*Bioinformatics*, 35(21), 2019, 4239–4246
doi: 10.1093/bioinformatics/btz219
Advance Access Publication Date: 27 March 2019
Original Paper

OXFORD

Sequence analysis

# Graph analysis of fragmented long-read bacterial genome assemblies

**Pierre Marijon [1,*], Rayan Chikhi[2] and Jean-Stéphane Varré[3]**

[1]Inria, Université de Lille, CNRS, Centrale Lille, UMR 9189 – CRIStAL, Lille F-59000, France, [2]Institut Pasteur, C3BI USR 3756 IP CNRS, Paris 75015, France, [3]Université de Lille, CNRS, Centrale Lille, Inria, UMR 9189 – CRIStAL, Lille F-59000, France

*To whom correspondence should be addressed.

Associate Editor: John Hancock

Received on October 9, 2018; revised on February 19, 2019; editorial decision on March 14, 2019; accepted on March 26, 2019

## Abstract

**Motivation:** Long-read genome assembly tools are expected to reconstruct bacterial genomes nearly perfectly; however, they still produce fragmented assemblies in some cases. It would be beneficial to understand whether these cases are intrinsically impossible to resolve, or if assemblers are at fault, implying that genomes could be refined or even finished with little to no additional experimental cost.

**Results:** We propose a set of computational techniques to assist inspection of fragmented bacterial genome assemblies, through careful analysis of assembly graphs. By finding paths of overlapping raw reads between pairs of contigs, we recover potential short-range connections between contigs that were lost during the assembly process. We show that our procedure recovers 45% of missing contig adjacencies in fragmented `Canu` assemblies, on samples from the NCTC bacterial sequencing project. We also observe that a simple procedure based on enumerating weighted Hamiltonian cycles can suggest likely contig orderings. In our tests, the correct contig order is ranked first in half of the cases and within the top-three predictions in nearly all evaluated cases, providing a direction for finishing fragmented long-read assemblies.

**Availability and implementation:** `https://gitlab.inria.fr/pmarijon/knot`.

**Contact:** pierre.marijon@inria.fr

**Supplementary information:** Supplementary data are available at *Bioinformatics* online.

## 1 Introduction

Third-generation DNA sequencing using PacBio and Oxford Nanopore instruments is increasingly becoming a go-to technology for constructing reference genomes of non-model prokaryotes and eukaryotes. Longer sequencing reads allow in principle to overcome the reconstruction problems posed by genomic repetitions (Bresler *et al.*, 2013). Direct assembly of second-generation (Illumina) sequencing data typically also results in high consensus accuracy yet generally more fragmented bacterial assemblies (Bankevich *et al.*, 2012). The large-scale ongoing NCTC project aims to assemble and make publicly available 3000 bacterial strains sequenced using PacBio (https://www.sanger.ac.uk/resources/downloads/bacteria/nctc/).

Recent works have demonstrated single-contig long-read assemblies of bacterial chromosomes (Koren and Phillippy, 2015; Loman *et al.*, 2015). Therefore, it is natural to ask whether genome assembly is now a solved problem with long reads (See e.g. https://flxlexblog.wordpress.com/2013/07/05/de-novo-bacterial-genome-assembly-a-solved-problem/), at minimum for smaller genomes such as bacteria. It turns out that in several cases, bacterial assemblies remain fragmented into a handful of contigs, even with long-read sequencing and recent assembly techniques. Deciding whether an assembly instance is resolved is not always clear due to the presence of plasmids, contaminants and unplaced low-quality reads. In this work, an assembly is considered to be *resolved* if the number of contigs classified as chromosomal is equal to the expected number of chromosomes (generally just one, in the bacterial case).

To date, the NCTC project contains 1735 samples for which 1136 have been assembled by the consortium, and among these, 599 (34%) are unresolved according to the criteria above (as in February 2019). Later in this article, we will see that even when using multiple recent tools, many assemblies remain fragmented. Therefore there is a clear and unmet need for an investigation that determines whether those samples are intrinsically impossible to resolve, or whether current assembly methods are imperfect.

In this article, we have selected a subset of NCTC samples (see Section 5) and considered the outputs of three recent assemblers: `Canu`, `Miniasm`, and `HINGE`. We observe that instances where the assembly is fragmented can be challenging to further manually elucidate. In general, assemblers produce an assembly graph where nodes are contigs and edges reflect local sequence proximity in the genome (*adjacency*). In fragmented instances, the final assembly graph is sometimes uninformative due to the absence of edges between contigs, hindering further assembly finishing steps. In such cases, it would be tempting to conclude that the assembly is fragmented due to regions of insufficient sequencing coverage, with no way to determine a likely contig order. However, in a number of cases, we found that a lack of connectivity can be due to reads that were discarded early in the assembly pipeline. Here, we will show that contig adjacency information can be computationally recovered from the raw data.

To automatically investigate unresolved assemblies and propose directions for refinement, we introduce a set of *in silico* forensics operations for long-read assemblies, and we built a software framework. Our analyses are based solely on information present in the raw-sequencing data in addition to the contigs produced by a given assembly tool, and are not biased by any other source, e.g. a closely related reference genome. For validation purposes only and to explain some of our observations, we will align contigs to a ground truth reference when one is available. Our framework is first tested on synthetic data to illustrate a simple case of fragmentation due to heuristics in the `Canu` assembler. We then show on real data that our method helps recover useful adjacency information between contigs.

Going further, we demonstrate how to use this recovered information to provide likely assembly hypotheses using Hamiltonian paths, through a ranked list of contigs orderings. Obtaining a small set of possible orderings between contigs, knowing that the true genome order is likely one of them, can be instrumental to guide further genome finishing steps.

