Coverage-preserving sparsification of overlap graphs for long-read assembly

Chirag Jain

Department of Computational and Data Sciences, Indian Institute of Science, Bengaluru, India
Email: chirag@iisc.ac.in

Abstract. Read-overlap-based graph data structures play a central role in computing *de novo* genome assembly using long reads. Many assembly tools use the string graph model [Myers, Bioinformatics 2005] to sparsify overlap graphs. Graph sparsification improves accuracy by removing spurious and redundant connections. However, a graph model must be coverage-preserving, i.e., it must ensure that each chromosome can be spelled as a walk in the graph, given sufficient sequencing coverage. This property becomes even more important for diploid genomes, polyploid genomes and metagenomes where there is a risk of losing haplotype-specific information.

We develop a novel theoretical framework under which the coverage-preserving properties of a graph model can be analysed. We first prove that de Bruijn graph and overlap graph models are guaranteed to be coverage-preserving. We also show that the standard string graph model lacks this guarantee. The latter result is consistent with the observation made in [Hui et al. ISIT'16] that removal of contained reads during string graph construction can lead to coverage gaps. To remedy this, we propose practical heuristics that are well-supported by our theoretical results to sparsify overlap graphs. In our experiments conducted by using simulated long reads from HG002 human diploid genome, we find that 50 coverage gaps are introduced on average by ignoring contained reads from nanopore datasets. We tested the proposed heuristics for deciding which contained reads should be retained to avoid the coverage gaps. The proposed method retains a small fraction of contained reads ($1 - 2\%$) and closes majority of the coverage gaps.

1 Introduction

Long-read sequencing technologies have shown the greatest promise in computing high-quality haplotype-resolved genome assemblies [1,3,8,10,24,26,30,31,32]. Long-read accuracy has improved enormously over the last few years. For instance, PacBio HiFi sequencing technology yields consensus sequencing reads that are both long (averaging 10-25 kb) and highly accurate (averaging 99.8%) [29]. Similarly, modal raw read accuracy above 99% has been reported using reads from Oxford Nanopore Technology (ONT) sequencing [25]. Long and accurate reads are useful to sequence traditionally challenging diploid or polyploid genomes and facilitate haplotype-resolved *de novo* assembly [11,16,17,18]. The combination of PacBio and ONT reads has recently enabled the first complete gapless assembly of a human genome [22].

Genome assembly is the task of reconstructing the original sequence from a large number of reads. Read-overlap-based assembly algorithms work by constructing an *overlap graph* where each vertex corresponds to a read, and edges represent suffix-prefix overlaps between reads. Practical algorithms account for the following two challenges while computing genome assembly. First, it is usually possible to reconstruct different genomes using the same input set of reads, each of which is fully consistent with the data. Accordingly, problem formulations for genome assembly which seek a single genome reconstruction, e.g., by finding a Hamiltonian cycle in an overlap graph, or computing the shortest common superstring of input reads, are not used in practice [19]. Instead, assemblers compute contigs which are long, contiguous segments and unambiguously inferred to be part of the genome.

Second, the presence of genomic repeats and variable-length reads creates many spurious edges in the overlap graph. This challenge is often addressed by using graph sparsification heuristics that exclude vertices and edges which are likely to be redundant or false. The commonly used *string graph* model [21], for example, (i) removes reads that are *contained* as a substring of a longer read, (ii) ignores suffix-prefix overlaps that are shorter than the longest possible between a read pair, and (iii) removes transitive edges. More aggressive sparsification procedures also exist in the literature, for example, the *best overlap graph* approach proposed in [20] selects single best overlap per read end, where *best* is defined using some criterion. Inspite of the use of such graph sparsification techniques in most long-read assembly tools [3,4,6,7,12,13,23,28], theoretical understanding of these heuristics is fairly limited.

Assuming a genome is sequenced with sufficient coverage, then there must exist walks in an assembly graph that can spell the true sequences corresponding to each chromosome. An assembly graph which guarantees this property is said to be *coverage-preserving* (formally defined in the next section). Note that this property is not intended to check if the true sequences can be unambiguously reconstructed [2,27], but to check whether their information is preserved in the graph. In this paper, we make the following contributions:

1. We formulate the coverage-preserving property of genome assembly graph models, and perform a rigorous evaluation in three commonly-used models: (a) de Bruijn graphs, (b) overlap graphs and (c) string graphs. We prove that de Bruijn graphs and overlap graphs are guaranteed to be coverage preserving, but string graphs are not.
2. Graph sparsification is a critical step during genome assembly to prune the overlap graph, as it helps to compute longer contigs. We develop theoretical results to compute a sparse overlap graph while preserving the coverage-preserving property.
3. We extend the proposed theory into practical heuristics and a prototype software ContainX (github.com/at-cg/ContainX). The proposed heuristics are useful to identify redundant contained reads, i.e., the contained reads which can be excluded from overlap graph without violating the coverage-preserving property. We empirically show that the proposed heuristics can help improve assembly quality.
4. We conducted experiments by using error-free long read datasets simulated from diploid HG002 human genome assembly while matching read length distributions to real PacBio HiFi and ONT data. We show that 1 and 50 coverage gaps are introduced on average by removing all contained reads in HiFi and ONT datasets respectively. ContainX algorithm excludes $98 - 99\%$ contained reads from the graph and leaves ≤ 5 coverage gaps. This result compares favorably to the performance of other known solutions for this problem.

2 Preliminaries

Notation on strings. For a linear string $x = a_1 \dots a_n$ over alphabet Σ ={A,C,G,T}, $|x| = n$ is the length of x, $x[i] = a_i$ is the ith symbol of X, and $x[i : j] = a_i \dots a_j$ is the substring from position i to position j. Let x^i denote string x concatenated with itself i times. Define $pre_i(x)$ to be the first i characters of string x. String x is said to have a suffix-prefix overlap of length l with string y when a proper suffix of length l in x equals a proper prefix of y. Recall that a proper prefix or a proper suffix of a string cannot be equal to the string itself and is non-empty.

Circular genome model. We will use a circular genome model to avoid edge-effects associated with sequencing coverage. A circular string can be viewed as a traditional linear string which has the left- and the right-most symbols wrapped around and glued together. Circular string $z = \langle a_1 \dots a_n \rangle$ has length $|z| = n$. Substring $z[i : j]$, where $i \in [1, n], j \geq i$, equals the finite substring of the linear infinite string $(a_1 \dots a_n)^\infty$ from position i to position j. Note that offset j in $z[i : j]$ can be greater than $|z|$ because z is a circular string. Further, substring $z[i : j]$ is said to be *repetitive* iff there exists $z[i' : j'] = z[i : j], i' \neq i$. Suppose the true unknown genome is a set of circular strings, each representing a chromosome sequence. Let ϕ be a known upper bound on the maximum length of a chromosome in the genome. A sequence *read* sampled from the genome is a substring of one of the circular strings in the genome. An indexed multiset of the reads is represented using symbol $\mathcal{R}$. Read $x \in \mathcal{R}$ is labeled as a *contained* read if there exists a read $y \in \mathcal{R} \setminus \{x\}$ such that x is a substring of y. If y is also contained in x, then x and y are identical. In this case, break the tie by assuming that the read with lower index in $\mathcal{R}$ is contained within the read with higher index. The reads which contain read x are referred to as *parent* reads of x. The proposed framework assumes that there are no sequencing errors or reads have been error-corrected. In our theoretical results, we will also assume that DNA is a single-stranded molecule.

Assumption on sequencing coverage. Depth of sequencing must be sufficient enough for computing *de novo* genome assembly. We wish to ensure that reads "cover" the genome, and the length of suffix-prefix overlap between "consecutive" reads is above a certain threshold. One way to achieve this is following. We say that there is *sufficient coverage* over circular string z if there exist parameters $l_1, l_2 \in \mathbb{N}$ such that $l_1 > l_2$ and all intervals of length $l_1 - l_2$ in z include the start of at least one substring of length $\geq l_1$ that matches a read $\in \mathcal{R}$. See Figure 1 for an example. Let $\mathcal{C}(\mathcal{R}, l_1, l_2, \phi)$ be the set of all *candidate* circular strings of length $\leq \phi$ which satisfy the above coverage assumption using a given read set $\mathcal{R}$ and coverage parameters (l_1, l_2). The genome is a subset of $\mathcal{C}(\mathcal{R}, l_1, l_2, \phi)$.

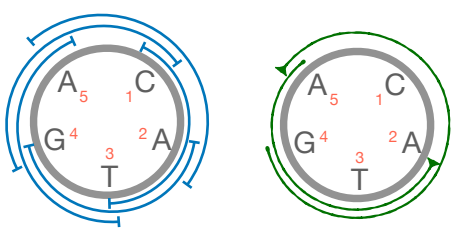

Fig. 1. Suppose $l_1 = 4$ and $l_2 = 1$. A circular string $z = \langle \text{CATGA} \rangle$ has five intervals of length $l_1 - l_2 = 3$ as shown in blue. Two reads ACATG and ATGA, which match substrings $z[5 : 9]$ and $z[2 : 5]$ respectively, are shown in green. z has sufficient coverage because all five intervals include at least one of the starting positions of the reads.

