Overlap-Based Genome Assembly from Variable-Length Reads

Joseph Hui, Ilan Shomorony, Kannan Ramchandran and Thomas A. Courtade
Department of Electrical Engineering and Computer Sciences, University of California, Berkeley
Email: {ude.yelekreb, ilan.shomorony, kannanr, courtade}@berkeley.edu

***Abstract*—Recently developed high-throughput sequencing platforms can generate very long reads, making the *perfect assembly* of whole genomes information-theoretically possible [1]. One of the challenges in achieving this goal in practice, however, is that traditional assembly algorithms based on the *de Bruijn* graph framework cannot handle the high error rates of long-read technologies. On the other hand, overlap-based approaches such as string graphs [2] are very robust to errors, but cannot achieve the theoretical lower bounds. In particular, these methods handle the variable-length reads provided by long-read technologies in a suboptimal manner. In this work, we introduce a new assembly algorithm with two desirable features in the context of long-read sequencing: (1) it is an overlap-based method, thus being more resilient to read errors than de Bruijn graph approaches; and (2) it achieves the information-theoretic bounds even in the variable-length read setting.**

I. Introduction

Current DNA sequencing technologies are based on a two-step process. First, tens or hundreds of millions of fragments from random unknown locations on the *target genome* are read via *shotgun sequencing*. Second, these fragments, called reads, are merged to each other based on regions of overlap using an *assembly algorithm*. As output, an assembly algorithm returns a set of *contigs*, which are strings that, in principle, correspond to substrings of the target genome. In other words, contigs describe sections of the genome that are correctly assembled.

Algorithms for sequence assembly can be mainly classified into two categories: approaches based on *de Bruijn graphs* [3] and approaches based on *overlap graphs* [2, 4, 5]. Following the short-read high-throughput trend of second-generation sequencers, assemblers based on de Bruijn graphs became popular. Roughly speaking, these assemblers operate by constructing a de Bruijn graph with vertex set given by the set of distinct K-mers extracted from the reads, and connecting two vertices via a directed edge whenever the corresponding K-mers appear consecutively in the same read. By construction, if the reads achieve sufficient coverage, the target genome corresponds to a Chinese Postman route on the graph, which is a path that traverses every edge at least once. The problem of finding the 'correct' Chinese postman route (thus determining the target genome) is complicated by the fact that repeated regions in the genome are condensed into single paths. Thus, to resolve repeats and obtain long contigs, a finishing step must be taken where the original reads are brought back and aligned onto the graph.

While the construction of the de Bruijn graph can be performed efficiently both in time and space, this approach has two main drawbacks. The first is that shredding the reads into K-mers renders the task of resolving repeats and obtaining long contigs more challenging. The second drawback is that the de Bruijn graph construction is not robust to read errors. Indeed, even small error rates will generate many chimeric K-mers (i.e., K-mers that are *not* substrings of the target genome). As a result, heuristics must be implemented in practice to clean up the graph.

This work was supported in part by NSF Grants CCF-1528132 and CCF-0939370 (Center for Science of Information).

On the other hand, overlap-based assembly algorithms typically operate by constructing an *overlap graph* with vertex set corresponding to the set of observed reads, where two vertices are connected if the suffix of one of the reads enjoys significant similarity with the prefix of the other (i.e,. two reads *overlap* by significant margin). This way, the target genome corresponds to a (generalized) Hamiltonian path on the graph, assuming sufficient coverage. By not breaking the reads into small K-mers, overlap-based approaches promise to generate less fragmented assemblies. Moreover, read errors have small impact if we restrict our attention to overlaps of sufficient length, implying that overlap-based assemblers can be more robust to read errors than their de Bruijn counterparts. Therefore, in the context of long-read third-generation sequencing (where error rates are high, and will continue to be for the foreseeable future [6]), overlap-based approaches are expected to play a central role.

In spite of their relevance in the context of long-read sequencing, our formal understanding of overlap-based algorithms is fairly limited. Under most natural formulations, extracting the correct sequence from the overlap graph becomes an NP-hard problem [7, 8]. Moreover, as the graph in general contains many spurious edges due to repeats in the target genome, formal analysis of these algorithms is difficult and very few of them have theoretical guarantees. One example is the work in [9], where an overlap-based algorithm is shown to have theoretical performance guarantees under the assumption of fixed-length reads. In practice this is never the case (e.g., PacBio reads can differ by tens of thousands of base pairs [5]), and processing the reads so that they all have the same length is usually suboptimal.

In this paper, we introduce an efficient, overlap-based assembly algorithm that handles variable-length reads and is guaranteed to reconstruct the target genome provided the reads satisfy the information-theoretic sufficient conditions proposed in [1].

II. Background and Definitions

In the genome assembly problem, the goal is to reconstruct a target sequence $g = (g[0], ..., g[G-1])$ of length G with symbols from the alphabet $\Sigma = \{\mathsf{A}, \mathsf{C}, \mathsf{G}, \mathsf{T}\}$. The sequencer produces a set of N *reads* $\mathcal{R} = \{\mathbf{r}_1, ..., \mathbf{r}_N\}$ from G, each of which is a substring of g. For ease of exposition, we assume a *circular* genome model to avoid edge-effects, so that a substring may wrap around to the beginning of g. Thus $g\,[5:3]$ denotes $g\,[5:G-1]\,g\,[0:3]$. The reads may be of arbitrary length. The goal is to design an *assembler*, which takes the set of reads $\mathcal{R}$ and attempts to reconstruct the sequence g.