## 2 Related works

Assembly forensics date back to the Sanger era, e.g. with the AMOSvalidate software (Phillippy *et al.*, 2008), which detects misassemblies within contigs using multiple sources of information (e.g. read coverage, properly mapped pairs, clipping). Other tools have been introduced for misassembly detection in Illumina data [REAPR (Hunt *et al.*, 2013), FRCbam (Vezzi *et al.*, 2012), Pilon (Walker *et al.*, 2014)] and for PacBio data [VALET (Olson *et al.*, 2017)] using similar principles. Completeness of an assembly can be estimated without any reference, using core genes as a proxy metric, e.g. with BUSCO (Simão *et al.*, 2015) or CheckM (Parks *et al.*, 2015) software. Finally, assembly likelihood metrics have been introduced to assess the fit of an assembly to a probabilistic model of sequencing, via remapping reads to the assembly (Clark *et al.*, 2013; Ghodsi *et al.*, 2013; Rahman and Pachter, 2013). For a more complete exposition, refer to a recent survey on metagenomics assembly validation (Olson *et al.*, 2017), that also largely applies to isolates.

For bacterial genomes specifically, several pipelines for *assembly finishing* have been developed (Bosi *et al.*, 2015). They usually take as input an assembly obtained with short-read data and align it to one or multiple close reference genomes, in order to find a contig ordering (Kremer *et al.*, 2017). Recent work has examined the cause of assembly fragmentation for seven bacterial genomes sequenced using PacBio sequencing, and rejected the hypothesis that gaps were caused by strong secondary DNA structure (Utturkar *et al.*, 2017). Instead, low coverage and repetitions appear to be the two main factors for contig termination.

To the best of our knowledge, little work has been carried to investigate assemblies based on the graph of assembled contigs or the initial string graph. Noteworthy exceptions are the Bandage software (an assembly graph visualization tool) (Wick *et al.*, 2015), and the `HINGE` assembler that implements automated repeat handling based on the assembly graph (Kamath *et al.*, 2017). We use Bandage extensively in the present work, and will consider datasets where even `HINGE` failed to produce a single-contig assembly.

## 3 Long-read assemblers

Several genome assemblers have been developed to process third-generation sequencing data, either stand-alone (Kamath *et al.*, 2017; Koren *et al.*, 2017; Li, 2016; Lin *et al.*, 2016) or in combination with Illumina data (Antipov *et al.*, 2016; Wick *et al.*, 2017; Ye *et al.*, 2016; Zimin *et al.*, 2013). In this work, we will focus on three recent stand-alone assemblers, chosen because of their widespread usage (`Canu`), automated graph analysis algorithms (`HINGE`) and speed/modularity (`Miniasm`). However the techniques are likely to be applicable to a broader set of assemblers.

### 3.1 Description of `Canu`, `Miniasm` and `HINGE`

The `Canu` (Koren *et al.*, 2017) assembler consists of three major steps: correction, trimming and contig creation. The first two steps should not be regarded as innocuous pre-processing steps, as they significantly impact the rest of the assembly process. The correction step uses MHAP to perform all-against-all read mapping then generates consensus reads with the falcon_sense tool (Chin *et al.*, 2016). `Canu` then performs overlapping of error-corrected reads with a legacy algorithm from the Celera assembler, named *ovl*. The trimming step detects hairpins, chimeric reads, and low-support regions and subsequently cuts reads. A 'unitigging' step is performed using `bogart`, a modified version of CABOG (Miller *et al.*, 2008), to produce a graph that records only the longest overlaps between corrected reads (termed BOG for 'Best Overlap Graph'). `Canu` generates contigs from this graph and improves their consensus accuracy by remapping all reads.

The `Miniasm` pipeline consists of two separate tools: `Minimap2` and `Miniasm` (Li, 2016). `Minimap2` finds overlaps between raw reads and outputs alignments. `Miniasm` trims low-coverage regions of reads, then constructs a string graph from `Minimap2` alignments that are suffix–prefix overlaps. `Miniasm` performs simplification on the graph inspired by short-read assembly: transitive reduction, tip removal, bubble popping and short overlaps removal based on a relative length threshold. After simplifications, non-branching paths are returned as contigs.

The `HINGE` (Kamath *et al.*, 2017) assembler uses raw uncorrected reads (similarly to `Miniasm`) to construct an overlap graph similar to the BOG of `Canu`. `HINGE` attempts to output finished

bacterial assemblies through improved repeat-resolution. In cases where there subsist repetitions that are not spanned by reads, `HINGE` provides a visualization of the resulting assembly graph for manual inspection.

### 3.2 Assembly graphs

Short-read and long-read assemblers output final assembly sequences in FASTA format, and an increasing number of tools also output an assembly graph in Graphical Fragment Assembly (GFA) format (https://github.com/GFA-spec/GFA-spec). A final long-read *assembly graph* typically consists of all contig sequences as nodes, and a set of overlaps between contigs as edges. Assembly graphs are seldom used by downstream tools, and are generally provided for the purpose of inspecting the assembly.

Most long-read assemblers start by constructing then analyzing a *string graph* (SG) of the reads (Myers, 2005), where each read is a node, and overlaps between reads are represented by edges to which additional information is attached (e.g. overlap length, overlap error rate). In addition, transitive reduction is performed on the edges and reads that are fully contained in others are discarded.

## 4 Materials and Methods

We hypothesized that the final contig graph produced by assemblers does not always reflect all the information present in the raw data, and may be missing overlaps or even genomic regions. We built a novel algorithmic framework to recover some of the 'missing' information and further analyze it. The main steps are presented in Figure 1, and the next sections describe them in more details.

