Sequence analysis

Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences

Heng Li

Medical Population Genetics, Broad Institute, Cambridge, MA 02142, USA

Associate Editor: Inanc Birol

Received on December 6, 2015; revised on March 14, 2016; accepted on March 14, 2016

Abstract

Motivation: Single Molecule Real-Time (SMRT) sequencing technology and Oxford Nanopore technologies (ONT) produce reads over 10 kb in length, which have enabled high-quality genome assembly at an affordable cost. However, at present, long reads have an error rate as high as 10–15%. Complex and computationally intensive pipelines are required to assemble such reads.

Results: We present a new mapper, minimap and a de novo assembler, miniasm, for efficiently mapping and assembling SMRT and ONT reads without an error correction stage. They can often assemble a sequencing run of bacterial data into a single contig in a few minutes, and assemble 45-fold Caenorhabditis elegans data in 9 min, orders of magnitude faster than the existing pipelines, though the consensus sequence error rate is as high as raw reads. We also introduce a pairwise read mapping format and a graphical fragment assembly format, and demonstrate the interoperability between ours and current tools.


Contact: hengli@broadinstitute.org

Supplementary information: Supplementary data are available at Bioinformatics online.

1 Introduction

High-throughput short-read sequencing technologies, such as Illumina, have empowered a variety of biological researches and clinical applications that would not be practical with the older Sanger sequencing. However, the short read length (typically a few hundred basepairs) has posed a great challenge to de novo assembly as many repetitive sequences and segmental duplications are longer than the read length and can hardly be resolved by short reads even with paired-end data (Alkan et al., 2011). Although with increased read length and improved algorithms we are now able to produce much better short-read assemblies than a few years ago, the contiguity and completeness of the assemblies are still not as good as Sanger assemblies (Chaisson et al., 2015).

The PacBio’s SMRT technology were developed partly as an answer to the problem with short-read de novo assembly. However, due to the high per-base error rate, around 15%, these reads were only used as a complement to short reads initially (Bashir et al., 2012; Koren et al., 2012; Ribeiro et al., 2012), until Chin et al. (2013) and Koren et al. (2013) demonstrated the feasibility of SMRT-only assembly. Since then, SMRT is becoming the preferred technology for finishing small genomes and producing high-quality Eukaryotic genomes (Berlin et al., 2015).

Oxford Nanopore Technologies (ONT) has recently offered another long-read sequencing technology. Although the per-base error rate was high at the early access phase (Quick et al., 2014), the latest data quality has been greatly improved. Loman et al. (2015) confirmed that we can achieve high-quality bacterial assembly with ONT data alone.

Published long-read assembly pipelines all include four stages: (i) all-vs-all raw read mapping, (ii) raw read error correction, (iii) assembly of error corrected reads and (iv) contig consensus polish. Stage (iii) may involve all-vs-all read mapping again, but as the error rate is much reduced at this step, it is easier and faster than stage (i). Table 1 shows the tools used for each stage. Notably, our tool minimap is a raw read overlapper and miniasm is an assembler. We do not correct sequencing errors, but instead directly produce unpolished and uncorrected contig sequences from raw read overlaps. The idea of correction-free assembly was inspired by talks given by Gene
Myers, Sikic et al. (personal communication) are also independently exploring such an approach.

As we can see from Table 1, each stage can be achieved with multiple tools. Although we have successfully combined tools into different pipelines, we need to change or convert the input/output formats to make them work together. Another contribution of this article is the proposal of concise mapping and assembly formats, which will hopefully encourage modular design of assemblers and the associated tools.

2 Methods

2.1 General notations

Let $\Sigma = \{A, C, G, T\}$ be the alphabet of nucleotides. For a symbol $a \in \Sigma$, $a$ is the Watson-Crick complement of $a$. A string $s = a_1a_2 \cdots a_n$ over $\Sigma$ is also called a DNA sequence. Its length is $|s| = n$; its reverse complement is $\overline{s} = \overline{a_2} \cdots \overline{a_n} = \ldots \overline{a_1}$. For convenience, we define strand function $\pi : \Sigma^* \times \{0, 1\} \rightarrow \Sigma^*$ such that $\pi(s, 0) = s$ and $\pi(s, 1) = \overline{s}$. Here $\Sigma^*$ is the set of all DNA sequences.

