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 prove that the standard string graph model lacks this guarantee. The latter result is consistent with the observation made in [Hui et al., 2016; Nurk et al., 2021] 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 using simulated long reads from HG002 human diploid genome, we find that 98 and 5 coverage gaps are introduced by ignoring contained reads from nanopore and HiFi long-read datasets respectively. We test 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 all coverage gaps. This compares favorably to other known solutions for this problem.

1 Introduction

Long-read sequencing technologies have shown the greatest promise in achieving high-quality haploid and diploid genome assemblies [1,3,8,11,21,23,27,28]. Long-read accuracy and lengths have improved enormously over the last few years. HiFi sequencing technology developed by Pacific Biosciences (PacBio) yields consensus sequencing reads that are both long (averaging 10-25 kb) and highly accurate (averaging 99.8%) [26]. Modal raw read accuracy above 99% was recently reported using reads from Oxford Nanopore Technology (ONT) [22]. Reads with N50 length above 50 kbp can be obtained using ultra-long sequencing protocol from ONT [10]. Long and accurate reads are useful to sequence traditionally challenging genomes and facilitate *de novo* assembly [14,15]. The combination of PacBio and ONT sequencing technologies has recently enabled the first complete gapless assembly of a human genome [19].

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 are used to 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 [16]. Instead, assemblers compute contigs which are long, contiguous segments which can unambiguously be inferred to be part of the genome.

Second, the presence of repeats in the sequenced genome creates many spurious edges in the overlap graph. This challenge is often addressed using graph sparsification heuristics that exclude vertices

and edges which are likely to be redundant or false. The *string graph* model [18], 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 [17] selects one *best* overlap per read end, where best is defined using some criterion. In spite of the use of such graph sparsification techniques in most long-read assembly tools [3,4,6,7,12,13,20,25], theoretical understanding of these heuristics is fairly limited.

Assuming 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 [2,24], but to check whether their information is preserved in the graph. We evaluate the coverage-preserving property in three commonly-used assembly graph 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. A visual example where "coverage gaps" are introduced in a string graph was first shown by Hui *et al.* [9].

Overlap graph is guaranteed to be coverage-preserving, but a majority of edges and vertices in it are either redundant or spurious in practice. Therefore, graph sparsification is crucial to prune the overlap graph, which helps to compute longer contigs. We develop theoretical results to compute a sparse overlap graph while preserving the coverage-preserving property. We next extend the proposed theory into practical heuristics and a prototype software ContainX (https://github.com/at-cg/ContainX). ContainX is designed to identify redundant contained reads, i.e., those contained reads which can be excluded from overlap graph without violating the coverage-preserving property. We conduct experiments using error-free long read datasets simulated from diploid HG002 human genome assembly whose read length distributions match real ONT and PacBio HiFi data. We observe that 98 and 5 coverage gaps are introduced by removing all contained reads in ONT and HiFi datasets respectively. ContainX algorithm retains $1-2\%$ contained reads within the graph and results in zero coverage gaps. We compare this performance against a competing heuristic implemented in a widely used assembler hifiasm [3]. We observe that hifiasm retains significantly fewer contained reads ($0.006-0.2\%$) in its graph compared to ContainX but results in 77 and 3 coverage gaps in the two datasets respectively. The results indicate that proposed heuristics can help improve assembly quality by preserving essential information within the graph.

This paper is organized as follows. Section 2 discusses notations and definitions. Section 3 includes proofs for coverage-preserving properties in de Bruijn graphs, overlap graphs and string graphs. The next section includes our theoretical results for safe sparsification techniques. Subsequently in Section 5, we propose practical heuristics. The final section includes our experiments using long-read datasets simulated from human genomes.

2 Preliminaries

Notation on strings. For a linear string $x = a_1 \dots a_n$ over alphabet $\Sigma = \{\text{A,C,G,T}\}$, $|x| = n$ 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 $z[i:j], i \in [1,n], j \geq i$ starts at position i in z and equals the finite substring of the linear infinite string $(a_1 \dots a_n)^\infty$ from position i to position j, where j can be $> |z|$. 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 $y \in \mathcal{R}$ such that x equals 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 the lower index in $\mathcal{R}$ is contained within the read with the 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 all our theoretical results, we will also assume that DNA is a single-stranded molecule.

Assumption on sequencing coverage. Typically depth of sequencing must be high enough for computing *de novo* genome assembly. 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 $\in \mathcal{R}$. A toy example is shown in Figure 1. This formulation will later be useful to ensure that reads "cover" the genome, and "consecutive" reads overlap by at least l_2 bases (formally proved in Theorem 2). 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 assumed to be a subset of $\mathcal{C}(\mathcal{R}, l_1, l_2, \phi)$.

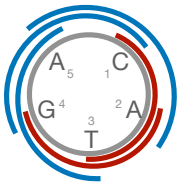

Fig. 1: Suppose $l_1 = 4$ and $l_2 = 1$. A circular string $z = \langle \text{CATGA} \rangle$ has five intervals of length $l_1 - l_2 = 3$ (shown using curved line segments). Suppose $|\mathcal{R}| = 2$, $\mathcal{R}[1] = \text{ATGA}$ and $\mathcal{R}[2] = \text{ACATG}$. Reads in $\mathcal{R}$ provide sufficient coverage over the circular string. The two red colored intervals include the beginning position of substring $z[2:5]$ that matches $\mathcal{R}[1]$. The other three intervals include the beginning position of substring $z[5:9]$ matching $\mathcal{R}[2]$.

