Sequence analysis

A spectral algorithm for fast \textit{de novo} layout of uncorrected long nanopore reads

Antoine Recanati\textsuperscript{1,*}, Thomas Brüls\textsuperscript{2,3,4} and Alexandre d’Aspremont\textsuperscript{1}

\textsuperscript{1}CNRS & D.I., UMR 8548, École Normale Supérieure, Paris, France, \textsuperscript{2}Commissariat à l’Energie Atomique et aux Energies Alternatives, Direction de la Recherche Fondamentale, Genoscope, \textsuperscript{3}UMR 8030, Centre National de la Recherche Scientifique and \textsuperscript{4}Université Paris-Saclay, Evry, France

*To whom correspondence should be addressed.

Received on October 5, 2016; revised on May 17, 2017; editorial decision on June 5, 2017; accepted on June 6, 2017

Abstract

\textbf{Motivation:} New long read sequencers promise to transform sequencing and genome assembly by producing reads tens of kilobases long. However, their high error rate significantly complicates assembly and requires expensive correction steps to layout the reads using standard assembly engines.

\textbf{Results:} We present an original and efficient spectral algorithm to layout the uncorrected nanopore reads, and its seamless integration into a straightforward overlap/layout/consensus (OLC) assembly scheme. The method is shown to assemble Oxford Nanopore reads from several bacterial genomes into good quality (\textit{\textgreater}99\% identity to the reference) genome-sized contigs, while yielding more fragmented assemblies from the eukaryotic microbe \textit{Saccharomyces cerevisiae}.

\textbf{Availability and implementation:} https://github.com/antrec/spectrassembler.

\textbf{Contact:} antoine.recanati@inria.fr

\textbf{Supplementary Information:} Supplementary data are available at \textit{Bioinformatics} online.

1 Introduction

\textit{De novo} whole-genome sequencing seeks to reconstruct an entire genome from randomly sampled sub-fragments whose order and orientation within the genome are unknown. The genome is oversampled so that all parts are covered multiple times with high probability.

High-throughput sequencing technologies such as Illumina substantially reduce sequencing cost at the expense of read length, which is typically a few hundred base pairs long (bp) at best. Yet, \textit{de novo} assembly is challenged by short reads, as genomes contain repeated sequences resulting in layout degeneracies when read length is shorter or of the same order than repeat length (Pop, 2004).

Recent long read sequencing technologies such as PacBio’s SMRT and Oxford Nanopore Technology (ONT) have spurred a renaissance in \textit{de novo} assembly as they produce reads over 10 kb long (Koren and Phillippy, 2015). However, their high error rate (\textit{\textless}15\%) makes the task of assembly difficult, requiring complex and computationally intensive pipelines.

Most approaches for long read assembly address this problem by correcting the reads prior to performing the assembly, while a few others integrate the correction with the overlap detection phase, as in the latest version of the Canu pipeline (Koren et al., 2016) [former Celera Assembler (Myers et al., 2000)].

\textit{Hybrid techniques} combine short and long read technologies: the accurate short reads are mapped onto the long reads, enabling a consensus sequence to be derived for each long read and thus providing low-error long reads [see e.g. Madoui et al. (2015)]. This method was shown to successfully assemble prokaryotic and eukaryotic genomes with PacBio (Koren et al., 2012) and ONT (Goodwin et al., 2015) data. \textit{Hierarchical assembly} follows the same mapping and consensus principle but resorts to long read data only, the rationale being that the consensus sequence derived from all erroneous long reads matching a given position of the genome should be accurate provided there is sufficient coverage and sequencing errors are reasonably randomly distributed: for a given base position on the genome, if 8 out of 50 reads are wrong, the majority vote still yields the correct base. Hierarchical methods map long reads against each other and derive, for each read, a consensus sequence based on all the reads that overlap it. Such an approach was implemented in
HGAP (Chin et al., 2013) to assemble PacBio SMRT data, and more recently by Loman et al. (2015), to achieve de novo assembly of *Escherichia coli* with ONT data exclusively.

Recently, Li (2016) showed that it is possible to efficiently perform de novo assembly of noisy long reads in only two steps, without any dedicated correction procedure: all-versus-all raw read mapping (with minimap) and assembly (with minimas). The minimas assembler is inspired by the Celera Assembler and produces unigens through the construction of an assembly graph. Its main limitation is that it produces a draft whose error rate is of the same order as the raw reads.

