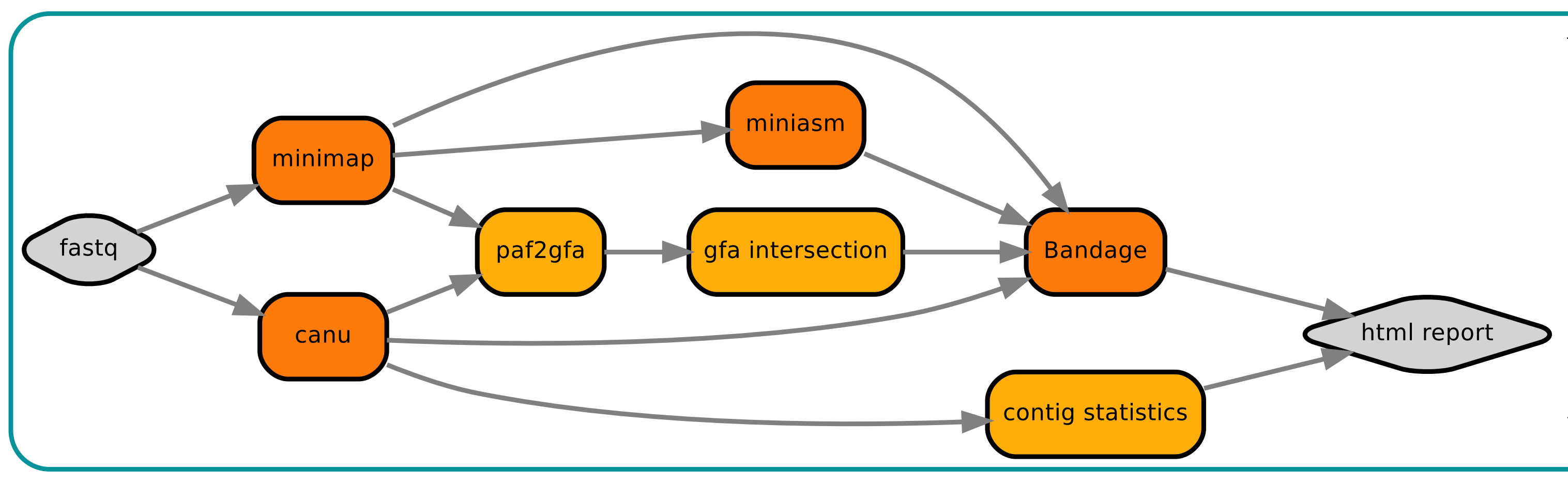
Debugging long-read genome assemblies using string graph analysis

Pierre MARIJON¹ , Jean-Stéphane VARRÉ² and Rayan CHIKHI²

¹ Inria, Université de Lille, CNRS, Centrale Lille, UMR 9189 - CRIStAL - Centre de Recherche en Informatique Signal et Automatique de Lille, F-59000 Lille, France

² Univ. Lille, CNRS, Centrale Lille, Inria, UMR 9189 - CRIStAL - Centre de Recherche en Informatique Signal et Automatique de Lille, F-59000 Lille, France

Third-generation long-read sequencing technologies tackle the repeat problem in genome assembly by producing reads that are long enough to span most repeat instances. In principle one expects that with such reads most bacterial genomes will be assembled into a single contig [1].
However in practice, some datasets fail to be perfectly assembled even with leading assemblers, and are fragmented into a handful of contigs.
As a mean to investigate those cases, we consider the string graphs that are generated by assemblers during intermediate stages of the assembly process. We seek to establish a coherent framework for analyzing these graphs to help us determine the biological causes that led the assembler to output shorter contigs.



We built an **assembly analysis software** that takes as input a set of 3rd generation reads, runs **Canu** [2] and **mini{asm|map}** [3] (but could include other assembly tools), and analyzes their outputs at several stages.
Canu and Mini{asm|map} have several differences:
i. **Canu** corrects reads before assembly while **mini{asm|map}** generates contigs directly from trimmed but uncorrected reads
ii. Different read overlappers (but related strategies), and different assembly heuristics.
Our **Snakemake** pipeline includes home-made graph assembly tools (light orange nodes) and provides the user with an HTML report.