Assembly graphs. De Bruijn graphs, overlap graphs and string graphs are popular graph-based frameworks used for *de novo* genome assembly. Let $G(V, E, \sigma, w)$ denote a directed assembly graph where vertices are labeled with strings, and edges indicate suffix-prefix overlap relations between the labels of connected vertices. Function $\sigma : V \to \Sigma^+$ assigns a string label to each vertex. Function $w : E \to \mathbb{N}$ assigns a weight to each edge. Weight of an edge $v_1 \to v_2$ signifies the length of prefix of $\sigma(v_1)$ that is not in the suffix-prefix match. In a de Bruijn graph $B_k(\mathcal{R})$, vertex set V corresponds to the set of all k-mer substrings in read set $\mathcal{R}$, and an edge of weight one exists from vertex v_1 to vertex v_2 if and only if $\sigma(v_1)[2 : k] = \sigma(v_2)[1 : k - 1]$. This data structure is also sometimes referred to as node-centric de Bruijn graph [5].

Overlap graph $O(\mathcal{R})$ is a directed multi-graph where vertices correspond to sequence reads and edges correspond to suffix-prefix overlaps among the reads. A directed edge $v_1 \to v_2$ with weight $|\sigma(v_1)| - l$ is drawn if and only if read $\sigma(v_1)$ has a suffix-prefix overlap of length l with read $\sigma(v_2)$. A subgraph of overlap graph

$O(\mathcal{R})$ that contains only the edges representing suffix-prefix overlaps of length $\geq k$ is denoted as $O_k(\mathcal{R})$. Myers [21] introduced a sparse variant of overlap graph structure called as string graph. Unlike overlap graphs which are defined as directed multi-graphs, a string graph, denoted as $S(\mathcal{R})$ is a directed graph. A string graph includes only the longest suffix-prefix overlap between a read pair. Second, vertices associated with contained reads are excluded from a string graph. Finally, if vertex v_1 connects to v_2, v_2 connects to v_3 and v_1 connects to v_3 using edges e_1, e_2 and e_3 respectively, and $w(e_1) + w(e_2) = w(e_3)$, then edge e_3 is excluded from the string graph because it is transitively deducible. Removal of transitive edges is referred to as transitive sparsification. In practice, a string graph has significantly fewer edges compared to the overlap graph which enables computation of longer assembly contigs. A subgraph of string graph $S(\mathcal{R})$ which only uses suffix-prefix overlaps of length $\geq k$ is denoted as $S_k(\mathcal{R})$.

The circular strings spelled by graphs. By performing a closed walk in an assembly graph, one can "spell" a circular string. Suppose $(v_1, e_1, v_2, e_2, \ldots, v_{n-1}, e_{n-1}, v_n), n > 1$ is a sequence of vertices and edges in a closed walk such that edge e_i connects v_i to v_{i+1} and $v_n = v_1$. The circular string spelled by this closed walk is $\langle\ pre_{w(e_1)}(\sigma(v_1))\ pre_{w(e_2)}(\sigma(v_2)) \ldots\ pre_{w(e_{n-1})}(\sigma(v_{n-1}))\ \rangle$. This is the circular string formed by concatenating labels of the vertices without spelling the overlapped substrings twice.

Definition 1. *Given set of reads $\mathcal{R}$, coverage parameters (l_1, l_2) and chromosome length threshold ϕ, graph $G(V, E, \sigma, w)$ is said to be* ***coverage-preserving*** *iff all candidate strings $\in \mathcal{C}(\mathcal{R}, l_1, l_2, \phi)$ can be spelled in the graph.*

3 Analysis of known graph models

3.1 De Bruijn graphs

Theorem 1. *de Bruijn graph $B_k(\mathcal{R})$ is coverage-preserving for all $k \leq l_2 + 1$.*

Proof. If a circular string can be spelled in $B_{l_2+1}(\mathcal{R})$, then it can certainly be spelled in $B_k(\mathcal{R})$, $\forall k < l_2 + 1$. Therefore, it suffices to prove the claim for $k = l_2 + 1$. We will prove that $B_{k=l_2+1}(\mathcal{R})$ is coverage-preserving using contradiction. Suppose there exists a circular string $z \in \mathcal{C}(\mathcal{R}, l_1, l_2, \phi)$ which cannot be spelled in $B_k(\mathcal{R})$. Let k_i $(1 \leq i \leq n)$ denote the k-mer substring starting from position i in z. If all k_is exist in the set of all k-mer substrings extracted from read set $\mathcal{R}$, then it is trivial to construct a closed walk in $B_k(\mathcal{R})$ which spells z. Therefore, at least one of the k_is must be missing. Consider the minimum i for which this is true. Next, consider the $(l_1 - l_2)$-long interval ending at the position i in the circular string z. Based on the coverage assumption, there is at least one read $\in \mathcal{R}$ which matches a substring of length $\geq l_1$ of z starting in this interval. Such a read must contain k-mer k_i as its substring. □

3.2 Overlap graphs

Unlike de bruijn graphs, overlap graphs require a more careful analysis due to variable-length suffix-prefix overlaps and variable-length vertex labels. The aim of this section is to prove the following.

Theorem 2. *Overlap graph $O_k(\mathcal{R})$ is coverage-preserving for all $k \leq l_2$.*

If a circular string can be spelled in $O_{l_2}(\mathcal{R})$, then it can be spelled in $O_k(\mathcal{R})$, $\forall k < l_2$. For a circular string $z \in \mathcal{C}(\mathcal{R}, l_1, l_2, \phi)$, we propose an algorithm to identify a closed walk in $O_{l_2}(\mathcal{R})$ which spells z. For this, we first need some definitions. Let $\mathcal{I}_i$ be the $(l_1 - l_2)$-long interval starting at position i in z, where i ranges from 1 to $|z|$. For each interval $\mathcal{I}_i$, let $z[\mathcal{I}_i.s : \mathcal{I}_i.e]$ be the substring of length $\geq l_1$ which starts in interval $\mathcal{I}_i$ and matches a read $\in \mathcal{R}$. If there are multiple such substrings available, pick the ones with the least value of $\mathcal{I}_i.s$, and then with the least value of $\mathcal{I}_i.e$. Let $\mathcal{I}_i.r$ denote the read that matches substring $z[\mathcal{I}_i.s : \mathcal{I}_i.e]$. Recall that a circular string is equivalent to any of its cyclic rotated version, therefore, assume wlog that $|\mathcal{I}_1.r| = \max_{i \in [1,|z|]} |\mathcal{I}_i.r|$. If $|\mathcal{I}_1.r| \geq |z| + l_2$, it is trivial to construct a valid closed walk by using a single vertex corresponding to read $\mathcal{I}_1.r$. In the following, we will focus on the case when $|\mathcal{I}_i.r| < |z| + l_2$ for all $i \in [1, |z|]$. The inequalities $l_1 \leq |\mathcal{I}_i.r|$ and $|\mathcal{I}_i.r| < |z| + l_2$ imply that the size of each interval, i.e., $l_1 - l_2$ is $< |z|$. Therefore, $|z| > 1$.

Lemma 1. *For any two consecutive intervals* $(\mathcal{I}_i, \mathcal{I}_{i+1})$ *in* $(\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_{|z|})$, $\mathcal{I}_i.e \geq \mathcal{I}_{i+1}.s + l_2 - 1$.

Proof. All intervals including $\mathcal{I}_i$ and $\mathcal{I}_{i+1}$ are of length $l_1 - l_2$. $\mathcal{I}_i.e - \mathcal{I}_{i+1}.s$ equals its minimum value $l_2 - 1$ when (a) $\mathcal{I}_i.s$ equals the first position in interval $\mathcal{I}_i$, (b) $\mathcal{I}_{i+1}.s$ equals the last position in interval $\mathcal{I}_{i+1}$, and (c) the substring $z[\mathcal{I}_i.s : \mathcal{I}_i.e]$ has minimum possible length l_1. □

Next we will identify a subsequence $(\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_p)$ of $(\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_{|z|})$ such that vertices corresponding to reads $(\mathcal{T}_1.r, \mathcal{T}_2.r, \ldots, \mathcal{T}_p.r, \mathcal{T}_1.r)$ can be connected to form a valid closed walk which spells z (Figure 2). Let $\mathcal{T}_1 = \mathcal{I}_1$. Suppose a partial subsequence $(\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_i = \mathcal{I}_j)$ has been computed so far. Continue by selecting the first interval among $(\mathcal{I}_{j+1}, \ldots, \mathcal{I}_{|z|})$ as $\mathcal{T}_{i+1}$ which satisfies the conditions $\mathcal{T}_{i+1}.s > \mathcal{T}_i.s$ and $\mathcal{T}_{i+1}.e > \mathcal{T}_i.e$. This selection procedure ensures that $\mathcal{T}_1.s < \mathcal{T}_2.s < \ldots < \mathcal{T}_p.s$ and $\mathcal{T}_1.e < \mathcal{T}_2.e < \ldots < \mathcal{T}_p.e$. Suppose $\mathcal{I}_q$ is the interval which ends at position $\mathcal{I}_1.s - 1$ if $\mathcal{I}_1.s > 1$ or else at position $|z|$ if $\mathcal{I}_1.s = 1$.