Here, we present a new method for computing the layout of raw nanopore reads, resulting in a simple and computationally efficient protocol for assembly. It takes as input the all-versus-all overlap information [e.g. from minimap, MHAP (Berlin et al., 2015) or DALIGNER (Myers, 2014)] and outputs a layout of the reads (i.e. their position and orientation in the genome). Like minimas, we compute an assembly from the all-versus-all raw read mapping, but achieve improved quality through a coverage-based consensus generation process, as in nanocorrect (Loman et al., 2015), although reads are not corrected individually in our case.

The method relies on a simple spectral algorithm akin to Google’s PageRank (Page et al., 1999) with deep theoretical underpinnings, described in Section 2.1. It has successfully been applied to consecutive-ones problems arising in physical mapping of genomes (Atkins and Middendorf, 1996), ancestral genome reconstructions (Jones et al., 2012), or the locus ordering problem (Cheema et al., 2010), but to our knowledge has not been applied to de novo assembly problems. In Section 2.2, we describe an assembler based on this layout method, to which we add a consensus generation step based on POA (Lee et al., 2002), a multi-sequence alignment engine. Finally, we evaluate this pipeline on prokaryotic and eukaryotic genomes in Section 3, and discuss possible improvements and limitations in Section 4.

2 Materials and methods

2.1 Layout computation

We lay out the reads in two steps. We first sort them by position, i.e. find a permutation $\pi$ such that read $\pi(1)$ will be positioned before read $\pi(2)$ on the genome. Then, we iteratively assign an exact position (i.e. leftmost basepair coordinate on the genome) to each read by using the previous read’s position and the overlap information.

The key step is the first one, which we cast as a seriation problem, i.e. we seek to reconstruct a linear order between $n$ elements using unsorted, pairwise similarity information (Atkins et al., 1998; Fogel et al., 2013). Here the $n$ elements are the reads, and the similarity information comes from the overlapper (e.g. from minimap).

The seriation problem is formulated as follows. Given a pairwise similarity matrix $A_{ij}$ and assuming the data has a serial structure, i.e. that there exists an order $\pi$ such that $A_{ij}(\pi(i) - \pi(j))^2$ decreases with $|i - j|$, seriation seeks to recover this ordering $\pi$ (see Fig. 1 for an illustration). If such an order $\pi$ exists, it minimizes the 2-SUM score,

$$2 - \text{SUM}(\pi) = \sum_{i=1}^{n} A_{ij}(\pi(i) - \pi(j))^2,$$

and the seriation problem can be solved as a minimization over the set of permutation vectors (Fogel et al., 2013). In other words, the permutation $\pi$ should be such that if $A_{ij}$ is high (meaning that $i$ and $j$ have a high similarity), then $(\pi(i) - \pi(j))^2$ should be low, meaning that the positions $\pi(i)$ and $\pi(j)$ should be close to each other. Conversely, if $A_{ij} = 0$, the positions of $i$ and $j$ in the new order may be far away without affecting the score. When using seriation to solve genome assembly problems, the similarity $A_{ij}$ measures the overlap between reads $i$ and $j$. In an ideal setting with constant read length and no repeated regions, two overlapping reads should have nearby positions on the genome. We therefore expect the order found by seriation to roughly match the sorting of the positions of the reads.

The problem of finding a permutation over $n$ elements is combinatorial. Still, provided the original data has a serial structure, an exact solution to seriation exists in the noiseless case (Atkins et al., 1998) using spectral clustering, and there exist several convex relaxations allowing explicit constraints on the solution (Fogel et al., 2013).

The exact solution is directly related to the well-known spectral clustering algorithm. Indeed, for any vector $x$, the objective in (1) reads

$$\min_{\pi} \pi^T L_A \pi$$

where $\pi$ is a permutation vector. Roughly speaking, the spectral clustering approach to seriation relaxes the constraint ‘$\pi$ is a permutation vector’ into ‘$\pi$ is a vector of $R^n$ orthogonal to the constant vector $1 = (1, \ldots, 1)^T$ with fixed norm. The problem then becomes

$$\min_{\{\pi \mid \pi^T 1 = 1\}} \pi^T L_A \pi$$

This relaxed problem is an eigenvector problem. Finding the minimum over normalized vectors $x$ yields the eigenvector associated to the smallest eigenvalue of $L_A$, but the smallest eigenvalue, 0, is associated with the eigenvector 1, from which we cannot recover any permutation. However, if we restrict $x$ to be orthogonal to 1, the solution is the second smallest eigenvector, called the Fiedler vector. A permutation is recovered from this eigenvector by sorting its coefficients: given $x = (x_1, x_2, \ldots, x_n)$, the algorithm outputs a permutation $\pi$ such that $x_{\pi(1)} \leq x_{\pi(2)} \leq \ldots \leq x_{\pi(n)}$. This procedure is summarized as Algorithm 1.