By convention, we call a $k$-long DNA sequence as a $k$-mer. We use the notation $s^k = a_{i-1} \cdots a_{i+k-1}$ to denote a $k$-long substring of $s$ starting at $i$. $\Sigma^k$ is the set of all $k$-mers.

2.2 Minimap

2.2.1 Overview of k-mer based sequence similarity search

BLAST (Altschul et al., 1997) and BLAT (Kent, 2002) are among the most popular sequence similarity search tools. They use one $k$-mer hash function $\phi : \Sigma^k \rightarrow Z$ to hash $k$-mers at the positions 1, $w+1, 2w+1, \ldots$ of a target sequence and keep the hash values in a hash table. Upon query, they use the same hash function on every $k$-mer of the query sequence and look up the hash table for potential matches. If there are one or multiple $k$-mers in a small window, these aligners extend the matches with dynamic programming to construct the final alignment.

DALIGNER (Myers, 2014) does not use a hash table. It instead identifies $k$-mer matches between two sets of reads by sorting $k$-mers and merging the sorted lists. DALIGNER is fast primarily because sorting and merging are highly cache efficient.

MHAP (Berlin et al., 2015) differs from others in the use of MinHash sketch (Broder, 1997). Briefly, given a read sequence $s$ and $m$ $k$-mer hash functions $\{\phi_i\}_{1 \leq i \leq m}$, MHAP computes $h_i = \min\{\phi_i(s^k) : 1 \leq i \leq |s| - k + 1\}$ with each hash function $\phi_i$, and takes list $(h_i)_{1 \leq i \leq m}$, which is called the sketch of $s$, as a reduced representation of $s$. Suppose $(h_i)$ and $(h'_i)$ are the sketches of two reads, respectively. When the two reads are similar to each other or have significant overlaps, there are likely to exist multiple $j$ such that $h_j = h'_j$. Potential matches can thus be identified. A limitation of MinHash sketch is that it always selects a fixed number of hash values regardless of the length of the sequences. This may waste space or hurt sensitivity when input sequences vary greatly in lengths.

Minimap is heavily influenced by all these works. It adopts the idea of sketch like MHAP but takes minimizers (Roberts et al., 2004; Schleimer et al., 2003) as a reduced representation instead; it stores $k$-mers in a hash table like BLAT and MHAP but also uses sorting extensively like DALIGNER. In addition, minimap is designed not only as a read overapper but also as a read-to-genome and genome-to-genome mapper. It has more potential applications.

2.2.2 Computing minimizers

Loosely speaking, a $(w, k)$-minimizer of a string is the smallest $k$-mer in a surrounding window of $w$ consecutive $k$-mers. Formally, let $\phi : \Sigma^w \rightarrow Z$ be a $k$-mer hash function. A double-strand $(w, k, \phi)$-minimizer, or simply a minimizer, of a string $s$, $|s| \geq w + k - 1$, is a triple $(h, i, r)$ such that there exists $\max(1, i - w + 1) \leq j \leq \min(i, |s| - w - k + 1)$ which renders

$$h = \phi(s^k, 1) = \min(\phi(s^k, 1), \phi(s^k, 2), \ldots, \phi(s^k, r)) : 0 \leq p < w, r' \in \{0, 1\}$$

Let $M(s)$ be the set of minimizers of $s$. Algorithm 1 gives the pseudocode to compute $M(s)$ in $O(w \cdot |s|)$ time. Our actual implementation is close to $O(|s|)$ in average case. It uses a queue to cache the previous minimals and avoids the loops at line 1 and 2 most of time. In practice, time spent on collecting minimizers is insignificant.