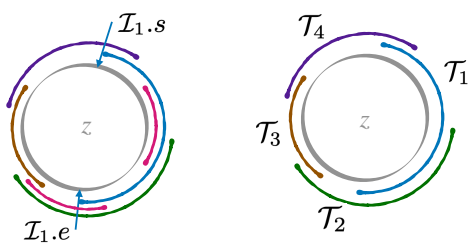

Fig. 2. A graphic illustration of the subsequence computed from the complete sequence of intervals $\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_{|z|}$. In the left figure, the gray-colored circle indicates circular string z, and the curved line segments indicate distinct substrings of z associated with intervals $\mathcal{I}_1, \mathcal{I}_2 \ldots, \mathcal{I}_{|z|}$. The right figure illustrates only the substrings associated with selected intervals in the subsequence $\mathcal{T}_1, \mathcal{T}_2, \mathcal{T}_3, \mathcal{T}_4$.

Lemma 2. *Length of the subsequence* $(\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_p)$ *identified by the above procedure is at least two.*

Proof. By contradiction. Assume $p = 1$, i.e., the computed subsequence is $\mathcal{T}_1 = \mathcal{I}_1$. Based on our previous arguments, we have $|\mathcal{I}_1.r| < |z| + l_2$, $l_1 - l_2 < |z|$ and $|z| > 1$. Interval $\mathcal{I}_q$ cannot contain position $\mathcal{I}_1.s$ because size of each interval, i.e., $l_1 - l_2$, is $< |z|$. Therefore, $\mathcal{I}_1.s < \mathcal{I}_q.s \leq |z|$. As interval $\mathcal{I}_q$ was not picked among the subsequence of intervals, $\mathcal{I}_1.e$ must be $\geq \mathcal{I}_q.e$. Next, we will identify the minimum possible value of $\mathcal{I}_q.e$ to get a lower bound of $\mathcal{I}_1.e$. $\mathcal{I}_q.e$ is lowest when the length of substring $z[\mathcal{I}_q.s : \mathcal{I}_q.e]$ is l_1 and $\mathcal{I}_q.s$ equals the first position in $\mathcal{I}_q$. Using these conditions, observe that $\mathcal{I}_q.e \geq \mathcal{I}_1.s + |z| + l_2 - 1$, and therefore, $\mathcal{I}_1.e \geq \mathcal{I}_1.s + |z| + l_2 - 1$. This implies that the length of substring $z[\mathcal{I}_1.s : \mathcal{I}_1.e]$ is $\geq |z| + l_2$. This is not possible when $|\mathcal{I}_1.r| < |z| + l_2$. □

Lemma 3. *For any two consecutive intervals* $(\mathcal{T}_i, \mathcal{T}_{i+1})$ *in* $(\mathcal{T}_1, \mathcal{I}_2, \ldots, \mathcal{T}_p)$, $\mathcal{T}_i.s < \mathcal{T}_{i+1}.s$, $\mathcal{T}_i.e < \mathcal{T}_{i+1}.e$ *and* $\mathcal{T}_i.e \geq \mathcal{T}_{i+1}.s + l_2 - 1$.

Proof. The first two inequalities $\mathcal{T}_i.s < \mathcal{T}_{i+1}.s$ and $\mathcal{T}_i.e < \mathcal{T}_{i+1}.e$ are guaranteed by the selection procedure of T_{i+1}. Suppose $\mathcal{T}_i = \mathcal{I}_j$ and $\mathcal{T}_{i+1} = \mathcal{I}_k$. From Lemma 1, we know that $\mathcal{I}_{k-1}.e \geq \mathcal{I}_k.s + l_2 - 1$. If $j = k - 1$, then inequality $\mathcal{T}_i.e \geq \mathcal{T}_{i+1}.s + l_2 - 1$ holds. Otherwise, $j < k - 1$ implies that $\mathcal{I}_{k-1}$ was not picked among the subsequence of intervals. This would happen in either of the following two situations (i) $\mathcal{I}_j.s = \mathcal{I}_{k-1}.s, \mathcal{I}_j.e = \mathcal{I}_{k-1}.e$, (ii) $\mathcal{I}_j.s < \mathcal{I}_{k-1}.s, \mathcal{I}_j.e \geq \mathcal{I}_{k-1}.e$. In either case, $\mathcal{I}_j.e \geq \mathcal{I}_{k-1}.e$. Therefore, $\mathcal{T}_i.e \geq \mathcal{T}_{i+1}.s + l_2 - 1$. □

Lemma 4. $\mathcal{T}_p.s < \mathcal{T}_1.s + |z|$, $\mathcal{T}_p.e < \mathcal{T}_1.e + |z|$ *and* $\mathcal{T}_p.e \geq \mathcal{T}_1.s + |z| + l_2 - 1$.

Proof. The first inequality $\mathcal{T}_p.s < \mathcal{T}_1.s + |z|$ holds because $\mathcal{T}_i.s \in [1, |z|]$ $\forall i \in [1, p]$. For the second inequality, recall that $|\mathcal{I}_1.r| = \max_{i \in [1,|z|]} |\mathcal{I}_i.r|$. If $\mathcal{T}_p.e \geq \mathcal{T}_1.e + |z|$, then $|\mathcal{I}_1.r| < |\mathcal{T}_p.r|$ which cannot be true. The third inequality can be proved by using the arguments from the proof of Lemma 2. If $q < |z|$, then the intervals $\mathcal{I}_{q+1}, \ldots, \mathcal{I}_{|z|}$ will contain $\mathcal{I}_1.s$ and won't be selected in the subsequence. As a result, either $p = q$ or $p < q$. In either case, $\mathcal{T}_p.e \geq \mathcal{I}_q.e$. $\mathcal{I}_q.e \geq \mathcal{I}_1.s + |z| + l_2 - 1$ implies that $\mathcal{T}_p.e \geq \mathcal{T}_1.s + |z| + l_2 - 1$. □

Lemmas 3 and 4 collectively prove that vertices associated with the reads $\mathcal{T}_1.r$, $\mathcal{T}_2.r$, $\ldots$, $\mathcal{T}_p.r, \mathcal{T}_1.r$ can be connected appropriately to build a valid closed walk in overlap graph $O_{l_2}(\mathcal{R})$. Along the walk, the non-overlapping prefix of read $\mathcal{T}_i.r$ $(1 \leq i < p)$ spells $z[\mathcal{T}_i.s : \mathcal{T}_{i+1}.s - 1]$. The prefix of the last read $\mathcal{T}_p.r$ spells $z[\mathcal{T}_p.s : |z|]$. When $\mathcal{T}_1.s > 1$, it also spells $z[1 : \mathcal{T}_1.s - 1]$. This proves Theorem 2.

3.3 String graphs

Theorem 3. *String graph $S_k(\mathcal{R})$ is not guaranteed to be coverage-preserving for any value of k.*

Proof. A counter-example suffices to support the claim. Assume $l_1 = 6, l_2 = 2$ and $\phi = 10$. Suppose count of reads $|\mathcal{R}| = 4, \mathcal{R}[1]$ =CACGTG, $\mathcal{R}[2]$ =CACGTGG, $\mathcal{R}[3]$ = TGTGCA and $\mathcal{R}[4]$ =TGGGCA (Figure 3). Accordingly, the two candidate circular strings are $\langle$CACGTGTG$\rangle$ (covered by $\mathcal{R}[1], \mathcal{R}[3]$) and $\langle$CACGTGGG$\rangle$ (covered by $\mathcal{R}[1], \mathcal{R}[2], \mathcal{R}[4]$). However, read $\mathcal{R}[1]$ is contained in read $\mathcal{R}[2]$ and the vertex corresponding to read $\mathcal{R}[1]$ is excluded from the string graph. As a result, the first candidate $\langle$CACGTGTG$\rangle$ cannot be spelled. □

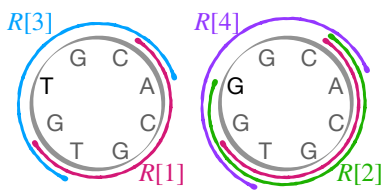

Fig. 3. A visualization of the counter-example used to support Theorem 3.

The significance of the above result is that despite sufficient sequencing coverage, string graphs may lose coverage over the genome. This limitation has been observed in recent articles [7,22]. Accordingly, the following questions need to be addressed: (a) does the lack of the above guarantee affect quality of long-read assembly in practice? and (b) does there exist an alternate sparse graph model derived from an overlap graph whose size is similar to the string graph in practice, but guaranteed to be coverage-preserving?

4 An alternative framework for sparsification of overlap graphs

String graph is a sub-graph of an overlap graph that loses the coverage-preserving guarantee after pruning selected vertices and edges. In this section, we propose techniques to compute a directed multi-graph structure which is also a sub-graph of overlap graph, and it is guaranteed to be coverage-preserving. Specifically, we propose *safe* graph sparsification rules for vertex and edge removal from overlap graph $O_k(\mathcal{R}), k \leq l_2$ which guarantee that all circular strings $\in \mathcal{C}(R, l_1, l_2, \phi)$ can be spelled in the sparse graph. Suppose $O'_k(\mathcal{R})$ equals a subgraph of $O_k(\mathcal{R})$ after performing a sequence of safe operations. Let $W_\phi(G)$ denote the set of circular strings of length $\leq \phi$ which can be spelled in graph G. From Theorem 2, we know that $\mathcal{C}(R, l_1, l_2, \phi) \subseteq W_\phi(O_k(\mathcal{R}))$. Therefore, the next two lemmas hold true.