In fact, (Atkins et al., 1998) showed that under the assumption that $A$ has a serial structure, Algorithm 1 solves the seriation problem exactly, i.e. recovers the order $\pi$ such that $A_{ij}(\pi(i) - \pi(j))^2$ decreases with $|i - j|$. This means that we solve the read ordering problem by simply solving an extremal eigenvalue problem, which has low complexity [comparable to principal component analysis (PCA)] and is efficient in practice (see Supplementary Fig. S1 and Table S1).

Once the reads are reordered, we can sequentially compute their exact positions (basepair coordinate of their left end on the genome) and orientation. We assign position 0 and strand ‘+’ to the first read.

![Fig. 1. A similarity matrix reordered with the spectral algorithm. The original matrix (left) has values that decrease when moving away from the diagonal. It is randomly permuted (right), and the spectral algorithm will find back the original ordering (Color version of this figure is available at Bioinformatics online.)](https://academic.oup.com/bioinformatics/article-abstract/33/20/3188/3866476)
and use the overlap information (position of the overlap on each read and mutual orientation) to compute the second read’s position and orientation, etc. More specifically, when computing the position and orientation of read \(i\), we use the information from reads \(i \in 1, \ldots, c\) to average the result, where \(c\) roughly equals the coverage, as this makes the layout more robust to misplaced reads. Note that overlapers relying on hashing, such as minimap and MHAP, do not generate alignments but still locate the overlaps on the reads, making this positioning step possible. Thanks to this ‘polishing’ phase, we would still recover the layout if two neighboring reads were permuted due to consecutive entries of the sorted Fiedler vector being equal up to the eigenvector computation precision, for example.

<table>
<thead>
<tr>
<th>Algorithm 1: Spectral ordering</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> Connected similarity matrix (A \in \mathbb{R}^{m \times n})</td>
</tr>
<tr>
<td>1. Compute Laplacian (L_A = \text{diag}(A) - A)</td>
</tr>
<tr>
<td>2. Compute second smallest eigenvector of (L_A), (\mathbf{x}^*)</td>
</tr>
<tr>
<td>3. Sort the values of (\mathbf{x}^*)</td>
</tr>
<tr>
<td><strong>Output:</strong> Permutation (\pi: \mathbf{x}^{<em>}_{[1]} \leq \mathbf{x}^{</em>}<em>{[2]} \leq \ldots \leq \mathbf{x}^{*}</em>{[n]})</td>
</tr>
</tbody>
</table>

2.2 Consensus generation

We built a simple assembler using this layout idea and tested its accuracy. It is partly inspired by the nanocorrect pipeline of Loman et al. (2015) in which reads are corrected using multiple alignments of all overlapping reads. These multiple alignments are performed with a Partial Order Aligner (POA) (Lee et al., 2002) multiple-sequence alignment engine. It computes a consensus sequence from the alignment of multiple sequences using a dynamic programming approach that is efficient when the sequences are similar (which is the case if we trim the sequences to align their overlapping parts). Specifically, we used SPOA, a Single Instruction Multiple Data implementation of POA developed in Vaser et al. (2016).

The key point is that we do not need to perform multiple alignment using all reads, since we already have a layout. Instead, we can generate a consensus sequence for, say, the first 3000bp of the genome by aligning the parts of the reads that are included in this window with SPOA, and repeat this step for the reads included in the window comprising the next 3000bp of the genome, etc. In practice, we take consecutive windows that overlap and then merge them to avoid errors at the edges, as shown in Figure 2. The top of the figure displays the layout of the reads broken down into three consecutive overlapping windows, with one consensus sequence generated per window with SPOA. The final assembly is obtained by iteratively merging the window \(k + 1\) to the consensus formed by the windows \(1, \ldots, k\).