A natural choice of hash function $\phi$ is to let $\phi(A) = 0$, $\phi(C) = 1$, $\phi(G) = 2$ and $\phi(T) = 3$ and for a $k$-mer $s = a_1 \cdots a_n$, define

$$\phi(s) = \phi(a_1) \times 4^{k-1} + \phi(a_2) \times 4^{k-2} + \cdots + \phi(a_n)$$

This hash function always maps a $k$-mer to a distinct 2$k$-bit integer. A problem with this $\phi$ is that poly-A, which is often highly

<table>
<thead>
<tr>
<th>Functionality</th>
<th>Program</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raw read overlap</td>
<td>BLASR</td>
<td>Chaisson and Tesler (2012)</td>
</tr>
<tr>
<td></td>
<td>DALIGNER</td>
<td>Myers (2014)</td>
</tr>
<tr>
<td></td>
<td>MHAP</td>
<td>Berlin et al. (2015)</td>
</tr>
<tr>
<td></td>
<td>GraphMap</td>
<td>Sovic et al. (2015)</td>
</tr>
<tr>
<td></td>
<td>minimap</td>
<td>this article</td>
</tr>
<tr>
<td></td>
<td>nanocorrect</td>
<td>Loman et al. (2015)</td>
</tr>
<tr>
<td>Assembly</td>
<td>wgs-assembler</td>
<td>Myers et al. (2000)</td>
</tr>
<tr>
<td></td>
<td>miimasn</td>
<td>this article</td>
</tr>
<tr>
<td></td>
<td>nanopolish</td>
<td>Loman et al. (2015)</td>
</tr>
</tbody>
</table>

Table 1. Tools for noisy long-read assembly

<table>
<thead>
<tr>
<th>Algorithm 1: Compute minimizers</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input: Parameter</strong> $w$ and $k$ and sequence $s$ with $</td>
</tr>
<tr>
<td><strong>Output:</strong> $(w,k)$-minimizers, their positions and strands</td>
</tr>
<tr>
<td><strong>Function</strong> MinimizerSketch($s$, $w$, $k$) begin</td>
</tr>
<tr>
<td>$M \leftarrow \emptyset$ [ NB: $M$ is a set; no duplicates ]</td>
</tr>
<tr>
<td>for $i$ from 1 to $</td>
</tr>
<tr>
<td>$m \leftarrow \infty$</td>
</tr>
<tr>
<td>for $j$ from 0 to $w - 1$ do</td>
</tr>
<tr>
<td>$(u, v) \leftarrow (\phi(s^k, i), \phi(s^k, j))$</td>
</tr>
<tr>
<td>if $u \neq v$ then</td>
</tr>
<tr>
<td>$m \leftarrow \min(m, \min(u, v))$</td>
</tr>
<tr>
<td>$M \leftarrow M \cup {m, \min(u, v)}$</td>
</tr>
<tr>
<td>for $j$ from 0 to $w - 1$ do</td>
</tr>
<tr>
<td>$(u, v) \leftarrow (\phi(s^k, i), \phi(s^k, j))$</td>
</tr>
<tr>
<td>if $u &lt; v$ and $u = m$ then</td>
</tr>
<tr>
<td>$M \leftarrow M \cup {(m, i, j, 0)}$</td>
</tr>
<tr>
<td>else if $v &lt; u$ and $v = m$ then</td>
</tr>
<tr>
<td>$M \leftarrow M \cup {(m, i, j, 1)}$</td>
</tr>
<tr>
<td>return $M$</td>
</tr>
</tbody>
</table>
enriched in genomes, always gets zero, the smallest value. We may oversample these non-informative poly-A and hurt practical performance. To alleviate this issue, we use function $\phi = \left( x \oplus (x \ll 21) \right) \ll mn$
\begin{align*}
x &\leftarrow x \ll 21 \\
x &\leftarrow x \ll 24 \\
x &\leftarrow (x + (x \ll 3) + (x \ll 8)) \ll mn \\
x &\leftarrow x \ll 14 \\
x &\leftarrow (x + (x \ll 2) + (x \ll 4)) \ll mn \\
x &\leftarrow x \ll 28 \\
x &\leftarrow (x + (x \ll 31)) \ll mn
\end{align*}
return $x$

Algorithm 2: Invertible integer hash function

Input: $p$-bit integer $x$
Output: hashed $p$-bit integer

Function InvertibleHash($x, p$) begin
$
\text{m} \leftarrow 2^p - 1 \\
x \leftarrow (x + (x \ll 21)) \ll \text{m} \\
x \leftarrow x \ll 24 \\
x \leftarrow (x + (x \ll 3) + (x \ll 8)) \ll \text{m} \\
x \leftarrow x \ll 14 \\
x \leftarrow (x + (x \ll 2) + (x \ll 4)) \ll \text{m} \\
x \leftarrow x \ll 28 \\
x \leftarrow (x + (x \ll 31)) \ll \text{m}
$
return $x$