### 4.1 Raw SG

First, we eliminate chimeric reads from the raw data based on overlaps found by `Minimap2` using a custom tool (https://gitlab.inria.fr/pmarijon/yacrd; P.Marijon *et al.*, manuscript in preparation, see Supplementary Figure S6). A SG is then constructed using overlaps between chimera-removed reads (here, overlaps found by minimap2). A stand-alone script was created to convert overlaps from the PAF format [defined in Li (2016)] to a graph in the GFA format (https://gitlab.inria.fr/pmarijon/fpa). Transitive reduction over the edges of this SG is performed using Myers' algorithm (Myers, 2005).

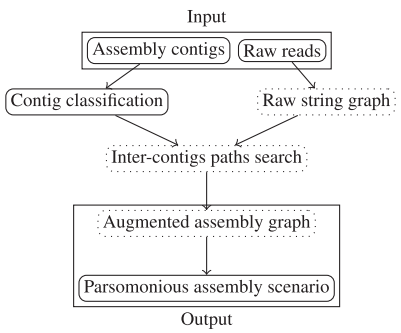


**Fig. 1.** The proposed framework takes as input raw long-read sequencing data and the output of an assembler. The (optional) contig classification step removes non-chromosomal contigs. A SG of raw reads is constructed, in which paths are searched between extremities of contigs, then are converted into links between contigs in an AAG. When such a graph is connected, putative contig orderings are reported. Dotted nodes represent elements that are automatically visualized in the HTML report

### 4.2 Contigs classification

In order to simplify analyses and focus on chromosomal contigs, we filter out contigs of plasmid origin and contigs of unknown taxonomic status (see Supplementary Methods Appendix 1). Contigs that were not marked as chromosomal are discarded. Note, however, that this contig classification step can be skipped in order to perform analysis of complete, unfiltered sets of contigs.

### 4.3 Computation of paths between contigs

An essential algorithmic component of our framework is the search for paths in the SG that uncover new connections between contigs. First, one read per contig extremity is identified among reads included in the SG: a read is selected such that both its incoming and outgoing neighbors also map at the same contig extremity (in order to avoid selecting dead-end nodes in the SG).

Then for each pair of contigs, shortest paths between reads at both extremities of each contig are computed in the SG using Dijkstra's algorithm. The length of a path is computed in nucleotides as follows: the sum of all reads lengths involved in the path minus all the overlaps between reads, as well as minus the overlaps between reads and contig extremities. If contigs overlap, the path length is reported as zero. Since we perform path search starting from each contig extremity, we may obtain two shortest paths for each pair of contigs, and only the shortest of those two is kept.

### 4.4 Augmented assembly graph

We transform a contig graph into a novel object, the *augmented assembly graph* (AAG), as follows. Nodes of the AAG are contig extremities. An edge is inserted between two nodes if a path has been found by the procedure in Section 4.3 between the two-contig extremities. Each edge is weighted by the corresponding path length. Additionally, zero-weight edges are created between both extremities of each contig.

Such a graph allows to explore adjacencies between contigs, beyond those present in the original contig graph, in order to formulate hypotheses regarding the ordering of contigs. At a certain contig extremity, and in absence of genomic repeats, low-weight edges likely reflect adjacent contigs, while high-weight edges likely correspond to SG paths that pass through other contig(s) (i.e. transitively redundant edges in the AAG). In the presence of repeats, low-weight edges do not necessarily show true adjacencies between contigs, as the true path may be longer. Yet one can observe that a path longer than the longest repeat in the genome necessarily reveals a distant link between two contigs (i.e. necessarily contigs which are truly non-adjacent on the genome), and also such path may go through another contig.

According to Treangen *et al.* (2009) most repetitions in bacteria are shorter than 10 kb. We thus categorize edges of the AAG into three groups according to their weight. Consider the path in the SG that led to the creation of the edge $e$ in the AAG between extremities of two different contigs $a$ and $b$. If the path is longer than 10 kb, and/or it contains at least one read that was involved in the construction of another contig $c$, the edge $e$ is named *distant*. Otherwise the edge $e$ is considered to reflect an adjacency between $a$ and $b$. If there is more than one edge outgoing from the extremity of $a$ or of $b$, the edge $e$ is named a *multiple adjacency* (likely revealing a putative repeat). Otherwise it is named a *single adjacency*.

### 4.5 Searching for parsimonious assembly scenarios

We sought to determine whether contigs could possibly be ordered directly using the AAG. In principle, we anticipate to recover a large

number of distant edges in the AAG, therefore it would be non-trivial to determine a contig order by direct inspection of the graph layout (e.g. see Figure 3). Given a connected AAG, our working hypothesis is that a minimum-weight Hamiltonian cycle may correspond to the correct contig order (note that having a connected AAG is a necessary condition for such a cycle to exist, but not a sufficient one). This is guided by the intuition that edges in the AAG with high weight are more likely to correspond to false connections due to repetitions or true paths between distant contigs. For simplicity, we search for Hamiltonian cycles and not paths, under the assumption that the genome is circular. We further require that any Hamiltonian cycle traverses all zero-weight edges corresponding to both extremities of each contig. Moreover, contigs mapping inside another one are not considered.

We designed an automated procedure to test this hypothesis, based on computing and sorting Hamiltonian cycles according to their total edge weights. In practice, some of the AAGs that we obtain are too complex, due to the presence of short contigs (see Section 6 for more details). Our pipeline excluded contigs shorter than 100 kb from the AAG before listing all Hamiltonian cycles. For validation purposes, when a reference genome is available, we mapped all chromosomal contigs against this reference to determine the true contig order. We then recovered the position of the true contig order within the list of orders given by Hamiltonian cycles.