A. Bridging conditions and Optimal assembly

In [1], the authors derive necessary and sufficient conditions for assembly in terms of *bridging* conditions of *repeats*. These conditions are used to characterize the information limit for the feasibility of the assembly problem. In this section, we recall the main ideas behind this characterization, which serve as motivation to our approach.

A *double repeat* of length $\ell \geq 0$ in g is a substring $x \in \Sigma^\ell$ appearing at distinct positions i_1 and i_2 in g; i.e., $g[i_1 : i_1+\ell-1] = g[i_2 : i_2+\ell-1] = x$. Similarly, a *triple repeat* of length ℓ is a substring x that appears at three distinct locations in s (possibly overlapping); i.e., $g[i_1 : i_1+\ell-1] = g[i_2 : i_2+\ell-1] = g[i_3 : i_3+\ell-1] = x$ for distinct i_1, i_2 and i_3 (modulo G, given the circular assumption on g). If x is a double repeat but not a triple repeat, we say that it is *precisely a double repeat*. A double repeat x is *maximal* if it is not a substring of any strictly longer double repeat. Finally, if $x = g\,[i_1 : i_1 + l] = g\,[i_2 : i_2 + l]$ and $y = g\,[j_1 : j_1 + l'] = g\,[j_2 : j_2 + l']$ for some $i_1, i_2, l, j_1 j_2, l'$ where x, y are maximal and $i_1 < j_1 < i_2 < j_2$, then x and y form an *interleaved repeat*. Examples are shown in Fig. 1.

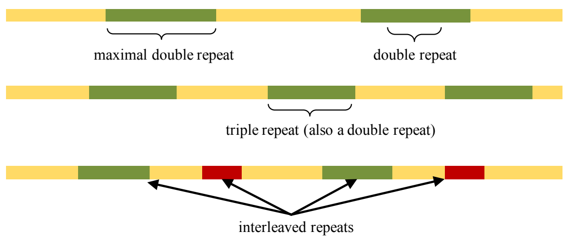


Fig. 1. Examples of various kinds of repeats.

A repeat consists of several *copies*, starting at distinct locations i_1, i_2, and so forth. A read $r = g\,[j_1 : j_2]$ is said to *bridge* a copy $g\,[i : i + l]$ if $j_1 < i$ and $j_2 > i + l$, as illustrated in Fig. 2. A

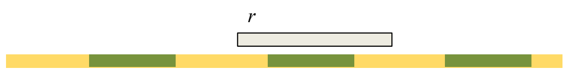


Fig. 2. A read bridging one copy of a triple repeat.

repeat is *bridged* if at least one copy is bridged by some read, and *all-bridged* if every copy is bridged by some read. A set of reads $\mathcal{R}$ is said to cover g if every base in g is covered by some read. In the context of two reads r_1, r_2 which both contain some string of interest s, r_1 and r_2 are said to be *inconsistent* if, when aligned with respect to s, they disagree at some base, as illustrated in Fig. 3.

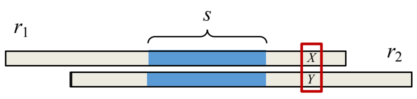


Fig. 3. Reads r_1 and r_2 are inconsistent with respect to the shared string s.

In [1], the authors proposed a de Bruijn graph-based assembly algorithm called MULTIBRIDGING and proved it to have the following theoretical guarantee, stated in terms of bridging conditions:

Theorem 1. [1] MULTIBRIDGING *correctly reconstructs the target genome g if $\mathcal{R}$ covers g and*

B1. Every triple repeat is all-bridged.

B2. Every interleaved repeat is bridged (i.e. of its four copies, at least one is bridged).

The motivation for appealing to conditions B1 and B2 stems from the observation that, under a uniform sampling model where N reads of a fixed length L are sampled uniformly at random from the genome, these conditions nearly match necessary conditions for assembly [1]. Motivated by this near-characterization of the information limits for perfect assembly and the advantages of overlap-based assembly for long-read technologies, we describe an overlap-based algorithm with the same performance guarantees. That is, provided conditions B1 and B2 are satisfied, our assembly algorithm will correctly reconstruct the target genome g. The analysis in [1] shows that, when B1 and B2 are not met, the assembly problem is likely to be infeasible, and there is inherent ambiguity in the target genome given the set of observed reads. In this sense, our algorithm can be considered to be a near-optimal overlap-based assembler.

III. Algorithm Overview

Let's begin with a description of the overall structure of our approach. The algorithm starts with a set $\mathcal{R}$ of variable-length reads, as illustrated in Fig. 4(a). Notice that in general $\mathcal{R}$ may contain many reads that are essentially useless - for example, a read consisting only of a single letter. Hence we begin by discarding some of these useless reads. A typical discarding strategy (used, for example, in the string graph approach [2, 4]), consists of simply discarding any read that is contained within another read. However, such reads can potentially encode useful information about the genome (see example in Figure 5). Thus, we first process the reads using a more careful rule described in Section IV to only throw away reads that are truly useless. This yields a trimmed-down set of reads as shown in Fig. 4(b).