Lemma 5. *Vertex u can be safely removed from graph $O'_k(\mathcal{R})$ if $W_\phi(O'_k(\mathcal{R})) = W_\phi(O'_k(\mathcal{R}) - u)$.*

Lemma 6. *Edge uv_l, i.e., the edge from vertex u to vertex v of weight l can be safely removed from graph $O'_k(\mathcal{R})$ if $W_\phi(O'_k(\mathcal{R})) = W_\phi(O'_k(\mathcal{R}) - uv_l)$.*

Lemma 7. *If vertex v_1 connects to v_2, v_2 connects to v_3 and v_1 connects to v_3 using edges e_1, e_2 and e_3 respectively such that $w(e_1) + w(e_2) = w(e_3)$ in graph $O'_k(\mathcal{R})$, then edge e_3 can be safely removed.*

Proof. The above property follows from the transitive sparsification step during string graph construction [21]. A circular string spelled by a closed walk using edge e_3 can be spelled by replacing e_3 with edges e_1 and e_2 instead. □

Lemma 8. *If a read $r \in \mathcal{R}$ is of length $< l_1$, then the vertex corresponding to read r can be safely removed.*

Proof. The above property is implied from the proof of Theorem 2. Only the reads of length $\geq l_1$ were required in the algorithm to build a closed walk that spell each circular string $\in \mathcal{C}(\mathcal{R}, l_1, l_2, \phi)$. □

One needs to be careful while filtering out contained reads as highlighted in the counter-example for Theorem 3. The next lemma suggests a condition for when it is safe to remove a contained read.

Lemma 9. *The vertex corresponding to a contained read $r \in \mathcal{R}$ can be safely removed from graph $O'_k(\mathcal{R})$ if all the following conditions are satisfied:*

1. *Read r is a substring of exactly one candidate circular string $z \in \mathcal{C}(\mathcal{R}, l_1, l_2, \phi)$.*
2. *Read r matches a non-repetitive substring of z.*
3. *Graph $O'_k(\mathcal{R})$ contains a vertex corresponding to a parent read of r.*

Proof. Let r_p be a parent read of read r whose corresponding vertex exists in graph $O'_k(\mathcal{R})$. Denote the vertices corresponding to reads r and r_p as u and u_p respectively. Recall that all reads are substrings of the genome, and the genome is a subset of $\mathcal{C}(\mathcal{R}, l_1, l_2, \phi)$. If r is a substring of exactly one circular string $z \in \mathcal{C}(\mathcal{R}, l_1, l_2, \phi)$, then r_p must be a substring of z and no other circular string in $\mathcal{C}(\mathcal{R}, l_1, l_2, \phi)$. We need to show that z can still be spelled after removing vertex u from graph $O'_k(\mathcal{R})$. Suppose $(v_1, e_1, v_2, e_2, \ldots, v_{n-1}, e_{n-1}, v_n)$, $n > 1, v_n = v_1$ is a closed walk that includes vertex u to spell z. WLOG, assume that $v_1 = u$. Note that neither of the vertices $v_2, v_3, \ldots, v_{n-1}$ equal u because read r matches a non-repetitive substring of z. Since $v_1 = u$, the first character of z is spelled by using the first character of r, and $z[1 : |r|]$ matches read r. Vertex $v_1 = u$ is said to span interval $\mathcal{A}_1$ of length $|r|$ starting from position one in z. Similarly, vertex v_2 spans interval $\mathcal{A}_2$ of length $|\sigma(v_2)|$ starting from position $w(e_1) + 1$ in z. In general, each vertex v_i in the closed walk spans a unique interval $\mathcal{A}_i$ with starting position $\in [1, |z|]$. Next, we will construct a new closed walk to spell z that uses vertex u_p instead of vertex u. The interval spanned by vertex u_p in z, say $\mathcal{A}_{new}$, can be uniquely identified because substring $z[1 : |r|]$ is non-repetitive. $\mathcal{A}_{new}$ subsumes $\mathcal{A}_1 = \mathcal{A}_n$ and possibly a few other adjacent intervals $\mathcal{A}_2, \mathcal{A}_3, \ldots$ and $\mathcal{A}_{n-1}, \mathcal{A}_{n-2}, \ldots$. All vertices associated with the subsumed intervals can be removed and replaced with u_p. With additional refinements, this procedure can be used to form a closed walk that spells z without using vertex u. □

It would be ideal to construct a "minimal" sparse overlap graph using the above rules. However, implementing these rules in practice is not trivial. Suppose all sequencing errors are corrected by using an appropriate error-correction algorithm, we need to estimate l_1, l_2 and ϕ parameters. An efficient algorithm to compute set of candidate circular strings $\mathcal{C}(\mathcal{R}, l_1, l_2, \phi)$ is unknown. Computing set of closed walks $W_\phi(G)$ efficiently in large overlap graphs is also challenging.

5 A proof-of-concept implementation

We build a practical algorithm for overlap graph sparsification that removes transitive edges by using the property in Lemma 7, and filters out a subset of contained reads by using heuristics that are inspired from Lemmas 5, 9. Our implementation is designed for error-free long-read sequencing data sampled from both strands of DNA. Reverse complement of a string x is denoted as $\bar{x}$. Reverse complements are handled by adding two separate vertices for each read: first for the read as given and second for its reverse complement. If vertex v corresponds to a read in either its forward or reverse-complemented orientation, then $\bar{v}$ refers to the vertex corresponding to the read in the other orientation. Therefore, $\overline{\sigma(v)} = \sigma(\bar{v})$. Graph $G(V, E, \sigma, w)$ satisfies the following two properties: (i) $\forall v \in V, \bar{v} \in V$, and (ii) For each edge $e \in E$, say from v_1 to v_2, an edge $\bar{v_1} \rightarrow \bar{v_2}$ of weight $|\sigma(v_2)| - (|\sigma(v_1)| - w(e))$ belongs to E.

Our implementation requires the following inputs from a user for the initial graph construction: (i) error-free long reads, (ii) minimum overlap length threshold, (iii) exact suffix-prefix overlaps, and (iv) exact match information of each contained read to each of its parent read. The minimum overlap length cutoff is set to 5 kbp by default. For our experiments, we used minimap2 [14] to compute all-vs-all read alignments. Overlaps available from minimap2 were filtered to satisfy the alignment length and 100% identity constraints. The first step in our implementation is to build an overlap graph from the input suffix-prefix overlaps, and separately label reads as either contained or non-contained. For each contained read r, a set of 4-tuples is used to save match information with the parent reads. For instance, tuple $(p, i, j, +)$ indicates that read r matches substring $p[i : j]$ of read $p \in \mathcal{R}$. Similarly, tuple $(p, i, j, -)$ indicates that read r matches substring $\bar{p}[i : j]$, where $\bar{p}$ is the reverse complement of read $p \in \mathcal{R}$.

5.1 Transitive sparsification of a directed multi-graph. The transitive sparsification property in Lemma 7 is inspired from the Myers's string graph formulation [21]. However, their linear $O(|E|)$ expected

time algorithm [21] to label transitive edges is not applicable here because our graph is a multi-graph, i.e., graph can have multiple edges between the same source and target vertices. We use a simple $O(|E|D^2)$ time algorithm where D is the maximum out-degree of vertices in G.

Algorithm 1: Labeling of transitive edges

```
Input: Graph G(V, E, σ, w)
for e1 = v1 → v2 ∈ E do
    for e2 = v1 → v3 ∈ E do
        for e3 = v3 → v4 ∈ E do
            if v4 = v2 and w(e1) = w(e2) + w(e3) then
                Mark e1 as transitive edge
```

5.2 Remove non-repetitive haplotype-specific contained reads. Lemma 9 suggests that a contained read can be removed if (i) it is a substring of only a single haplotype, (ii) it is sampled from a non-repetitive genomic region, and (iii) at least one parent read is available in the graph. The third condition is automatically satisfied because there is at least one parent read of each read which is non-contained, and therefore, will not be removed. We propose a heuristic to check the first two conditions from all-versus-all read alignments computed using Hifiasm. For each read r, we inspect multiple sequence alignment (MSA) of read r and other reads overlapping with read r. We check if the count of reads in the MSA does not greatly exceed the sequencing coverage to ensure that the read is sampled from a non-repetitive genomic region. For a diploid genome, we additionally check for the presence of a heterozygous variant in the MSA (Supplementary Figure 2). If both the conditions are satisfied, we remove the two vertices corresponding to read r and its reverse complement respectively.

5.3 k-mer-based filtering heuristic for contained reads. We process the remaining contained reads using a second filter. In theory, we can try enumerating all possible string walks with and without contained read. Comparing the two will inform whether the read can be safely removed or not (Lemma 5). However, enumerating all possible string walks at chromosome length scale may not be computationally tractable in large graphs. We address this by implementing a k-mer-based heuristic. We first compute the set of k-mers observed in bounded-length string walks from the vertex associated with contained read (Supplementary Figure 1). The set is denoted as κ_1. Next, we compute the union of set of k-mers observed in bounded-length string walks from the vertices associated with the parent reads of the contained read. This set is denoted as κ_2. If $\kappa_1 \subseteq \kappa_2$, we mark the contained read for removal. This heuristic estimates whether there exists a string which cannot be spelled after removal of the contained read.