### 4.6 Assembly report generation

We implemented a Snakemake (Koster and Rahmann, 2012) pipeline that takes as input raw reads, contigs produced by an assembler, and optionally a contig graph. The pipeline follows steps described Figure 1, then generates an HTML report for easy inspection. Companion tools to compute AAG edge classification and to perform Hamiltonian path search are also provided.

## 5 Results

### 5.1 Datasets

In order to illustrate our methods using a simple yet non-trivial case of assembly graph analysis, we simulated long reads from a linearized reference genome of *Terriglobulus roseus* (NC_018014.1, 5.2 Mbp). This genome contains an unusual 460 kb repeat that is challenging for assembly tools. We used LongISLND (Lau *et al.*, 2016), with 20× sequencing coverage and 9 kb mean read length (Supplementary Table S9).

To investigate real datasets, we mined the NCTC project which consists of 1735 bacterial strains (as of February 2019) sequenced using PacBio technology. For each dataset, the NCTC consortium had built an assembly using HGAP and Circlator (Hunt *et al.*, 2015) followed by a manual correction step. We estimate, based on visual inspection of 159 NCTC fragmented `HINGE` assemblies (https://web.stanford.edu/ gkamath/NCTC/report.html) out of 997, that assembly graphs are missing contig adjacency information in 69% of the fragmented assemblies of `HINGE` and `Miniasm`, i.e. around 13% of all NCTC datasets (including those that assemble perfectly). Among datasets for which both `Canu` and `HINGE` failed to produce a single contig per chromosome, we selected 19 datasets where the assembly made by NCTC contains as many chromosomal contigs as the number of expected chromosomes (i.e. is resolved), 24 datasets where the NCTC assembly is unresolved, and finally 2 datasets that were not yet assembled by NCTC. See Supplementary Table S2 for a complete list of the 45 datasets. All datasets were assembled with `Canu` version 1.7 and `Miniasm` version 0.2.

`Canu` contigs were classified according to Section 4.2. On average for each dataset, 10.2% (resp. 6.4%) of the `Canu` (resp. `Miniasm`) contigs are marked as plasmid, 13.7% (resp. 12.2%) do not match any bacteria in the Blast database and are therefore marked as of undefined origin, and the remaining 76.0% (resp. 81.3%) of contigs are classified as chromosomal and are further considered for analysis. Full classification results are presented in Supplementary Tables S6 and S7.

We further investigated whether the assemblies could somehow be combined, e.g. by improving `Canu` assemblies using `Miniasm` contigs. We have performed a simple test to evaluate this possibility (see Supplementary Section Appendix 2) and could not straightforwardly improve assemblies this way.

### 5.2 Assembly graph analysis of a synthetic low-coverage dataset

This section gives an introductory overview of the analyses that our method performs on the *T.roseus* simple synthetic dataset described above. `Canu` produced three contigs of total length 4.7 Mbp. A ≈ 500 kb region is missing from the assembly. `Miniasm` produced seven contigs and the `HINGE` assembler (commit 8613194) was not able to produce an assembly, likely because of the low coverage (20×).

Since the SG has a single connected component (Figure 2b) but both the BOG and the contig graph of `Canu` have multiple connected components (Figure 2a), assembly fragmentation can be explained by reads that have been discarded at the BOG construction stage of `Canu`. The coloring of the SG using the connected components of `Canu` BOG (Figure 2b) further suggests an ordering of contigs. Note that the `Canu` contig graph is uninformative on this dataset, as it contains no edges between contigs.

We performed path analysis as per Section 4.3. Figure 2d shows the length of paths in SG found between reads at `Canu` contigs extremities. Since a reference genome is available, the true order of contigs is reported on the figure but note that path analysis does not need this information. We find that the `Canu` contigs named tig8 and tig4 overlap in the SG. tig1 and tig8 are linked by a long path involving 491922 bp. This long path can be explained by looking at how tig1 has been built by `Canu`: the path goes through a large 'loop' (see Supplementary Figure S2) which corresponds to a repeat in the reference (Figure 2c). The repeat (of length 460 kb) was not resolved by `Canu`, leading to a region of about 440 kb missing from the assembly between tig1 and tig8, which explains why the shortest path between both contigs contains as many as 491922 bp. We further checked that the path of length 755235 bp between tig1 and tig4 indeed contains reads from tig8, and is therefore redundant. By aligning raw reads and `Canu` corrected reads to the reference genome, we observe a drop of raw reads coverage (around 8×) in the region between tig8 and tig4. This likely explains why `Canu` failed to connect both contigs.

As a side note, a `Canu` assembly of the same dataset with twice higher read coverage (40×) yielded a two-contig assembly, also with same pattern as in between tig8 and tig4. An older version of `Canu` (1.6) fully resolved the 40× dataset into a single contig, likely due to changes in how reads are corrected and trimmed between version 1.6 and 1.7.