The computational complexity for aligning \(N\) sequences of length \(L\) with POA, with an average divergence between sequences \(\epsilon\), is roughly \(O(mNL^2)\), with \(m \approx (1 + 2\epsilon)\). With 10% of errors, \(m\) is close to 1. If each window of size \(L_w\) contains about \(C\) sequences, the complexity of building the consensus in a window is \(O(mCL)^2\). We compute \(L_d/L_w\) consensus windows, with \(L_d\) the length of the genome (or contig), so the overall complexity of the consensus generation is \(O(mCL_d^2L_w)\). We therefore chose in practice a window size relatively small, but large enough to prevent mis-assemblies due to noise in the layout, \(L_w = 3\text{kbp}\).

2.3 Overlap-based similarity and repeats handling

In practice, we build the similarity matrix \(A\) as follows. Given an overlap found between the \(i\)-th and \(i\)-th reads, we set \(A_{ij}\) equal to the overlap score (or number of matches, given in tenth column of minimap or fourth column of MHAP output file). Such matrices are sparse: a read overlaps with only a few others (the number of neighbors of a read in the overlap graph roughly equals the coverage).

Overlaps can be removed but only a few others (the number of neighbors of a read in the overlap graph roughly equals the coverage). There is no sparsity requirement for the algorithm to work, however sparsity lowers RAM usage since we store the \(n \times n\) similarity matrix with about \(n \times C\) non-zero values, with \(C\) the coverage. In such cases, the ordered similarity matrix is band diagonal.

Unfortunately, the correctly ordered (sorted by position of the reads on the backbone sequence) similarity matrix contains outliers outside the main diagonal band (see Fig. 3) that corrupt the ordering. These outliers are typically caused by either repeated subsequences or sequencing noise (error in the reads and chimeric reads), although errors in the similarity can also be due to hashing approximations made in the overlap algorithm. We use a threshold on the similarity values and on the length of the overlaps to remove them. The error-induced overlaps are typically short and yield a low similarity score (e.g. number of shared min-mers), while repeat-induced overlaps can be as long as the length of the repeated region. By weighting the similarity, the value associated to repeat-induced overlaps can be lowered. Weighting can be done with, e.g. the –weighted option in MHAP to add a tf-idf style scaling to the MinHash sketch, making repetitive k-mers less likely to cause a match between two sequences, or with default parameters with minimap. In the Supplementary Material, we describe experiments with real, corrected and simulated reads to assess the characteristics of such overlaps and validate our method. Supplementary Figure S2 shows that although the overlap scores and lengths are lower for outliers than for inliers on average, the distributions of these quantities intersect. As shown in Supplementary Figure S3, the experiments indicate that all false-overlaps can be removed with a stringent threshold on the overlap length and score. However, removing all these short or low score overlaps will also remove many true overlaps. For bacterial genomes, the similarity graph can either remain connected or be broken into several connected components after a threshold-based outlier removal, depending on the initial coverage. Supplementary Figure S3 illustrates the empirical
observation that the coverage needs to be above 60× to keep the graph connected while removing all outliers. Most outliers can be similarly removed for real and synthetic data from *S. cerevisiae*, although a few outliers, probably harboring telomeric repeats, remain at the ends of chromosomes after thresholding.

There is thus a tradeoff to be reached depending on how many true overlaps one can afford to lose. With sufficient coverage, a stringent threshold on overlap score and length will remove both repeat-induced and error-induced overlaps, while still yielding a connected assembly graph. Otherwise, aggressive filtering will break the similarity graph into several connected components. In such a case, since the spectral algorithm only works with a connected similarity graph, we compute the layout and consensus separately in each connected component, resulting in several contigs. To set the threshold sufficiently high to remove outliers but small enough to keep the number of contigs minimal, we used a heuristic based on the following empirical observation, illustrated in Supplementary Figure S4. The presence of outliers in the correctly (based on the positions of the reads) ordered band diagonal matrix imparts an increased bandwidth (maximum distance to the diagonal of non-zero entries) on the matrix reordered with the spectral algorithm. We can therefore run the spectral algorithm, check the bandwidth in the reordered matrix, and increase the threshold if the bandwidth appears too large (typically larger than twice the coverage).

In practice, we chose to set the threshold on the overlap length to 3.5 kb, and removed the overlaps with score in the lowest 40% quantile (90 and 95%, respectively) for C ≤ 60X (resp. 60X ≤ C ≤ 100X and C ≥ 100X). As indicated in Algorithm 2, we let this threshold increase if indicated by the bandwidth heuristic.

