Economic Genome Assembly from Low Coverage Illumina and Nanopore Data

Thomas Gatter
Bioinformatics Group, Department of Computer Science, University of Leipzig, Germany
Interdisciplinary Center of Bioinformatics, University of Leipzig, Germany
thomas@bioinf.uni-leipzig.de

Sarah von Löhneysen
Bioinformatics Group, Department of Computer Science, University of Leipzig, Germany
Interdisciplinary Center of Bioinformatics, University of Leipzig, Germany
lsarah@bioinf.uni-leipzig.de

Polina Drozdova
Institute of Biology, Irkutsk State University, Russia
drozdovapb@gmail.com

Tom Hartmann
Bioinformatics Group, Department of Computer Science, University of Leipzig, Germany
Interdisciplinary Center of Bioinformatics, University of Leipzig, Germany
tom@bioinf.uni-leipzig.de

Peter F. Stadler
Bioinformatics Group, Department of Computer Science, University of Leipzig, Germany
Interdisciplinary Center of Bioinformatics, University of Leipzig, Germany
Max-Planck-Institute for Mathematics in the Sciences, Leipzig, Germany
Institut for Theoretical Chemistry, University of Vienna, Austria
Facultad de Ciencias, Universidad National de Colombia, Bogotá, Colombia
Santa Fe Institute, NM, USA
studla@bioinf.uni-leipzig.de

Abstract

Ongoing developments in genome sequencing have caused a fundamental paradigm shift in the field in recent years. With ever lower sequencing costs, projects are no longer limited by available raw data, but rather by computational demands. The high complexity of eukaryotic genomes in concordance with increasing data sizes creates unique demands on methods to assemble full genomes. We describe a new approach to assemble genomes from a combination of low-coverage short and long reads. `LazyB` starts from a bipartite overlap graph between long reads and restrictively filtered short-read unitigs, which are then reduced to a long-read overlap graph G. Instead of the more conventional approach of removing tips, bubbles, and other local features, `LazyB` stepwisely extracts subgraphs whose global properties approach a disjoint union of paths. First, a consistently oriented subgraph is extracted, which in a second step is reduced to a directed acyclic graph. In the next step, properties of proper interval graphs are used to extract contigs as maximum weight paths. These are translated into genomic sequences only in the final step. A prototype implementation of `LazyB`, entirely written in python, not only yields significantly more accurate assemblies of the yeast and fruit fly genomes compared to state-of-the-art pipelines but also requires much less computational effort. Our findings demonstrate a new low-cost method that enables the assembly of even large genomes with low computational effort.

2012 ACM Subject Classification Theory of computation → Discrete optimization; Applied computing → Computational genomics

Keywords and phrases Nanopore sequencing, Illumina sequencing, genome assembly, spanning tree, unitigs, anchors

Digital Object Identifier 10.4230/LIPIcs.WABI.2020.10

Supplementary Material The `LazyB` prototype is available at `https://github.com/TGatter/LazyB`.

Funding *Thomas Gatter*: DFG STA 850/19-2 within SPP 1738; German Federal Ministery of Education and Research 031A538A, de.NBI-RBC.
Polina Drozdova: RSF / Helmholtz Association 18-44-06201; German Academic Exchange Service (DAAD).
Tom Hartmann: German Federal Ministry of Education and Research (BMBF 031L0164C).

1 Introduction

The assembly of genomic sequences from high throughput sequencing data has turned out to be a difficult computational problem in practice. Recent approaches combine cheap short-read data (typically using Illumina technology) with long reads produced by PacBio or Nanopore technologies. Although the short-read data are highly accurate and comparably cheap to produce, they are insufficient even at (very) high coverage due to repetitive elements. Long-read data, on the other hand, are comparably expensive and have much higher error rates. HiFi PacBio reads derived from repeat sequencing of circularized elements rival short read accuracy but at vastly increased costs.

Several assembly techniques have been developed recently for *de novo* assembly of large genomes from high-coverage (50× or greater) PacBio or Nanopore reads. Recent state-of-the-art methods employ a hybrid assembly strategy using Illumina reads to correct errors in the longer PacBio reads prior to assembly. For instance, the 32 Gb axolotl genome was produced in this manner [26].

Traditional assembly strategies can be classified into two general categories [21]. The Overlap-layout-consensus (OLC) assembly model attempts to find all pairwise matches between reads, using sequence similarity as a metric for overlaps. A general layout is constructed and post-processed in various ways. Most notably, overlaps can be transformed into assembly graphs such as string graphs. This method is flexible to read length and can be adapted to the diverse error models of different sequencing technologies. However, finding all overlaps is very expensive, especially for increasing read sizes.

In de Bruijn graph based strategies, reads are deconstructed to fixed length k-mers, representing nodes with edges between them for each $k-1$ overlap. Ideally, a de Bruijn graph represents exactly one Eulerian path per chromosome, although this property is generally violated in practice even by light sequencing errors. With the help of specialized hashing strategies k-mers can be efficiently stored and constructed. Thus, de Bruijn graphs require much less memory than OLC strategies. An overall speed up can be attributed to the absence of an all-*vs*-all comparison step. However, as k has to be chosen smaller than read size, contiguity information is lost. With increasing error rates in reads, de Bruijn graphs tend to become less useful, as k-mers become also less accurate.

Long read only and hybrid assembly strategies also largely align to these two categories, although some more unique methods have emerged over the years. `Canu` [18] and `Falcon` [5] implement classic OLC, albeit both error-correct long reads before creating a string graph. MinHash filters can significantly reduce the costs of comparisons, but overall complexity remains high. `Wtdbg2` [28] also follows OLC, but utilizes de Bruijn like graphs based on sparse k-mer mapping for comparison. It avoids all-*vs*-all mapping by matching reads that share k-mers under the assumption that even under high error rates correct pairs share more k-mer than those with spurious matches. `Shasta` [29] implements a full de Bruijn graph strategy by transforming k-mers into a run-length encoding that is more robust to sequencing errors in long reads. Newer versions of `Canu` also implement a similar encoding [27].

Classic de Bruijn methods have been adapted to combine both long and short reads into a hybrid assembly. Long reads can serve as "bridging elements" in the same way as mate pairs to resolve paths in (short read) assembly graphs [2, 33].

Under the assumption that short-read assemblies are cheap and reliable, various workflows have been proposed to integrate both kinds of data also for OLC like approaches. As a general goal, these programs aim to avoid the costly all-*vs*-all comparison to create the assembly graph by various heuristics. `MaSuRCA` [36] attempts to join both long and short reads into longer super-reads by chaining unique k-mers, as such creating fewer reads to compare for overlap. `WENGAN` [7] first creates full short-read contigs that are then scaffolded by synthetic mate pairs generated out of the long reads. `Flye` [17], even more uniquely, assembles intentionally erroneous contigs that are concatenated to a common sequence. Self-mapping then reveals repeats that can be resolved much like in a traditional assembly graph. In `HASLR` [11] an assembly graph like structure is defined combing both short and long reads. Short reads are assembled into contigs that, after k-mer filtering to remove repeats, are aligned to long reads. In the resulting backbone graph, short-read contigs serve as nodes that are connected by an edge if they map onto the same long read. While different to e.g. string graphs, standard tip and bubble removal algorithms are applied to remove noise. Contigs are extracted as paths. `TULIP` [13] implements a very similar strategy, however, does not assemble short reads into full contigs. Instead, the gaps between mate-pairs are closed if possible with sufficiently rare k-mers, resulting in relative short but unique seeds that serve in the same capacity. In both cases, consensus construction of the resulting sequence is trivial. Edges define fixed regions on groups of long reads that can be locally aligned for each edge along a path.

`DBG2OLC` [35] is methodologically most closely related to `LazyB`, however, both approaches differ in various key features (which becomes obvious over the course of this paper). `DBG2OLC` assembles short reads to full contigs with the advise to avoid repeat resolving techniques such as gap closing or scaffolding as they introduce too many errors. Contigs are then aligned against long reads. Each long read implies a neighborhood of contigs. Mappings are corrected prior to graph construction via consistency checks over all neighborhoods for each contig, i.e., contigs are required to map in the same order on all long reads. This technique can help to remove both spuriously matched contigs and chimeric long reads, but requires adequate coverage to allow for effective voting. Notably, here, long reads serve as nodes, with edges representing contigs mapping to both. Nodes that map a subset of contigs of another node are removed as they are redundant. The resulting graph can be error corrected by classic tip and bubble removal, after which paths are extracted as contigs, following the edge with the best overlap at each step.

`LazyB` implements an alternative approach to assembling genomes from a combination of long-read and short-read data. We avoid the expensive direct all-*vs*-all comparison of the error-prone long-read data, the difficult mapping of individual short reads against the long reads, and the conventional techniques to error-correct de Bruijn or string graphs. As we shall see, this is not only possible but also adds the benefit of producing rather good assemblies with surprisingly low requirements on the coverage of both short and long reads. Our methods lends itself in particular to the exploratory assembly of large numbers of species.

2 Strategy

Instead of a "total data" approach, we identify "anchors" that are nearly guaranteed to be correct and use an, overall, greedy-like workflow to obtain very large long-read contigs. To this end, the initial overlap graph is oriented and then edited in several steps to graph classes approaching the desired union of paths. The strategy of `LazyB` is outlined in Fig. 1.

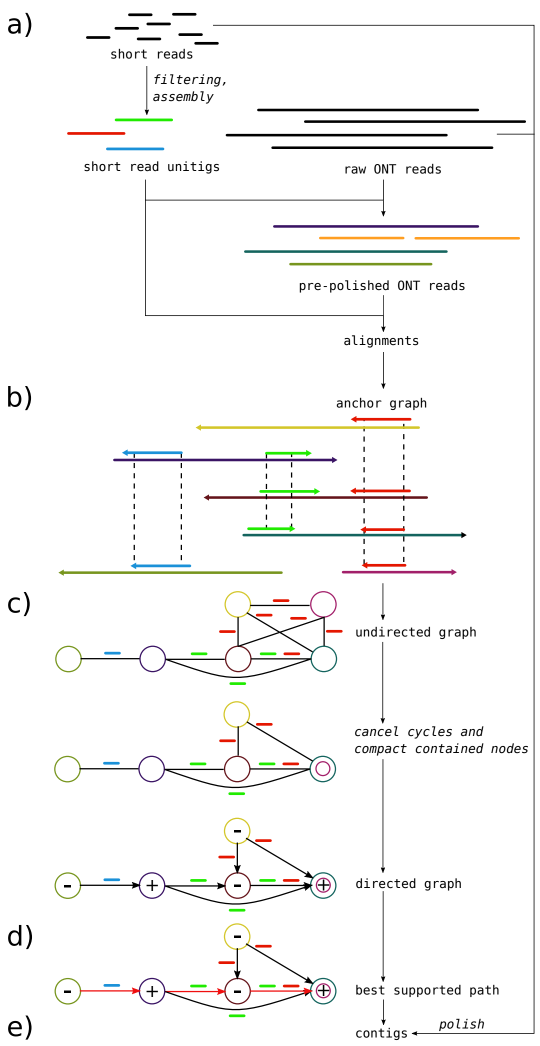

Figure 1 Overview of the `LazyB` assembly pipeline. a) Short Illumina reads are filtered to represent only near unique k-mers and subsequently assembled into unambiguous unitigs. Long Nanopore reads (ONT) can be optionally scrubbed to include only regions consistent to at least one other read. For larger data sets scrubbing can be handled on subsets efficiently. Mapping unitigs against Nanopore reads yields unique "anchors" between them b). An undirected graph c) is created by adding Nanopore reads as nodes and edges between all pairs of reads sharing an "anchor". Each edge is assigned a *relative orientation*, depending on whether the "anchor" maps in the same direction on both Nanopore reads. Cycles with a contradiction in orientation have to be removed before choosing a node at random and directing the graph based on its orientation. As Nanopore reads that are fully contained within another do not yield additional data, they can be collapsed. Contigs are extracted as maximally supported paths for each connected component d). Support in this context is defined by the number of consistent overlaps transitive to each edge. Final contigs e) can be optionally polished using established tools.