Assembly graphs. De Bruijn graphs, overlap graphs and string graphs are popular graph-based frameworks used for *de novo* genome assembly. $G(V, E, \sigma, w)$ is a directed assembly graph where vertices are labeled with strings, and edges indicate suffix-prefix overlap relations between the labels of connected vertices. Function $\sigma : V \rightarrow \Sigma^+$ assigns a string label to each vertex. Function $w : E \rightarrow \mathbb{N}$ assigns a weight to each edge. Weight of an edge $v_1 \rightarrow v_2$ 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 $= 1$ 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 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 $v_1 \rightarrow v_2$ 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 [18] 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. String graph $S(\mathcal{R})$ has significantly fewer edges compared to overlap graph $O(\mathcal{R})$ in practice which enables computation of longer assembly contigs.

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 of length $n-1$ 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 $\langle\, pre_{w(e_1)}(\sigma(v_1))\, pre_{w(e_2)}(\sigma(v_2)) \ldots\, pre_{w(e_{n-1})}(\sigma(v_{n-1}))\, \rangle$. In other words, this is the circular string formed by concatenating labels of the vertices along the closed walk 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 $G(V, E, \sigma, w)$.*

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 by 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 in 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 the 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. If a circular string can be spelled in $O_{l_2}(\mathcal{R})$, then it can be spelled in $O_k(\mathcal{R})$, $\forall k < l_2$. In the following, we will show that $O_{l_2}(\mathcal{R})$ is coverage-preserving. 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, 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 $\mathcal{I}_i$ 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 index of the read that matches the substring $z[\mathcal{I}_i.s : \mathcal{I}_i.e]$ and has the highest index. 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 using a single vertex corresponding to read $\mathcal{I}_1.r$. In the following, we will focus on the case when $|\mathcal{I}_i.r| < |z| + l_2$ $\forall i \in [1, |z|]$. 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 $< |z|$ and $|z| > 1$.

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 $\mathcal{I}_{i+1}$ have length $l_1 - l_2$. The lower bound of $l_2 - 1$ for $\mathcal{I}_i.e - \mathcal{I}_{i+1}.s$ is achieved 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 $\mathcal{I}_1.s = 1$.

Lemma 2. *Length of the subsequence $(\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_p)$ identified by the stated 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, recall $|\mathcal{I}_1.r| < |z| + l_2$, $l_1 - l_2 < |z|$ and $|z| > 1$. Interval $\mathcal{I}_q$ cannot contain position $\mathcal{I}_1.s$ because size of each interval 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 $|\mathcal{I}_1.r| < |z| + l_2$. □

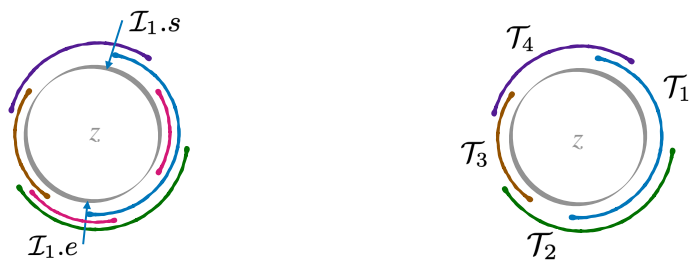

Fig. 2: A graphic illustration of the subsequence computed from the complete sequence of intervals $\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_{|z|}$. In the left figure, the gray-colored circle indicates circular string z, and the curved line segments indicate distinct substrings of z associated 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 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} in the algorithm. 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 $\mathcal{T}_p.e \geq \mathcal{T}_1.s + |z| + l_2 - 1$. □

Lemmas 3 and 4 collectively prove that vertices associated with the reads $\mathcal{T}_1.r$, $\mathcal{T}_2.r$, ..., $\mathcal{T}_p.r, \mathcal{T}_1.r$ 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 $z[\mathcal{T}_i.s : \mathcal{T}_{i+1}.s - 1]$. 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 $z[1 : \mathcal{T}_1.s - 1]$. The complete result is summarized as a theorem below.

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

3.3 String graphs

Theorem 3. *String graph* $S(\mathcal{R})$ *is not guaranteed to be coverage-preserving.*

Proof. We will provide two examples to illustrate this (Figure 3). Assume $l_1 = 6, l_2 = 2$ and $\phi = 10$ in both the examples.
Example 1: Suppose count of reads $|\mathcal{R}| = 2, \mathcal{R}[1]$ =CATATA, $\mathcal{R}[2]$ =TATACA. Accordingly, the set of candidate circular strings $\mathcal{C}(\mathcal{R}, l_1, l_2, \phi)$ is $\{\langle \text{CATATA} \rangle, \langle \text{CATATATA} \rangle\}$. The string graph $S(\mathcal{R})$ includes two vertices u, v such that $\sigma(u) = \mathcal{R}[1]$, $\sigma(v) = \mathcal{R}[2]$. Edges $u \to v$ and $v \to u$ have weights 2 and 4 respectively. Edge $u \to v$ of weight 4 is not considered because it does not correspond to the longest suffix-prefix overlap of $\sigma(u)$ with $\sigma(v)$. The graph $S(\mathcal{R})$ spells $\langle \text{CATATA} \rangle$ in a closed walk $u \to v \to u$. However, $S(\mathcal{R})$ does not spell $\langle \text{CATATATA} \rangle$.
Example 2: Suppose count of reads $|\mathcal{R}| = 4, \mathcal{R}[1]$ =CACGTG, $\mathcal{R}[2]$ =CACGTGG, $\mathcal{R}[3]$ =TGTGCA and $\mathcal{R}[4]$ =TGGGCA. Accordingly, the two candidate circular strings are $\langle \text{CACGTGTG} \rangle$ (covered by $\mathcal{R}[1], \mathcal{R}[3]$) and $\langle \text{CACGTGGG} \rangle$ (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 $\langle \text{CACGTGTG} \rangle$ cannot be spelled. □