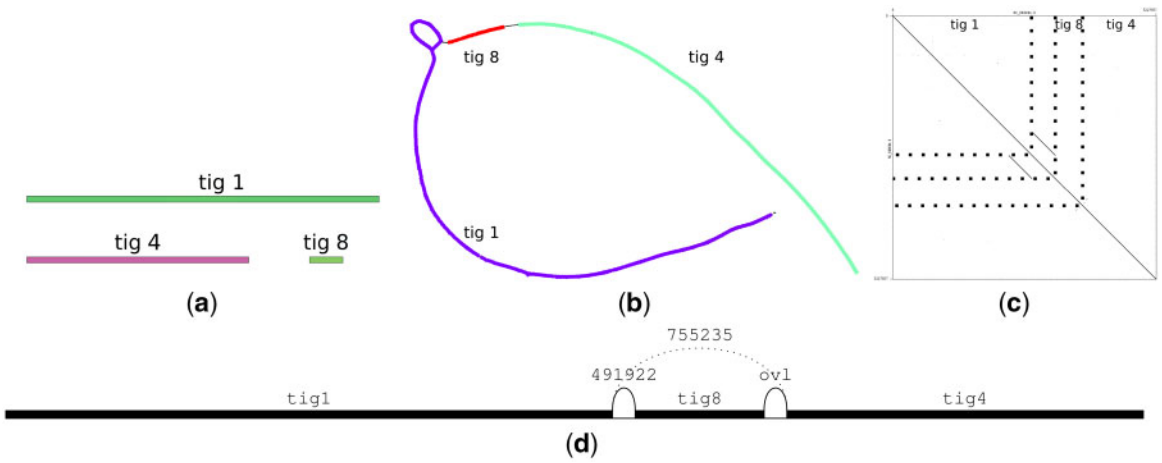


**Fig. 2.** Graph analysis of a synthetic dataset (*T.roseus*). (**a**) Contig graph produced by `Canu` (visualized using Bandage): three contigs, no edge. (**b**) SG built from `Minimap2` overlaps, on which connected components of the `Canu` BOG are colored. (**c**) Dot-plot of the *T.roseus* genome (NC_018014.1) aligned against itself, showing a long tandem repeat. (**d**) The AAG with `Canu` contigs ordered according to their position on the *T.roseus* reference. If two contigs overlap, no length is given and instead the link is labeled 'ovl'

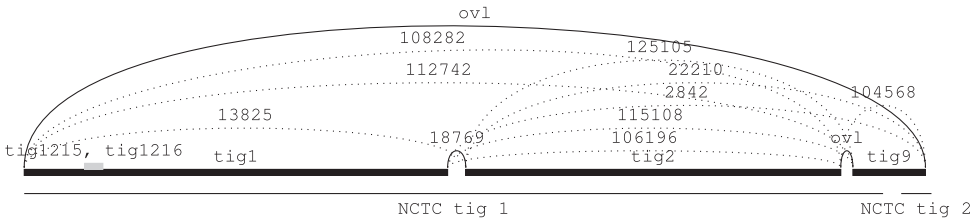


**Fig. 3.** Mapping of `Canu` contigs (bold horizontal lines) against NCTC12123 assembly (the two thin horizontal lines). Links between contigs give the length (in bp) of the shortest path in SG between reads at extremities. If two contigs overlap, no length is given and instead the link is labeled with 'ovl'. Plain links are paths that are compatible with the sequential order of contigs given by mapping to the NCTC assembly, and dotted links are all other paths

**Table 1.** Assemblies and contig graphs statistics for an excerpt of nine NCTC datasets (full tables in Supplementary Tables S2 and S3), consisting of eight datasets where Hamiltonian cycle search succeeded, and the NCTC5050 dataset discussed in Section 5

| | NCTC contigs | | | `Canu` contigs | | | No. of nodes | No. of dead-ends in | | No. of edges in AAG | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| NCTC ID | chr | pls | und | chr | pls | und | in AAG | Contig graph | AAG | Total | Single adjacency | Multiple adjacency |
| NCTC10006 | 1 | 0 | 0 | 3 | 0 | 0 | 4 | 2 | 2 | 4 | 2 | 0 |
| NCTC10332 | 1 | 0 | 0 | 12 | 0 | 0 | 8 | 8 | 4 | 24 | 0 | 3 |
| NCTC10444 | 1 | 0 | 0 | 7 | 0 | 0 | 8 | 3 | 0 | 24 | 0 | 6 |
| NCTC10702 | 1 | 1 | 1 | 3 | 3 | 0 | 4 | 4 | 4 | 4 | 0 | 0 |
| NCTC12123 | 2 | 3 | 0 | 5 | 4 | 1 | 6 | 4 | 1 | 12 | 1 | 2 |
| NCTC12132 | 1 | 0 | 0 | 2 | 0 | 2 | 4 | 4 | 2 | 4 | 1 | 0 |
| NCTC13125 | 1 | 2 | 4 | 6 | 3 | 1 | 6 | 0 | 0 | 12 | 0 | 4 |
| NCTC13463 | 1 | 1 | 4 | 5 | 2 | 2 | 4 | 0 | 0 | 3 | 2 | 0 |
| NCTC5050 | 2 | 3 | 0 | 4 | 2 | 3 | 6 | 6 | 0 | 12 | 3 | 0 |

*Note:* AAGs are constructed using a SG built from `Minimap2` overlaps and `Canu` contig extremities. The 'contig graph' column corresponds to the final assembly graph produced by `Canu`; 'chr': number of chromosomal contigs; 'pld': number of plasmid contigs; 'und': number of other contigs. Note that some of `Canu` 'chr' contigs may be contained in others, therefore the '# nodes in AAG' column corresponds to twice the number of non-contained contigs.

## 5.3 Investigation of 45 unresolved NCTC assemblies

We performed the same type of analysis on the 45 NCTC samples. A `Minimap2` AAG was constructed for each dataset using SG and `Canu` contig extremities. Assembly and AAG statistics are presented in Table 1 for an excerpt of the dataset. Full statistics and more details are given in Supplementary Tables S2, S6 and S7. There we observe that the number of contigs in `Canu` and `Miniasm` assemblies is generally higher than in the assemblies made by NCTC. Nevertheless the sum of lengths of chromosomal contigs is about the same in all assemblies (Supplementary Table S8).

### 5.3.1 Case study of two NCTC datasets

We closely examine two NCTC datasets that contain interesting patterns, through the lens of a ground truth obtained by remapping `Canu` contigs against respective NCTC assemblies using BWA-mem (Li, 2013).

*NCTC12123* This dataset was assembled into five chromosomal contigs by Canu, including two contigs that are contained in others and are automatically discarded by our pipeline (see Figure 3).