6 Experimental results

Our evaluation addresses two questions: (a) How many coverage gaps are introduced if we follow the standard string graph model, i.e., discard all contained reads during graph construction?, and (b) How well does the proposed implementation (hereon referred to as ContainX) perform when compared to string graph as well as other existing methods?

6.1 Experimental setup

Read simulation from human haploid and diploid genomes. We used two human genome assemblies for read simulation. The first is a CHM13 draft haploid genome assembly (v2.0) provided by the T2T consortium (GenBank id GCA_009914755.4) [22]. This haploid genome assembly of size 3.1 Gbp has 25 contigs and N50 length 50.6 Mbp. The second assembly is a long-read diploid genome assembly[1] of HG002

[1] ftp://ftp.dfci.harvard.edu/pub/hli/hifiasm-phase/v2/HG002.hifiasm.trio.0.16.1.hap1.fa.gz
ftp://ftp.dfci.harvard.edu/pub/hli/hifiasm-phase/v2/HG002.hifiasm.trio.0.16.1.hap2.fa.gz

human sample computed using Trio-hifiasm [3]. This diploid genome assembly of size 6.0 Gbp has 970 contigs and N50 length 57.8 Mbp. From these genome assemblies, we simulated eight *error-free* read sets whose length distributions are compatible with either real PacBio HiFi or ONT sequencing data. Seqrequester (https://github.com/marbl/seqrequester) is used to simulate reads from random start positions in both forward and reverse orientations. Seqrequester allows users to specify a desired read length distribution. We used HiFi and ONT read length distributions from publicly-available long-read datasets of HG02080 human sample [16]. Four long-read sets were simulated from haploid genome assembly with 20-fold coverage. The other four long-read sets were simulated from diploid genome assembly with 30-fold coverage (15-fold per haplotype). Length statistics and coverage information of the simulated read sets are shown in Table 1. The commands used to run the tools are listed in Supplementary Table 1.

Id	Count of reads	Total bases (Gbp)	N50 length	Minimum read length	Maximum read length	No. of genome intervals with zero coverage
HAPLOID-20x-ONT-1	3,697,647	62.3	40,098	3	572,359	0
HAPLOID-20x-ONT-2	3,699,867	62.3	40,089	3	544,992	0
HAPLOID-20x-HiFi-1	2,913,761	62.3	21,314	9	48,708	0
HAPLOID-20x-HiFi-2	2,914,174	62.3	21,310	45	48,708	0
DIPLOID-30x-ONT-1	5,348,694	89.5	39,715	1	544,992	3
DIPLOID-30x-ONT-2	5,350,643	89.5	39,732	2	572,359	5
DIPLOID-30x-HiFi-1	4,199,478	89.5	21,295	1	48,708	2
DIPLOID-30x-HiFi-2	4,199,129	89.5	21,298	1	48,708	2

Table 1. Properties of the simulated long-read sequencing datasets including their size and length statistics.

6.2 Assessment of coverage gaps caused by removing contained reads

Benchmarking procedure. We developed a method to estimate coverage gaps introduced by removal of contained reads. We computed all-versus-all read overlaps by using minimap2 (v2.23) [14] to identify the set of contained reads. A read is labeled as contained if it matched a proper substring of a longer read. We can have false negatives because minimap2 uses k-mer-based heuristics. However, there cannot be false positives here because each overlap is verified using sequence alignment. In the second step, the set of non-contained reads was mapped to the genome. For the diploid genome, the set of non-contained reads was mapped to both paternal and maternal haplotypes, one by one. Minimap2 parameters were adjusted to enable reporting of multiple best alignments per read. Whenever a read aligned end-to-end against a genomic interval with 100% alignment identity, we recorded the interval. Finally, all the genomic intervals where no alignment coverage was observed were extracted using bedtools (v2.29.1). Not all of these can be considered as coverage gaps caused by removal of contained reads. The sequencing coverage and read lengths at the two extreme ends of each contig are expected to be lower than the whole-genome average due to edge effect. Therefore, the intervals that overlap with either the first 25 kbp or the last 25 kbp bases of a contig are not considered. Similarly, the intervals that overlap with segments of the genome with zero sequencing coverage are also not considered. The remaining intervals are the coverage gaps introduced by removal of contained reads.

Results. The count of coverage gaps observed in the four datasets are shown in Table 2. We observe that there are zero coverage gaps introduced in haploid genome for both ONT and HiFi read sets. In theory, haploid genomes can suffer coverage gaps due to identical repeats in the genome. However, removal of contained reads does not cause any harm in practice. In diploid genomes, however, we observe 1 and $46 - 54$ coverage gaps introduced in HiFi and ONT datasets respectively. Compared to a haploid genome, coverage gaps in diploid genomes are more likely to happen because a longer read sampled from one haplotype can subsume all reads sampled from the homologous loci of the other haplotype, especially in regions with low heterozygosity. Moreover, ONT read length distribution is highly non-uniform which leads to a higher fraction of contained

reads, and therefore, more coverage gaps. See Supplementary Figure 3 for long-read length distributions. We further investigated whether the observed gaps are clustered in a particular chromosome, but we do not observe such behavior. The coordinates of these coverage gaps in the HG002 assembly were mapped to the coordinates of GRCh38 human genome reference by using paftools [14]. The complete lists of these coordinates are provided in Supplementary Tables 2-5. We also evaluated heterozygosity rate by checking the count of heterozygous variants in these gaps. These gaps collectively span 1.26 million bases on GRCh38 genome reference when all the four diploid datasets are considered together. However, Dipcall [15] reported only 83 heterozygous variants which is an order of magnitude lower relative to the HG002 whole-genome average rate 0.1%. This observation confirms our expectation that the coverage gaps are more likely to happen in regions of low heterozygosity.

Id	Count of contained reads	Coverage gap statistics		
		Count	Mean length	Maximum length
HAPLOID-20x-ONT-1	3,160,926	0	-	-
HAPLOID-20x-ONT-2	3,161,210	0	-	-
HAPLOID-20x-HiFi-1	1,897,005	0	-	-
HAPLOID-20x-HiFi-2	1,895,182	0	-	-
DIPLOID-30x-ONT-1	4,550,993	46	11,227	53,490
DIPLOID-30x-ONT-2	4,552,757	54	14,369	101,371
DIPLOID-30x-HiFi-1	2,506,248	1	1,808	1,808
DIPLOID-30x-HiFi-2	2,503,577	1	224	224

Table 2. Coverage gap statistics for the gaps introduced by removal of contained reads. Count of contained reads is estimated from all-to-all read alignments computed by using minimap2.

6.3 Evaluation of the proposed overlap graph sparsification algorithm

Benchmarking procedure. We evaluate the performance of our proof-of-concept implementation ContainX that determines a subset of contained reads to be retained. We tested ContainX using diploid datasets. We checked the performance by observing the following two parameters: (a) Count of coverage gaps caused by removing contained reads that are estimated to be redundant, and (b) count of "junction" reads, i.e., the reads which correspond to vertices having either > 1 incoming or > 1 outgoing edges in the graph. Retaining more contained reads implies that their corresponding vertices and edges are retained in the graph. This may result in more number of junction reads which can further result in shorter unitigs. It is important to look at the two parameters simultaneously because the first parameter can be optimized by removing all contained reads whereas the second parameter can be optimized by retaining all contained reads. An ideal solution should optimize both by doing a careful selection of contained reads. To measure the first parameter, we used minimap2 to compute end-to-end 100% identical read alignments of the retained contained reads against the two haplotypes. We checked if these alignments closed the coverage gaps that were previously caused by discarding all contained reads.

Competing methods. We compared ContainX with four other methods. The first method, called as *Retain-all*, simply retains all contained reads. The second method, called as *Remove-all*, discards all contained reads. The third method is from Hui *et al.* [9]. It removes a contained read r if two of its parent reads are *inconsistent*. Two parent reads are said to be inconsistent if, when aligned with respect to read r, they disagree at some base. We also included Hifiasm [3] in our benchmark. Unlike previous methods which stop at graph construction, Hifiasm is an full-fledged genome assembler that incorporates multiple heuristics (e.g., tip removal, bubble popping etc.) for graph sparsification. Hifiasm builds its initial overlap graph without using contained reads. It uses a heuristic to retain a few selected contained reads which can connect ends of two unitigs.

Dataset	Method	Count of retained contained reads	Count of junction reads	Count of coverage gaps
DIPLOID-30x-ONT-1	Retain-all	2,762,064	2,540,591	0
	Hui-2016	2,489,204	2,281,451	0
	Remove-all	0	38,882	46
	ContainX	28,470	53,855	2
	Hifiasm	4,043	1,690	33
DIPLOID-30x-ONT-2	Retain-all	2,761,180	2,542,829	0
	Hui-2016	2,487,524	2,282,955	0
	Remove-all	0	38,472	54
	ContainX	28,388	53,823	5
	Hifiasm	3,684	1,656	39
DIPLOID-30x-HiFi-1	Retain-all	2,498,380	3,387,941	0
	Hui-2016	2,455,406	3,321,695	0
	Remove-all	0	158,428	1
	ContainX	39,766	184,071	0
	Hifiasm	164	36,866	0
DIPLOID-30x-HiFi-2	Retain-all	2,495,781	3,385,499	0
	Hui-2016	2,452,456	3,319,128	0
	Remove-all	0	158,513	1
	ContainX	39,951	184,622	0
	Hifiasm	149	37,235	1

Table 3. Performance comparison of ContainX with four other methods using long-read diploid datasets.