The key idea to obtain the overlap graph is to start from a collection $\mathcal{S} := \{s_i\}$ of pre-assembled, high-quality sequences that are unique in the genome. These serve as "anchors" to determine overlaps among the long reads $\mathcal{R} := \{r_j\}$. In practice, $\mathcal{S}$ can be obtained by assembling Illumina data with fairly low coverage to the level of unitigs only. The total genomic coverage of $\mathcal{S}$ only needs to be large enough to provide anchors between overlapping long reads, and it is rigorously filtered to be devoid of repetitive and highly similar sequences.

Mapping a short read $s \in \mathcal{S}$ against the set $\mathcal{R}$ of long reads implies (candidate) overlaps $r_1 - r_2$ between two long reads (as well as their relative orientation) whenever an s maps to both r_1 and r_2. Thus we obtain a directed overlap graph G of the long reads without an all-*vs*-all comparison of the long reads.

A series of linear-time filtering and reduction algorithms then prunes first the underlying undirected overlap graph and then the directed version of the reduced graph. Its connected components are reduced to near-optimal directed acyclic graphs (DAGs) from which contigs are extracted as best-supported paths. In the following sections we describe the individual steps in detail. In comparison to `DBG2OLC` we avoid global corrections of short-read mappings, but instead rely on the accuracy of assembled unitigs and a series of local corrections. For this, we utilize previously unreported properties of the class of alignment graphs used by both tools. This allows `LazyB` to operate reliably even on very low coverage. Variations of the dataset dependent assembly options have little impact on the outcome. In contrast of complicated setup of options required for tools such as `DBG2OLC`, `LazyB` comes with robust defaults.

3 Theory and Methods

3.1 Preprocessing

A known complication of both PacBio and Nanopore technologies are chimeric reads formed by the artificial joining of disconnected parts of the genome [23] that may cause mis-assemblies [34]. Current methods dealing with this issue heavily rely on raw coverage [22] and hence are of little use for our goal of a low-coverage assembler. In addition, start- and end-regions of reads are known to be particularly error-prone. We pre-filter low quality regions, but only consider otherwise problematic reads later at the level of the overlap graph.

Short-read (Illumina) data are preprocessed by adapter clipping and trimming. A set $\mathcal{S}$ of high quality fragments is obtained from a restricted assembly of the short-read data. The conventional use case of assembly pipelines aims to find a minimal set of contigs in trade-off to both correctness and completeness. For our purposes, however, completeness is of little importance and fragmented contigs are not detrimental to our workflow, as long as their lengths stay above a statistical threshold. Instead, correctness and uniqueness are crucial. We therefore employ three filtering steps: (1) Using a k-mer profile, we remove all k-mers that are much more abundant than the expected coverage since these are likely part of repetitive sequences. This process can be fully automated (see Appendix B).(2) In order to avoid ambiguities only branch-free paths are extracted from the assembly graph. Moreover, a minimal path length is required for secure anchors. The de Bruijn based assembler `ABySS` [30] allows to assemble up to unitig stage, implementing this goal. Since repeats in general lead to branch-points in the de Bruijn graph, repetitive sequences are strongly depleted in unitigs. While in theory, every such assembly requires a fine tuned k-mer size, a well known factor to be influential on assembly quality, we found overall results to be mostly invariant of this parameter. To test this, we systematically varied the k-mer-size for `ABySS`. Nevertheless, we found little to no effect on the results of `LazyB` (Fig. 2). As assembly stops at unitigs, error rates and genome coverage stay within a narrow range as long as the unitigs are long enough. (3) Finally, the set $\mathcal{R}$ of long reads is mapped against the unitig set. At present we use `minimap2` [20] for this purpose. Regions or whole unitigs significantly exceeding the expected coverage are removed from $\mathcal{S}$ because they most likely are repetitive or at least belong to families of very similar sequences such as multi-gene families. Please note that all repetitive elements connected to a unique region within a single long read may still be correctly assembled (see Appendix C).

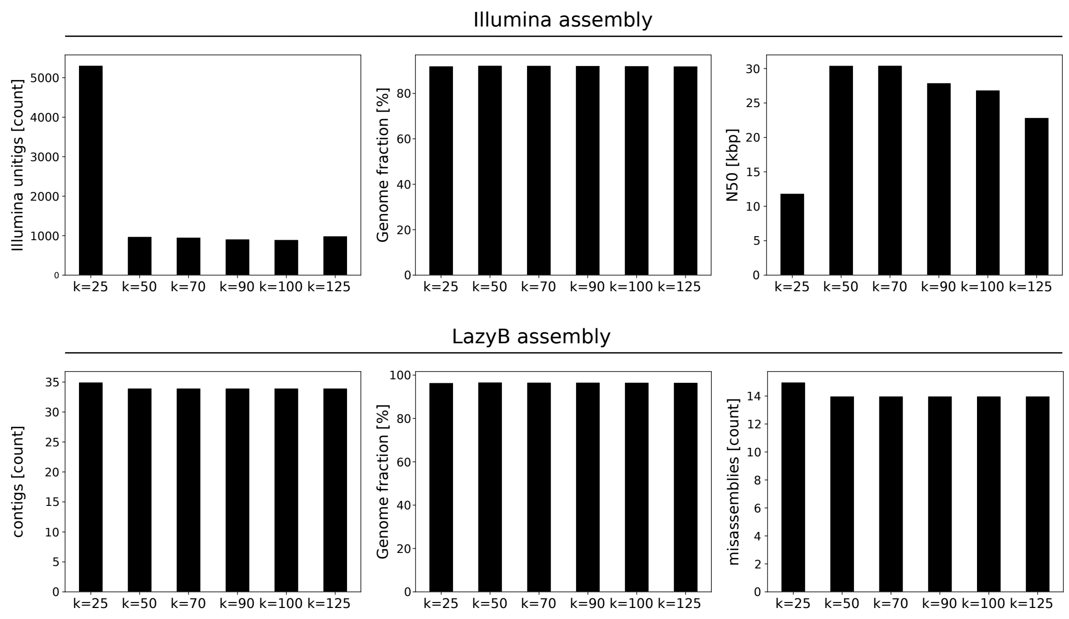

Figure 2 Assembly statistics as a function of the k-mer size used to construct unitigs from the short-read data for yeast. Top: Illumina unitigs (left: number of unitigs; middle: fraction of the reference genome covered; right: N50 values); bottom: final `LazyB` assembly at ~11× long reads (left: number of unitigs; middle: fraction of the reference genome covered; right: number of mis-assemblies).

3.2 Overlap Graph for Long Reads

As a result we obtain a set of *significant matches* $\mathcal{V} := \{(s,r) \in \mathcal{S} \times \mathcal{R} \mid \delta(s,r) \geq \delta_*\}$ whose matching score $\delta(s,r)$ exceeds a user-defined threshold δ_*. The *long-read overlap graph* G has the vertex set $\mathcal{R}$. Conceptually, two long reads overlap, i.e., there should be an undirected edge $r_1r_2 \in E(G)$ if and only if there is an $s \in \mathcal{S}$ such that $(s,r_1) \in \mathcal{V}$ and $(s,r_2) \in \mathcal{V}$. In practice, however, we employ a more restrictive procedure:

For distinct long reads $r_1, r_2 \in \mathcal{R}$ with $(s,r_1),(s,r_2) \in \mathcal{V}$ the sequence intervals on s that match intervals on r_1 and r_2 are denoted with $[i,j]$ and $[k,l]$, respectively. The intersection $[i,j] \cap [k,l]$ is the interval $[\max\{i,k\}, \min\{j,l\}]$ if $k \leq j$ and the empty interval otherwise. Note that if $[i,j] \cap [k,l]$ is not empty, then it corresponds to a direct match of r_1 and r_2. The expected bit score for the overlap is estimated as

$$\omega(s,r_1,r_2) := \begin{cases} 0 & \text{if } [i,j] \cap [k,l] = \varnothing; \\ \frac{1}{2}(\min\{j,l\} - \max\{i,k\} + 1)\left(\frac{\delta(s,r_1)}{(j-i+1)} + \frac{\delta(s,r_2)}{(l-k+1)}\right) & \text{otherwise.} \end{cases} \tag{1}$$

For a given edge $r_1r_2 \in E(G)$ there may be multiple significant matches, mediated by a set of unitigs $\mathcal{S}_{r_1r_2} := \{s \in \mathcal{S} \mid (s,r_1),(s,r_2) \in \mathcal{V}\}$. In ideal data they are all consistent with respect to orientation and co-linear location. In real data, however, this may not be the case.

For each significant match $(s,r) \in \mathcal{V}$ we define the *relative orientation* $\theta(s,r) \in \{+1,-1\}$ of the reading directions of the short-read scaffold s relative to the long read r. The relative reading direction of the long reads (as suggested by s) is thus $\theta_s(r_1,r_2) = \theta(s,r_1) \cdot \theta(s,r_2)$.