The assembly is made of two large contigs (tig1 and tig2) and a shorter one (tig9) totaling 4.78 Mbp. Miniasm produces also five chromosomal contigs, including three small ones. Both Canu and Miniasm contig graphs are made of two components. HINGE produces a single-component assembly graph but does not resolve it (because it detects multiple possible traversals). Finally, the NCTC assembly consists of two chromosomal contigs: one being 4.69 Mbp long and the other 21 kb long. Contigs tig1 and tig2 both map over the large NCTC contig, while tig9 maps to both NCTC contigs. Using the AAG on Canu contigs (see Figure 3), one can observe that a number of scaffolding scenarios could be made following this graph. Interestingly, based on the mapping of the three contigs on the larger contig of the NCTC assembly, edges of smaller weight (i.e. shortest paths) tend to be associated with true contig adjacencies. In this example, low-weight Hamiltonian cycles (Section 4.5) yield two likely contig orders (see Supplementary Figure S3). This SG analysis thus enabled to retrieve an adjacency that was missed by Canu. It also confirms the multiple traversals prediction of HINGE, further reducing the number of putative contig orders to only two.

*NCTC5050* This dataset is assembled into four chromosomal contigs by Canu, including one that is contained in another. The Canu contig graph is 'fully' fragmented as each contig is its own connected component. There is no reference genome for this strain, and we chose as ground truth the NCTC assembly consisting of two contigs. One is entirely covered by a Canu contig, and the other contains the three remaining contigs (see Supplementary Figure S4). In the following, $x_s$ and $x_e$ denote left (resp. right) extremities of a contig $x$. We found *single* (i.e. non-repeat) adjacencies between $\text{tig1}_s$/$\text{tig23}_s$, $\text{tig1}_e$/$\text{tig10}_s$, $\text{tig10}_e$/$\text{tig23}_s$ that were confirmed by mapping to the longest contig from the NCTC assembly. Together, these single adjacencies suggest a putative scaffolding scenario: tig1—tig10—tig23 (reversed). This scenario is also the top-ranked one proposed by our Hamiltonian path search procedure (see below).

We also mapped corrected and raw reads to the junction for validation (see Supplementary Figure S5). We observe a drop of coverage at this location (see reads mapping in Supplementary Figure S4) that is likely the cause of assembly fragmentation. Therefore, again in this dataset the path search operation enabled to recover a link between contigs that was discarded by the assembler due to a drop in sequencing coverage.

#### 5.3.2 Path search enables to recover adjacency between contigs

Table 1 reports statistics of paths found between Canu contigs by our method for a subset of 9 NCTC datasets (for the full dataset, see Supplementary Table S3). We first focus on unambiguous contig adjacencies recovered by our pipeline. Single adjacency edges are only found in six out of nine datasets, yet across the entire dataset of 45 samples, 60.4% of all single adjacency edges (43 in total) are found in samples that have a sequencing coverage below 38×, and only 17 single adjacency edges are found in datasets with coverage above 38×. This is likely due to the error-correction step in assemblers that is less effective in low-coverage datasets (even when the true-sequencing coverage is given to the assembler as a parameter), which in turn causes assembly fragmentation. Our method therefore enables to recover single adjacency edges between contigs that were fragmented due to this effect.

To measure whether the Canu contig graphs could be used as-is to recover contig order, we counted the number of contig extremities that are not linked to any other extremity (i.e. *dead-ends*). Those are contigs for which no chromosomal order can be reliably inferred. In 35 out of the 45 datasets (seven out of nine in Table 1), the Canu contig graph has some dead-end extremities (between 1 and 23). In principle dead-ends extremities should not exist in circular bacterial assembly graphs, except for linear chromosomes. Assemblers, here Miniasm and Canu, do not report all true contig adjacencies. In contrast, our method enables to recover some of these adjacencies and lower the number of dead-ends in 23 out the 37 datasets (and all but one dataset in Table 1).

**Table 2.** Average statistics of AAGs using a SG built from Minimap2 overlaps on Canu contigs across the 38 NCTC datasets with two or more contigs, after size and classification filters

| Mean number of | |
|---|---|
| Canu contigs | 4.32 |
| Edges in AAG | 32.67 |
| Theoretical max. edges in AAG | 41.83 |
| Distant edges | 28.64 |
| All adjacency edges | 4.02 |
| Single adjacency edges | 1.16 |
| Multiple adjacency edges | 2.86 |
| Dead-ends in Canu contigs | 4.94 |
| Dead-ends in AAG, adjacency edges | 2.70 |

*Note:* All rows are as per definitions in Section 4.4. 'Theoretical max. edges': number of possible edges in each AAG. 'Dead-ends in AAG, adjacency edges': number of dead-ends in the AAG when only adjacency edges are considered, i.e. distant edges are deleted.

Table 2 summarizes average AAG statistics over all 38 datasets on Canu contigs (per-dataset results in Supplementary Table S3). Results for Miniasm contigs are shown in Supplementary Tables S4 and S3. On average, Canu contig graphs contain 4.32 nodes (5.86 extremities), among which 4.94 extremities are dead-ends. The AAG enables to reduce the number of dead-end extremities to 2.7 (45% lower), through the discovery of 1.16 single adjacency edges and 2.86 multiple adjacency edges in the AAG per dataset on average. The reduction is also significant for Miniasm contigs but not as high (31%, Supplementary Table S4). Note that these adjacencies are 'real' in the sense that they are all supported by paths of overlapping reads of total nucleotide length less than 10 kb, yet a number of them may be caused by repetitions. An upper bound on the ability to mine paths in the SG is given by the theoretical maximal number of edges in the AAG (41.83 edges). Our method is on average 78% close to this bound for Canu contigs (resp. 90.1% for Miniasm) as it discovered 32.67 edges per dataset (resp. 85.1). We note that large fraction (87%) of discovered edges were classified as distant edges, yet the remaining adjacency edges are informative as they significantly contribute to removing dead-ends in the contig graph.