Results. We start with the evaluation of the first four methods; Hifiasm is discussed separately. The reads with no suffix-prefix overlap (above user-specified length) are excluded from the graph by default in all the four methods. In all four datasets, Retain-all method retained the highest number of contained reads as expected (Table 3). It resolved all coverage gaps, but the corresponding graph has significantly higher number of junction reads compared to Remove-all. Next, Hui-2016 filtering method appears to be conservative, i.e., it typically ends up retaining a majority of contained reads. In several cases, the contained read may have parent reads which agree at all bases (e.g., when they come from homozygous region of the genome). There may also be cases when there is only a single parent read available. In all such scenarios, Hui-2016 heuristic will choose to retain the contained reads. This heuristic resolved all the coverage gaps. Remove-all method has the fewest junction reads among the four methods but has the highest number of unresolved coverage gaps as expected. ContainX delivered favorable performance compared to the three methods. Using both datasets, ContainX retained $1-2\%$ contained reads compared to Retain-all, yet it successfully resolved majority of the coverage gaps. ContainX resulted in only $2-5\%$ junction reads when compared to Retain-all. This suggests that ContainX heuristics are promising in terms of improving assembly quality by avoiding coverage gaps. Finally, Hifiasm retained fewer contained reads than ContainX but it failed to resolve a majority of coverage gaps. This suggests that there is further scope to improve Hifiasm algorithm. The unitig graph of Hifiasm has the least number of junction reads because it does additional graph pruning which is necessary for computing longer unitigs. Incorporating ContainX heuristics inside Hifiasm code can be an interesting direction to explore in future.

The proposed graph sparsification algorithms implemented in ContainX are easy-to-implement, fast and space-efficient. The k-mer-based heuristic uses multiple threads by considering each contained read independently. The time required by the ContainX heuristics using a multi-core AMD processor is about ten minutes. This is insignificant compared to the main time-consuming step during long-read assembly, i.e., computing all-vs-all read alignments.

Acknowledgements

We thank Haowen Zhang, Mile Sikic and Robert Vaser for useful feedback. Haowen suggested the use of hifiasm code for one of the proposed heuristics. We thank Haoyu Cheng for helping us to modify hifiasm code. Brian Walenz helped to resolve an issue in Seqrequester code for long-read simulation. We also thank Sergey Nurk for highlighting this research problem during team discussions as part of the T2T consortium. This research is supported in part by funding from the Indian Institute of Science and allocation of U.S. National Energy Research Scientific Computing Center (NERSC) computing resources.

References

1. Bankevich, A., Bzikadze, A.V., Kolmogorov, M., Antipov, D., Pevzner, P.A.: Multiplex de bruijn graphs enable genome assembly from long, high-fidelity reads. Nature Biotechnology pp. 1–7 (2022)
2. Bresler, G., Bresler, M., Tse, D.: Optimal assembly for high throughput shotgun sequencing. In: BMC bioinformatics. vol. 14, pp. 1–13. BioMed Central (2013)
3. Cheng, H., Concepcion, G.T., Feng, X., Zhang, H., Li, H.: Haplotype-resolved de novo assembly using phased assembly graphs with hifiasm. Nature methods **18**(2), 170–175 (2021)
4. Cheng, H., Jarvis, E.D., Fedrigo, O., Koepfli, K.P., Urban, L., Gemmell, N.J., Li, H.: Robust haplotype-resolved assembly of diploid individuals without parental data. arXiv preprint arXiv:2109.04785 (2021)
5. Chikhi, R., Rizk, G.: Space-efficient and exact de bruijn graph representation based on a bloom filter. Algorithms for Molecular Biology **8**(1), 1–9 (2013)
6. Chin, C.S., Peluso, P., Sedlazeck, F.J., Nattestad, M., Concepcion, G.T., Clum, A., Dunn, C., O'Malley, R., Figueroa-Balderas, R., Morales-Cruz, A., et al.: Phased diploid genome assembly with single-molecule real-time sequencing. Nature methods **13**(12), 1050–1054 (2016)
7. Feng, X., Cheng, H., Portik, D., Li, H.: Metagenome assembly of high-fidelity long reads with hifiasm-meta. arXiv preprint arXiv:2110.08457 (2021)
8. Garg, S., Fungtammasan, A., Carroll, A., Chou, M., Schmitt, A., Zhou, X., Mac, S., Peluso, P., Hatas, E., Ghurye, J., et al.: Chromosome-scale, haplotype-resolved assembly of human genomes. Nature biotechnology **39**(3), 309–312 (2021)
9. Hui, J., Shomorony, I., Ramchandran, K., Courtade, T.A.: Overlap-based genome assembly from variable-length reads. In: 2016 IEEE International Symposium on Information Theory (ISIT). pp. 1018–1022. IEEE (2016)
10. Jarvis, E.D., Formenti, G., Rhie, A., et al.: Automated assembly of high-quality diploid human reference genomes. bioRxiv (2022). https://doi.org/10.1101/2022.03.06.483034, `https://www.biorxiv.org/content/early/2022/03/06/2022.03.06.483034`
11. Jung, H., Winefield, C., Bombarely, A., Prentis, P., Waterhouse, P.: Tools and strategies for long-read sequencing and de novo assembly of plant genomes. Trends in plant science **24**(8), 700–724 (2019)
12. Koren, S., Walenz, B.P., Berlin, K., Miller, J.R., Bergman, N.H., Phillippy, A.M.: Canu: scalable and accurate long-read assembly via adaptive k-mer weighting and repeat separation. Genome research **27**(5), 722–736 (2017)
13. Li, H.: Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences. Bioinformatics **32**(14), 2103–2110 (2016)
14. Li, H.: Minimap2: pairwise alignment for nucleotide sequences. Bioinformatics **34**(18), 3094–3100 (2018)
15. Li, H., Bloom, J.M., Farjoun, Y., Fleharty, M., Gauthier, L., Neale, B., MacArthur, D.: A synthetic-diploid benchmark for accurate variant-calling evaluation. Nature methods **15**(8), 595–597 (2018)
16. Liao, W.W., Asri, M., Ebler, J., Doerr, D., Haukness, M., Hickey, G., Lu, S., Lucas, J.K., Monlong, J., Abel, H.J., et al.: A draft human pangenome reference. bioRxiv (2022)
17. Logsdon, G.A., Vollger, M.R., Eichler, E.E.: Long-read human genome sequencing and its applications. Nature Reviews Genetics **21**(10), 597–614 (2020)
18. Marx, V.: Long road to long-read assembly. Nature Methods **18**(2), 125–129 (2021)
19. Medvedev, P., Pop, M.: What do eulerian and hamiltonian cycles have to do with genome assembly? PLoS Computational Biology **17**(5), e1008928 (2021)
20. Miller, J.R., Delcher, A.L., Koren, S., Venter, E., Walenz, B.P., Brownley, A., Johnson, J., Li, K., Mobarry, C., Sutton, G.: Aggressive assembly of pyrosequencing reads with mates. Bioinformatics **24**(24), 2818–2824 (2008)
21. Myers, E.W.: The fragment assembly string graph. Bioinformatics **21**(suppl_2), ii79–ii85 (2005)
22. Nurk, S., Koren, S., Rhie, A., Rautiainen, M., Bzikadze, A.V., Mikheenko, A., Vollger, M.R., Altemose, N., Uralsky, L., Gershman, A., et al.: The complete sequence of a human genome. Science **376**(6588), 44–53 (2022)

23. Nurk, S., Walenz, B.P., Rhie, A., Vollger, M.R., Logsdon, G.A., Grothe, R., Miga, K.H., Eichler, E.E., Phillippy, A.M., Koren, S.: Hicanu: accurate assembly of segmental duplications, satellites, and allelic variants from high-fidelity long reads. Genome research **30**(9), 1291–1305 (2020)
24. Porubsky, D., Ebert, P., Audano, P.A., Vollger, M.R., Harvey, W.T., Marijon, P., Ebler, J., Munson, K.M., Sorensen, M., Sulovari, A., et al.: Fully phased human genome assembly without parental data using single-cell strand sequencing and long reads. Nature biotechnology **39**(3), 302–308 (2021)
25. Sereika, M., Kirkegaard, R.H., Karst, S.M., Michaelsen, T.Y., Sørensen, E.A., Wollenberg, R.D., Albertsen, M.: Oxford Nanopore R10.4 long-read sequencing enables the generation of near-finished bacterial genomes from pure cultures and metagenomes without short-read or reference polishing. Nature Methods (2022). https://doi.org/10.1038/s41592-022-01539-7, https://doi.org/10.1038/s41592-022-01539-7
26. Shafin, K., Pesout, T., Lorig-Roach, R., Haukness, M., Olsen, H.E., Bosworth, C., Armstrong, J., Tigyi, K., Maurer, N., Koren, S., et al.: Nanopore sequencing and the shasta toolkit enable efficient de novo assembly of eleven human genomes. Nature biotechnology **38**(9), 1044–1053 (2020)
27. Shomorony, I., Kim, S.H., Courtade, T.A., Tse, D.N.: Information-optimal genome assembly via sparse read-overlap graphs. Bioinformatics **32**(17), i494–i502 (2016)
28. Vaser, R., Šikić, M.: Time-and memory-efficient genome assembly with raven. Nature Computational Science **1**(5), 332–336 (2021)
29. Wenger, A.M., Peluso, P., Rowell, W.J., Chang, P.C., Hall, R.J., Concepcion, G.T., Ebler, J., Fungtammasan, A., Kolesnikov, A., Olson, N.D., et al.: Accurate circular consensus long-read sequencing improves variant detection and assembly of a human genome. Nature biotechnology **37**(10), 1155–1162 (2019)
30. Wick, R.R., Judd, L.M., Cerdeira, L.T., Hawkey, J., Méric, G., Vezina, B., Wyres, K.L., Holt, K.E.: Trycycler: consensus long-read assemblies for bacterial genomes. Genome biology **22**(1), 1–17 (2021)
31. Yang, X., Zhao, X., Qu, S., Jia, P., Wang, B., Gao, S., Xu, T., Zhang, W., Huang, J., Ye, K.: Haplotype-resolved chinese male genome assembly based on high-fidelity sequencing. Fundamental Research (2022)
32. Zimin, A.V., Puiu, D., Luo, M.C., Zhu, T., Koren, S., Marçais, G., Yorke, J.A., Dvořák, J., Salzberg, S.L.: Hybrid assembly of the large and highly repetitive genome of aegilops tauschii, a progenitor of bread wheat, with the masurca mega-reads algorithm. Genome research **27**(5), 787–792 (2017)