The position of a significant match (s,r) defined on the unitig s on interval $[i,j]$ corresponds to an interval $[i',j']$ on the long read r that is determined by the alignment of s to r. Due to the large number of randomly distributed InDels in the Nanopore data,

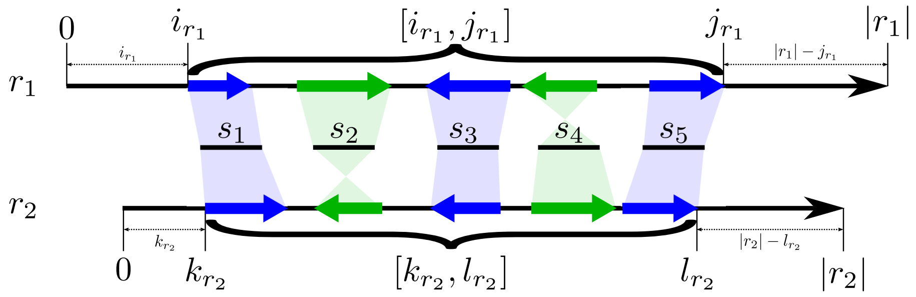

Figure 3 Construction of the overlap of two long reads r_1 and r_2 (long black arrows) from all unitigs $\mathcal{S}_{r_1r_2} := \{s_1, ..., s_5\}$ (short black bars) that match to both r_1 and r_2. A significant match (s, r) of $s \in \mathcal{S}_{r_1r_2}$ on $r \in \{r_1, r_2\}$ is illustrated by blue and green thick arrows on r. The relative orientation of (s, r) is indicated by the direction of its arrow, that is, $\theta(s, r) = +1$ (resp. $\theta(s, r) = -1$) if its arrow points to the right (resp. left). The subsets $\mathcal{S}^1_{r_1r_2} := \{s_1, s_3, s_5\}$ (unitigs with blue significant matches) and $\mathcal{S}^2_{r_1r_2} := \{s_2, s_4\}$ (unitigs with green significant matches) of $\mathcal{S}_{r_1r_2}$ are both inclusion-maximal and consists of pairwise consistent unitigs. The set $\mathcal{S}^1_{r_1r_2}$ maximizes $\Omega(r_1, r_2)$ and thus determines the overlap. It implies $\theta(r_1, r_2) = +1$. Moreover, i_{r_1} (resp. j_{r_1}) is the minimal (resp. maximal) coordinate of significant matches of unitigs from $\mathcal{S}^1_{r_1r_2}$ on r_1. The corresponding coordinates on r_2 are k_{r_2} and l_{r_2}, respectively. The spanning intervals $[i_{r_1}, j_{r_1}]$ and $[k_{r_2}, l_{r_2}]$ define the overlap of r_1 and r_2. In this example we have $i_{r_1} > k_{r_2}$ and $|r_1| - j_{r_1} > |r_2| - l_{r_2}$, implying that r_2 extends r_1 neither to the left or right and thus, edge r_1r_2 is contracted in G.

the usual dynamic programming alignment strategies fail to produce accurate alignments. This is also the case for `minimap2` [20], our preliminary choice, as it only chains short, high quality matches into larger intervals. Although more accurate alignments would of course improve the local error rate of the final assembled sequence, we expect very little impact on the overall assembly that is not effected in large by small local errors. We therefore record only the matching intervals and use a coordinate transformation τ_r that estimates the position $\tau_r(h) \in [i', j']$ for some $h \in [i, j]$ by linear interpolation:

$$\tau_r(h) := \begin{cases} j' - (j - h)\frac{j'-i'+1}{j-i+1} & \text{if } j - h \le h - i; \\ i' + (h - i)\frac{j'-i'+1}{j-i+1} & \text{if } j - h > h - i. \end{cases} \tag{2}$$

The values of $\tau_r(h)$ are rounded to integers and used to determine intersections of matches. We write $[i, j]_r := [\tau_r(i), \tau_r(j)]$ for the interval on r corresponding to an interval $[i, j]$ of s.

▶ **Definition 1.** *Two unitigs s, s' in $\mathcal{S}_{r_1r_2}$ are* consistent *if (i) $\theta_s(r_1, r_2) = \theta_{s'}(r_1, r_2)$, (ii) the relative order of $[i^s, j^s]_{r_1}$, $[k^{s'}, l^{s'}]_{r_1}$ on r_1 and $[i^s, j^s]_{r_2}$, $[k^{s'}, l^{s'}]_{r_2}$ on r_2 is the same.*

For distinct long reads $r_1, r_2 \in \mathcal{R}$, Definition 1 enables us to determine $m \ge 1$ subsets $\mathcal{S}^1_{r_1r_2}, ..., \mathcal{S}^m_{r_1r_2}$ of $\mathcal{S}_{r_1r_2}$ such that each is maximal with respect to inclusion and contains only unitigs that are pairwise consistent with respect to r_1 and r_2. In addition, we may require that the difference between the distances of consecutive corresponding intervals on r_1 and r_2, respectively, is sufficiently similar. Computing the set $\mathcal{S} \in \{\mathcal{S}^1_{r_1r_2}, ..., \mathcal{S}^m_{r_1r_2}\}$ that maximizes the total bit score $\sum_{s\in\mathcal{S}} \omega(s, r_1, r_2)$ amounts to a chaining problem that can be solved in quadratic time by dynamic programming [25]. An edge r_1r_2 is inserted into G if the optimal total bit score $\Omega(r_1, r_2) := \sum_{s\in\mathcal{S}} \omega(s, r_1, r_2)$ exceeds a user-defined threshold. The *signature* $\theta(r_1, r_2)$ of the edge $r_1r_2 \in E(G)$ is the common value $\theta_s(r_1, r_2)$ for all $s \in \mathcal{S}$.

For each edge $r_1r_2 \in E(G)$ we determine $s, s' \in \mathcal{S}$ such that $\tau_{r_1}(i^s)$ is the minimal and $\tau_{r_1}(j^{s'})$ is the maximal coordinate of the matching intervals on r_1. Hence, the interval $[i^s, j^{s'}]_{r_1}$ spans all matching intervals on r_1. The corresponding pair of coordinates,

$\tau_{r_2}(k^s)$ and $\tau_{r_2}(l^{s'})$, spans the matching intervals on r_2. In particular, the interval $[k^s, l^{s'}]_{r_2}$ (resp. $[l^{s'}, k^s]_{r_2}$) spans both matching intervals on r_2 if $\theta(r_1, r_2) = 1$ (resp. $\theta(r_1, r_2) = -1$). For the sake of a clear notation, let $[i_{r_1}, j_{r_1}] := [i^s, j^{s'}]_{r_1}$ and $[k_{r_2}, l_{r_2}]$ be the "spanning" interval on r_2, i.e., either $[k_{r_2}, l_{r_2}] := [k^s, l^{s'}]_{r_2}$ or $[k_{r_2}, l_{r_2}] := [l^{s'}, k^s]_{r_2}$. Intervals $[i_{r_1}, j_{r_1}]$ and $[k_{r_2}, l_{r_2}]$ specify the known overlapping regions between r_1 and r_2, see also Fig. 3 for an illustration. If $\theta(r_1, r_2) = +1$ then r_1 *extends* r_2 *to the left* if $i_{r_1} > k_{r_2}$ and *to the right* if $|r_1| - j_{r_1} > |r_2| - l_{r_2}$. For $\theta(r_1, r_2) = -1$ the corresponding conditions are $i_{r_1} > |r_2| - k_{r_2}$ and $|r_1| - j_{r_1} > l_{r_2}$, respectively. If r_1 does not extend r_2 to either side then r_1 is completely contained in r_2 and does not contribute to the assembly. If r_1 extends r_2 on both sides, r_2 is fully contained, respectively. In both cases we contract the edge between r_1 and r_2 in G. Otherwise, if r_1 extends r_2 to the left and r_2 extends r_1 to the right we record $r_1 \to r_2$ and accordingly, if r_2 extends r_1 to the left and r_1 extends r_2 to the right we note $r_1 \leftarrow r_2$.

The result of this construction is a long-read-overlap graph G whose vertices are the non-redundant long reads and whose edges $r_1 r_2$ record (1) the relative orientation $\theta(r_1, r_2)$, (2) the bit score $\Omega(r_1, r_2)$, (3) the local direction of extension, and (4) the overlapping interval.

3.3 Consistent Orientation of Long Reads

For perfect data it is possible to consistently determine the reading direction of each read relative to the genome from which it derives. This is not necessarily the case in real-life data. The relative orientation of two reads is implicitly determined by the relative orientation of overlapping reads, i.e., by the signature $\theta(r_1, r_2)$ of the edge $r_1 r_2 \in E(G)$. To formalize this idea we consider a subset $D \subseteq E(G)$ and define the *orientation* of D as $\theta(D) := \prod_{r_1 r_2 \in D} \theta(r_1, r_2)$. For a disjoint union of two edge sets D and D' we therefore have $\theta(D \,\dot\cup\, D') = \theta(D')\theta(D)$ and, more generally, their symmetric different $D \oplus D'$ satisfies $\theta(D \oplus D') = \theta(D)\theta(D')$ since the edges in $D \cap D'$ appear twice in $\theta(D)\theta(D')$ and thus contribute a factor $(\pm 1)^2 = 1$.

▶ **Definition 2.** *Two vertices $r_1, r_2 \in V(G)$ are* orientable *if $\theta(P) = \theta(P')$ holds for any two paths P and P' connecting r_1 and r_2 in G. We say that G is* orientable *if all pairs of vertices in G are orientable.*

▶ **Lemma 3.** *G is orientable if and only if every cycle C in G satisfies $\theta(C) = 1$.*

Proof. Let r, r' be two vertices of G and write $\mathcal{C}(r, r')$ for the set of all cycles that contain r and r'. If $r = r'$ or $\mathcal{C}(r, r') = \emptyset$, then r and r' are orientable by definition. Now assume $r \neq r'$, $\mathcal{C}(r, r') \neq \emptyset$, and consider a cycle $C \in \mathcal{C}(r, r')$. Clearly, C can be split into two edge-disjoint path C_1 and C_2 both of which connect r and r'. If r and r' are orientable, then $\theta(C_1) = \theta(C_2)$ and thus $\theta(C) = 1$. If r and r' are not orientable, then there is a pair of path P_1 and P_2 connecting r and r' such that $\theta(P_1) = -\theta(P_2)$. Since $P_1 \oplus P_2 = \dot\bigcup_{i=1}^{k} C_i$ is an edge-disjoint union of cycles C_i we have $-1 = \theta(P_1)\theta(P_2) = \prod_{i=1}^{k} \theta(C_i)$ and thus there is least one cycle C_i with $\theta(C_i) = -1$ in G. ◀

The practical importance of Lemma 3 is the implication that only a small set of cycles needs to be considered since every graph G with c connected components has a cycle basis comprising $|E| - |V| - c$ cycles. Particular cycles bases, known as *Kirchhoff bases*, are obtained from a spanning tree T of G as the set $\mathcal{B}$ of cycles C_e consisting of the edge $e \in E \setminus T$ and the unique path in T connecting the endpoints of e. Every cycle C of G can then be written as $C = \bigoplus_{e \in C \setminus T} C_e$, see e.g. [15].

▶ **Theorem 4.** *Let $\mathcal{B}$ be a cycle basis of G. The graph G is orientable if and only if $\theta(C) = 1$ for all $C \in \mathcal{B}$.*

Proof. The theorem follows from Lemma 3 and the fact that every cycle C in G can be written as an $\oplus$-sum of basis cycles, i.e., $\theta(C) = 1$ for every cycle in C if and only if $\theta(C') = 1$ for every basis cycle $C' \in \mathcal{B}$. ◀

Theorem 4 suggests the following, conservative heuristic to extract an orientable subgraph from G:

(1) Construct a maximum weight spanning tree T_G of G by using the Ω-scores as edge weights. Tree T_G can easily be obtained using, e.g., Kruskal's algorithm [19].