Algorithm 3: Index target sequences

Input: Set of target sequences $T = \{s_1, \ldots, s_T\}$
Output: Minimizer hash table $H$

Function Index($T$, $w$, $k$) begin
$H \leftarrow$ empty hash table
for $t \leftarrow 1$ to $T$ do
\begin{align*}
&\text{foreach } (h, i, \iota) \in M \\
&\quad \quad \quad \{[h] \leftarrow [h] \cup \{(t, i, \iota)\}\}
\end{align*}
return $H$

2.2.3 Indexing

Algorithm 3 describes indexing target sequences. It keeps minimizers of all target sequences in a hash table where the key is the minimizer hash and the value is a set of target sequence index, the position of the minimizer and the strand (packed into one 64-bit integer).

In implementation, we do not directly insert minimizers to the hash table. Instead, we append minimizers to an array of two 64-bit integers (one for minimizer sequence and one for position) and sort the array after collecting all minimizers. The hash table keeps the intervals on the sorted array. This procedure dramatically reduces heap allocations and cache misses, and is supposedly faster than direct hash table insertion.

2.2.4 Mapping

Given two sequences $s$ and $s'$, we say we find a minimizer hit $(h, x, i, i')$ if there exist $(h, i, \iota) \in M(s)$ and $(h, f, \iota') \in M(s')$ with $x = i \oplus i'$ ($\oplus$ is the XOR operator). Here $h$ is the minimizer hash value, $x$ indicates the relative strand and $i$ and $i'$ are the positions on the two sequences, respectively. We say two minimizer hits $(h_1, x_1, i_1, i'_1)$ and $(h_2, x_2, i_2, i'_2)$ are $\epsilon$-away if $1 \times x = 0$ and $(|i_1 - i'_1| - |i_2 - i'_2|) < \epsilon$ or $2 \times x = 1$ and $(|i_1 + i'_1| - |i_2 + i'_2|) < \epsilon$. Intuitively, $\epsilon$-away hits are approximately colinear within a band of width $\epsilon$ (500bp by default). Given a set of minimizer hits $(h, x, i, i')$, we can cluster $i - i'$ for $x = 0$ or $i' - i$ for $x = 1$ to identify long colinear matches. This procedure is inspired by Hough Transformation mentioned by Sovic et al. (2015).

Algorithm 4 gives the details of the mapping algorithm. The loop at line 1 collects minimizer hits between the query and all the target sequences. The loop at line 2 performs a single-linkage clustering to group approximately colinear hits. Some hits in a cluster may not be colinear because two minimizer hits within distance $\epsilon$ are always $\epsilon$-away. To fix this issue, we find the maximal colinear subset of hits by solving a longest increasing sequencing problem (line 3). This subset is the final mapping result. In practical implementation, we set thresholds on the size of the subset (4 by default) and the number of matching bases in the subset to filter poor mappings (100 for read overlapping).

2.3 Assembly graph

Two strings $v$ and $w$ may be mapped to each other based on their sequence similarity. If $v$ can be mapped to a substring of $w$, we say $w$ contains $v$. If a suffix of $v$ and a prefix of $w$ can be mapped to each other, we say $v$ overlaps $w$, written as $v \rightarrow w$. If we regard strings $v$ and $w$ as vertices, the overlap relationship defines a directed edge
between them. The length of $v \rightarrow w$ equals the length of $v$’s prefix that is not in the prefix-suffix match.

Let $G = (V, E, l)$ be a graph without multi-edges, where $V$ is a set of DNA sequences (vertices), $E$ a set of overlaps between them (edges) and $l : E \rightarrow \mathbb{R}_+$ is the edge length function. $G$ is said to be Watson-Crick complete if (i) $\forall v, \tilde{v} \in V$, $\tilde{v} \in V$ and (ii) $\forall v \rightarrow w \in E$, $\tilde{w} \rightarrow \tilde{v} \in E$. $G$ is said to be containment-free if any sequence $v$ is not contained in other sequences in $V$. If $G$ is both Watson–Crick complete and containment-free, it is an assembly graph. By definition, any vertex $v$ has a complement vertex $\overline{v}$ in the graph and any edge $v \rightarrow w$ has a complement edge $w \rightarrow \overline{v}$. Let $\deg^+(v)$ be the outdegree of $v$ and $\deg^- (v)$ be the indegree. It follows that $\deg^+(v) = \deg^- (v)$.