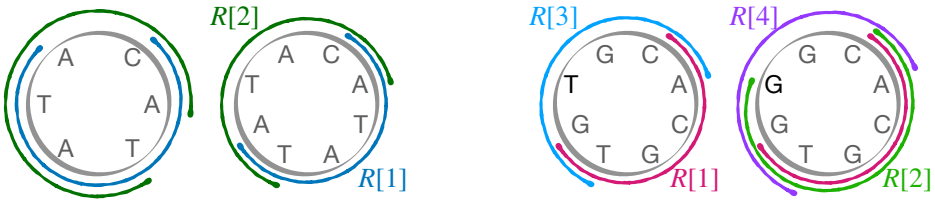

Fig. 3: A visualization of the two examples used in the proof of Theorem 3. The first example (left) uses two reads, and the second example (right) uses four reads. In each of the two examples, there are two candidate circular strings.

The practical significance of the above result is that despite sufficient sequencing coverage, string graphs may lose coverage over the genome. This limitation of string graphs has been pointed out in recent articles [7,9,19]. However, the following questions remain open: (a) Does the lack of the above guarantee in string graph affect quality of long reads-based genome assembly in practice? and (b) Does there exist an alternate sparse graph model of k-overlap graph whose size is similar to a 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. However, after pruning selected vertices and edges, it loses the coverage-preserving guarantee. 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 is directly inspired from the transitive sparsification step during string graph construction. A string spelled by a closed walk which uses edge e_3 can be spelled by replacing e_3 with edges e_1 and e_2 instead, i.e., $W_\phi(O'_k(\mathcal{R})) = W_\phi(O'_k(\mathcal{R}) - e_3)$. □

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 which spells a 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 proof of 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 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 vertex corresponding to contained read r as u. 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 that uses vertex u_p associated with parent read r_p. 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 based on the vertex and edge removal rules that are listed above as lemmas. However, there are multiple challenges associated with practical implementation of these rules. Suppose all sequencing errors are corrected by using an appropriate error-correction algorithm, estimating l_1, l_2 and ϕ remains non-trivial. Even if correct values of the parameters are estimated using some algorithm, an efficient algorithm to compute candidate circular strings $\mathcal{C}(\mathcal{R}, l_1, l_2, \phi)$ is unknown. Computing set of closed walks $W_\phi(G)$ in overlap graphs is also non-trivial.

5 A proof-of-concept implementation

While computing haploid, diploid or polyploid genome assembly, preserving each haplotype sequence as a walk in the assembly graph is a prerequisite for computing accurate assembly. We seek to build a sparse overlap graph that (i) removes transitive edges by using the property in Lemma 7, and (ii) filters out a subset of contained reads by using heuristics that are inspired from Lemmas 5 and 9. Our implementation currently works on error-corrected 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 $G(V, E, \sigma, w)$ is said to be *Watson-Crick complete* if the following two properties are satisfied: (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 containment overlaps among reads. The minimum overlap length cutoff is set to 5 kbp by default, and is applicable to both suffix-prefix and containment overlaps. For our experiments, we relied on minimap2 tool to compute the 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 a Watson-Crick complete 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 saved to represent containment overlap 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 [18]. However, the linear $O(|E|)$ expected time algorithm developed by Myers to label transitive edges is not applicable here because our graph is a multi-graph, i.e., graph can have multiple edges with the same source and target vertices. We use

a simple $O(|E|D^2)$ time algorithm (Algorithm 1) where D is the maximum out-degree of a vertex. The algorithm iterates over all edges $\in E$ (Line 1). Suppose e_1 denotes the edge being processed with source vertex v_1 and target vertex v_2. The next loop iterates over edges originating from v_1. Edges which connect an out-neighbor of vertex v_1 to vertex v_2 are evaluated (Line 4). It can be easily proved that if the input graph is Watson-Crick complete, then the resulting graph after removing transitive edges is Watson-Crick complete.