#### 5.3.3 Contig order search retrieves parsimonious assembly scenarios

While the work done in the previous section helps to recover contig adjacencies, the presence of multiple adjacency edges due to repetitions often prevents us from unambiguously inferring a contig order. We applied the Hamiltonian cycle procedure presented in Section 4.5 to determine likely contig orderings. Figure 4 shows orderings sorted by weight across 23 datasets on which the method could successfully be executed (connected AAG, low number of edges).

A ground truth is known in only 8 of those datasets. Among them, the lowest-weight scenario is ranked first in 3 datasets, 2nd in 2 datasets, 3rd in 1, 4th in 1 and 38th in the last one.

These results suggest that the correct assembly scenario is likely to be one of the top predictions made by our parsimonious

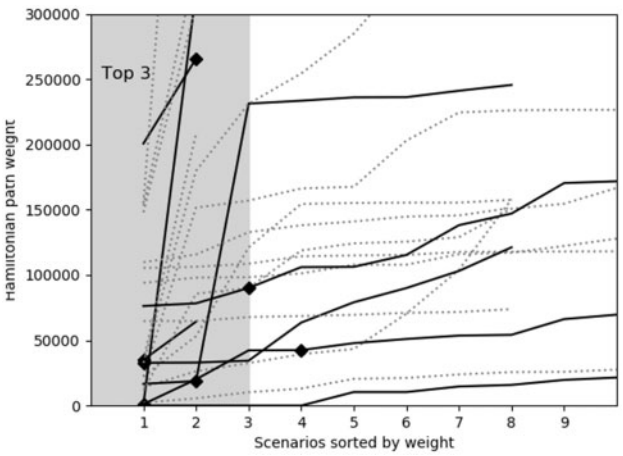


**Fig. 4.** Weights of scenarios in AAGs. Each curve corresponds to the sorted list of Hamiltonian cycles, sorted by weight. If a ground truth is known, a diamond symbol marks the correct assembly scenario

Hamiltonian cycle procedure. However finding many fragmented datasets that also have a ground truth is inherently difficult, thus further work is needed to confirm this hypothesis. Also, datasets where several scenarios have similar weights (i.e. curves that 'plateau' in Figure 4) will possibly be more challenging to resolve using this method. Yet for many samples with fragmented assemblies, parsimonious assembly scenarios are a promising approach to explore a limited number of hypotheses that could further be validated using long-range PCR to finish the genome.

## 6 Discussion

We presented a set of concepts to provide novel insights on fragmented long-read bacterial genome assemblies. By searching for paths of overlapping raw reads between extremities of contigs, we construct an AAG that recovers unreported adjacencies between contigs. We demonstrate several usages of this graph: to provide a more informative representation of fragmented assemblies, to examine repeat structures, and to propose likely contig orderings. In our tests, the AAGs of NCTC datasets recover edges for nearly half (45%) of the dead-end nodes in `Canu` contig graphs, on average. We further show a link between the lowest-weight Hamiltonian cycles in the AAG and the true contig order. We highlight that our method solely relies on the raw data and information produced by assemblers at various stages of their pipelines and, when our contig classification step is skipped, no reference genome nor external information (e.g. genome map, BLAST database) are used.

Our method hinges on directly constructing a SG on the raw reads, after a relatively conservative chimera removal step. Doing so avoid biases that may be introduced in the read trimming and error-correction steps of an assembler. Indeed, overlaps between reads may become shorter or even absent after error-correction. For instance on the 45 NCTC datasets that we analyzed, the number of edges in SGs built from `Canu` error-corrected reads is reduced by 41.4% compared with the SGs of raw reads. We have classified edges in the AAG, by considering their underlying nucleotide lengths and whether they contain reads that belong to other contigs. To go further, one could define confidence metrics, e.g. based on local graph structures.

Due to a combination of engineering choices and the inherent difficulty of visualizing large assembly graphs, our software has only been tested on bacterial genomes and is unlikely to readily run on larger genomes. However, the techniques presented here (AAG, path search between contig extremities, weighted Hamiltonian cycles) are not specific to bacterial assembly, and should in principle be applicable to small and large eukaryotes. However more work would be needed, e.g. to scale path search to thousands of contigs, refine thresholds (contig filter, adjacency edges), handle inter-chromosomal repeats, and an evaluation of the relevance of Hamiltonian cycles for larger genomes.

We stress that our techniques currently do not aim at detecting misassemblies within contigs. We also did not focus on the difficulty of running multiple assembly programs, but we note that the process has previously been reported to be challenging (Lariviere *et al.*, 2018). Our work is also orthogonal to assembly reconciliation (Alhakami *et al.*, 2017), which consists of constructing a higher-contiguity assembly by merging the results of multiple assemblers.

No attempt was made to optimize the detection of overlaps between reads though this could be a direction for improvement. Finally, automatic post-assembly improvements based on the AAG would be a natural extension of this work. One could use the AAG to design an oracle that suggests a limited number of (long-range) PCR experiments for resolving individual repeats.

## Acknowledgement

The authors are grateful to Samarth Rangavittal, Monika Cechova, Jason Chin, Jason Hill and Christopher W. Wheat for discussions that led to this project, Gautam Kamath for guidance on reproducing NCTC analyses with `HINGE`, Antoine Limasset and anonymous reviewers for helpful comments on the manuscript.

## Funding

This work was supported by an Inria doctoral grant and the INCEPTION project (PIA/ANR16CONV0005).