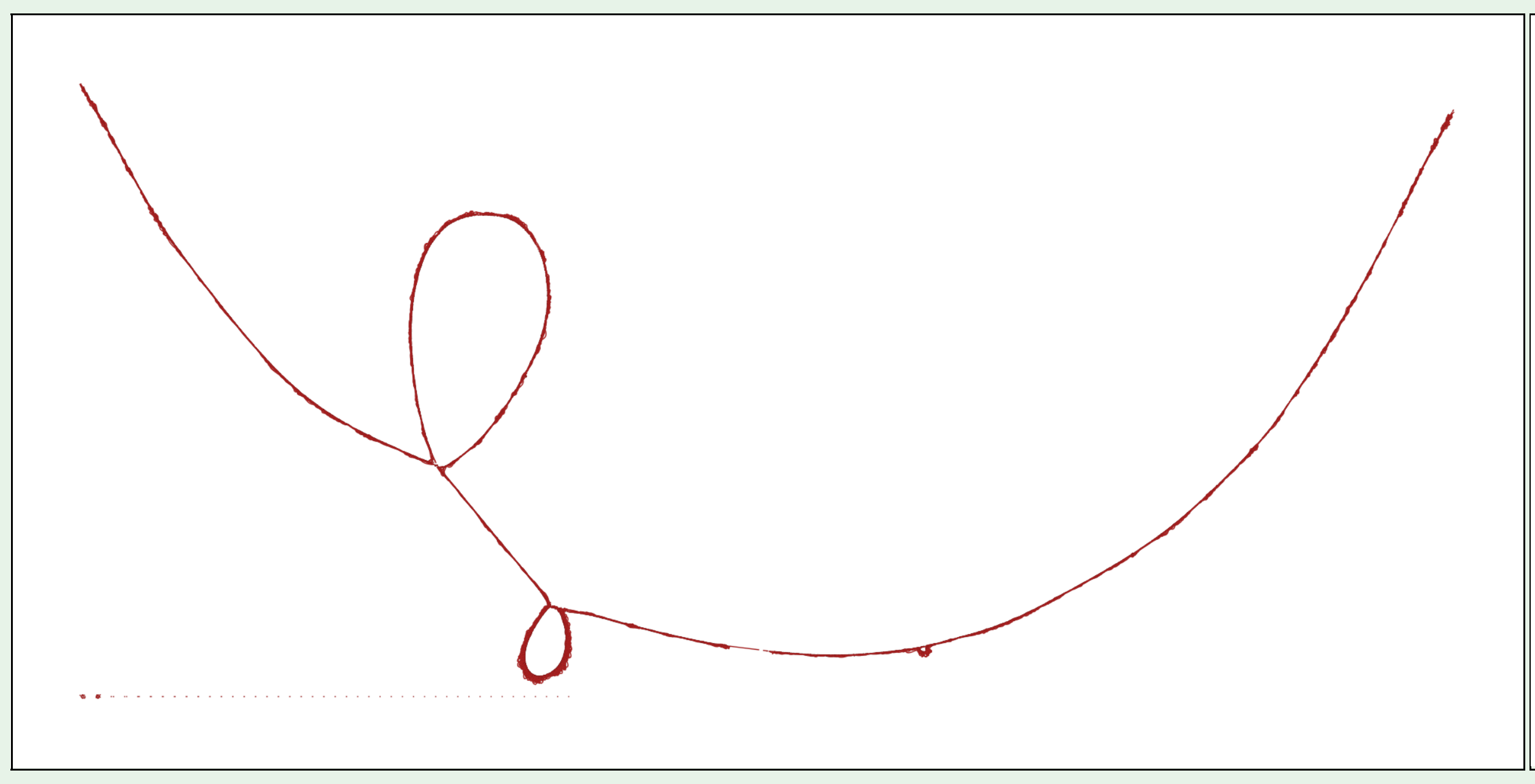
Assembly report for : longislnd/20x/terriglobus_roseus.fastq (simulated)

The **PAF** (Pairwise Alignment Format) file: all read overlaps seen by **Canu**.
i. **Canu** (MHAP) computes a signature for each read and uses it to get an estimate of the Jaccard distance between two reads
ii. **Canu** then performs overlaps for all pairs of reads with low Jaccard distance
We converted the **PAF** file to a graph (**GFA** format) for easier manipulation and representation.