The next step, the read extension, is the most complex part of the algorithm. For each read, we consider its potential successors and predecessors and carefully decide whether it can be extended to the right and to the left in an unambiguous way. Whenever B1 is satisfied, our extension algorithm is guaranteed to extend all reads correctly. Moreover, we can keep extending the reads in both directions until we hit the end of a double repeat. At this point we are not sure how to proceed and we stop, obtaining a set of extended reads as shown in Fig. 4(c).

In the third step, we merge reads that contain certain unique "signatures" and must belong together. Although the example in Fig. 3 does not show it, in this step we may also merge nonidentical reads. If a double repeat is bridged by some read, this merging process will merge the bridging read with the correct reads to the left and right, thus "resolving" the repeat. The merging operation produces a new set of reads as illustrated in Fig. 4(d). At this point the only remaining ambiguity comes from unbridged double repeats.

Finally, we resolve the residual ambiguity by constructing a graph. Notice that for each unbridged double repeat, we have two reads going in, and two going out, but we do not know the correct matching. We express this structure as a graph, where each long read is a node and each unbridged double repeat is also a (single) node, as illustrated in Fig. 4(e). Since each of the unbridged double repeats has in- and out-degree two, the graph is Eulerian, and contains at least one Eulerian cycle. Whenever condition B2

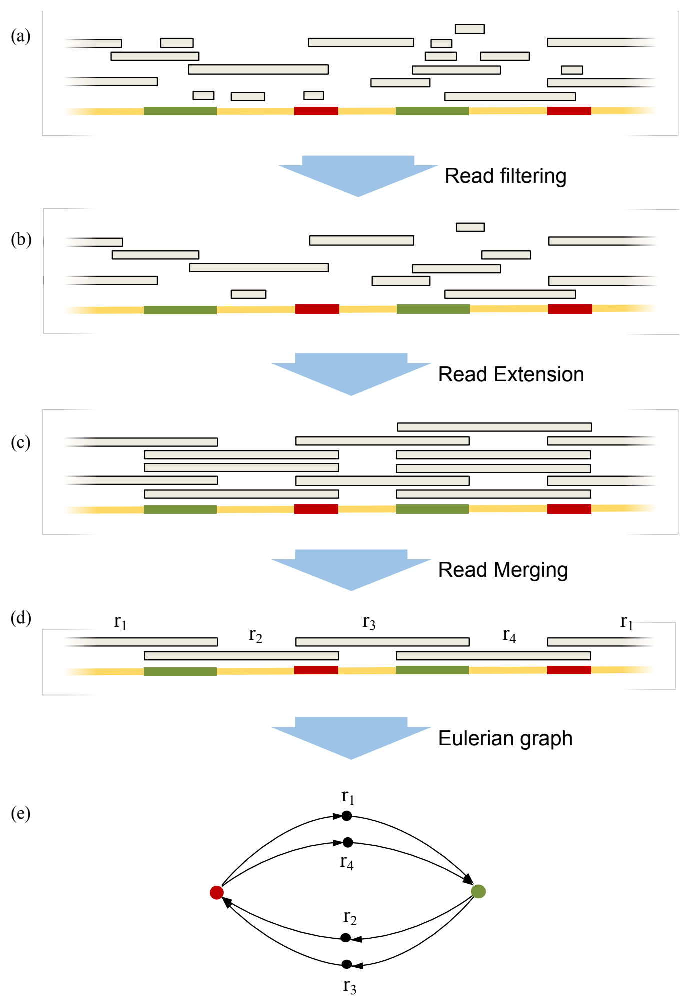


Fig. 4. The steps of the assembly algorithm.

is also satisfied, this cycle is unique, and corresponds to the true ordering of the long reads, yielding the true sequence.

In the next two sections, we will describe the algorithm in detail. In Section IV, we describe the three read processing steps: read filtering, extension and merging. Then in Section V, we present the final step where we construct the Eulerian graph and extract the genome sequence g from it. We refer to the appendix for detailed proofs.

IV. Processing Variable-Length Reads

A basic question that arises when dealing with variable-length reads is how to handle reads that are entirely contained in other reads; i.e., a read r_1 that is a substring of another read r_2. An intuitive idea would be to simply discard all such reads, as they seemingly contain no additional information for assembly.

However, as shown in Fig. 5, discarding all contained reads is in general suboptimal as it can create holes in the coverage, making perfect assembly from the remaining reads infeasible. Here, r_1 is contained within r_2, because r_2 bridges a repeat which in turn contains r_1. However, deleting r_1 causes the left copy to no longer be covered by any read.

We start instead with a more careful treatment of contained reads, described in Algorithm 1. As it turns out, this procedure preserves valuable properties of the set of reads $\mathcal{R}$, which will

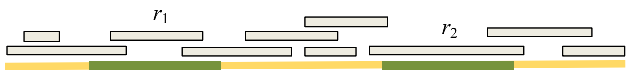


Fig. 5. Removing read r_1, which is contained in r_2, creates a coverage gap.

Algorithm 1 Contained read filtering

```
1: Input: $\mathcal{R}$
2: for $r \in \mathcal{R}$ do
3:     if r is contained in two reads that are inconsistent with each other then
4:         Remove $r$ from $\mathcal{R}$
5: Output: Updated $\mathcal{R}$
```

allow us to achieve perfect assembly. This is stated in the following lemma, whose proof we defer to the appendix.