An assembly graph has the same topology as a string graph (Myers, 2005), though the interpretation of the vertex set $V$ is different. In a string graph, $V$ is the set of the two ends of sequences, not the set of forward and reverse-complement sequences. De Bruijn graph can be regarded as a special case of overlap graph. It is also an assembly graph.

In an assembly graph, an edge $v \rightarrow w$ is transitive if there exist $v \rightarrow u$ and $u \rightarrow w$. Removing a transitive edge does not affect the connectivity of the graph. A vertex $v$ is a tip if $\deg^+(v) = 0$ and $\deg^- (v) > 0$. The majority of tips are caused by artifacts or missing overlaps. A bubble is a directed acyclic subgraph with a single source $v$ and a single sink $w$ having at least two paths between $v$ and $w$, and without connecting the rest of the graph. The bubble is tight if $\deg^+(v) > 1$ and $\deg^- (w) > 1$. A bubble may be caused by missing overlaps or by variants between haplotypes in multi-ploidy samples or paralogs. It is preferred to collapse bubbles for high contiguity, though this introduces loss of information.

### 2.4 Miniasm

#### 2.4.1 Trimming reads

Raw read sequences may contain artifacts such as untrimmed adapters and chimaera. The first step of assembly to reduce such artifacts by examining read-to-read mappings. For each read, miniasm computes per-base coverage based on good mappings against other reads (longer than 2000 bp with at least 100 bp non-redundant bases on overlaps or paralogs. It is preferred to collapse bubbles for high contiguity, though this introduces loss of information.

#### 2.4.2 Generating assembly graph

For each trimmed read, miniasm applies Algorithm 5 to classify the mapping (see also Fig. 1 for the explanation of input variables). It ignores internal matches, drops contained reads and adds overlaps to the assembly graph. For a pair of reads, miniasm uses the longest overlap only to avoid multi-edges.

#### 2.4.3 Graph cleaning

After constructing the assembly graph, miniasm removes transitive edges (Myers, 2005), trims tipping unitigs composed of few reads (4 by default) and pops small bubbles (Zerbino and Birney, 2008). Algorithm 6 detects bubbles where the longest path is shorter than $d$ (50 kb by default). It is adapted from Kahn’s topological sorting algorithm (Kahn, 1962). It starts from the potential source and visits a vertex when all its incoming edges are visited before. Algorithm 6 only detects bubbles. We can keep track of the optimal parent vertex at line 1 and then backtrack to collapse bubbles to a single path. Fermi (Li, 2012) uses a similar algorithm except that it keeps two optimal paths through the bubble. Onodera et al. (2013) and Brankovic et al. (2015) have also independently found similar algorithms.

In addition, if $v \rightarrow w_1$ and $v \rightarrow w_2$ exist and $\ell(v \rightarrow w_1) < \ell(v \rightarrow w_2)$, miniasm removes $v \rightarrow w_2$ if $|v| - \ell(v \rightarrow w_2)/|w| - \ell(v \rightarrow w_1)$ is small enough (70% by default). When there are longer overlaps, shorter overlaps after transitive reduction may be due to repeats. However, non-repetitive overlaps may also be removed at a small chance, which leads to missing overlaps and misassemblies.

#### 2.4.4 Generating unitig sequences

If there are no multi-edges in the assembly graph, we can use $v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_k$ to represent a path consisting of $k$ vertices. The sequence spelled from this path is the concatenation of vertex substrings:

$$v_1, \ell(v_1 \rightarrow v_2) \circ v_2, \ell(v_2 \rightarrow v_3) \circ \cdots \circ v_{k-1}, \ell(v_{k-1} \rightarrow v_k) \circ v_k,$$

where $\|v_i, j\|$ is the substring between $i$ and $j$ inclusive, and $\circ$ is the string concatenation operator.