Algorithm 1: Labeling of transitive edges

```
   Input: Graph G(V, E, σ, w)
1  for e1 = v1 → v2 ∈ E do
2      for e2 = v1 → v3 ∈ E do
3          for e3 = v3 → v4 ∈ E do
4              if v4 = v2 and w(e1) = w(e2) + w(e3) then
5                  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 develop a heuristic to check the first two conditions by modifying the error-correction module of hifiasm code. The error-correction module in hifiasm performs all-versus-all read alignment. For a target read r, hifiasm inspects the alignments of other reads with the target read that are above a cutoff sequence identity and length. For *each* position in the target read, the following parameters are tracked by hifiasm: (a) count of overlapping alignments of other reads, (b) count of overlapping alignments of other reads that have equal nucleotide symbol as read r, (c) count of overlapping alignments of other reads that have a different nucleotide symbol from read r, and (d) count of overlapping alignments with a gap symbol '$-$' at the position. These four quantities are denoted using symbols *cnt*, *cnt_match*, *cnt_mismatch* and *cnt_gap* respectively. In addition, hifiasm also estimates global "homozygous sequencing coverage" of the given sequencing data which is equivalent to the sum of coverage over individual haplotypes. This quantity is denoted as *hom_cov*. If the target read r is contained, and the sequenced genome is diploid, we remove the two vertices corresponding to read r and its reverse complement respectively if both the following conditions are satisfied:

1. $r[i].cnt < hom_cov \times 1.5 \;\; \forall i \in [1, |r|]$
2. $\exists i$ such that $r[i].cnt_match \geq hom_cov \times 0.25$ and $r[i].cnt_mismatch \geq hom_cov \times 0.25$

The first condition is used to estimate whether read r is non-repetitive and the second condition estimates the presence of at least one heterozygous SNP in read r. The second condition is not needed if the sequenced genome is haploid. Our code is implemented such that it accepts from user a path to a file with read ids of non-repetitive haplotype-specific reads. Such reads which are also contained are filtered out before graph construction. In our experiments, this file is pre-computed separately by using the modified hifiasm code. The modified hifiasm code can be accessed online[1].

5.3 k-mer-based filtering heuristic for contained reads. To check if a contained read which is not yet removed is essential, 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 is not an easy task. We address this by implementing a k-mer-based heuristic. We first compute the set of k-mers observed in bounded-length walks from the vertex associated with contained read. The set will be denoted as κ_1. Next, we compute the union of set of k-mers observed in bounded-length walks from the vertices associated with the parent reads of the contained read. This set will be denoted as κ_2. If $\kappa_1 \subseteq \kappa_2$, we

[1] https://github.com/cjain7/hifiasm/tree/hifiasm_dev_debug

mark the contained read for removal. This heuristic is used to estimate whether there exists a string which cannot be spelled due to removal of the contained read. A precise description of the heuristic follows.

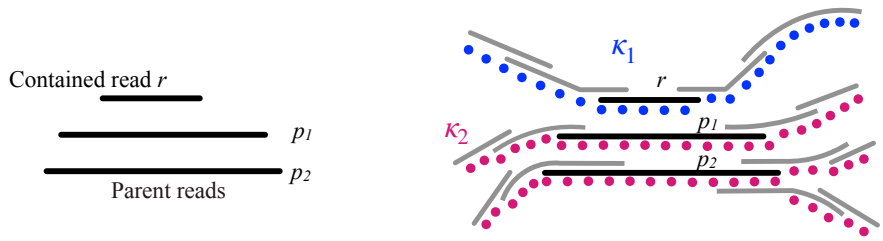

Fig. 4: A graphic illustration of how k-mer sets κ_1 and κ_2 are computed from contained read r and its parent reads respectively. Small colored circles indicate the k-mers extracted from the strings.

Let $\mathcal{L}(v, i, d)$ be the set of linear strings of length $\leq d$ that can be spelled by a walk starting from position i in $\sigma(v)$, $1 \leq i \leq |\sigma(v)|$, $v \in V$. For instance, if one of the possible walks is (v_1, e_1, v_2), then it can be used to spell suffix of $\sigma(v_1)$ from position i, followed by a prefix of the non-overlapping suffix of v_2. Suppose v and $\bar{v}$ are the vertices corresponding to a contained read r and its reverse complement respectively. Then $\kappa_1 = \mu_k(\mathcal{L}(v, 1, d)) \cup \mu_k(\mathcal{L}(\bar{v}, 1, d))$, where function μ_k returns the set of k-mers in a given set of strings. By using $\bar{v}$ along with v, walks starting from v and ending at v are automatically considered. Next, we build set κ_2 of the union of k-mers found within the bounded-length walks using parent reads of r. All tuples associated with containment overlaps of read r are processed one by one. Suppose tuple (p, i, j, o) indicates containment of r within parent read p; o specifies what orientation of p should be used. If $o =$'+', let u be the vertex corresponding to read p. Otherwise, let u be the vertex corresponding to reverse complement of read p. We add k-mers found within the bounded-length walks using read p, i.e., $\mu_k(\mathcal{L}(u, i, d)) \cup \mu_k(\mathcal{L}(\bar{u}, |\sigma(u)| - j + 1, d))$ into set κ_2. After all the tuples have been processed, we decide whether to mark the contained read for removal by checking if $\kappa_1 \subseteq \kappa_2$. In order to make sure that the heuristic finishes fast, our implementation uses $k = 16$ and $d = 3|r|$ while processing contained read r. All contained reads can be processed independently in parallel if multiple threads are available.