Lemma 1. *Suppose $\mathcal{R}$ covers s and B1 and B2 hold. After the filtering procedure in Algorithm 1, $\mathcal{R}$ still covers s, B1 and B2 still hold, and in addition,*

B3. No read in $\mathcal{R}$ is a triple repeat in s.

After filtering out unnecessary reads, we move to the read extension step. The main idea is to consider one read at a time, and keep extending it in both directions according to other overlapping reads. Due the existence of repeats in s, however, we cannot always confidently determine the next base, so we stop when this is no longer possible. We describe this in Algorithm 2.

Algorithm 2 Extension Algorithm

```
1: Input: $\mathcal{R}$ after filtering from Algorithm 1
2: for $r \in \mathcal{R}$ do
3:     BifurcationFound $\leftarrow$ False
4:     $t \leftarrow r$
5:     while BifurcationFound = False do
6:         $s \leftarrow$ longest proper suffix of t that appears (anywhere) in another read, but preceded by a distinct symbol
7:         $X \leftarrow$ symbol of $t$ preceding $s$
8:         $U \leftarrow$ {segments $XsK$ appearing in $\mathcal{R}$ for some $K \in \Sigma$}
9:         if $U = \emptyset$ then
10:            $t \leftarrow s$
11:        else if $|U| = 1$ then
12:            $r \leftarrow rA$, where $U = \{XsA\}$
13:            $t \leftarrow XsA$
14:        else if $|U| = 2$ then
15:            BifurcationFound $\leftarrow$ True
16:            $U_{\text{right}}(r) \leftarrow \{XsA, XsB\}$
17:        else
18:            B1 must have been violated
19:     Repeat for left extensions (obtaining $U_{\text{left}}(r)$ instead)
20: Output: Set of extended reads $\mathcal{R}$.
```

Algorithm 2 works by finding reads that overlap with r, and using them to determine what the possible next bases are. For each read r, in line 6, we carefully choose a suffix s and then look for occurrences of XsK for some $K \in \Sigma$ in any other reads to form the set U, as illustrated in Fig. 6. We will later prove that the suffix s always exists. If U is empty, we return to line 6 and consider a shorter suffix of r. If U has a single element XsA, we extend r by A. If U has two elements XsA

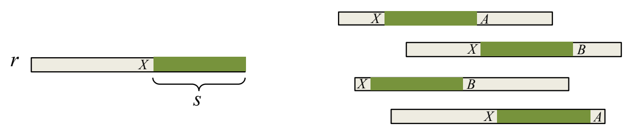


Fig. 6. After the suffix s of r is defined in line 6 of Algorithm 2, we look for reads containing a string XsK for some $K \in \Sigma$. In this case, we would have $U = \{XsA, XsB\}$.

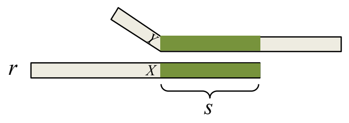


Fig. 7. In line 6 of Algorithm 2, we consider the longest suffix s of r (or the longest suffix that is shorter than the previously considered s) that appears in another read preceded by a distinct symbol.

and XsB, we conclude that we must be at the end of a repeat and a "bifurcation" should happen. So we set BifurcationFound to be true, and exit the loop.

A key aspect of this procedure is the selection of the suffix s, which determines the size of the match we are looking for. Intuitively, if a read overlaps with r by a large amount, we should trust that it gives us the correct next base, whereas if a read overlaps with r by a small amount, this is likely to be a spurious match. To determine the amount of overlap that is "enough" to be trustworthy, we look for a suffix s of r that appears on a different read preceded by a different symbol, as shown in Fig. 7. To understand the choice of s, consider the following definition.

Definition 1. *A read r's triple-suffix is the longest suffix of r that is a triple repeat in the genome.*

A read r's triple suffix z tells us the minimum overlap that we consider reliable. Although we cannot always determine this quantity exactly, it turns out that the suffix s chosen in line 6 is always an overestimate.

Theorem 2. *Suffix s is always at least as long as r's triple-suffix.*

The reason why an s that is at least as long as r's triple-suffix is trustworthy is that, whenever conditions B1, B2 and B3 are satisfied, our extension operations are never in error. Hence Algorithm 2 never produces a read that could not have come from the genome, nor does it cause the set of reads to violate any of our initial conditions.

Theorem 3. *The Extension Algorithm produces a set of reads that continue to obey constraints B1, B2, and B3.*

After extending our reads in Algorithm 2, we have a set of reads that end at precisely-double repeats, as illustrated in Fig. 4(c). These repeats make the correct next base ambiguous. However, although the next base itself is ambiguous, finding the precisely-double repeats still allows us to resolve some additional ambiguity. We do so by merging reads together in Algorithm 3. Notice that in Algorithm 2, whenever we found a bifurcation in line 15, we recorded "signatures" $U_{\text{right}}(r) = \{XsA, XsB\}$ that should identify the two possible extensions of r to the right (and $U_{\text{left}}(r)$ for the possible left extensions). In Algorithm 3 we use these signatures to guide the merging operations.