In a transitive reduced graph, a unitig (Myers et al., 2000) is a path $v_1 \rightarrow v_2 \rightarrow \cdots \rightarrow v_k$ such that $\deg^+(v_i) = \deg^- (v_i+1) = 1$ and (i) $v_1 = v_k$ or (ii) $\deg^+(v_1) = 1$ and $\deg^- (v_k) = 1$. Its sequence is the sequence spelled from the path. Intuitively, a unitig is a maximal path on which adjacent vertices can be ‘unambiguously merged’ without affecting the connectivity of the original assembly graph.
As miniasm does not correct sequencing errors, the error rate of unitig sequence is the same as the error rate of the raw input reads. We have not implemented such a consensus tool yet.

### 2.5 Formats: pairwise read mapping format and graphical fragment assembly format

#### 2.5.1 Pairing mapping format

Pairwise read mapping format (PAF) is a lightweight format keeping the key mapping information (Table 2). Miniasm outputs mappings in PAF, which are taken by miniasm as input for assembly. We also provide scripts to convert DALIGNER, MHAP and SAM formats to PAF.

#### 2.5.2 Graphical fragment assembly format

Graphical fragment assembly format (GFA) is a concise assembly format (Table 3; http://bit.ly/gfaspec) initially proposed by us prior to miniasm and later improved by community (Melsted et al., personal communication). GFA has an explicit relationship to an assembly graph—an ‘S’ line in the GFA corresponds to a vertex and its complement in the graph; an ‘L’ line corresponds to an edge and its complement. GFA is able to represent graphs produced at all the stages of an assembly pipeline, from initial read overlaps to the unitig relationship in the final assembly.

FASTG (http://bit.ly/fastgfmt) is another assembly format prior to GFA. It uses different terminologies. A vertex in an assembly graph is called an edge in FASTG, and an edge is called an adjacency. In FASTG, subgraphs can be nested, though no tools work with nested graphs due to technical complications. In addition, with nesting, one assembly graph can be represented in distinct ways, which we regard as a limitation of FASTG.

### 2.6 Evaluating the layout accuracy

Miniasm outputs the approximate positions of trimmed reads on the resulting unitigs. We extract these reads, map to the true assembly with miniasm (option: ‘-L100 -m0 -w5’) and select the best mapping for each read. For a read $i$, let utg$_i$ be the unitig name and rank, be its index on utg, (i.e. read $i$ is the rank$_i$th read on the unitig). If two reads $i$ and $j$ are mapped adjacently on the true assembly, we say the adjacency is w-consistent, if (i) utg$_i$ = utg$_j$, and |rank$_i$ - rank$_j$| < $w$, or (ii) both read $i$ and $j$ are the first or the last $w$ reads of some unitigs. We use $w = 5$ to detect large structural misassemblies.

### 3 Results

#### 3.1 The accuracy of minimap

We mapped a human PacBio run ‘m130928_232712_42213_*.*’ (http://bit.ly/chm1p5c3) with minimap and BWA-MEM (Li, 2013) against GRCh37 plus decoy sequences (http://bit.ly/GRCh37d5). We started from 23 235 reads (131 Mb), filtered out 7593 reads (10 Mb) without ≥2 kb BWA-MEM alignments, and further dropped 815 reads (11 Mb) with two or more ≥2 kb chimeric alignments and 598 reads (4 Mb) with mapping quality below 10. Of the remaining reads, we found only 2.0% not overlapping the best minimap mapping of the same read. The majority of them hit to the decoy sequence where defining the true alignment is challenging as decoy is enriched with incomplete segments of centromeric repeats. If we exclude hits to the decoy, the percentage drops to 0.7%. On this input, minimap is 50 times faster than BWA-MEM, while finding similar best mapping positions. This experiment evaluates both the sensitivity and the specificity of minimap: if minimap had low sensitivity, it would miss the BWA-MEM mapping completely; if minimap had low specificity, its best mapping would often be a wrong mapping.

To test the sensitivity for read overlapping, we aligned all reads from PBcR-PB-ec (Table 4) against the reference genome with BWA-MEM, extracted reads with mapping quality ≥10, and identified ≥2kb overlaps between the extracted reads based on their positions on the reference genome. Minimap finds 93% of these overlaps. It is more sensitive than MHAP in its sensitive mode (78%) but less than DALIGNER (98%).