Finally, we added a filtering step to remove reads that have non-zero similarity with several sets of reads located in distant part of the genome, such as chimeric reads. These reads usually overlap with a first subset of reads at a given position in the genome, and with another distinct subset of reads at another location, with no overlap between these distinct subsets. We call such reads ‘connecting reads’, and they can be detected from the similarity matrix by computing, for each read (index i), the set of its neighbors in the graph $N_i = \{ j : A_{ij} > 0 \}$. The subgraph represented by A restricted to $N_i$ is either connected (there exists a path between any pair of edges), or split into separate connected components. In the latter case, we keep the overlaps between read i and its neighbor that belong to only one of these connected components (the largest one).

3 Results

3.1 Data

We tested this pipeline on ONT and PacBio data. The bacterium *Acinetobacter baylyi* ADP1 and the yeast *Saccharomyces cerevisiae* S288C were sequenced at Genoscope with Oxford Nanopore’s MiniION device using the R7.3 chemistry, together with an additional dataset of *S. cerevisiae* S288C using the R9 chemistry. Only the 2D high quality reads were used. The *S. cerevisiae* S288C ONT sequences were deposited at the European Nucleotide Archive (http://www.ebi.ac.uk/ena) where they can be accessed under Run accessions ERR1539069 to ERR1539080, while *Acinetobacter baylyi* ADP1 sequences will be made available on https://github.com/antrec/spectrassembler. We also used the following publicly available data: ONT *Escherichia coli* by Loman et al. (2015) (http://bit.ly/loman006 - PCR12D pass dataset), and PacBio E. coli K-12 PacBio P6C4, and *S. cerevisiae* W303 P4C2. Their key characteristics are given with the assembly results in Table 1, and read length histograms are given in Supplementary Figure S5. For each dataset, we also used the reads corrected and trimmed by the Canu pipeline as an additional dataset with low error-rate. The results on these corrected datasets are given in Supplementary Figures S6 and S7 and Tables S2 and S4.

3.2 Layout

3.2.1 Bacterial genomes

minimap was used to compute overlaps between raw reads (we obtained similar results with MHAP and DALIGNER). The similarity matrix preprocessed as detailed in Section 2.3 yielded a few connected components for bacterial genomes. The reads were successfully ordered in each of these, as one can see in Figure 4 for *E. coli*, and in Supplementary Figure S6 for the other datasets.

3.2.2 Eukaryotic genome

For the *S. cerevisiae* genome, the threshold on similarity had to be set higher than for bacterial genomes because of a substantially higher number of repetitive regions and false overlaps, leading to a more fragmented assembly. Most of them are correctly reordered with the spectral algorithm, see Figure 5 and Supplementary Figure S7.
contig. Sufficient overlap was left between the contig sequences to small number of connected components, each of them yielding a the two bacterial genomes, the first round of layout produced a Once the layout was established, the method described above was

3.3.1 Recovering contiguity

3.3 Consensus
3.3.1 Recovering contiguity

Once the layout was established, the method described above was used to assemble the contigs and generate a consensus sequence. For the two bacterial genomes, the first round of layout produced a small number of connected components, each of them yielding a contig. Sufficient overlap was left between the contig sequences to find their layout with a second iteration of the algorithm and produce a single contig spanning the entire genome. The number of con- tigs in the yeast assemblies can be reduced similarly. The fact that the first-pass contigs overlap even though they result from breaking the similarity graph into several connected components might seem counter-intuitive at first sight. However, note that when cutting an edge $A_{ij}$ results in the creation of two contigs (one containing $i$ and