As in the case of the extension algorithm, in the appendix we show that Algorithm 3 does not make any mistakes:

Algorithm 3 Merging Algorithm

```
1: Input: $\mathcal{R}$ after extension from Algorithm 2
2: for $r \in \mathcal{R}$ do
3:     Let $\{XsA, XsB\} = U_{\text{right}}(r)$. Merge all reads
         with $XsA$ as $r_1$ and all reads with $XsB$ as $r_2$
4:     If $r$ is inconsistent with $r_1$, merge it with $r_2$
5:     If $r$ is inconsistent with $r_2$, merge it with $r_1$
6:     If $r$ is contained in both $r_1$ and $r_2$, discard it.
7:     Canonical successors: $S(r) \leftarrow \{r_1, r_2\}$
8:     Repeat for left extensions (and compute canonical
         predecessors $P(r)$ instead)
9: Output: New set of long reads $\widetilde{\mathcal{R}}$ and two canonical succes-
     sors $S(r)$ and predecessors $P(r)$ for $r \in \widetilde{\mathcal{R}}$
```

Theorem 4. *The Merging Algorithm produces a set of reads that continue to obey constraints B1, B2, and B3.*

Although we loop over $r \in \mathcal{R}$ in the algorithm, we point out that strictly speaking this loop is not well-defined as we are modifying the set $\mathcal{R}$ as we loop through it. We present the algorithm in this way for simplicity. In reality, one would process reads in a queue, and additionally reprocess certain reads as necessary (whenever their successors are merged).

V. Building an Eulerian graph from extended reads

After the merging part of the algorithm, we obtain a set of long reads $\widetilde{\mathcal{R}}$ that stretch between pairs of unbridged repeats, as illustrated in Figure 4(c). In addition, Algorithm 3 outputs, for every long read r, a pair of canonical successors, say r_1 and r_2. From the canonical successor/predecessor relationships, we will construct the final Eulerian graph G that will allow us to figure out the correct ordering of the long reads. First, we present several technical observations that guarantee that the construction of G is well defined and will satisfy certain properties.

To begin, let's consider the current state of the set of reads. The following lemma about the canonical successors of a read r (and the analogous statement for predecessors) follows by construction from Algorithm 3.

Lemma 2. *After Algorithm 3, each read r has two canonical successors r_1 and r_2 such that:*

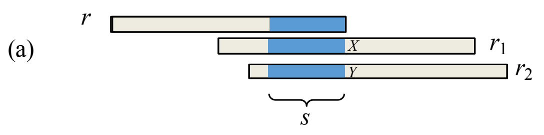
(a) r has a suffix s that is precisely a double repeat, and such that r_1 and r_2 contain sX, sY for some $X \neq Y$ (see Fig. 8(b)), and no other read contains sK for any K.
(b) r is not contained within both r_1 and r_2.
(c) r is consistent with both r_1 and r_2.

Our eventual goal is to show that the reads can be grouped into (non-disjoint) groups of four that all overlap on a particular substring, as shown in Fig. 8(b). First, we show that a read's two successors must have the same overlap.

Lemma 3. *A read r has the same overlap z with its successors r_1 and r_2, and is contained in neither.*

Now we can demonstrate another type of symmetry: predecessors and successors are opposites in the natural sense.

Corollary 1. *If r_1 is one of r's canonical successors, then r is one of r_1's canonical predecessors.*



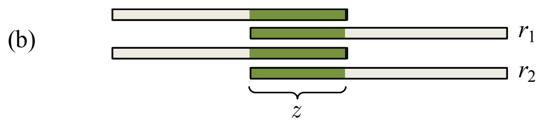


Fig. 8. From the original set of reads (a), merging produces sets of reads that overlap precisely at unbridged double repeats (b).

With these technical observations, we can prove the following theorem, which shows that every read in the graph can be grouped into a four-read structure as the one shown in Fig. 8(b).

Theorem 5. *If r has successors r_1 and r_2, then r_1 and r_2 both have predecessors r and r' for some r'.*

Proof. Suppose r has successors r_1 and r_2. By Corollary 1, r is one of r_1's predecessors; let the other be r'. By Lemma 3, r overlaps by z with both r_1 and r_2, and by the analogous version of Lemma 3 for predecessors, r' also overlaps by z wth r_1.

All of these reads contain zK or Kz for some $K \in \Sigma$. Thus if we define s as in Lemma 2(a) and s' as in the predecessor version of Lemma 2(a), all these reads contain sK or Ks' for some $K \in \Sigma$. Thus, r_1 and r_2 are the successors of r and r', and vice versa, as shown in Fig. 9(a). □

Note that for any four-read configuration implied by Theorem 5, the mutual overlap z must be an unbridged double repeat, since no read other than these four contain zK or Kz for $K \in \Sigma$.

Now we have groups of reads matched in this manner: two start and two end reads, where the end reads are the start reads' successors, and vice versa. We will now construct a graph on all reads. First, for each read we will create a node. Second, for every group, we create a node corresponding to the unbridged repeat, and edges as shown in Fig. 9(b).