(2) Construct a Kirchhoff cycle basis $\mathcal{B}$ from T_G.

(3) For every cycle $C \in \mathcal{B}$, check whether $\theta(C) = -1$. If so, find the Ω-minimum weighted edge $\hat{e} \in C$ and remove it from $E(G)$ and (possibly) from T_G if $\hat{e} \in E(T_G)$. Observe that if $\hat{e} \notin E(T_G)$, then T_G stays unchanged. If $\hat{e} \in E(T_G)$, then the removal of $\hat{e}$ splits T_G into two connected components. We restrict G to the connected components of T_G.

This procedure yields a not necessarily connected subgraph G' and a spanning forest $T_G \cap E(G')$ for G'.

▶ **Lemma 5.** *Let G be an undirected graph and let G'' be a connected component of the residual graph G' produced by the heuristic steps (1)-(3). Then (i) G'' is orientable and (ii) $T_G \cap E(G'')$ is an Ω-maximal spanning tree of G''.*

Proof. Removal of an edge e from a spanning tree T of G partitions T into two components with vertex sets V_1 and V_2. Let $G_1 = G[V_1]$ and $G_2 = G[V_2]$ be the corresponding induced subgraphs of G. The cut in G induced by e is $E(G) \setminus (E(G_1) \cup E(G_2))$. Clearly $T_1 = T \cap E(G_1)$ and $T_2 = T \cap E(G_2)$ are spanning trees of G_1 and G_2, respectively. The restrictions $\mathcal{B}_1$ and $\mathcal{B}_2$ of the Kirchhoff basis $\mathcal{B}$ to cycles with non-tree edges $e \in E(G_1)$ or $e \in E(G_2)$ form a Kirchhoff basis of G_1 and G_2, respectively. Now consider G_1' is obtained from G_1 by removing a set of non-tree edges, then $\mathcal{B}_1'$ obtained from $\mathcal{B}_1$ by removing the cycles with these non-tree edges is a cycle basis of G_1' and T_1 is still a spanning tree of G_1. The Ω-weights of $T_1 = T \cap E(G_1)$ and $T_2 = T \cap E(G_2)$ must be maximal, since otherwise a heavier spanning tree T of G could be constructed by replacing T_1 or T_2 by a heavier spanning tree of G_1 or G_2. The arguments obviously extend to splitting G_i by cutting at an edge of T_i. Since the heuristic removes all non-tree edges e with $\theta(C_e) = -1$, Theorem 4 implies that each component G'' is orientable. When removing $e \in T$, the corresponding cut edges are removed, the discussion above applies and thus $T \cap E(G'')$ is Ω-maximal. ◀

From here on, we denote a connected component of G' again by G and write T_G for its maximum Ω-weight spanning tree, which by Lemma 5 is just the restriction of the initial spanning tree to G. We continue by defining an orientation φ for the long reads. To this end, we pick an arbitrary $r_* \in V(G)$ and set $\varphi(r_*) := +1$. For each $r \in V(G)$ we set $\varphi(r) := \prod_{e \in \text{path}(r_*, r)} \theta(e)$, where path$(r_*, r)$ is the unique path connecting r_* and r in T_G. We can now define an equivalent graph $\tilde{G}$ with the same vertices and edges as G and orientations $\tilde{\theta}(e) = +1$ for $e \in T_G$ and $\tilde{\theta}(e) := \varphi(x_e)\varphi(y_e)$ for all non-tree edges $e = x_e y_e \notin T_G$. We note that the vertex orientations can be computed in $\mathcal{O}(|\mathcal{R}|)$ time along T_G. Since $\theta(C_e) = \tilde{\theta}(e)$ for every $C_e \in \mathcal{B}$, we can identify the non-orientable cycles in linear time.

3.4 Reduction to a DAG

We next make use of the direction of extension of long read r_1 and r_2 defined by the mutual overhangs in the case that $r_1 r_2$ is an edge in G. We write $\vec{G}$ for the directed version of a connected component G of the residual graph G' constructed above. For each edge $r_1 r_2 \in E(G)$ we create the corresponding edge $e \in E(\vec{G})$ as

$$e := \begin{cases} r_1 r_2 & \text{if } \varphi(r_1) = +1 \text{ and } r_1 \to r_2 \text{ or } \varphi(r_1) = -1 \text{ and } r_1 \leftarrow r_2; \\ r_2 r_1 & \text{if } \varphi(r_1) = +1 \text{ and } r_1 \leftarrow r_2 \text{ or } \varphi(r_1) = -1 \text{ and } r_1 \to r_2. \end{cases} \tag{3}$$

In perfect data, $\vec{G}$ is a directed interval graph. Recall that we have contracted edges corresponding to nested reads (i.e., intervals). Therefore, $\vec{G}$ is a proper interval graph or indifference graph. Thus there is an ordering $\prec$ of the vertices (long reads) that satisfies the *umbrella property* [12]: $r_1 \prec r_2 \prec r_3$ and $r_1 r_3 \in E(\vec{G})$ implies $r_1 r_2, r_2 r_3 \in E(\vec{G})$. A "normal interval representation" and a linear order $\prec$ of the reads, can be computed in $\mathcal{O}(|\mathcal{R}|)$ time [24]. Again, we cannot use these results directly due to the noise in the original overlap graph.

First we observe that $\vec{G}$ should be acyclic. Our processing so far, however, does not guarantee acyclicity since $\vec{G}$ still may contain some spurious edges due to unrecognized repetitive elements. The obvious remedy is to remove a (weight-)minimal set of directed edges. This FEEDBACK ARC SET problem, however, is NP-complete, see [3] for a recent overview. We therefore resort to a heuristic that makes use of our expectations on the structure of $\vec{G}$: In general we expect multiple overlaps of correctly placed reads, i.e., r is expected to have several incoming edges from its predecessors and several outgoing edges exclusively to a small set of succeeding reads. In contrast, we expect incorrect edges to appear largely in isolation. This suggest to adapt Khan's topological sorting algorithm [14]. In its original version, it identifies a source u, i.e., a vertex with in-degree 0, appends it to the list W of ordered vertex and then deletes all its out-edges. It stops with "fail" when no source can be found before the sorting is complete, i.e., W does not contain all vertices of the given graph, indicating that a cycle has been encountered. In our setting we need to identify the best approximation to create a new source in this case. Denote by $N_+(W)$ denotes the out-neighborhood of the already sorted set W. The set $K := (V \setminus W) \cap N_+(W)$ of not yet sorted out-neighbors of W are the candidates for the next source. For each $u \in K$ we distinguish incoming edges xu from $x \in W$, $x \in K$, and $x \in V \setminus (W \cup K)$ and consider two cases:

(1) There is a $u \in K$ without an in-edge xu from some other $x \in K$. Then we choose among these the vertex $\hat{u}$ with the largest total Ω-weight incoming from W because $\hat{u}$ then overlaps with most of the previously sorted reads.

(2) If for each $u \in K$ there is an in-edge xu from some other $x \in K$, then the candidate set K forms a strongly connected digraph. In this case we choose the candidate $\hat{u} \in K$ with the largest difference of Ω-weights incoming from W and K, i.e., $\hat{u} := \arg\max_{u \in K} \sum_{w \in W} \Omega(w,u) - \sum_{k \in K \setminus \{u\}} \Omega(k,u)$.

In either case we remove from $\vec{G}$ the edges incoming from $V \setminus W$ into $\hat{u}$ and proceed. If multiple sources are available we always pick the one with largest Ω-weight incoming from W. As a consequence, incomparable paths in $\vec{G}$ are sorted contiguously. The result of the modified Kahn algorithm is a directed acyclic graph $\overrightarrow{G}$.

3.5 Golden Paths

For perfect data, $\overrightarrow{G}$ (and already $\vec{G}$) has a single source and a single sink vertex, corresponding to the left-most and right-most long reads r' and r'', respectively. Furthermore, every directed path connecting r' and r'' is a *golden path*, that is, a sequence of overlapping intervals that covers the entire chromosome. Even more stringently, every read $r \neq r', r''$ has at least one predecessor and at least one successor in $\overrightarrow{G}$. The acyclic graph $\overrightarrow{G}$ therefore has a unique topological sorting, i.e., its vertices are totally ordered. As before, we cannot expect that $\overrightarrow{G}$ has these properties for real-life data.

Ploidy in eukaryotes may constitute a valid exception to this assumption, as differences in chromosomes ideally also cause diverging structures. However, given the high error rate of long reads, low sequence variation can only be differentiated in very high coverage scenarios; these explicitly are not targeted by `LazyB`. High accuracy short read assemblies originating from different alleles thus can be expected to match equally well to the same long reads given their low quality. Therefore, also ploidy variation will normally be merged to a single consensus. Accordingly, we did not detect any mayor duplication issues in the human, fly, or yeast.

A transitive reduction H° of some directed graph H is a subgraph of H with as few edges as possible such that two vertices x and y are connected by a directed path in H° if and only if they are connected by a directed path in H. It is well-known that each acyclic digraph has a unique transitive reduction [1, Thm. 1]. This property enables us to call an edge e of an acyclic digraph H *redundant* if $e \notin E(H^\circ)$.

Consider a proper interval graph H, an induced subgraph F of H, and recall that H is an acyclic digraph. Since H satisfies the umbrella property, every redundant edge $uw \in E(H)$ is part of some triangle. We also observe that F has a unique topological sorting and its *triangle reduction* $F^\triangle$, obtained by removing all edges $uw \in E(F)$ for which there is a vertex v with $uv, vw \in E(F)$, is a path. In fact, $F^\triangle$ is an induced path in the triangle reduction $H^\triangle$ of H.

This deduction suggests to identify maximal paths in the triangle reduction $\overrightarrow{G}^\triangle$ of the directed acycling graph $\overrightarrow{G}$ as contigs. Since the topological sorting along any such path is unique, it automatically identifies any redundant non-triangle edges along a path.

On imperfect data $\overrightarrow{G}^\triangle$ differs from a unique golden path by bubbles, tips, and crosslinks (see Appendix A). Tips and bubbles predominantly are caused by edges that are missing e.g. due to mapping noise between reads that belong to a shared contig region. Hence, any path through a bubble or superbubble yields essentially the same assembly of the affected region and thus can be chosen arbitrarily, whereas tips may prematurely end a contig. Node-disjoint alternative paths within a (super-)bubble start and end in the neighborhood of the original path. Tips either originate or end in neighborhood of the chosen path.