### Table 1. Assembly results of the spectral method, compared to Miniasm, Canu and Racon, across the different datasets

<table>
<thead>
<tr>
<th></th>
<th>Miniasm</th>
<th>Spectral</th>
<th>Canu</th>
<th>Miniasm + Racon</th>
<th>Miniasm + Racon (2 iter.)</th>
<th>Spectral + Racon</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A. baylyi</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ONT R7.3, 28x</td>
<td>3 598 621</td>
<td>3 598 621</td>
<td>3 598 621</td>
<td>3 598 621</td>
<td>3 598 621</td>
<td>3 598 621</td>
</tr>
<tr>
<td>Total contigs (#)</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Aln. bases ref (kbp)</td>
<td>3 345 (95.74%)</td>
<td>3 396 (99.93%)</td>
<td>3 395 (99.90%)</td>
<td>3 397 (99.95%)</td>
<td>3 397 (99.95%)</td>
<td>3 398 (99.99%)</td>
</tr>
<tr>
<td>Misassemblies (#)</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Avg. identity</td>
<td>87.31</td>
<td>98.17</td>
<td>97.59</td>
<td>98.18</td>
<td>98.36</td>
<td>98.42</td>
</tr>
<tr>
<td><strong>E. coli</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ONT R7.3, 30x</td>
<td>4 641 652</td>
<td>4 641 652</td>
<td>4 641 652</td>
<td>4 641 652</td>
<td>4 641 652</td>
<td>4 641 652</td>
</tr>
<tr>
<td>Total contigs (#)</td>
<td>4 759 346</td>
<td>4 662 043</td>
<td>4 625 543</td>
<td>4 564 823</td>
<td>4 566 438</td>
<td>4 629 112</td>
</tr>
<tr>
<td>Aln. bases ref (kbp)</td>
<td>4 433 (93.14%)</td>
<td>4 624 (99.18%)</td>
<td>4 625 (100.00%)</td>
<td>4 643 (99.91%)</td>
<td>4 640 (99.93%)</td>
<td>4 629 (100.00%)</td>
</tr>
<tr>
<td>Misassemblies (#)</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Avg. identity</td>
<td>98.42</td>
<td>99.87</td>
<td>99.46</td>
<td>99.46</td>
<td>99.46</td>
<td>99.46</td>
</tr>
<tr>
<td><strong>S. cerevisiae</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ONT R7.3, 68x</td>
<td>11 217 105</td>
<td>11 123 218</td>
<td>11 124 953</td>
<td>11 926 664</td>
<td>11 926 191</td>
<td>11 126 363</td>
</tr>
<tr>
<td>Total contigs (#)</td>
<td>11 813 544</td>
<td>11 795 644</td>
<td>11 217 497</td>
<td>11 128 279</td>
<td>11 129 086</td>
<td>11 750 114</td>
</tr>
<tr>
<td>Aln. bases ref (kbp)</td>
<td>11 566 (95.14%)</td>
<td>11 138 (99.06%)</td>
<td>11 129 (99.42%)</td>
<td>11 128 (99.41%)</td>
<td>11 128 (99.41%)</td>
<td>11 128 (99.41%)</td>
</tr>
<tr>
<td>Misassemblies (#)</td>
<td>0</td>
<td>7</td>
<td>34</td>
<td>18</td>
<td>19</td>
<td>11</td>
</tr>
<tr>
<td>Avg. identity</td>
<td>96.61</td>
<td>96.64</td>
<td>99.54</td>
<td>99.54</td>
<td>99.46</td>
<td>99.46</td>
</tr>
<tr>
<td><strong>S. cerevisiae</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ONT R9, 86x</td>
<td>11 734 150</td>
<td>11 795 644</td>
<td>11 217 497</td>
<td>11 128 279</td>
<td>11 129 086</td>
<td>11 750 114</td>
</tr>
<tr>
<td>Total contigs (#)</td>
<td>11 733 150</td>
<td>11 795 644</td>
<td>11 217 497</td>
<td>11 128 279</td>
<td>11 129 086</td>
<td>11 750 114</td>
</tr>
<tr>
<td>Aln. bases ref (kbp)</td>
<td>11 549 (98.43%)</td>
<td>11 129 (99.93%)</td>
<td>11 130 (99.92%)</td>
<td>11 129 (99.92%)</td>
<td>11 129 (99.92%)</td>
<td>11 129 (99.92%)</td>
</tr>
<tr>
<td>Misassemblies (#)</td>
<td>0</td>
<td>23</td>
<td>39</td>
<td>18</td>
<td>19</td>
<td>36</td>
</tr>
<tr>
<td>Avg. identity</td>
<td>98.10</td>
<td>99.20</td>
<td>99.10</td>
<td>99.10</td>
<td>99.10</td>
<td>99.10</td>
</tr>
<tr>
<td><strong>E. coli</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PacBio, 161x</td>
<td>4 843 211</td>
<td>4 731 239</td>
<td>4 670 125</td>
<td>4 633 228</td>
<td>4 645 420</td>
<td>4 674 460</td>
</tr>
<tr>
<td>Total contigs (#)</td>
<td>4 845 211</td>
<td>4 731 239</td>
<td>4 670 125</td>
<td>4 653 228</td>
<td>4 645 420</td>
<td>4 674 460</td>
</tr>
<tr>
<td>Aln. bases ref (kbp)</td>
<td>4 437 (95.60%)</td>
<td>4 618 (99.48%)</td>
<td>4 642 (100.00%)</td>
<td>4 641 (100.00%)</td>
<td>4 641 (100.00%)</td>
<td>4 642 (100.00%)</td>
</tr>
<tr>
<td>Misassemblies (#)</td>
<td>0</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Avg. identity</td>
<td>98.72</td>
<td>99.54</td>
<td>99.87</td>
<td>99.87</td>
<td>99.87</td>
<td>99.87</td>
</tr>
<tr>
<td><strong>S. cerevisiae</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PacBio, 127x</td>
<td>12 266 420</td>
<td>12 346 258</td>
<td>12 070 971</td>
<td>12 052 148</td>
<td>12 052 148</td>
<td>12 052 148</td>
</tr>
<tr>
<td>Total contigs (#)</td>
<td>12 266 420</td>
<td>12 346 258</td>
<td>12 070 971</td>
<td>12 052 148</td>
<td>12 052 148</td>
<td>12 052 148</td>
</tr>
<tr>
<td>Aln. bases ref (kbp)</td>
<td>11 250 (92.54%)</td>
<td>11 918 (98.03%)</td>
<td>12 023 (98.90%)</td>
<td>12 025 (98.91%)</td>
<td>12 025 (98.91%)</td>
<td>12 025 (98.91%)</td>
</tr>
<tr>
<td>Misassemblies (#)</td>
<td>0</td>
<td>57</td>
<td>76</td>
<td>61</td>
<td>59</td>
<td>68</td>
</tr>
<tr>
<td>Avg. identity</td>
<td>98.41</td>
<td>99.43</td>
<td>99.72</td>
<td>99.72</td>
<td>99.72</td>
<td>99.72</td>
</tr>
</tbody>
</table>