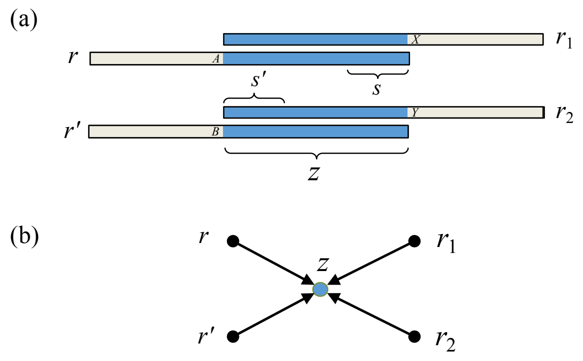


Fig. 9. (a) All read overlaps occur in a four-read configuration. (b) The four-read configurations are used to construct a graph.

The correct genome corresponds to some Eulerian cycle through this graph in the natural sense, because every read r must be succeeded by either r_1 or r_2 and r' will then be succeeded by the other one, which determines an Eulerian cycle. Finally, we have the following result regarding the uniqueness of Eulerian cycles, which is proved in the appendix.

Lemma 4. *Suppose a graph G is Eulerian and every node has in-degree and out-degree at most 2. If there are multiple distinct Eulerian cycles in G, then any Eulerian cycle must visit two vertices u and v in an interleaved manner; i.e., u, v, u, v.*

Since, in our construction, only the unbridged double repeat nodes have degree more than one, Lemma 4 implies that our constructed graph only has multiple Eulerian cycles if s has unbridged interleaved repeats. We conclude that if B1 is satisfied our constructed Eulerian graph has a unique Eulerian cycle, which must correspond to the genome sequence g. Finally, we confirm that the entire algorithm can be implemented efficiently.

Theorem 6. *Suppose that we have N reads with read lengths bounded by some fixed constant $L_{\max}$, and that the coverage depth $c \triangleq \left(\sum_{i=1}^{N} L_i\right)/G$ (the average number of reads covering a symbol in g) is a constant. Then the algorithm can be implemented in $O\left(N^3\right)$ time.*

VI. Concluding Remarks

In this paper, we described an overlap-based assembly algorithm with performance guarantees under the assumption of error-free reads. However, the overlap-based nature of the algorithm makes it amenable to be modified to handle read errors. The algorithm relies on string operations such as testing whether one string is contained within another and whether two strings share a substring, which have natural approximate analogues. For example, instead of testing whether two strings overlap, we can test whether they approximately overlap (with few errors). The bridging conditions then translate to approximate analogues (e.g. instead of triple repeats being all-bridged, we must have *approximate* triple repeats being all-bridged), and the sufficiency proofs should naturally translate to this scenario.

We point out that real sequencing datasets are generally large and require linear- or near-linear-time algorithms, making the $O(N^3)$ complexity guaranteed by Theorem 6 impractical. However, we expect that minor changes in the algorithm design and analysis can yield an algorithm which is, in practice, at most quadratic and possibly near-linear.

VII. Appendix

Here we present several proofs referred to throughout the text.

A. Proof of Lemma 1

Proof. Suppose that r is a triple repeat. In particular, r must be contained in some maximal triple repeat s. If all copies of s are preceded by the same base, say X, then Xs would also be a triple repeat, meaning that s would not be maximal. Thus, s must be preceded by at least two different bases, say X and Y. If B1 holds, then s is all-bridged, so Xs and Ys must appear in two reads, and these two reads contain r and are inconsistent with each other, so that r is removed. Thus, all triple repeats are removed (B3).

Now suppose r was removed. r appears in two inconsistent reads, so it is at least a double repeat. If r is a triple repeat, it must be all-bridged, so that r is subsumed in one of the bridging reads and can be discarded. Otherwise, r is precisely a double repeat. Suppose it is contained within both r_1 and r_2. Since the two reads are inconsistent, they must correspond precisely to the two distinct locations where r may be in the genome. r must be subsumed within either r_1 and r_2, and can be discarded. Thus, removing a read never violates B1 or B2. □

B. Proof of Theorem 2

Proof. First we prove a useful lemma.

Lemma 5. *If r's triple suffix z is a proper suffix of t before line 6, then it will be a suffix of s after line 6.*

Subproof of Lemma 5. Let r's triple-suffix be a string z preceded by X. Then Xz cannot also be a triple repeat (otherwise it would be a triple-suffix longer than z, a contradiction). Thus, of the three copies of z, they cannot all be preceded by X; all but one or two must be preceded by a different base, say Y, as shown below.

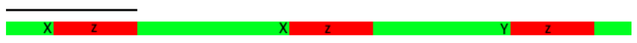


Fig. 10. Xs is at most a double repeat.

Since this copy must be bridged, there must be a read containing the string "Yz". Also, z is a proper suffix of t by assumption. Thus, z satisfies the conditions described in line 6; and since line 6 searches for the longest string satisfying those conditions, s will be at least as long as r's triple-suffix. That is to say, z is a suffix of s. □

With this lemma, we can proceed by induction.

First we consider the base case, when we set s on line 6 with $t = r$. Firstly, r is guaranteed to have some triple-suffix z for nontrivial genomes, since each base in A,G,C,T should appear at least three times in the genome; this means that at least the last base of r is a triple repeat. Now r's triple-suffix z cannot be r itself, by B3; thus z is a proper suffix of $r = t$. This satisfies the conditions of Lemma 5.

Then we consider all three cases in the next code block.