Crosslinks represent connections between two proper contigs by spurious overlaps, caused, e.g., by repetitive elements that have escaped filtering. As crosslinks can occur at any positions, a maximal path may not necessarily follow the correct connection and thus may introduce chimeras into the assembly. As a remedy we measure how well an edge e fits into a local region that forms an induced proper interval graph. Recall that the out-neighborhood of each vertex in a proper interval graph induces a transitive tournament. For real data, however, the subgraph $\overrightarrow{G}[N_+(r)]$ induced by the out-neighbors of r may in general violate this expectation. The problem of finding the maximum transitive tournament in an acyclic graph is NP-hard [8]. An approximation can be obtained, however, using the fact that a transitive tournament has a unique directed Hamiltonian path. Finding a longest path in a DAG only requires linear time. Thus candidates for transitive tournaments in $\overrightarrow{G}[N_+(r)]$ can be retrieved efficiently as the maximal path P_{rq} in $\overrightarrow{G}[N_+(r)]$ that connects r with an endpoint q, i.e., a vertex without an outgoing edge within $\overrightarrow{G}[N_+(r)]$. Clearly, it suffices to consider the maximum path problem in the much sparser DAG $\overrightarrow{G}^\triangle[N_+(r)]$. The induced subgraph

$\overrightarrow{G}^{\triangle}[P_{rq}]$ with the largest edge set $H_r := E(\overrightarrow{G}^{\triangle}[P_{rq}])$, i.e., $q := \arg\max_p |E(\overrightarrow{G}^{\triangle}[P_{rp}])|$, serves as approximation for the maximal transitive tournament and is used to define the *interval support* of an edge $e \in E(\overrightarrow{G})$ as

$$\nu(e) := \sum_{r \in V(\overrightarrow{G}): e \in H_r} (|H_r| - d(r,e) - 1)\,. \tag{4}$$

Here, $d(r,e)$ is the minimal number of edges in the unique path from r to e in the path formed by the edges in H_r. The interval support can be interpreted as the number of triangles that support e as lying within an induced proper interval graph. It suffices to compute $\nu(e)$ for $e \in E(\overrightarrow{G}^{\triangle})$. We observed empirically that determining the best path with respect to $\nu(e)$ (rather than weight Ω of the spanning tree edges) results in contigs with a better solution quality. Taken together, we arrive at the following heuristic to iteratively extract meaningful paths (see also Appendix D):

i) Find the longest path $\mathbf{p} = r_1, \dots, r_n$ in $\overrightarrow{G}^{\triangle}$ such that at every junction, we choose the incoming and outgoing edges e with maximal interval support $\nu(e)$.

ii) Add the path $\mathbf{p}$ to the contig set if it is at least two nodes long and neither the in-neighborhood $N_-(r_1)$ nor the out-neighborhood $N_+(r_n)$ are marked as previously visited in $\overrightarrow{G}$. Otherwise, we have found a tip if one of $N_-(r_1)$ or $N_+(r_n)$ was visited before and a bubble if both were visited. Such paths are assumed to have arisen from more complex crosslinks and can be added to the contig set if they exceed a user-defined minimum length.

iii) The path $\mathbf{p}$ is marked visited in $\overrightarrow{G}$ and all corresponding nodes and edges are deleted from $\overrightarrow{G}^{\triangle}$.

iv) The procedure terminates when $\overrightarrow{G}^{\triangle}$ is empty.

As the result, we obtain a set of paths, each defining a contig.

3.6 Consensus Sequence

The final step is the retrieval of a consensus sequence for each path $\mathbf{p}$. This step is more complicated than usual due to the nature of our initial mappings. While we enforce compatible sets of unitigs for each pair of long reads, a shared unitig between edges does not necessarily imply the same genomic coordinate. (i) Unitigs can be long enough that we gain triples $r_i, r_{i+1}, r_{i+2} \in V(\mathbf{p})$ such that an $s \in \mathcal{S}_{r_i r_{i+1}} \cap \mathcal{S}_{r_{i+1} r_{i+2}}$ exists but r_i and r_{i+2} share no interval on s. Such triples can occur chained. (ii) Unitigs of genomic repeats may remain in the data. Such unitigs may introduce pairwise distinct edges e_i, e_j, e_k that appear in this order, denoted by $e_i \prec e_j \prec e_k$, along the path $\mathbf{p}$ such that $s \in \mathcal{S}_{e_i}$ and $s \in \mathcal{S}_{e_k}$ but $s \notin \mathcal{S}_{e_j}$, therefore creating disconnected occurrences of s. (iii) Similarly, proximal repeats may cause inversions in the order of two unitigs $s, s' \in \mathcal{S}_{e_i} \cap \mathcal{S}_{e_k}$, w.l.o.g $e_i \prec e_k$. This scenario cannot appear on neighboring edges, as the shared node has a unique order of s and s'. Hence, either s or s' must be missing in an intermediary edge e_l due to the consistency constraints in the original graph, resulting in a situation as described in (ii). (iv) Finally, true matches of unitigs may be missing for some long reads due to alignment noise, which may also yield a situation as in (ii).

To address (i), we collect all instances of a unitig in the path independent of its context. We create an undirected auxiliary graph U_s with a vertex set$V(U_s) := \{e \in E(\mathbf{p}) \mid s \in \mathcal{S}_e\}$. We add edges for all edge-pairs that share an overlap in s. Any clique in this graph then represents a set of edges that share a common interval in s. We assign each edge a unique cluster index c_s^e, according to a minimal size clique decomposition. As finding a set of maximal cliques is NP-hard, we instead resort to a $\mathcal{O}(|V|/(\log|V|)^2)$ heuristic [4]. We address (ii-iv) with the help of a second index g_s^e, where $g_s^{e_i} \neq g_s^{e_k}$ for two edges e_i, e_k if and only if an edge e_j exists such that $e_i \prec e_j \prec e_j$ and $s \notin \mathcal{S}_{e_j}$.

Finally, we can now create a multigraph M consisting of vertex triples $\{(s, c_s^e, g_s^e) \mid s \in \mathcal{S}_e \text{ with } e \in E(\mathbf{p})\}$. We add edges $(s, c_s^e, g_s^e) \to (s', c_s'^e, g_s'^e)$ if and only if $s \prec s'$ on an edge e and no element s'' exists such that $s \prec s'' \prec s'$. The resulting graph is cycle free and thus uniquely defines the positions of all unitigs. Nodes represent the sequence of the common interval on the unitig s as attributed to the clique c_s^e. Edges represent the respective sequence of long reads between s and s', or a negative offset value if unitigs overlap. We take an arbitrary node in M and set its interval as the reference point. Positions of all other nodes are progressively built up following a topological order in this graph. If multiple edges exist between two nodes in this process a random but fixed edge is chosen to estimate the distance between nodes. As now all sequence features are embedded in the same coordinate system, an arbitrary projection of the sequence is set as the reference contig, retaining unitigs were possible due to their higher sequence quality. At the same time, we can map the features of each long read to their respective position in this newly constructed reference. This information can be directly fed into consensus based error correction systems such as `racon` [32].

4 Experimental Results

To demonstrate the feasibility of our assembly strategy we applied `LazyB` to publicly available datasets (see Appendix E) [9, 16, 31] for three well studied model organisms, baker's yeast (*S. cerevisiae*, genome size 12 Mb), fruit fly (*D. melanogaster*, genome size 140 Mb) and human (*H. sapiens*, genome size 3 Gb). The data were downsampled to approximately 5× and 10× nanopore coverage for long reads, respectively, and Illumina coverage sufficient for short-read anchors. We compare results to the most widespread competing assembler `Canu` [18], also highlighting the disadvantage of long read only strategies, `DBG2OLC`'s [35] implementing the most closely related concept, as well as the recent competitor `HASLR` [11] based on also a similar strategy. For comparison, we also provide the statistics for short-read only assemblies created with `ABySS` [30] on the same sets of reads used to create the "anchors" to show the advantage of hybrid assembly even at a low coverage of long reads. Quality was assessed via alignment to a reference genome by the `QUAST` tool [10]; see Table 1. `LazyB` produced consistently better results than `Canu`, increasing genomic coverage at a lower contig count. Due to our inclusion of accurate short-read unitigs, overall error counts are also significantly lower. Most notably, `Canu` was unable to properly operate at the 5× mark for both data sets. Only insignificant portions of yeast could be assembled, accounting for less than 15% of the genome. `Canu` completely failed for fruit fly, even after adapting settings to low coverage. Even at 5×, `LazyB` already significantly reduces the number of contigs compared to the respective short-read assemblies, while retaining a reasonably close percentage of genome coverage. At only 10× coverage for fruit fly, we were able to reduce the contig count 10-fold at better error rates. For human, `LazyB` manages at 39-fold decrease of the number contigs, albeit at a loss of greater 10% coverage. This difference appears to be a consequence of the high fragmentation of unitigs in the abundant repeat regions of the genome, rendering them too unreliable as anchors. Results are indeed in line with unitig coverage. While `HASLR` produced the fewest mis-assemblies, it creates significantly more and shorter contigs that cover a much smaller fraction of the genome. As a consequence it has the least favorable N50 values of all tools. For fruit fly at 10×, it results in four times as many contigs and

Table 1 Assessment of assembly qualities for LazyB, Canu, and short-read only assemblies for two model organisms. LazyB outperforms Canu in all categories, while significantly reducing contig counts compared to short-read only assemblies. While HASLR is more accurate, it covers significantly lower fractions of genomes at a higher contig count and drastically lower N50. While DBG2OL produces few contigs at a high N50 for higher coverage cases, it calls significantly more mis-assemblies. Mismatches and InDels are given per 100kb. Accordingly, errors in LazyB's unpolished output constitute < 1% except for human. Column descriptions: X coverage of sequencing data, **compl**eteness of the assembly. **#ctg** number of contigs, **#MA** number of mis-assemblies (breakpoints relative to the reference assembly) **M**is**M**atches and **InDels** relative to the reference genomes. N50 of **correctly assembled** contigs (minimal length of a correctly assembled contig needed to cover 50% of the genome, also named NGA50; omitted when $<$ 50% is correctly recalled).

Org.	X	Tool	compl.[%]	#ctg	#MA	MM	InDels	N50
yeast	~5×	LazyB	90.466	127	9	192.56	274.62	118843
		Canu	14.245	115	5	361.47	2039.15	-
		HASLR	64.158	111	1	14.87	34.86	60316
		DBG2OLC	45.645	53	20	2066.64	1655.92	-
	~11×	LazyB	97.632	33	15	193.73	300.20	505126
		Canu	92.615	66	15	107.00	1343.37	247477
		HASLR	92.480	57	1	7.89	33.91	251119
		DBG2OLC	97.689	38	25	55.06	1020.48	506907
	~80×	Abyss	95.247	283	0	9.13	1.90	90927
fruit fly	~5×	LazyB	71.624	1879	68	446.19	492.43	64415
		Canu	-	-	-	-	-	-
		HASLR	24.484	1407	10	31.07	58.96	-
		DBG2OLC	25.262	974	141	1862.85	969.26	-
	~10×	LazyB	80.111	596	99	433.37	486.28	454664
		Canu	49.262	1411	275	494.66	1691.11	-
		HASLR	67.059	2463	45	43.83	84.89	36979
		DBG2OLC	82.52	487	468	739.47	1536.32	498732
	~45×	Abyss	83.628	5811	123	6.20	8.31	67970
human	~10×	LazyB	67.108	13210	2915	1177.59	1112.84	168170
	~43×	Unitig	69.422	4146090	252	93.07	13.65	338
	~43×	Abyss	84.180	510315	2669	98.53	25.03	7963

covers 10% less of the genome, with a 12 times lower N50. While an improvement to Canu, it also struggles on datasets with low Nanopore coverage. DBG2OLC shows the greatest promise compared to our own method, but similarly fails to operate well on very low coverage datasets. For yeast at 5×, less then 50% the genome can be reconstructed. In fruit fly even less then 25% can be assembled at about 2 times the error rate of LazyB. At 10×, DBG2OLC reconstruct a similar proportion of the genome, albeit at high error rates. While it produces about 100 fewer contigs for fruit fly, this achievement is offset by over 350 (4.7 times more) mis-assemblies.