Note: For the spectral method, we give the results after contig merging (see Section 3.3.1); the number of contigs before this post-processing is given between parentheses. Racon’s use here can be seen as a polishing phase for the sequences outputted by the spectral method and Miniasm. To keep both assemblers on an equal footing, we compared Spectral + Racon to two iterations of Miniasm + Racon (since one pass of Miniasm does not implement any consensus). The best results in terms of average identity are highlighted in bold (but other metrics should also be used to compare the assemblies). Canu clearly outperforms the spectral method on PacBio data, while both assemblers yield comparable results on the ONT datasets.

### 3.3 Consensus

#### 3.3.1 Recovering contiguity

Once the layout was established, the method described above was used to assemble the contigs and generate a consensus sequence. For the two bacterial genomes, the first round of layout produced a small number of connected components, each of them yielding a contig. Sufficient overlap was left between the contig sequences to find their layout with a second iteration of the algorithm and produce a single contig spanning the entire genome. The number of con- tigs in the yeast assemblies can be reduced similarly. The fact that the first-pass contigs overlap even though they result from breaking the similarity graph into several connected components might seem counter-intuitive at first sight. However, note that when cutting an edge $A_{ij}$ results in the creation of two contigs (one containing $i$ and
the other $j$), the sequence fragment at the origin of the overlap between the two reads is still there on both contigs to yield an overlap between them in the second iteration. Alternatively, we found the following method useful to link the contigs’ ends: 1. extract the ends of the contig sequences, 2. compute their overlap with minimap, 3. propagate the overlaps to the contig sequences, 4. use miniasm with all pre-selection parameters and thresholds off, to just concatenate the contigs (see Supplementary Material, implementation and reproducibility).

3.3.2 Consensus quality evaluation

We first investigated the quality of the consensus sequences derived in each window. Figure 6 and Supplementary Figure S8 highlight the correcting effect of the consensus. Supplementary Figure S9 suggests that the error-rate in the consensus windows depends mainly on the local coverage. We then compared our results to those obtained with other long reads assemblers: Miniasm, Canu and Racon (Vaser et al., 2016). Racon takes a draft assembly, the raw reads, and a mapping of the reads to the draft assembly as input. We used it with the draft assembly produced by Miniasm [as done by Vaser et al. (2016)]. We label this method ‘Miniasm + Racon’ in our results. We also used Racon with the draft assembly derived by our method (‘Spectral + Racon’ method), using Minimap to map the raw reads to the draft assemblies before using Racon. A summary of assembly reports generated with DNAdiff (Kurtz et al., 2004) and QUAST (Gurevich et al., 2013) are given in Table 1 and Supplementary Table S3. Briefly, the assemblies displayed between 98 and 99% average identity to their reference genome, with errors mostly consisting in deletions. Misassemblies were rare in reconstructed bacterial genomes but more frequent in assembled yeast genomes, where they mostly consisted in translocations and relocations caused by either deletions and/or misplaced reads in the layout.