(a) If we proceed to line 9, we assign t = s and loop. We set t = s only if $U = \emptyset$, that is, s is unbridged. We prove by contradiction that r's triple suffix z is a proper suffix of t. Suppose not; then t is a suffix of z. Since z is a triple repeat, it must be bridged, and t, a substring of z, must also be bridged, a contradiction. Again, this allows us to use Lemma 5, and we are done.

(b) If we proceed to line 11, we set t = XsA. Since s is at least as long as r's triple-suffix, sA is at least as long as rA's triple suffix. If not, some XsA is a triple repeat, but then Xs is also a triple repeat that is longer than s, a contradiction. Now since sA is at least as long as rA's triple suffix, and sA is a proper suffix of t = XsA, rA's triple suffix will also be a proper suffix of XsA, so we can use Lemma 5.

(c) If we proceed to line 14, we exit the loop, so that the invariant is irrelevant.

(d) If we proceed to line 17, we see at least three different strings (say) XsA, XsB, and XsC. Thus Xs is a triple repeat, and it is longer than s, a contradiction to the inductive hypothesis. Thus this line can never be reached.

Note that this guarantees that line 6 never fails, because z is always a suffix of t and therefore there is at least one valid candidate for the value s. □

C. Proof of Theorem 3

Proof. The only place where the set of reads is modified is line 12. Here, s is a double repeat, because both Xs and Ys are substrings of some read. Suppose s is precisely a double repeat. Then, since s appears both in Xs and in Ys, both Xs and Ys appear only once in the genome (and s appears once in each, i.e., twice in total), as shown in Fig. 11. Since Xs appears once

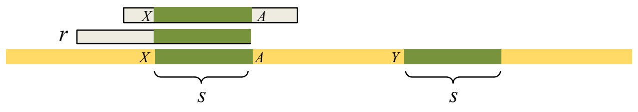


Fig. 11. Extension step when s is a double repeat.

in the genome and a read contains XsA, the next base of r must be A.

Alternatively, suppose s is not precisely a double repeat, i.e. it is a triple repeat. Then s must be all-bridged by assumption B1. If r were succeeded by some base other than A (say, B), then the string XsB would appear in the genome. Since s is all-bridged, XsB would also appear in some read. But it does not, so r again must be suceeded by A, as shown in Fig. 12. Thus, we

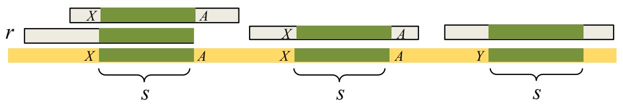


Fig. 12. Extension step when s is a triple repeat.

can extend r by A safely. Clearly, a correct extension operation cannot violate any of the conditions B1, B2, and B3. □

D. Proof of Theorem 4

Proof. We are given $U(r) = \{XsA, XsB\}$ which were carried over from the extension algorithm. We claim that both XsA and XsB appear only once in the genome each. If either of them appeared more than once, then Xs would be a triple repeat that is longer than s, and thus longer than r's triple repeat (by Theorem 2), a contradiction.

So, since XsA and XsB appear only once each in the genome, we can merge all reads with XsA into a single read, and similarly with XsB. Also, since Xs appears only twice in the genome, if r is inconsistent with r_1, it must be aligned with r_2 and can be merged with r_2; and vice versa. If r is contained in both r_1 and r_2, it cannot contribute to B1, B2 and B3, and can be discarded. □

E. Proof of Lemma 3

Proof. Suppose towards a contradiction that r has two successors r_1 and r_2, where r has strictly greater overlap with r_1 than with r_2, with z being the shorter overlap, as shown in Fig. 13. Read

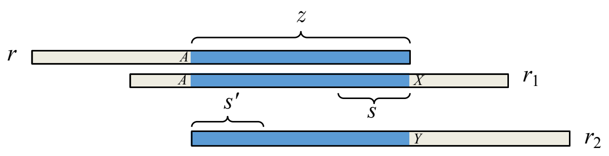


Fig. 13. Two successors with different overlaps produces a contradiction.

r_2 may not contain r, otherwise both r_1 and r_2 would contain

r, contradicting Lemma 2(b). Now, if we let s be the suffix of r as in Lemma 2(a), s must be a suffix of z. Since s is precisely a double repeat (once followed by X and once followed by Y), zX only appears once in the genome. Now if we apply the predecessor version of Lemma 2 to r_2, r_2 begins with some s' and has predecessors containing As' and Bs' respectively. Since s' is precisely a double repeat, it cannot contain zX (which only appears once) and must be a prefix of z. Since s' can only be preceded by A or B, we can assume wlog that r_1 contains As' as depicted. But then r and r_1 both contain As', contradicting (the predecessor version of) Lemma 2(a), which states that only one read may contain As'. Thus a read r must have the same overlap with both successors, and cannot be contained in either, or it would be contained in both, contradicting Lemma 2. □

F. Proof of Corollary 1

Proof. By (the predecessor version of) Lemma 2, r_1 begins with a double repeat s' and has predecessors containing As' and Bs' respectively, as shown in Fig. 14. If we let z be the overlap

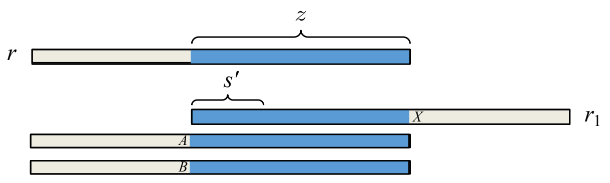


Fig. 14. Successors/predecessors are symmetric.

between r and r_1 and X be the symbol following z in r_1, as we argued in the proof of Lemma 3, zX can only appear once in the genome. Hence, since s' is a double repeat, it must be a prefix of z.