The resource footprint of LazyB is small enough to run on an off-the-shelf desktop machine or even a laptop. The total effort is, in fact, dominated by the computation of the initial unitig set from the short reads. We expect that an optimized re-implementation of LazyB will render its resource consumption negligible. Compared to the competing Canu assembler, the combination of ABySS and the python-prototype of LazyB is already more than a factor of 60 faster. In terms of memory, given precomputed unitigs LazyB also requires 3 – 18 times less RAM than Canu, see Table 4. Most notably, we were able to assemble the human genome within only 3 days, while Canu could not be run within our resource constraints. HASLR shows a similar distribution of running times between tasks, overall operating slightly faster. We could not process our human test set with HASLR. A human DBG2OLC assembly can be estimated to take several weeks without manual parallelization for a single set of parameters, with authors recommending several possible alternatives for optimization. We therefore include only the results for LazyB here, and leave a more detailed comparison of the performance for very complex genomes for a proper follow-up experiment.

5 Discussion and Outlook

We demonstrated here the feasibility of a new strategy for sequence assembly with low coverage long-read data. Already the non-optimized prototype LazyB, written entirely in python, not only provides a significant improvement of the assembly but also requires much less time and memory than state-of-the-art tools. This is achieved by avoiding both a correction of long reads and an all-*vs*-all comparison of the long reads. Instead, we use rigorously filtered short-read unitigs as anchors to sparsifying the complexity of full string-graphs construction. LazyB then uses a series of fast algorithms to consistently orient this sparse overlap graph, reduce it to a DAG, and sort it topologically, before extracting contigs as maximum weight paths. This workflow relies on enforcing properties of overlap graphs that have not been exploited in this manner in competing sequence assembly methods.

The prototype implementation leaves several avenues for improvements. We have not attempted here to *polish* the sequence but only to provide a common coordinate system defined on the long reads into which the short-reads unitigs are unambiguously embedded to yield high-quality parts of the LazyB-assembly. The remaining intervals are determined solely by long-read data with their high error rate. Multiple edges in the multigraph constructed in the assembly step correspond to the same genome sequence, hence the corresponding fragments of reads can be aligned. This is also true for alternative paths between two nodes. This defines a collection of alignments distributed over the contig, similar to the situation in common polishing strategies based on the mapping of (more) short-read data or long reads to a preliminary assembly. Preliminary tests with off-the-shelf tools such as racon [32], however, indeed improve sequence identity but also tend to introduce new translocation breakpoints. We suspect this is the consequence of InDels being much more abundant than mismatches in Nanopore data, which is at odds with the Needleman–Wunsch alignments used by polishing tools.

A prominent category of mis-assemblies within the LazyB contigs are inherited from chimeric reads. This therefore suggests an iterative approach: Subsampling the long-read set will produce more fragmented contigs, but statistically remove chimeric reads from the majority of replicate assemblies. Final contigs are constructed in a secondary assembly step by joining intermediary results. It might appear logical to simply run LazyB again to obtain a "consensus" assembly, where intermediary contigs play the role of longer reads with mapped anchors. In preliminary tests, however, we observed that this results in defects that depend on the sampling rate. The question of how to properly design the majority calling to construct a consensus assembly remains yet to be answered.

References

1 Alfred V. Aho, Michael R. Garey, and Jeffrey D. Ullman. The transitive reduction of a directed graph. *SIAM Journal on Computing*, 1:131–137, 1972. doi:10.1137/0201008.

2 Dmitry Antipov, Anton Korobeynikov, Jeffrey S McLean, and Pavel A Pevzner. HYBRIDSPADES: an algorithm for hybrid assembly of short and long reads. *Bioinformatics*, 32:1009–1015, 2016. doi:10.1093/bioinformatics/btv688.

3 Ali Baharev, Hermann Schichl, and Arnold Neumaier. An exact method for the minimum feedback arc set problem. Technical report, University of Vienna, 2015.

4 Ravi Boppana and Magnús M. Halldórsson. Approximating maximum independent sets by excluding subgraphs. *BIT Numerical Mathematics*, 32:180–196, 1992. doi:10.1007/BF01994876.

5 Chen-Shan Chin, Paul Peluso, Fritz J Sedlazeck, Maria Nattestad, Gregory T Concepcion, Alicia Clum, Christopher Dunn, Ronan O'Malley, Rosa Figueroa-Balderas, Abraham Morales-Cruz, Grant R. Cramer, Massimo Delledonne, Chongyuan Luo, Joseph R. Ecker, Dario Cantu, David R. Rank, and Michael C. Schatz. Phased diploid genome assembly with single-molecule real-time sequencing. *Nature methods*, 13:1050–1054, 2016. doi:10.1038/nmeth.4035.

6 Yun Sung Cho, Hyunho Kim, Hak-Min Kim, Sungwoong Jho, JeHoon Jun, Yong Joo Lee, Kyun Shik Chae, Chang Geun Kim, Sangsoo Kim, Anders Eriksson, et al. An ethnically relevant consensus korean reference genome is a step towards personal reference genomes. *Nature Comm.*, 7:13637, 2016. doi:10.1038/ncomms13637.

7 Alex Di Genova, Elena Buena-Atienza, Stephan Ossowski, and Marie-France Sagot. WENGAN: Efficient and high quality hybrid *de novo* assembly of human genomes. Technical Report 840447, bioRxiv, 2019. doi:10.1101/840447.

8 Kunal Dutta and C. R. Subramanian. Induced acyclic tournaments in random digraphs: sharp concentration, thresholds and algorithms. *Discussiones Mathematicae Graph Theory*, 34:467–495, 2014. doi:10.7151/dmgt.1758.

9 Francesca Giordano, Louise Aigrain, Michael A. Quail, Paul Coupland, James K. Bonfield, Robert M. Davies, German Tischler, David K. Jackson, Thomas M. Keane, Jing Li, Jia-Xing Yue, Gianni Liti, Richard Durbin, and Zemin Ning. *De novo* yeast genome assemblies from MinION, PacBio and MiSeq platforms. *Scientific Reports*, 7:1–10, 2017. doi:10.1038/s41598-017-03996-z.

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

11 Ehsan Haghshenas, Hossein Asghari, Jens Stoye, Cedric Chauve, and Faraz Hach. HASLR: Fast hybrid assembly of long reads. Technical Report 921817, bioRxiv, 2020. doi:10.1101/2020.01.27.921817.

12 Pinar Heggernes, Daniel Meister, and Charis Papadopoulos. A new representation of proper interval graphs with an application to clique-width. *Electronic Notes in Discrete Mathematics*, 32:27–34, 2009. doi:10.1016/j.endm.2009.02.005.

13 Hans J. Jansen, Michael Liem, Susanne A. Jong-Raadsen, Sylvie Dufour, Finn-Arne Weltzien, William Swinkels, Alex Koelewijn, Arjan P. Palstra, Bernd Pelster, Herman P. Spaink, et al. Rapid de novo assembly of the European eel genome from nanopore sequencing reads. *Scientific reports*, 7:7213, 2017. doi:10.1038/s41598-017-07650-6.

14 Arthur B. Kahn. Topological sorting of large networks. *Communications of the ACM*, 5:558–562, 1962. doi:10.1145/368996.369025.

15 Telikepalli Kavitha, Christian Liebchen, Kurt Mehlhorn, Dimitrios Michail, Romeo Rizzi, Torsten Ueckerdt, and Katharina A. Zweig. Cycle bases in graphs: characterization, algorithms, complexity, and applications. *Computer Science Review*, 3:199–243, 2009. doi:10.1016/j.cosrev.2009.08.001.

16 Hui-Su Kim, Sungwon Jeon, Changjae Kim, Yeon Kyung Kim, Yun Sung Cho, Jungeun Kim, Asta Blazyte, Andrea Manica, Semin Lee, and Jong Bhak. Chromosome-scale assembly

comparison of the korean reference genome KOREF from PromethION and PacBio with Hi-C mapping information. *GigaScience*, 8:giz125, 2019. doi:10.1093/gigascience/giz125.

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

18 Sergey Koren, Brian P Walenz, Konstantin Berlin, Jason R. Miller, Nicholas H. Bergman, and Adam M. Phillippy. Canu: scalable and accurate long-read assembly via adaptive k-mer weighting and repeat separation. *Genome Research*, 27:722–736, 2017. doi:10.1101/gr.215087.116.

19 Joseph B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7:48–50, 1956. doi:10.1090/S0002-9939-1956-0078686-7.

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

21 Zhenyu Li, Yanxiang Chen, Desheng Mu, Jianying Yuan, Yujian Shi, Hao Zhang, Jun Gan, Nan Li, Xuesong Hu, Binghang Liu, Bicheng Yang, and Wei Fan. Comparison of the two major classes of assembly algorithms: overlap–layout–consensus and de-bruijn-graph. *Briefings Funct. Genomics*, 11:25–37, 2012. doi:10.1093/bfgp/elr035.

22 Pierre Marijon, Rayan Chikhi, and Jean-Stéphane Varré. yacrd and fpa: upstream tools for long-read genome assembly. Technical Report 674036, bioRxiv, 2019. doi:10.1101/674036.

23 Samuel Martin and Richard M. Leggett. Alvis: a tool for contig and read ALignment VISualisation and chimera detection. Technical Report 663401, BioRxiv, 2019. doi:10.1101/663401.

24 George B. Mertzios. A matrix characterization of interval and proper interval graphs. *Applied Mathematics Letters*, 21:332–337, 2008. doi:10.1016/j.aml.2007.04.001.

25 Burkhard Morgenstern. A simple and space-efficient fragment-chaining algorithm for alignment of DNA and protein sequences. *Applied Mathematics Letters*, 15:11–16, 2002. doi:10.1016/S0893-9659(01)00085-4.

26 Sergej Nowoshilow, Siegfried Schloissnig, Ji-Feng Fei, Andreas Dahl, Andy WC. Pang, Martin Pippel, Sylke Winkler, Alex R. Hastie, George Young, Juliana G. Roscito, Francisco Falcon, Dunja Knapp, Sean Powell, Alfredo Cruz, Han Cao, Bianca Habermann, Michael Hiller, Elly M. Tanaka, and Eugene W. Myers. The axolotl genome and the evolution of key tissue formation regulators. *Nature*, 554:50–55, 2018. doi:10.1038/nature25458.