6 Experimental results

Our focus in the experiments is on evaluating the importance and performance of heuristics for deciding which contained reads should be retained during overlap graph construction. We specifically focus on 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? In order to preserve genome coverage while keeping graph density low, suffix-prefix overlaps also have to be carefully filtered using an appropriate minimum length cutoff, but we do not consider this issue here.

6.1 Experimental setup

Read simulation from human haploid and diploid genomes. We use 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) [19]. It contains complete gapless assembled sequences of chromosomes 1-22, X from CHM13 human cell line and chromosome Y from HG002 human cell line. This haploid genome assembly of size 3.1 Gbp has 25 contigs and N50 length 50.6 Mbp. The

second genome assembly is a publicly available long-read diploid genome assembly[2] of HG002 human cell line computed using hifiasm-trio (v.0.16.1). It was computed using PacBio HiFi long reads and parent-child data to sort haplotypes during assembly [3]. This diploid genome assembly of size 6.0 Gbp has 970 contigs and N50 length 57.8 Mbp. From both the haploid and diploid genome assemblies, we simulate two sets of *error-free* reads whose length distributions are compatible with real PacBio HiFi and ONT sequencing data by using seqrequester tool (https://github.com/marbl/seqrequester). This tool simulates reads in both forward and reverse orientations. It allows users to specify a read length distribution. We use HiFi and ONT read length distributions from publicly-available long-read sequencing datasets of HG02080 human cell line provided by the HPRC consortium. HiFi and ONT long-read sets are simulated from haploid genome assembly with 20-fold coverage. Similarly, HiFi and ONT long-read sets are 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. Commands used to run the tools are listed in Supplementary Table 1.

Id	Count of reads	Total bases (Gbp)	N50 length	Minimum length	Maximum length	Count of uncovered haplotype intervals
HAPLOID-20x-ONT	3,697,647	62.3	40,098	3	572,359	0
HAPLOID-20x-HiFi	2,913,761	62.3	21,314	9	48,708	0
DIPLOID-30x-ONT	5,348,230	89.5	39,752	1	544,992	2
DIPLOID-30x-HiFi	4,199,015	89.5	21,299	1	48,708	1

Table 1: Properties of the simulated long-read sequencing datasets including their size and length statistics. During the simulation, each read id stores metadata specifying its location of origin and orientation. Using this information, we also list the count of genomic intervals from where no reads were sampled.

6.2 Assessment of coverage gaps caused by removing contained reads

Benchmarking procedure. We use the following method to estimate coverage gaps introduced by removal of contained reads. First, we compute all-versus-all read overlaps using minimap2 (v2.23) to identify the set of contained reads. A read is marked as contained if it matches a proper substring of a longer read. There cannot be false positives here because overlaps are verified using an alignment algorithm. However, we can have false negatives because minimap2 uses k-mer-based heuristics. Typically false negatives happen when a contained read is too short (e.g., < 100 bp). In the second step, the set of non-contained reads is mapped to the genome. For the diploid genome, the set of non-contained reads is mapped to both paternal and maternal haplotypes, one by one. Minimap2 parameters are adjusted to enable multiple alignments per read. Whenever a read aligns end-to-end against a genomic interval with 100% alignment identity, we record the interval. Finally, all the genomic intervals where no alignment coverage is observed are extracted using bedtools (v2.29.1). Not all of these can be considered as coverage gaps. 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 identified as coverage gaps introduced by removal of contained reads. Whenever one or more coverage gaps are observed, we validate the cause by repeating the above steps without removing contained reads and ensure that the same coverage gaps are not observed. We additionally verify that all the reads simulated from a gap interval are marked as contained. The validation steps are unnecessary if an exact algorithm is used instead of minimap2.