Moreover, r cannot be contained within r_1 by Lemma 3. Since by (the predecessor version of) Lemma 2, only the predecessors of r_1 may contain Ks' for some $K \in \Sigma$, r must be one of the predecessors of r_1. □

G. Proof of Lemma 4

Proof. Since G contains an Eulerian cycle, $d_{in}(v) = d_{out}(v)$ for every v (the in- and out-degrees are equal). Moreover, any node v with $d_{in}(v) = d_{out}(v) = 1$ can only be traversed by an Eulerian cycle in a unique way, so it can be "condensed", and the Eulerian cycles in the resulting graph G' are in one-to-one correspondence with the Eulerian cycles in G. Similarly, if a node v has a self-loop, v can be condensed, since there is a unique way in which any Eulerian cycle can traverse it. Therefore, we may prove the lemma assuming the special case where $d_{in}(v) = d_{out}(v) = 2$ for every v, and there are no self-loops.

To prove the forward direction, suppose an Eulerian cycle C visits u and v in an interleaved manner. Then C can be partitioned into four paths, two from u to v and two from v to u. By considering the two possible orderings of these four paths into a cycle, we obtain two distinct Eulerian cycles.

For the backwards direction, suppose no such nodes u and v exist. If after the condensation steps described above, $n = 1$, the Eulerian cycle must be unique. So suppose $n \geq 2$, and consider the order in which C visits the nodes in $v_1, ..., v_n$, each one twice, starting from v_1. Suppose v_a is the first node to be visited twice. Since there is no self-loop, there must be a node v_b that was visited between the first and second visits to v_a. Therefore, v_a and v_b are visited in an interleaved manner, which is a contradiction. □

H. Proof of Theorem 6

Proof. In many portions of the algorithm, we will need to find bridging reads. To make this more efficient, we can preprocess each read by hashing all of its substrings, which takes $O(N)$ time. Using this hash table, we can implement Algorithm 1 in constant time per read, or $O(N)$ time.

In Algorithm 2, we can loop only $O(G) = O(N)$ times for each read, because either t becomes shorter or the read becomes longer every time we loop; and if we loop G times we have already assembled the entire genome and can terminate. Each loop takes constant time using our hash table to find bridging reads. Thus the total time is $O\left(N^2\right)$.

After Algorithm 2, we will recompute the hash table. This step takes $O\left(N^3\right)$ time because the reads may have grown in size up to $O(G)$. Then, in Algorithm 3, we perform some additional constant-time bridging checks, and merge some reads. Each merge takes $O(N)$ time and reduces the number of reads by one, thus this phase is $O\left(N^2\right)$ time.

Finally, we construct a graph and traverse an Eulerian cycle in it, which takes linear time. This completes the algorithm. □

References

[1] G. Bresler, M. Bresler, and D. Tse, "Optimal assembly for high throughput shotgun sequencing," *BMC Bioinformatics*, 2013.
[2] E. W. Myers, "The fragment assembly string graph," *Bioinformatics*, vol. 21, pp. 79–85, 2005.
[3] P. A. Pevzner, H. Tang, and M. S. Waterman, "An eulerian path approach to dna fragment assembly," *Proceedings of the National Academy of Sciences*, vol. 98, no. 17, pp. 9748–9753, 2001.
[4] J. T. Simpson and R. Durbin, "Efficient construction of an assembly string graph using the fm-index," *Bioinformatics*, vol. 26, pp. 367–373, 2010.
[5] C.-S. Chin, D. H. Alexander, P. Marks, A. A. Klammer, J. Drake, C. Heiner, A. Clum, A. Copeland, J. Huddleston, E. E. Eichler *et al.*, "Nonhybrid, finished microbial genome assemblies from long-read smrt sequencing data," *Nature methods*, vol. 10, no. 6, pp. 563–569, 2013.
[6] G. Myers, "Efficient local alignment discovery amongst noisy long reads," in *Algorithms in Bioinformatics*. Springer, 2014, pp. 52–67.
[7] N. Nagarajan and M. Pop, "Parametric complexity of sequence assembly: theory and applications to next generation sequencing," *Journal of computational biology*, vol. 16, no. 7, pp. 897–908, 2009.
[8] P. Medvedev, K. Georgiou, G. Myers, and M. Brudno, "Computability of models for sequence assembly," in *Algorithms in Bioinformatics*. Springer, 2007, pp. 289–301.
[9] I. Shomorony, S. Kim, T. Courtade, and D. Tse. Information-Optimal Assembly via Sparse Read-Overlap Graphs. [Online]. Available: http://www.eecs.berkeley.edu/~courtade/pdfs/NSG.pdf
[10] J. Hui, I. Shomorony, K. Ramchandran, and T. Courtade, "Genome Assembly from Variable-Length Reads," 2016. [Online]. Available: http://www.eecs.berkeley.edu/~courtade/pdfs/VarLengthISIT2016.pdf