k-mer size selection for genome assembly. *Bioinformatics*, 30(1):31–37, 2014. doi:10.1093/bioinformatics/btt310.

9 Chen-Shan Chin and Asif Khalak. Human genome assembly in 100 minutes. bioRxiv, 2019. URL: https://www.biorxiv.org/content/biorxiv/early/2019/07/17/705616.full.pdf.

10 Nicolaas Govert De Bruijn. A combinatorial problem. *Proc. Section of Sciences of the Koninklijke Nederlandse Akademie van Wetenschappen te Amsterdam*, 49(7):758–764, 1946. URL: https://research.tue.nl/files/4442708/597473.pdf.

11 Diego Díaz-Domínguez and Gonzalo Navarro. Efficient construction of the bwt for repetitive text using string compression. *Information and Computation*, 294:105088, 2023. doi:10.1016/j.ic.2023.105088.

12 Barış Ekim, Bonnie Berger, and Rayan Chikhi. Minimizer-space de Bruijn graphs: Whole-genome assembly of long reads in minutes on a personal computer. *Cell Systems*, 12(10):958–968, 2021. doi:10.1016/j.cels.2021.08.009.

13 Alexey Gurevich, Vladislav Saveliev, Nikolay Vyahhi, and Glenn Tesler. QUAST: quality assessment tool for genome assemblies. *Bioinformatics*, 29(8):1072–1075, 2013. doi:10.1093/bioinformatics/btt086.

14 Ramana M. Idury and Michael S. Waterman. A new algorithm for DNA sequence assembly. *Journal of Computational Biology*, 2(2):291–306, 1995. doi:10.1089/cmb.1995.2.291.

15 Mikhail Kolmogorov, Jeffrey Yuan, Yu Lin, and Pavel A Pevzner. Assembly of long, error-prone reads using repeat graphs. *Nature biotechnology*, 37(5):540–546, 2019. doi:10.1038/s41587-019-0072-8.

16 Tak Wah Lam, Ruiqiang Li, Alan Tam, Simon Wong, Edward Wu, and Siu-Ming Yiu. High throughput short read alignment via bi-directional BWT. In *Proc. IEEE international conference on bioinformatics and biomedicine (BIBM)*, pages 31–36. IEEE Computer Society, 2009. doi:10.1109/BIBM.2009.42.

17 Heng Li. Exploring single-sample SNP and INDEL calling with whole-genome de novo assembly. *Bioinformatics*, 28(14):1838–1844, 2012. doi:10.1093/bioinformatics/bts280.

18 Heng Li. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics*, 34(18):3094–3100, 2018. doi:10.1093/bioinformatics/bty191.

19 Yu Lin and Pavel A. Pevzner. Manifold de Bruijn graphs. In Dan Brown and Burkhard Morgenstern, editors, *Proc. 14th International Workshop on Algorithms in Bioinformatics (WABI)*, Lecture Notes in Computer Science, pages 296–310. Springer, 2014. doi:10.1007/978-3-662-44753-6_22.

20 Zhiqiang Liu, Xue Li andLei Xie, Bin Wang, Shihua Zhou, Ben Cao, Pan Zheng, and Qiang Zhang. Dvoug enables robust dna sequence assembly and reconstruction with a dynamic, variable-order graph. *Cell Reports Methods*, 5(12):101243, 2025. doi:10.1016/j.crmeth.2025.101243.

21 Eugene W Myers. Toward simplifying and accurately formulating fragment assembly. *Journal of Computational Biology*, 2(2):275–290, 1995. doi:10.1089/cmb.1995.2.275.

22 Gonzalo Navarro and Veli Mäkinen. Compressed full-text indexes. *ACM Comput. Surv.*, 39(1):2, 2007. doi:10.1145/1216370.1216372.

23 Sergey Nurk, Brian P. Walenz, Arang Rhie, Mitchell R. Vollger, Glennis A. Logsdon, Robert Grothe, Karen H. Miga, Evan E. Eichler, Adam M. Phillippy, and Sergey Koren. Hicanu: accurate assembly of segmental duplications, satellites, and allelic variants from high-fidelity long reads. *Genome Research*, 30(9):1291–1305, 2020. URL: https://genome.cshlp.org/content/30/9/1291.

24 Jannik Olbrich. Fast and Memory-Efficient BWT Construction of Repetitive Texts Using Lyndon Grammars. In Anne Benoit, Haim Kaplan, Sebastian Wild, and Grzegorz Herman, editors, *Proc. 33rd Annual European Symposium on Algorithms (ESA)*, volume 351 of *LIPIcs*, pages 60:1–60:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPIcs.ESA.2025.60.

25 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):132, 2016. URL: https://link.springer.com/article/10.1186/s13059-016-0997-x.

26 Yu Peng, Henry C.M. Leung, Siu-Ming Yiu, and Francis Y.L. Chin. IDBA–a practical iterative de Bruijn graph de novo assembler. In Bonnie Berger, editor, *Proc. 14th Annual International Conference on Research in Computational Molecular Biology (RECOMB)*, volume 6044 of *Lecture Notes in Computer Science*, pages 426–440. Springer, 2010. doi:10.1007/978-3-642-12683-3_28.