Supplementary Information

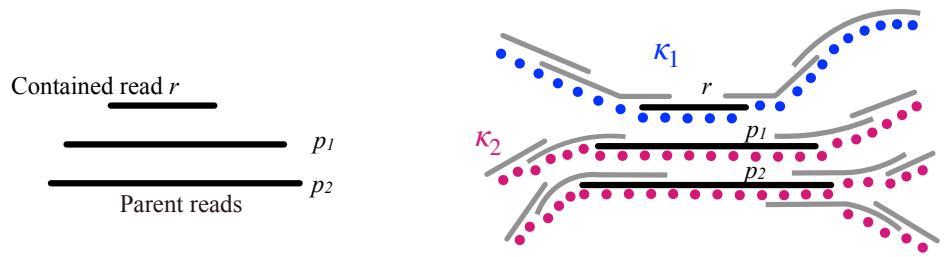

Fig. 1. An illustration for how k-mer sets κ_1 and κ_2 are computed using bounded-length string walks from contained read r and its parent reads, respectively, in an overlap graph. Colored dots indicate the k-mers extracted from the strings.

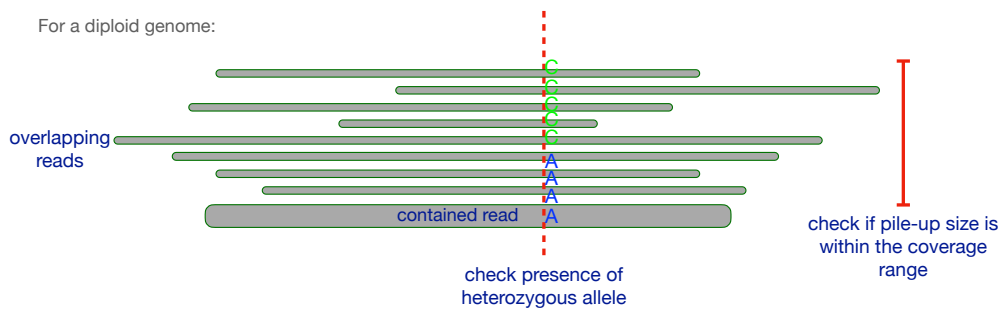

Fig. 2. An illustration suggesting how multiple sequence alignment is used to detect non-repetitive haplotype-specific contained reads. In our implementation, the set of non-repetitive haplotype-specific reads is precomputed by using modified hifiasm code (https://github.com/cjain7/hifiasm/tree/hifiasm_dev_debug)

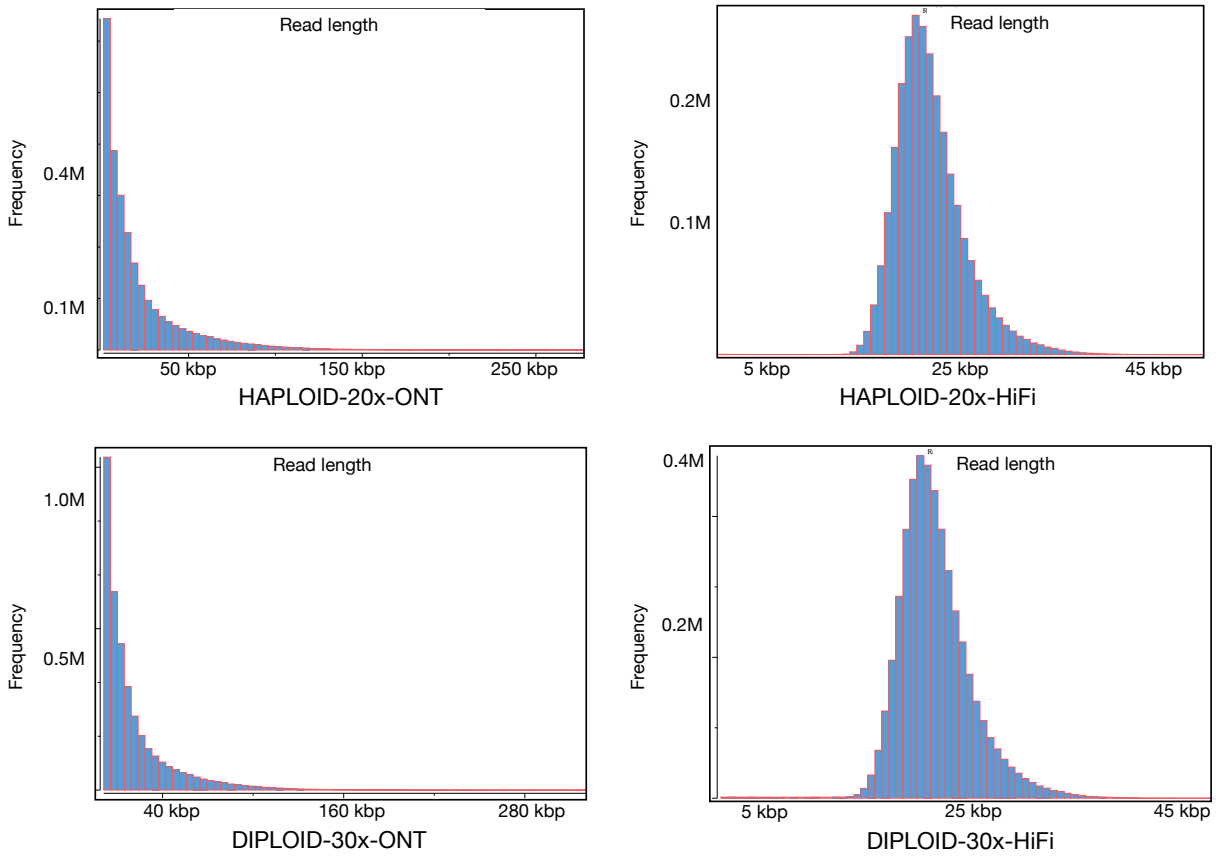

Fig. 3. Histogram of the number of reads with each read length. Reads of length < 1000 bp are not considered. The above plots show that read length distribution of ONT datasets is highly skewed with a large number of reads of short length. HiFi read lengths are within a narrow range compared to ONT. These are known characteristics of the two sequencing technologies.

Tool	**Purpose**	**Commands**
seqrequester (commit:1ee6fd9)	Read simulation	`seqrequester simulate -truncate -genome HG002.hap1.fa -genomesize 2935689000 -coverage 15 -distribution histogram.txt`
minimap2 (v2.23)	All-vs-all read alignments	`minimap2 -t 32 -w 101 -k 27 -g 500 -B 8 -O 8,48 -E 4,2 -cx ava-ont reads.fasta reads.fasta > overlaps.paf`
	Read mapping	`-t 32 -N 50 -cx map-ont reference.fasta reads.fasta > mm2.paf`
ContainX (commit:23e1feb)	Run ContainX	`ContainX -p hetreads.txt -d dump.gfa -t 32 -n usefulContainedReads.txt reads.fasta overlaps.paf`
	Run Retain-all	`ContainX -I 0 -d dump.gfa -t 32 -n usefulContainedReads.txt reads.fasta overlaps.paf`
	Run Remove-all	`ContainX -I 0 -c -d dump.gfa -t 32 -n usefulContainedReads.txt reads.fasta overlaps.paf`
	Run Hui-2016	`ContainX -h -d dump.gfa -t 32 -n usefulContainedReads.txt reads.fasta overlaps.paf`
hifiasm (v0.16.1)	Test hifiasm	`hifiasm -o output --write-paf -t 32 reads.fasta`
Modified hifiasm (commit:59bcfd5)	For ContainX heuristic	`hifiasm --dbg-het-cnt -o output --write-paf -t 32 reads.fasta`
Purpose		**Commands**

Fetch contained read IDs from all-vs-all read alignments		`cat overlaps.paf | awk '{if ($3 == 0 && $2 == $4 && $2 < $7 && $10 == $11) print $0}'> tmp`
		`cat overlaps.paf | awk '{if ($8 == 0 && $7 == $9 && $7 < $2 && $10 == $11) print $0}'>> tmp`
		`cat tmp | sort | uniq > containedIDs.txt`

Fetch contained read IDs used in hifiasm unitig graph		`grep -P "^A\t" output.bp.r_utg.gfa | grep -o -P "read=[^\t]*" | sort > hifiasm.readIDs.txt` `comm -12 hifiasm.readIDs.txt containedIDs.txt > hifiasm.containedreadIDs.txt`
Fetch count of junction reads in hifiasm unitig graph		`grep "^L" output.bp.r_utg.gfa | cut -f2 | sort | uniq -c | grep -v "1 " | wc -l`
Fetch haplotype-specific non-repetitive read IDs using modified hifiasm code		`cat output.het_cnt.log | tr -d ">" | awk '{if ($2 > 0) {print $1}}'> selectedReadIDs.txt`

Table 1. Some important commands and tools that were used in this study. Bash script files are also provided along with containX code.