Results. Table 2 reports the count of coverage gaps observed in the four datasets. We observe that coverage gaps are not introduced in haploid genome for both ONT and HiFi read sets. In theory,

[2] 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

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, we observe 98 and 5 coverage gaps introduced in ONT and HiFi datasets respectively. Compared to a haploid genome, coverage gaps in diploid genomes are more likely to happen because it is easy to have scenarios where a longer read sampled from a haplotype contains all reads sampled from the homologous loci of the other haplotype. Moreover, ONT read length distribution is highly non-uniform which leads to a higher fraction of contained reads, and therefore, more coverage gaps. Read length distributions are shown in Supplementary Figure 1. We further investigate whether the observed gaps are clustered in a particular region of the genome, but we do not observe such behavior. The genomic coordinates of these coverage gaps within the HG002 assembly are lifted over to GRCh38 human genome reference by using paftools. The lists of these coordinates are provided in Supplementary Tables 2 and 3.

Id	Count of contained reads	Coverage gap statistics		
		Count	Mean length	Maximum length
HAPLOID-20x-ONT	3,160,926	0	-	-
HAPLOID-20x-HiFi	1,897,005	0	-	-
DIPLOID-30x-ONT	4,551,987	98	7,912	71,802
DIPLOID-30x-HiFi	2,506,031	5	1,101	2,709

Table 2: Coverage gap statistics for the gaps introduced by removal of contained reads. Count of contained reads is estimated from read overlaps computed by minimap2. The true count of contained reads is expected to be higher because minimap2 may not be sensitive to short reads (e.g., < 100 bp).

6.3 Evaluation of the proposed 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. Here we focus only on the two diploid datasets. We check the performance by observing the following two parameters.

1. Count of "junction" reads, i.e., the reads which correspond to vertices having either > 1 incoming or > 1 outgoing edges in the graph. A junction read indicates ambiguity in the assembly graph where one vertex connects to multiple others.
2. Count of coverage gaps caused by removing selected contained reads.

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 second parameter, we rely on minimap2 to compute end-to-end 100% identical read alignments of the retained contained reads against the two haplotypes. We check if these alignments result in closing the coverage gaps that were caused by removing all contained reads.

Competing methods. We compare 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 has been suggested by 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 r, they disagree at some base. This can happen, for example, when the two parent reads originate from different haplotypes and contain a heterozygous variant. We also include 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. Hifiasm-meta extends hifiasm for metagenome assembly, and uses a different heuristic for retaining contained reads [7]. It retains a contained read r when all parent reads of r are predicted to be originated

from haplotypes different from r. A comparison with hifiasm-meta is skipped because it is designed explicitly for metagenomic data. We implemented the first three methods, i.e., Retain-all, Remove-all and Hui-2016 within ContainX code; these can be run by passing appropriate command line parameters (Supplementary Table 1).

Dataset	Method	Count of retained contained reads	Count of junction reads	Count of coverage gaps
DIPLOID-30x-ONT	Retain-all	2,762,354	2,540,960	0
	Hui-2016	2,489,736	2,282,235	0
	Remove-all	0	38,267	98
	ContainX	28,234	53,008	0
	hifiasm	3,319	1,607	77
DIPLOID-30x-HiFi	Retain-all	2,499,031	3,386,900	0
	Hui-2016	2,455,294	3,319,981	0
	Remove-all	0	158,988	5
	ContainX	40,951	184,887	0
	hifiasm	138	37,108	3

Table 3: Performance comparison of ContainX with four other methods using two datasets DIPLOID-30x-ONT and DIPLOID-30x-HiFi. We measure the count of contained reads retained by each method, count of junction reads in the constructed graph and count of unresolved coverage gaps. The first four methods stop at overlap graph construction whereas hifiasm is a commonly used genome assembly tool that outputs a unitig graph.