27 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. *bioRxiv*, 2020. doi:10.1101/2020.03.14.992248.

28 Jue Ruan and Heng Li. Fast and accurate long-read assembly with wtdbg2. *Nature Methods*, 17:155–158, 2020. doi:10.1038/s41592-019-0669-3.

29 Kishwar Shafin, Trevor Pesout, Ryan Lorig-Roach, Marina Haukness, Hugh E Olsen, Colleen Bosworth, Joel Armstrong, Kristof Tigyi, Nicholas Maurer, Sergey Koren, Fritz J. Sedlazeck, Tobias Marschall, Simon Mayes, Vania Costa, Justin M. Zook, Kelvin J. Liu, Duncan Kilburn, Melanie Sorensen, Katy M. Munson, Mitchell R. Vollger, Evan E. Eichler, Sofie Salama, David Haussler, Richard E. Green, Mark Akeson, Adam Phillippy, Karen H. Miga, Paolo Carnevali, Miten Jain, and Benedict Paten. Efficient *de novo* assembly of eleven human genomes using PromethION sequencing and a novel nanopore toolkit. Technical Report 715722, BioRxiv, 2019. doi:10.1101/715722.

30 Jared T. Simpson, Kim Wong, Shaun D. Jackman, Jacqueline E. Schein, Steven JM. Jones, and Inanç Birol. ABySS: a parallel assembler for short read sequence data. *Genome Research*, 19:1117–1123, 2009. doi:10.1101/gr.089532.108.

31 Edwin A. Solares, Mahul Chakraborty, Danny E. Miller, Shannon Kalsow, Kate Hall, Anoja G. Perera, JJ. Emerson, and R. Scott Hawley. Rapid low-cost assembly of the *Drosophila*

melanogaster reference genome using low-coverage, long-read sequencing. *G3: Genes, Genomes, Genetics*, 8:3143–3154, 2018. doi:10.1534/g3.118.200162.

32 Robert Vaser, Ivan Sović, Niranjan Nagarajan, and Mile Šikić. Fast and accurate de novo genome assembly from long uncorrected reads. *Genome Research*, 27:737–746, 2017. doi:10.1101/gr.214270.116.

33 Ryan R. Wick, Louise M. Judd, Claire L. Gorrie, and Kathryn E. Holt. Unicycler: resolving bacterial genome assemblies from short and long sequencing reads. *PLOS Computational Biology*, 13:e1005595, 2017. doi:10.1371/journal.pcbi.1005595.

34 Ryan R. Wick, Louise M. Judd, and Kathryn E. Holt. Deepbinner: Demultiplexing barcoded Oxford Nanopore reads with deep convolutional neural networks. *PLOS Computational Biology*, 14:e1006583, 2018. doi:10.1371/journal.pcbi.1006583.

35 Chengxi Ye, Christopher M Hill, Shigang Wu, Jue Ruan, and Zhanshan Sam Ma. DBG2OLC: efficient assembly of large genomes using long erroneous reads of the third generation sequencing technologies. *Scientific reports*, 6:31900, 2016. doi:10.1038/srep31900.

36 Aleksey V Zimin, Guillaume Marçais, Daniela Puiu, Michael Roberts, Steven L Salzberg, and James A Yorke. The MaSuRCA genome assembler. *Bioinformatics*, 29:2669–2677, 2013. doi:10.1093/bioinformatics/btt476.

A Definitions of Alignment Graph Defects

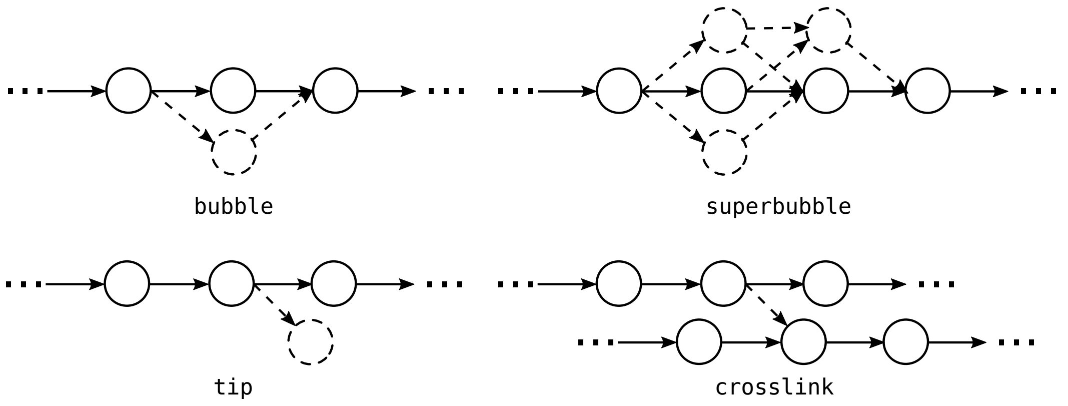

Figure 4 Examples of assembly graph defects in $\overrightarrow{G}^{\triangle}$.

On ideal data, $\overrightarrow{G}^{\triangle}$ would consist of a unique golden path. For real data, however, it also also harbors bubbles, tips, and crosslinks. We briefly define these types of imperfections here. Given two nodes $s, t \in \overrightarrow{G}^{\triangle}$, an $s - t$ path is a path starting in s and ending in t. A *simple bubble* consists of two vertex disjoint $s - t$ paths. This construct can be extended to *super-bubbles*, defined as a set of $s - t$ paths, exactly including all nodes reachable from s without passing t and *vice versa*. Bubbles and superbubbles are primarily the result of unrecognized overlaps. *Tips* are “side branches” that do not reconnect with the dominating paths and thus have distinct end-points. *Crosslinks*, finally, are connecting edges between two golden paths. As tips themselves may also be subject to mild noise, and crosslinks may occur near the start- or end-sites of the true paths, both are not always easily distinguished. Hence, we apply the heuristic filtering steps described in the main text.

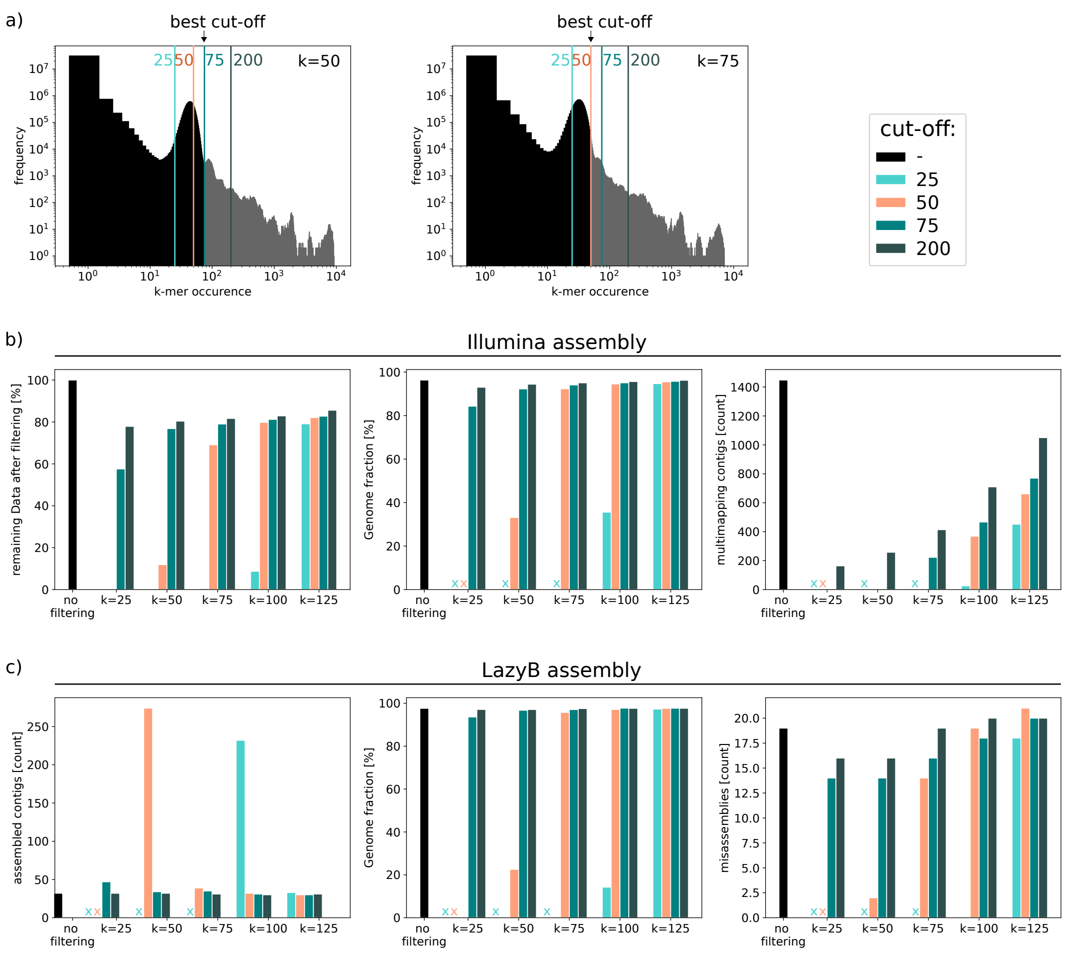

Figure 5 Assembly statistics of yeast as a function of the k-mer size and maximal occurrence cut-off used to remove very frequent k-mers from short reads prior to unitig assembly. a) k-mer profiles for k=50 bp and k=75 bp. Cut-offs restrict short reads to different degrees. Note logarithmic axes. b) Illumina unitigs (left: percentage of remaining short-read data; middle: fraction of the reference genome covered; right: number of unitigs mapping multiple times to reference). c) final `LazyB` assembly left: number of unitigs; middle: fraction of the reference genome covered; right: number of mis-assemblies). x: not enough data to assemble.

B Influence of Short Read Filtering

The strategies for filtering short-read data have a larger impact than the choice of the k-mer size for unitig assembly (Fig. 5). This is not surprising given that both chimeric unitigs and unitigs that harbor repetitive DNA elements introduce spurious edges into G and thus negatively influence the assembly. In order to exclude short reads that contain highly frequent k-mers, the maximal tolerated occurrence has to be set manually and is dependent on the k-mer size. Setting the cut-off right next to the main peak in the profiles has turned out to be a good estimate. After assembling short reads, unitigs are mapped to long reads and a coverage profile over the length of every unitig is calculated. Unitigs with maximal coverage above interquartile range $IQR \times 1.5 + Q3$ are considered outliers. However, regions below coverage threshold (Q3) spanning more than 500 bp can be “rescued”. This filter step effectively reduces ambiguous regions, especially when no previous filtering is applied (Fig. 6). Combining both short-read filter improves the assembly quality; see Table 2.