#### 3.2 Assembling bacterial genomes

We evaluated the performance of miniasm on 17 bacterial datasets (Table 4) with command line ‘minimap -Sw5 -L100 -m0 reads.fa reads.fa | miniasm -r reads.fa -’. Miniasm is able to derive a single contig per chromosome/plasmid for all but four datasets: 3 extra ≥50 kb contigs for ERS554120, and 1 extra contig for ERS605484, PBcR-ONT-ec and MAP-006-pcr-1 each. In the dotter plot between the assembly and the reference genome (similar to Fig. 2), no large-scale misassemblies are observed. We also applied the method in Section 2.6. Except ERS473430, the miniasm layouts are 5-consistent with the reference assemblies. For ERS473430, the NCTC project page claimed the sample has a plasmid. Miniasm gives two contigs, but the NCTC assembly has one contig only. The difference in layout may be an error in the NCTC assembly.
Table 2. Pairwise mapping format (PAF)

<table>
<thead>
<tr>
<th>Col</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>string</td>
<td>Query sequence name</td>
</tr>
<tr>
<td>2</td>
<td>int</td>
<td>Query sequence length</td>
</tr>
<tr>
<td>3</td>
<td>int</td>
<td>Query start coordinate (BED-like)</td>
</tr>
<tr>
<td>4</td>
<td>int</td>
<td>Query end coordinate (BED-like)</td>
</tr>
<tr>
<td>5</td>
<td>char</td>
<td>‘+’ if query and target on the same strand; ‘–’ if opposite</td>
</tr>
<tr>
<td>6</td>
<td>string</td>
<td>Target sequence name</td>
</tr>
<tr>
<td>7</td>
<td>int</td>
<td>Target sequence length</td>
</tr>
<tr>
<td>8</td>
<td>int</td>
<td>Target start coordinate on the original strand</td>
</tr>
<tr>
<td>9</td>
<td>int</td>
<td>Target end coordinate on the original strand</td>
</tr>
<tr>
<td>10</td>
<td>int</td>
<td>Number of matching bases in the mapping</td>
</tr>
<tr>
<td>11</td>
<td>int</td>
<td>Number bases, including gaps, in the mapping</td>
</tr>
<tr>
<td>12</td>
<td>int</td>
<td>Mapping quality (0–255 with 255 for missing)</td>
</tr>
</tbody>
</table>

PAF is TAB-delimited text format with each line consisting of the above fixed fields. When the alignment is available, column 11 equals the total number of sequence matches, mismatches and gaps in the alignment. Column 10 divided by column 11 gives the alignment identity. If the detailed alignment is not available, column 10 and 11 can be approximate. PAF may optionally have additional fields in the SAM-like typed key-value format (Li et al., 2009).

We have also run the PBcR pipeline (Berlin et al., 2015). PBcR requires a spec file. We took ‘pacbio.spec’ from the PBcR-PB-ec sample example and ‘oxford.spec’ from PBcR-ONT-ec, and applied them to all datasets based on their data types. MAP* datasets only provide FASTA sequences for download. We assigned quality 9 to all bases as PBcR requires base quality. PBcR assembled all PacBio datasets without extra contigs longer than 50 kb—better than minimism. However, on the ONT datasets, PBcR produced more fragmented assemblies for MAP-006-2, MAP-006-pcr-1 and MAP-006-pcr-2; the PBcR-PB-ec dataset is 300 kb shorter.

With four CPU cores, it took minimasm 14 s to assemble the 30-fold PBcR-PB-ec dataset and 2 minutes to assemble the 160-fold PB-ecoli dataset. PBcR, with four CPU cores, too, is about 700 times slower on PBcR-PB-ecoli and 60 times slower on PB-ecoli. It is slower on low-coverage data because PBcR automatically switches to the slower sensitive mode. Here we should remind readers that without an error correction stage, the contig sequences generated by minimasm are of much lower accuracy in comparison to PBcR. Nonetheless, minimasm is still tens of times faster than PBcR excluding the time spent on error correction.

### 3.3 Assembling a Caenorhabditis elegans genome

We assembled a 45-fold C.elegans dataset (Table 4). With 16 CPU cores, minimiasm assembled the data in 9 min, achieving an N50 size 2.8 Mb. From the dot plot (Fig. 2), we observed three structural misassemblies (readers are advised to zoom into the vector graph to see the details). PacBio has assembled the same dataset with HGAP3 (Chin et al., 2013). HGAP3 produces shorter contigs (N50 = 1.6 Mb), but does not incur large-scale misassemblies visible from the dot plot between the C.elegans reference genome and the contigs.