Results. We will focus on the performance of the first four methods and discuss hifiasm separately. Those reads without any suffix-prefix overlap above user-specified length threshold are excluded from the graph by default in all the four methods. In both datasets, Retain-all method retains the highest number of contained reads as expected. It resolves all coverage gaps, but the corresponding graph has significantly higher junction reads compared to Remove-all. Next, Hui-2016 filtering method appears to be conservative, i.e., it 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. The heuristic resolves 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. ContainX delivers favorable performance compared to the three methods. Using both datasets, ContainX retains $1-2\%$ contained reads compared to Retain-all, yet it successfully resolves the coverage gaps. ContainX results 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. Hifiasm retains fewer contained reads than ContainX but it fails to resolve a majority of coverage gaps in both datasets. 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 will be an interesting direction to explore. However, this would require us to modify the k-mer based heuristic (section 5.3) in ContainX to tolerate sequencing errors because error-corrected reads may not be 100% accurate in practice.

The proposed graph sparsification algorithms implemented in ContainX are easy-to-implement, fast and space-efficient. The k-mer-based heuristic is parallelized by considering each contained read independently. The additional time required by all the ContainX heuristics using a multi-core AMD processor is about ten minutes. This is insignificant compared to other time-consuming steps in long-read assembly such as computing all-vs-all read alignments.

Acknowledgements

We thank Haowen Zhang, Mile Sikic and Robert Vaser for useful discussions on this topic. 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 Telomere-to-Telomere consortium. We benefited from publicly-available data released by the T2T and HPRC consortiums. This research is supported in part by funding from the Indian Institute of Science.

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. Jain, M., Koren, S., Miga, K.H., Quick, J., Rand, A.C., Sasani, T.A., Tyson, J.R., Beggs, A.D., Dilthey, A.T., Fiddes, I.T., et al.: Nanopore sequencing and assembly of a human genome with ultra-long reads. Nature biotechnology 36(4), 338–345 (2018)
11. Jarvis, E.D., Formenti, G., Rhie, A., et al.: Automated assembly of high-quality diploid human reference genomes. bioRxiv (2022), `https://www.biorxiv.org/content/early/2022/03/06/2022.03.06.483034`
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. 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)
15. Marx, V.: Long road to long-read assembly. Nature Methods 18(2), 125–129 (2021)
16. Medvedev, P., Pop, M.: What do eulerian and hamiltonian cycles have to do with genome assembly? PLoS Computational Biology 17(5), e1008928 (2021)
17. 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)
18. Myers, E.W.: The fragment assembly string graph. Bioinformatics 21(suppl_2), ii79–ii85 (2005)

19. 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. bioRxiv (2021)
20. 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)
21. 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)
22. 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 near-perfect bacterial genomes from pure cultures and metagenomes without short-read or reference polishing. bioRxiv (2021)
23. 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)
24. 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)
25. Vaser, R., Šikić, M.: Time-and memory-efficient genome assembly with raven. Nature Computational Science 1(5), 332–336 (2021)
26. 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)
27. 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)
28. 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

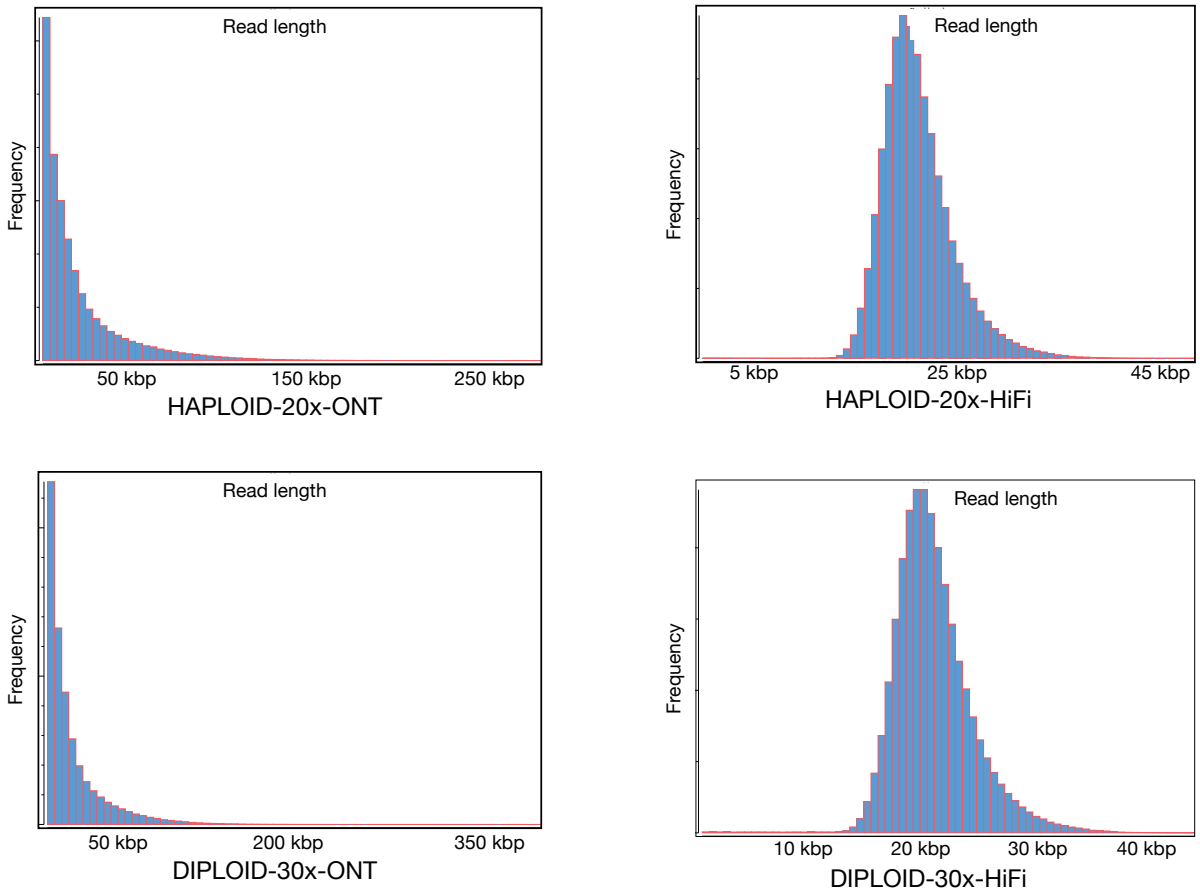

Fig. 1: Histogram of the number of reads with each read length. Reads of length < 1000 bp are not considered. The above plots show that read length distribution of ONT datasets is highly skewed with a large number of reads of short length. HiFi read lengths are within a narrow range compared to ONT. 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 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
h2tg000008l	17659995	17662994	chr1	24706209	24709211
h1tg000009l	29612179	29617307	chr1	35788560	35793686
h2tg000020l	41115665	41116209	chr1	93064227	93064771
h2tg000020l	41116558	41120625	chr1	93065120	93069187
h2tg000020l	41120686	41138205	chr1	93069248	93086768
h1tg000009l	101013468	101016805	chr1	107142594	107145931
h1tg000009l	101016860	101017556	chr1	107145986	107146682
h2tg000020l	59907093	59916148	chr1	111854650	111863706
h2tg000020l	59916227	59920210	chr1	111863785	111867764
h2tg000020l	59920253	59923114	chr1	111867807	111870666
h2tg000022l	50206283	50206838	chr1	178376789	178377344
h1tg000012l	30370436	30373042	chr1	180128653	180131258
h1tg000012l	58710849	58716360	chr1	208575630	208581141
h2tg000029l	54846436	54863658	chr2	13585562	13602801
h2tg000029l	54837359	54846246	chr2	13602991	13611916
h2tg000029l	54827118	54837304	chr2	13611971	13622152