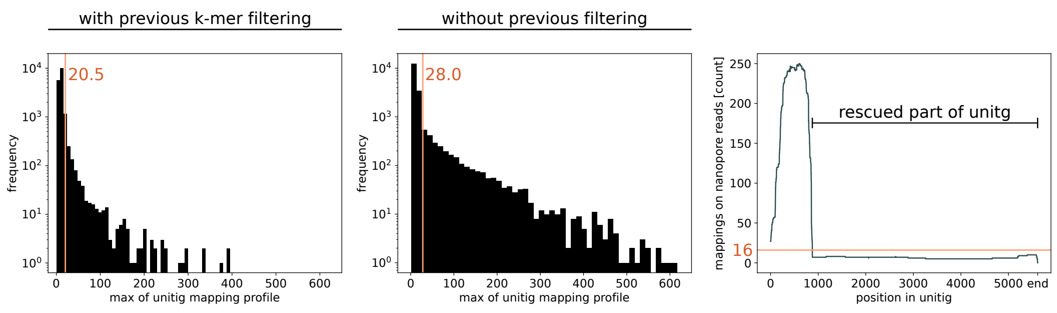

Figure 6 Exclusion of unitigs based on very high mapping coverage. Thresholds are IQR×1.5+Q3. Shown are maximal values of coverage profiles for unitigs assembled with (left) and without (middle) previous k-mer filtering. Note the logarithmic axes. right: exemplary profile; only the high-coverage peak is excluded. Threshold is Q3.

Table 2 Impact of short-read filtering strategies on `LazyB` assembly quality in fruit fly. Column descriptions: **compl**eteness of the assembly, **#ctg** number of contigs, **#MA** number of misassemblies (breakpoints relative to the reference assembly).

Filter strategy	**compl.**[%]	**#ctg**	**#MA**
no filter	82.81	457	302
***k*-mer filter**	80.66	567	104
unitig filter	80.71	563	108
***k*-mer and unitig filter**	80.11	596	99

C Validation of `Minimap2` Anchor Alignments

Classic alignment tools, even those specifically advertised for this purpose, rely on scoring schemes that cannot accurately represent the high InDel profiles of long-read data. Instead, they rely on seeds of high quality matches that are then chained with high error tolerance. Currently, `minimap2` is one of the most commonly used tools for this purpose. Since we do not have a gold set of perfect data, we can only roughly estimate the influence of this heuristic on the `LazyB` alignment quality in a related experiment. Specifically, we test consistency of anchor alignments on pairs of long reads to direct alignments of both reads for fruit fly. Consistency is validated at the level of relative orientation, the offset indicated by both alignment methods, the portion of overlap that can be directly aligned and whether direct alignment of the long reads is possible at all. Different relative orientations were observed only in very small numbers. Changes in the offset by more then 5% of the longer read length are equally rare (Fig. 7). However, requiring a direct alignment of at least 75% of the overlap region marks 4.6% of the anchor links as incorrect. Removing those has a negative effect on the final `LazyB` assembly and in particular tends to break correct contigs apart; see Table 3. In our test set 7.7% of direct alignments of two anchor-linked long reads gave no result. In these cases, expected overlaps are rather short (Fig. 7). We therefore tested whether the assembly could be improved by excluding those connections between long reads for which no alignment could be calculated despite the presence of an overlap of at least 1 kbp (3.7%). We found, however, that this procedure also causes the loss of correct edges in G.

Summarizing, we observe three facts: (1) The overwhelming number of pairs is consistent and therefore true. (2) Removing inconsistent edges from the assembly not only does not improve the results but results are worse on average. (3) While we can manually identify some incorrect unitig matches, the mappings produced by `minimap2` are too inconsistent

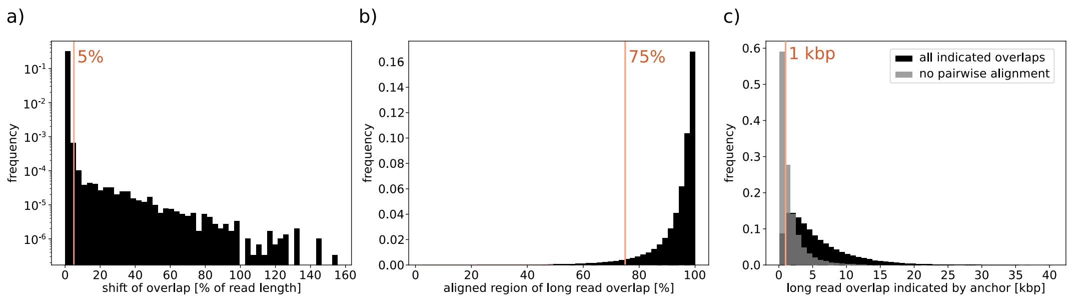

Figure 7 Consistency test of anchor-linked long-read overlaps to direct alignments of both reads on fruit fly. a) Frequencies of shifted offsets (% of the longer read); changes up to 5% are tolerated; note logarithmic axis. b) Frequencies of the percentage at which the direct alignment covers the overlap. A minimum of 75% is set for consistency. c) Long read pairs where no direct alignment is possible tend to have shorter anchor-indicated overlaps. Connections that cannot be confirmed via direct alignments despite an expected overlap of at least 1 kbp are excluded.

for proper testing. Since we have no proper methods to identify such false positives we also cannot properly estimate the number of false negatives, i.e., missing matches in the graph $\vec{G}$, e.g. by computing a transitive completion.

Overall, our main results together with (1) indicate that a high level trust in the anchors mapping in warranted. We also conclude that `minimap2` is sufficient for our purposes. However, the data also suggest that the assembly would profit from a more accurate handling of the alignments.

Table 3 Assessment of different parameters to verify long-read overlaps and their impact on `LazyB` assembly quality on fruit fly. Overlaps are indicated by anchors and evaluated by pairwise long-read alignments. They are considered valid if: the relative direction suggested by the anchor matches that of the pairwise alignment (direction); the offset is sufficiently similar for both methods (offset); at least 75% of the overlap is found as direct alignment (incomplete mapping); the overlap indicated by the anchor is less than or equal to 1 kbp or a pairwise alignment is possible (no mapping). Column descriptions: **compl**eteness of the assembly, **#ctg** number of contigs, **#MA** number of mis-assemblies (breakpoints relative to the reference assembly).

Varification parameters	**compl.**[%]	**#ctg**	**#MA**
direction	80.13	608	111
direction + offset	80.08	622	103
direction + offset + incomplete mapping	80.04	1263	121
no mapping	80.15	801	113

D Alternative Heuristic for Maximum Induced Transitive Tournament

We found that $\nu(e)$ provides a better heuristic than the initial bit scores $\Omega(e)$ for the extraction of the paths. Most plausibly, one is interested transitive tournaments as an indication for the correct assembly path. Since this is a computationally difficult problem, we described in the main text a heuristic based on longest paths in $\vec{G}^{\triangle}$, that is equivalent for perfect data. Here we briefly sketch an alternative heuristic operating directly on the edges of $\vec{G}$.

Here, we denote by $N_{+}(r)$ and $N_{-}(r)$ the set of out- and in-edges of r in $\vec{G}$. We note that the out-neighbors of r form an induced proper interval graph if and only if they are an acyclic tournament (AT). With noisy data, we therefore ask for the maximal AT $\vec{K}_r$ with source r.

Unfortunately, this task is NP-hard even in acyclic graphs [8]. We therefore resort to a heuristic making use of the topological order in $N_+(r)$ inherited from the topological order of $\overrightarrow{G}$ and process nodes in increasing order starting with r: (i) Initialize a list $\mathcal{L}$ of candidates with the pair (H, A) where $H = \{r\}$ and $A = N_+(r)$. (ii) For each candidate $(H, A) \in \mathcal{L}$ consider the $r_i \in A$ in topological order and append (H'_i, A'_i) to $\mathcal{L}$, where $H' = H \cup \{r_i\}$ and $A' = A \cap N_+(r_i)$. (iii) Select the candidate with maximum cardinality $|H_r|$.

E Evaluation of Real Data Sets

We re-used data sets from previously published benchmarks of Nanopore assemblies. For yeast (*S. cerevisiae*) we used Nanopore sets ERR1883389 for lower coverage, ERR1883399 for higher coverage, and short-reads set ERR1938683, all from bioproject PRJEB19900 [9]. For comparison we use the reference genome R64.2.1 of strain S288C from the SGD. For the fruit fly (*D. melanogaster*) we used the Oxford Nanopore and Illumina raw data of bioproject PRJNA433573 [31], and the FlyBase reference genome 6.30 (`http://www.flybase.org`). On Human we use SRX6356866-8 on bioproject PRJNA549351 [16] for long reads and SRA292482 [6] for short reads. We compare against reference GRCh38.p13. `QUAST` [10] is a specialized tool to evaluate the quality of assemblies. We report statistics without further processing. Table 4 summarizes the resource requirements for the assembly of the yeast, fruit fly, and human data set.

Table 4 Assessment of running times for `LazyB` and `Canu`. Resources for `LazyB` are given in three steps: 1) `ABySS` unitig assembly; 2) Mapping of unitigs to long reads and 3) `LazyB` itself. Step 1) is often not needed as short-read assemblies are available for many organisms. Resources are only compared for yeast and fruit fly, as `Canu` cannot be run for human in sensible time and resource-constraint on our machine. As all tools except `LazyB` and `DBG2OL` are parallelized, running times are given as the sum of time spent by all CPUs. `ABySS` greatly dominates the `LazyB` pipeline. Nevertheless, `LazyB` is faster on a factor of > 60 to `Canu` and ≈ 3 to `DBG2OL`.

Organism	**X**	**Tool**	**Runtime** (dd:hh:mm:ss)	**RAM** (MB)
yeast		`ABySS`	00:00:11:03	2283
	~5×	Mapping	00:00:00:05	540
		`LazyB`	00:00:00:30	136
		`ABySS` + Mapping + `LazyB`	00:00:11:38	2283
		`Canu`	00:10:23:55	2617
		`HASLR`	00:00:06:44	4922
		`DBG2OL`	00:00:31:46	1141
	~11×	Mapping	00:00:00:15	1544
		`LazyB`	00:00:01:46	362
		`ABySS` + Mapping + `LazyB`	00:00:13:04	2283
		`Canu`	00:13:44:16	6779
		`HASLR`	00:00:08:09	4922
		`DBG2OL`	00:00:51:13	1264
fruit fly		`ABySS`	00:02:32:39	25344
	~5×	Mapping	00:00:02:43	6433
		`LazyB`	00:00:08:33	613
		`ABySS` + Mapping + `LazyB`	00:02:43:55	25344
		`Canu`	02:13:51:39	7531
		`HASLR`	00:01:30:33	5531
		`DBG2OL`	00:07:58:22	6151
	~10×	Mapping	00:00:06:11	9491
		`LazyB`	00:00:11:57	2241
		`ABySS` + Mapping + `LazyB`	00:02:50:47	25344
		`Canu`	07:04:08:28	7541
		`HASLR`	00:01:43:21	5553
		`DBG2OL`	02:07:32:01	17171