When we take the C.elegans reference genome as the truth, the method in Section 2.6 also identifies the three structural misassemblies. The method additionally finds eight intra-unigene and one inter-unigene inconsistencies. In all cases, minimasm agrees with HGAP3, suggesting these inconsistencies may be true structural variations between the reference strain and the sequenced strain.

We have also tried PBcR on this dataset. Based on the intermediate progress report, we estimated that with 16 CPU cores, it would take a week or so to finish the assembly in the automatically chosen ‘sensitive’ mode.

For this dataset, minimap takes 27 GB RAM at the peak. As minimap loads 4 Gbp bases to index, the peak RAM will be capped around 27 GB. The memory used by minimasm is proportional to the number of overlaps. Although it only takes 1.3 GB RAM here, it will become the limiting factor for larger datasets.

### 3.4 Switching read overlappers

Miniasm also works with other overlappers when we convert their output format to PAF. On the 30-fold PBcR-PB-ec dataset, we are
able to produce a single contig with DALIGNER (option -k15–h50), MHAP (option –pacbio-sensitive) and GraphMap (option -w owler). DALIGNER is the fastest, taking 65 s with four CPUs. Minimap is five times as fast on this dataset and is 18 times as fast on PB-ecoli at 160-fold. Minimap is faster on larger datasets possibly because without staging all possible hits in RAM, minimap is able to process more reads in a batch while a large batch usually helps performance. We should note that DALIGNER generates alignments while minimap does not. Minimap would probably have a similar performance if it included an alignment step.

4 Discussions

Miniasm implements the ‘O’ and ‘L’ steps in the Overlap-Layout-Consensus (OLC) assembly paradigm. It confirms long noisy reads can be assembled without an error correction stage, and without this stage, the assembly process can be greatly accelerated and simplified, while achieving comparable contiguity and large-scale accuracy to existing pipelines, at least for genomes without excessive repetitive sequences. Although without the ‘C’ step, miniasm cannot produce high-quality consensus for many analyses, it opens the door to ultrafast assembly if we can develop a fast consensus tool matching the speed of minimap and miniasm. In addition, MinION has a ‘read-until’ mode, allowing users to pause sequencing and reload samples. Fast layout by miniasm could already help to decide if enough data have been collected.

Our main concern with miniasm is that when we look at a low-identity match between two noisy reads, it is difficult to tell whether the low identity is caused by the stochastically higher base error rate on reads, or because reads come from two recent segmental duplications. In comparison, error correction takes the advantage of multiple reads and in theory has more power to distinguish high error rate from duplications/repeats. Bacteria and C.elegans evaluated in this article are repeat sparse. We are yet to know the performance of miniasm given repeat-rich genomes. In addition, miniasm has not been optimized for large repeat-rich genomes. It reads all hits into RAM, which may not be practical when there are too many. We need to filter repetitive hits, introduce disk-based algorithms (e.g. for sorting) or stream hits before removing contained reads. Working with large complex genomes will be an important future direction.

Oxford Nanopore is working on PromethION and PacBio will ship PacBio Sequel later this year. Both sequencers promise significantly reduced sequencing cost and increased throughput, which may stimulate the adoption of long-read sequencing and subsequently the development of long-read mappers and assemblers. We hope in this process, the community could standardize the input and output formats of various tools, so that a developer could focus on a component he or she understands best. Such a modular approach has been proved to be fruitful in the development of short-read tools—in fact, the best short-read pipelines all consist of components developed by different groups—and will be equally beneficial to the future development of long-read mappers and assemblers.

Acknowledgements

We thank Pall Melsted for maintaining the GFA spec and are grateful to Gene Myers, Jason Chin, Adam Phillippy, Jared Simpson, Zamin Iqbal, Nick Loman and Ivan Sovic for their presentations, talks, comments on social media and unpublished works which have greatly influenced and helped the development of minimap and miniasm.

Funding

NHGRI U5HG003037; NIH GM100233.

Conflict of Interest: none declared.

References