3.3.3 Optical mapping

After the first iteration of the bacterial genome assembly pipeline, overlaps between the first-pass contigs were sufficient to find their layout. It should be anticipated however that not all overlaps might be apparent in some cases, e.g. if too many reads were removed during the preprocessing step. One attractive option is to use optical mapping (Aston et al., 1999) to layout the contigs. We had such an optical map available for the A. baylyi genome, and implemented the algorithm of Nagarajan et al. (2008) to map the contigs to the restriction map, which led to the same layout as the one identified from our two-round assemblies (data not shown), thus providing a ‘consistency check’ for the layout. We suggest in Supplementary Figure S10 and Table S5 that optical maps could be particularly valuable for the ordering of contigs from more structurally complex eukaryotic genomes such as S. cerevisiae.

4 Discussion

We have shown that seriation based layout algorithms can be successfully applied to de novo genome assembly problems, at least for genomes harboring a limited number of repeats.

In a similar vein to the recent report about the miniasm assembly engine (Li, 2016), our work confirms that the layout of long reads...
can be found without prior error correction, using only overlap information generated from raw reads by tools such as minimap, MHAP or DALIGNER. However, unlike miniasm, which does not derive a consensus but instead concatenates the reads into a full sequence, we take advantage of read coverage to produce contigs with a consensus quality on par with that achieved by assembly pipelines executing dedicated error-correction steps. The results of Table 1 appear promising. For example, our assembler combined with Racon yields among the highest average identities with the reference for the ONT datasets. In terms of speed however, our pipeline is clearly outperformed by Miniasm, but also by Miniasm+Racon, the latter improving overall accuracy. Still, compared to approaches implementing error correction steps, we gain significant speed-ups by highly localizing the error correction and consensus generation processes, which is made possible by knowledge of the layout. We believe that tools such as Miniasm and Racon are implemented in a much more efficient way than our own, but the layout method itself is efficient (see Supplementary Table S1) and is known to be scalable as it relies on the same algorithmic core as Google’s PageRank.

The main limitation of our layout algorithm is its sensitivity to outliers in the similarity matrix, hence the need to remove them in a pre-processing phase. Higher coverage and quality of the input reads, both expected in the near future, would likely improve the robustness of our pipeline. Still, for eukaryotic genomes, we found that some outliers require additional information to be resolved (see Supplementary Figure S3), which could be provided in the future by extracting topological information from the assembly graph.

In the meantime, our pipeline behaves like a draft generating assembler for prokaryotic genomes, and a first-pass unitigger for eu- karyotic genomes. Importantly, the overall approach is modular and can integrate other algorithms to increase layout robustness or consensus quality, as illustrated here by the integration of Racon as an optional polishing module.

Our original contribution here consists in the layout computation. The spectral OLC assembler we built on top of it could be enhanced in many ways. We have shown that the spectral algorithm is suited to find the layout for bacterial genomes, even though there is room left for performance improvements on repeat-rich eukaryotic genomes.

For these eukaryotic genomes, it could make sense to use the spectral algorithm jointly with other assembly engines (e.g. Miniasm or Canu), to check the consistency of connected components before they are assembled. Our consensus generation method is coarsen- grained for now and does not take into account statistical properties of ONT sequencing errors. Nevertheless, the three components (O, L and C) of the method being independent, an external and more refined consensus generation process could readily be plugged after the overlap and layout computations to further improve results and increase accuracy.

Acknowledgements

T.B. would like to thank Genoscope’s sequencing (Laboratoire de Séquençage) and bioinformatics (Laboratoire d’Informatique Scientifique) teams for sharing some Acinetobacter baylyi ADP1 and Saccharomyces cerevisiae S288C MinION data, and is grateful to Oxford Nanopore Technologies Ltd for granting Genoscope access to its MinION device via the MinION Access Programme. A.A. and A.R. would like to acknowledge support from the chaîne Économie des nouvelles données, the data science joint research initiative with the fonds AXA pour la recherche and a gift from Société Générale Cross Asset Quantitative Research.

Funding

This work was supported by the European Research Council (project SIPA).

Conflict of Interest: none declared.

References