h2tg000029l	54822598	54826959	chr2	13622311	13626677
h2tg000090l	1316761	1319651	chr2	125000830	125003720
h2tg000090l	1319712	1320159	chr2	125003781	125004228
h1tg000001l	3451094	3456547	chr2	134015951	134021405
h1tg000001l	3456674	3457554	chr2	134021532	134022416
h1tg000001l	3457669	3459520	chr2	134022531	134024382
h2tg000010l	61815406	61817900	chr3	135165843	135168337
h2tg000010l	61809262	61815291	chr3	135168452	135174481
h1tg000034l	40413488	40417003	chr3	156529822	156533336
h1tg000034l	39179603	39180165	chr3	157766711	157767273
h1tg000034l	39177706	39179549	chr3	157767327	157769152
h1tg000042l	99703608	99707806	chr4	88437925	88442123
h1tg000042l	99702335	99703452	chr4	88442279	88443396
h1tg000042l	99692989	99702259	chr4	88443472	88452742
h1tg000042l	99687118	99692842	chr4	88452889	88458612
h1tg000042l	99675115	99686903	chr4	88458827	88470615
h1tg000042l	99665374	99675049	chr4	88470681	88480356
h1tg000233l	49140	63441	chr4	143945306	143959587
h1tg000233l	42432	49087	chr4	143959640	143966295
h1tg000233l	39432	42374	chr4	143966353	143969295
h1tg000233l	35967	39382	chr4	143969345	143972760
h1tg000233l	32340	35909	chr4	143972818	143976387
h1tg000233l	26349	32120	chr4	143976607	143982378
h2tg000014l	9777751	9780610	chr4	173058405	173061264
h2tg000014l	9772559	9777568	chr4	173061447	173066457
h2tg000014l	9761827	9772500	chr4	173066516	173077180
h1tg000047l	10336355	10341296	chr5	59457068	59462009
h1tg000016l	1092584	1114355	chr5	173628346	173650117
h1tg000016l	1092016	1092533	chr5	173650168	173650685
h1tg000016l	1087493	1091880	chr5	173650821	173655210
h2tg000021l	24955470	24958763	chr6	84748430	84751724
h1tg000036l	33433002	33436160	chr6	124387112	124390270
h2tg000098l	8073287	8079304	chr6	162690162	162696178
h2tg000098l	8065911	8073241	chr6	162696224	162703556
h2tg000098l	8062981	8065857	chr6	162703610	162706486
h2tg000098l	8062485	8062930	chr6	162706537	162706982
h2tg000049l	13161895	13171843	chr7	20941799	20951747
h2tg000049l	13138468	13161793	chr7	20951849	20975184
h1tg000067l	4397303	4424502	chr7	96643024	96670223
h1tg000067l	4390592	4397242	chr7	96670284	96676930
h2tg000011l	77122669	77124296	chr8	121002065	121003707
h2tg000011l	77124497	77150568	chr8	121003908	121029947
h2tg000011l	77150725	77154708	chr8	121030104	121034087
h2tg000041l	28320360	28320904	chr9	28303090	28303634
h2tg000041l	28320964	28321097	chr9	28303694	28303827
h2tg000041l	28321146	28334330	chr9	28303876	28317053
h2tg000041l	28334389	28337657	chr9	28317112	28320380
h2tg000041l	28337992	28356121	chr9	28320715	28338860
h1tg000059l	5770345	5770357	chr9	35608310	35608322
h1tg000059l	5768912	5770290	chr9	35608377	35609755
h2tg000001l	36703187	36721975	chr10	79571186	79589932
h2tg000001l	54101795	54104390	chr10	96885499	96888094
h2tg000030l	18866061	18896358	chr11	18815028	18845326
h1tg000022l	34145414	34154517	chr13	52597310	52606415
h1tg000022l	34141673	34145359	chr13	52606470	52610156
h1tg000022l	34129031	34141620	chr13	52610209	52622808
h1tg000022l	34096561	34128974	chr13	52622865	52655313
h2tg000016l	17595883	17597622	chr13	103819406	103821145
h2tg000016l	17597840	17602011	chr13	103821363	103825534
h1tg000017l	22355208	22361604	chr14	42061121	42067515
h1tg000017l	22361662	22367547	chr14	42067573	42073463
h2tg000039l	22881683	22885980	chr16	78640452	78644747
h1tg000039l	14482108	14486811	chr17	60172516	60177220
h1tg000039l	14486854	14492943	chr17	60177263	60183349
h1tg000039l	14493000	14494622	chr17	60183406	60185028
h1tg000039l	14494779	14499038	chr17	60185185	60189444
h1tg000039l	14499097	14513638	chr17	60189503	60204044
h1tg000020l	49910720	49911198	chr18	28498334	28498812
h1tg000020l	49896784	49910635	chr18	28498897	28512749
h1tg000020l	49895008	49896693	chr18	28512840	28514525
h1tg000020l	39567017	39568547	chr18	38840453	38841983
h1tg000020l	28513955	28520766	chr18	49875181	49881999
h1tg000020l	28510761	28513905	chr18	49882049	49885218
h1tg000020l	28507172	28510686	chr18	49885293	49888805
h1tg000020l	28501118	28507004	chr18	49888973	49894859
h1tg000051l	20673752	20745554	chr20	42402367	42474152
h1tg000051l	20604049	20615688	chr20	42532285	42543959
h1tg000051l	20602762	20604004	chr20	42544004	42545246
h1tg000051l	20586158	20602666	chr20	42545342	42561853
h1tg000051l	20565997	20586104	chr20	42561907	42581998
h1tg000051l	20531106	20565773	chr20	42582222	42616892
h1tg000060l	15684891	15691550	chr21	28704023	28710682

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

HG002 genome assembly			GRCh38 genome reference		
Contig id	Start	End	Chromosome id	Start	End
h1tg000006l	27133762	27136471	chr6	27176513	27179214
h1tg000006l	32104840	32106053	chr6	32157328	32158541
h2tg000021l	76487955	76488007	chr6	136341956	136342008
h1tg000023l	31436783	31437725	chr7	132419515	132420457
h2tg000121l	2147514	2148104	chr9	41487272	41487862

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