27 Pavel A. Pevzner, Haixu Tang, and Michael S. Waterman. An eulerian path approach to DNA fragment assembly. *Proceedings of the National Academy of Sciences*, 98(17):9748–9753, 2001. doi:10.1073/pnas.171285098.

28 Martin Raab and Angelika Steger. "Balls into bins" — a simple and tight analysis. In Michael Luby, José D. P. Rolim, and Maria J. Serna, editors, *Proc. 2nd International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM)*, volume 1518 of *Lecture Notes in Computer Science*, pages 159–170, 1998. doi:10.1007/3-540-49543-6_13.

29 Mikko Rautiainen and Tobias Marschall. MBG: Minimizer-based sparse de Bruijn graph construction. *Bioinformatics*, 37(16):2476–2478, 2021. doi:10.1093/bioinformatics/btab004.

30 Alexandru I. Tomescu and Paul Medvedev. Safe and complete contig assembly through omnitigs. *Journal of Computational Biology*, 24(6):590–602, 2017. doi:10.1089/cmb.2016.0141.

31 Aaron M. Wenger, Paul Peluso, William J. Rowell, Chang, et al. Accurate circular consensus long-read sequencing improves variant detection and assembly of a human genome. *Nature Biotechnology*, 37:1155–1162, 2019. doi:10.1038/s41587-019-0217-9.

32 Daniel R. Zerbino and Ewan Birney. Velvet: Algorithms for de novo short read assembly using de bruijn graphs. *Genome Research*, 18(5):821–829, 2008. doi:10.1101/gr.074492.107.

A Appendix

Table A1 General parameter set for the experiments. For `Ryu`, we divide the general case of choosing the pair (ℓ, h) manually and the parameters chosen for calculating the theoretical best. In both cases, a parameter `-r` has been set to discard (ℓ, h)-tigs smaller than 1000 bases to speed up calculations. The number of threads for `Ryu` is set to 4 since a number of threads larger than the size of the considered alphabet will not yield any improvements.

Tool	Parameters	Values	Notes
`Ryu`	ℓ	9 9 13 13 17 17 20 24	Threads = 4 `-r = 1000`
	h	16 17 22 25 28 31 26 32	
	Confidence Rate	0.95	
	Error rate	0.001	
	Estimated Length	depends on the dataset	
`Bcalm2`	`-kmer-size`	16 20 32 40 64 96 128 150 160 192 200 224 256	Threads = 24
	`-abundance-min`	7 9 11 13 15	
`Flye`		No specified parameters	Threads = 24
`Hifiasm`		No specified parameters	Threads = 24

Table A2 Comparison between all the instances of `Ryu` with `Hifiasm`, `Flye` and the best two instances of `Bcalm2`. For `Ryu`, ★, ◆, and ▲ indicate the best, fastest, and theoretical optimum assemblies, respectively. The fastest (◆) assembly coincides with the recommended parameters on all datasets.

Dataset	Tool	Parameter	N50 (Kbp)	Misassemblies	Largest Contig (Kbp)	Total Contigs
ECOLI	Bcalm2	128,13	1.20	0	7.07	3,301
		224,15	2.19	0	15.12	2,659
	Flye		1,182.54	1	2,087.87	7
	Hifiasm		4,665.56	1	4,665.56	1
	Ryu	9,16	13.29	0	25.67	11
		9,17	13.29	0	25.67	10
		13,22	4,654.42	1	4,654.42	8
		13,25	24.55	0	24.55	7
		17,28	4,640.70	1	4,640.70	6
		17,31 ★	4,665.84	1	4,665.84	4
		20,26	4,640.70	1	4,640.70	6
		20,28 ▲	4,640.70	1	4,640.70	4
		24,32	4,640.70	1	4,640.70	5
		100,150 ◆	4,640.70	3	4,640.70	4
YEAST	Bcalm2	224,13	1.43	0	9.66	9,753
		256,15	0.89	1	15.85	22,545
	Flye		759.17	8	1,471.70	23
	Hifiasm		138.67	488	1,357.40	890
	Ryu	9,16	47.43	87	505.03	3,790
		9,17	58.66	85	590.06	3,673
		13,22	70.44	70	391.44	2,508
		13,25 ★	104.14	64	626.48	2,322
		17,28	71.48	66	381.41	1,825
		17,31	100.67	60	511.24	1,675
		20,26	60.30	77	322.16	1,848
		23,33 ▲	65.33	73	322.15	1,476
		24,32	64.83	74	322.15	1,484
		225,338 ◆	23.31	21	322.04	2,565
HUMAN	Bcalm2	32,7	1.73	0	25.15	827,231
		40,7	1.93	0	28.41	846,616
	Flye		5,953.40	238	26,830.28	2,085
	Hifiasm		3,967.22	241	33,485.86	1,453
	Ryu	3,5 ★◆	253.98	55	1,900.07	62,016
		5,9	84.33	80	674.80	141,087
		9,16	18.72	11	169.04	221,144
		9,17	19.03	74	169.04	212,741
		13,22	6.04	0	75.77	103,080
		13,25	6.46	33	78.88	88,931
		15,20 ▲	3.92	0	53.08	61,922
		17,28	3.12	2	53.08	34,444
		17,31	3.32	5	62.87	31,029
		20,26	2.53	0	47.72	25,504
		24,32	2.22	0	35.24	20,426