HG002 genome assembly			GRCh38 genome reference		
Contig id	Start	End	Chromosome id	Start	End
h1tg000009l	110900070	110905875	chr1	117037725	117043529
h2tg000022l	15152870	15160560	chr1	213447093	213454782
h1tg000068l	2866076	2870654	chr1	231372139	231376767
h2tg000043l	14918403	14921887	chr2	73264698	73268182
h2tg000043l	2156839	2174983	chr2	86011544	86029687
h2tg000023l	7722607	7738066	chr3	7727629	7743090
h2tg000010l	82369634	82395207	chr3	114613975	114639545
h2tg000034l	15546170	15547725	chr4	24679008	24680563
h2tg000014l	98646906	98657209	chr4	84170623	84180925
h2tg000014l	21807903	21814254	chr4	161020948	161027327
h2tg000017l	10313522	10314556	chr5	164430844	164431883
h1tg000006l	20492323	20506277	chr6	20516506	20530473
h1tg000036l	62513086	62523058	chr6	95294581	95304553
h1tg000036l	24475930	24479551	chr6	133346301	133349923
h1tg000036l	21309213	21329934	chr6	136496415	136517130
h1tg000036l	8446464	8499954	chr6	149308719	149362388
h2tg000049l	9904947	9912852	chr7	24201066	24208971
h1tg000010l	26527715	26546509	chr7	27410220	27429019
h1tg000067l	14144866	14160305	chr7	86898659	86914094
h1tg000023l	40382649	40397054	chr7	141372589	141386990
h1tg000003l	74679982	74689676	chr8	76860952	76870646
h2tg000011l	53131095	53153351	chr8	97022436	97044663
h1tg000021l	31311351	31311574	chr8	109880707	109880930
h2tg000001l	20933873	20947186	chr10	63801174	63811239
h2tg000001l	20933873	20947186	chr10	63811868	63813558
h1tg000007l	39009214	39010750	chr10	77733511	77735047
h2tg000001l	53657444	53660889	chr10	96441169	96444614
h1tg000032l	26710243	26711572	chr11	108418486	108419815
h1tg000032l	24316455	24321327	chr11	110809522	110814394
h1tg000026l	81896783	81946294	chr12	41557194	41606710
h1tg000026l	69278071	69293106	chr12	54217901	54232937
h2tg000004l	17612446	17614118	chr12	54572215	54573889
h1tg000026l	39760070	39761374	chr12	83726976	83728279
h1tg000026l	38832150	38833350	chr12	84657866	84659066
h1tg000017l	10534178	10543269	chr14	30228290	30237381
h2tg000003l	81495165	81500085	chr14	100294778	100299697
h1tg000017l	83021339	83027909	chr14	102667673	102674243
h2tg000002l	49073036	49081755	chr15	28042576	28051295
h2tg000002l	48832864	48844715	chr15	28279362	28291210
h1tg000005l	46151572	46153525	chr15	30938120	30940068
h2tg000002l	15585368	15615044	chr15	61449613	61479284
h1tg000123l	5886845	5887779	chr16	9342670	9343604
h2tg000057l	199241	206544	chr17	6652296	6659599
h1tg000027l	14645437	14651469	chr20	14625327	14631355
h2tg000059l	15875485	15884831	chr21	28368387	28377734
h1tg000064l	15871721	15877214	chr22	34851127	34856633

Table 2. Start–stop coordinates of coverage gaps caused by removing contained reads in DIPLOID-30x-ONT-1 dataset. The coordinates in HG002 assembly are also translated (i.e., lifted over) to GRCh38 genome reference.

HG002 genome assembly			GRCh38 genome reference		
Contig id	Start	End	Chromosome id	Start	End
h1tg000009l	18493205	18504097	chr1	24689133	24700023
h1tg000009l	86945009	86955138	chr1	93087776	93097904
h1tg000009l	103508009	103514037	chr1	109640847	109646875
h1tg000012l	64787435	64888806	chr1	214650325	214751706
h1tg000081l	4756106	4770890	chr2	73204290	73219075
h1tg000081l	11019519	11024843	chr2	79484322	79489661
h2tg000025l	52097255	52097889	chr2	212663821	212664455
h1tg000001l	92817961	92818455	chr2	223349132	223349626
h1tg000004l	8873984	8877229	chr3	8874669	8877914
h2tg000184l	232777	248699	chr3	74871732	74887654
h2tg000034l	4797508	4823424	chr4	13932709	13958691
h1tg000042l	114230605	114234253	chr4	73904437	73908085
h1tg000042l	104113807	104122161	chr4	84022694	84031047
h2tg000014l	92028013	92033684	chr4	90811125	90816796
h2tg000014l	82459823	82460036	chr4	100390877	100391090
h2tg000014l	70702729	70707859	chr4	112148705	112153851
h1tg000042l	15091152	15118100	chr4	173050753	173077668
h1tg000016l	91734766	91742004	chr5	83043812	83051050
h2tg000017l	90073347	90073761	chr5	84683858	84684272
h1tg000006l	20609242	20654818	chr6	20633454	20679068
h2tg000018l	25740048	25747436	chr6	33724209	33731597
h1tg000006l	50115047	50118074	chr6	50067746	50070778
h2tg000021l	5153975	5157462	chr6	64965355	64968838
h1tg000036l	90457178	90459095	chr6	67337810	67339727
h1tg000010l	20186319	20234628	chr7	21071775	21120082
h2tg000049l	12785813	12787547	chr7	21326099	21327801
h2tg000049l	9850119	9868340	chr7	24245575	24263817
h1tg000145l	1110399	1110479	chr7	67358338	67358418
h2tg000006l	26473188	26544931	chr7	132862940	132934640
h2tg000006l	26231773	26253441	chr7	133155335	133177002
h1tg000023l	40495078	40505896	chr7	141485319	141496136
h1tg000003l	46208131	46209187	chr8	48451474	48452530
h1tg000003l	64446587	64457392	chr8	66634485	66645289
h2tg000011l	22611658	22627754	chr8	67277359	67293455
h1tg000059l	13139707	13143077	chr9	28243394	28246761
h2tg000041l	37246637	37249901	chr9	37218351	37221615
h1tg000007l	11801161	11811351	chr10	104918759	104928949
h2tg000038l	50521189	50532240	chr11	39142190	39153262
h1tg000011l	44597468	44597862	chr11	46713582	46713976
h2tg000007l	24589917	24627125	chr11	110494690	110531902
h2tg000037l	1789315	1816591	chr13	19277044	19304307
h2tg000037l	12969569	12969897	chr13	30439954	30440282
h2tg000037l	19506191	19509196	chr13	36983851	36986858
h1tg000022l	42499978	42505754	chr13	44248785	44254562
h2tg000037l	30583828	30599471	chr13	48052070	48067716
h2tg000037l	33124120	33126064	chr13	50591608	50593552
h1tg000033l	20360199	20363658	chr13	107116486	107119945
h2tg000003l	26788478	26790995	chr14	45599428	45601940
h2tg000002l	2549333	2575965	chr15	74425030	74451662
h1tg000005l	1570767	1570875	chr15	75451659	75451767
h1tg000069l	22949634	22972649	chr16	78611705	78634697
h1tg000030l	14192878	14193075	chr17	3288793	3288990
h2tg000224l	30443	72254	chr19	43484700	43525330
h1tg000060l	29858829	29903277	chr21	14516790	14561239

Table 3. Start–stop coordinates of coverage gaps caused by removing contained reads in DIPLOID-30x-ONT-2 dataset.

HG002 genome assembly			GRCh38 genome reference		
Contig id	Start	End	Chromosome id	Start	End
h1tg000036l	57223158	57224966	chr6	100588432	100590240

Table 4. Start–stop coordinates of coverage gaps caused by removing contained reads in DIPLOID-30x-HiFi-1 dataset.

HG002 genome assembly			GRCh38 genome reference		
Contig id	Start	End	Chromosome id	Start	End
h2tg000002l	32863923	32864147	chr15	44197800	44198024

Table 5. Start–stop coordinates of coverage gaps caused by removing contained reads in DIPLOID-30x-HiFi-2 dataset.