## References

Alhakami,H. *et al.* (2017) A comparative evaluation of genome assembly reconciliation tools. *Genome Biol.*, **18**, 93.

Antipov,D. *et al.* (2016) hybridSPAdes: an algorithm for hybrid assembly of short and long reads. *Bioinformatics*, **32**, 1009–1015.

Bankevich,A. *et al.* (2012) SPAdes: a new genome assembly algorithm and its applications to single-cell sequencing. *J. Comput. Biol.*, **19**, 455–477.

Bosi,E. *et al.* (2015) MeDuSa: a multi-draft based scaffolder. *Bioinformatics*, **31**, 2443–2451.

Bresler,G. *et al.* (2013) Optimal assembly for high throughput shotgun sequencing. *BMC Bioinformatics*, **14**, S18.

Chin,C.-S. *et al.* (2016) Phased diploid genome assembly with single-molecule real-time sequencing. *Nat. Methods*, **13**, 1050–1054.

Clark,S.C. *et al.* (2013) ALE: a generic assembly likelihood evaluation framework for assessing the accuracy of genome and metagenome assemblies. *Bioinformatics*, **29**, 435–443.

Ghodsi,M. *et al.* (2013) De novo likelihood-based measures for comparing genome assemblies. *BMC Res. Notes*, **6**, 334.

Hunt,M. *et al.* (2013) REAPR: a universal tool for genome assembly evaluation. *Genome Biol.*, **14**, R47.

Hunt,M. *et al.* (2015) Circlator: automated circularization of genome assemblies using long sequencing reads. *Genome Biol.*, **16**, 294.

Kamath,G.M. *et al.* (2017) HINGE: long-read assembly achieves optimal repeat resolution. *Genome Res.*, **27**, 747–756.

Koren,S., and Phillippy,A.M. (2015) One chromosome, one contig: complete microbial genomes from long-read sequencing and assembly. *Curr. Opin. Microbiol.*, **23**, 110–120.

Koren,S. *et al*. (2017) Canu: scalable and accurate long-read assembly via adaptive k-mer weighting and repeat separation. *Genome Res*., **27**, 722–736.

Koster,J., and Rahmann,S. (2012) Snakemake–a scalable bioinformatics workflow engine. *Bioinformatics*, **28**, 2520–2522.

Kremer,F.S. *et al*. (2017) Approaches for in silico finishing of microbial genome sequences. *Genet. Mol. Biol*., **40**, 553–576.

Lariviere,D. *et al*. (2018) Understanding trivial challenges of microbial genomics: An assembly example. Cold Spring Harbor Laboratory, doi: 10.1101/347625.

Lau,B. *et al*. (2016) LongISLND: in silicosequencing of lengthy and noisy datatypes. *Bioinformatics*, **32**, 3829–3832.

Li,H. (2013) Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. *arXiv:1303.3997*.

Li,H. (2016) Minimap2 and Miniasm: fast mapping and *de novo* assembly for noisy long sequences. *Bioinformatics*, **32**, 2103–2110.

Lin,Y. *et al*. (2016) Assembly of long error-prone reads using de Bruijn graphs. *Proc. Natl. Acad. Sci. USA*, **113**, E8396–E8405.

Loman,N.J. *et al*. (2015) A complete bacterial genome assembled de novo using only nanopore sequencing data. *Nat. Methods*, **12**, 733–735.

Miller,J.R. *et al*. (2008) Aggressive assembly of pyrosequencing reads with mates. *Bioinformatics*, **24**, 2818–2824.

Myers,G. (2005) The fragment assembly string graph. *Bioinformatics*, **21**, ii79–ii85.

Olson,N.D. *et al*. (2017) Metagenomic assembly through the lens of validation: recent advances in assessing and improving the quality of genomes assembled from metagenomes. *Brief. Bioinform*. doi: 10.1093/bib/bbx098/4075034.

Parks,D.H. *et al*. (2015) CheckM: assessing the quality of microbial genomes recovered from isolates, single cells, and metagenomes. *Genome Res*., **25**, 1043–1055.

Phillippy,A.M. *et al*. (2008) Genome assembly forensics: finding the elusive mis-assembly. *Genome Biol*., **9**, R55.

Rahman,A., and Pachter,L. (2013) CGAL: computing genome assembly likelihoods. *Genome Biol*., **14**, R8.

Simão,F.A. *et al*. (2015) BUSCO: assessing genome assembly and annotation completeness with single-copy orthologs. *Bioinformatics*, **31**, 3210–3212.

Treangen,T.J. *et al*. (2009) Genesis, effects and fates of repeats in prokaryotic genomes. *FEMS Microbiol. Rev*., **33**, 539–571.

Utturkar,S.M. *et al*. (2017) A case study into microbial genome assembly gap sequences and finishing strategies. *Front. Microbiol*., **8**, 1272.

Vezzi,F. *et al*. (2012) Reevaluating assembly evaluations with feature response curves: GAGE and assemblathons. *PLoS One*, **7**, e52210.

Walker,B.J. *et al*. (2014) Pilon: an integrated tool for comprehensive microbial variant detection and genome assembly improvement. *PLoS One*, **9**, e112963.

Wick,R.R. *et al*. (2015) Bandage: interactive visualization of de novo genome assemblies. *Bioinformatics*, **31**, 3350–3352.

Wick,R.R. *et al*. (2017) Unicycler: resolving bacterial genome assemblies from short and long sequencing reads. *PLoS Comput. Biol*., **13**, e1005595.

Ye,C. *et al*. (2016) DBG2olc: efficient assembly of large genomes using long erroneous reads of the third generation sequencing technologies. *Sci. Rep*., **6**, 31900.

Zimin,A.V. *et al*. (2013) The MaSuRCA genome assembler. *Bioinformatics*, **29**, 2669–2677.