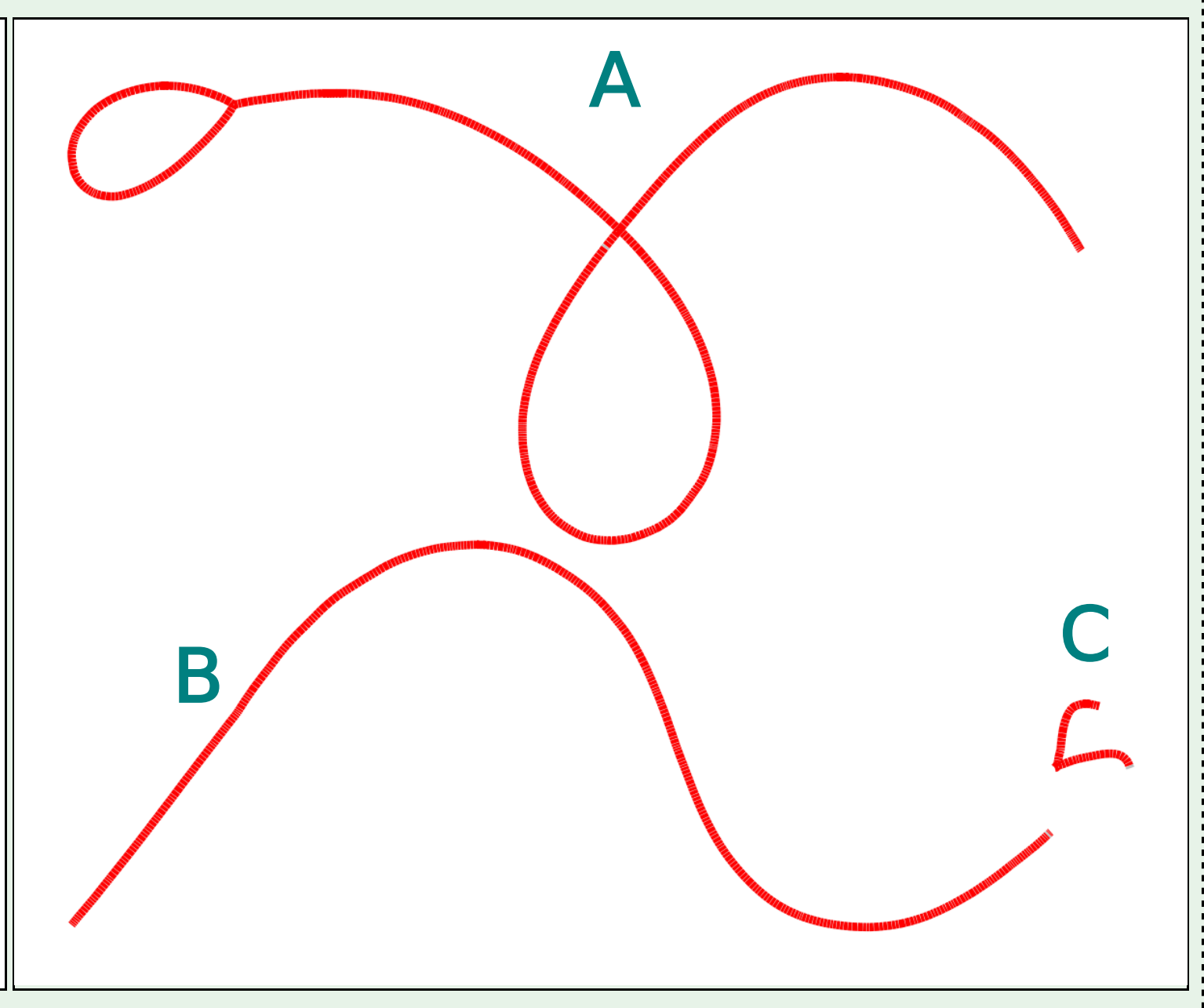
Canu

Number of reads	1019
Number of contigs	3
Total size	4769751

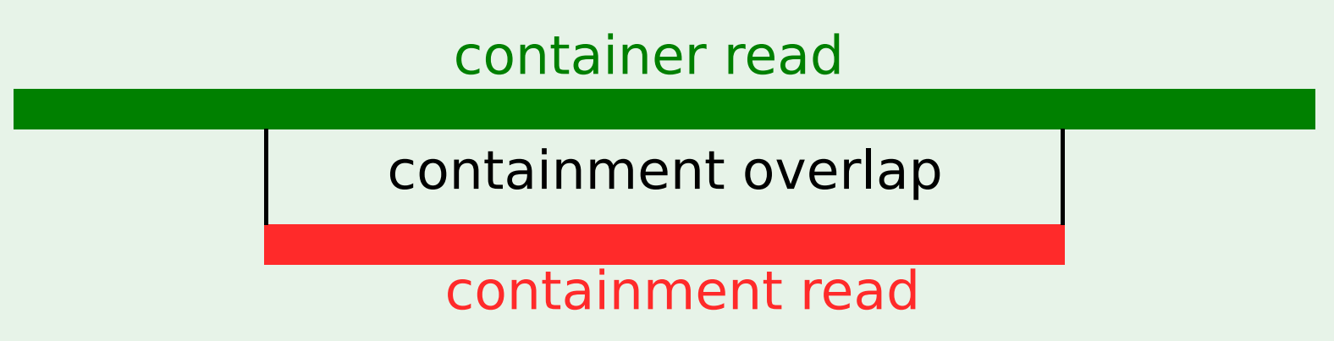
PAF



BOG



The **BOG** (Best Overlap Graph) is the main graph of **Canu** [2].
i. Canu keeps just the two best overlaps (the longest) for each read
ii. it checks that the overlap error rate is less than some dynamic threshold
iii. Canu constructs unitigs and contigs based on the **BOG**
Contained reads are removed:

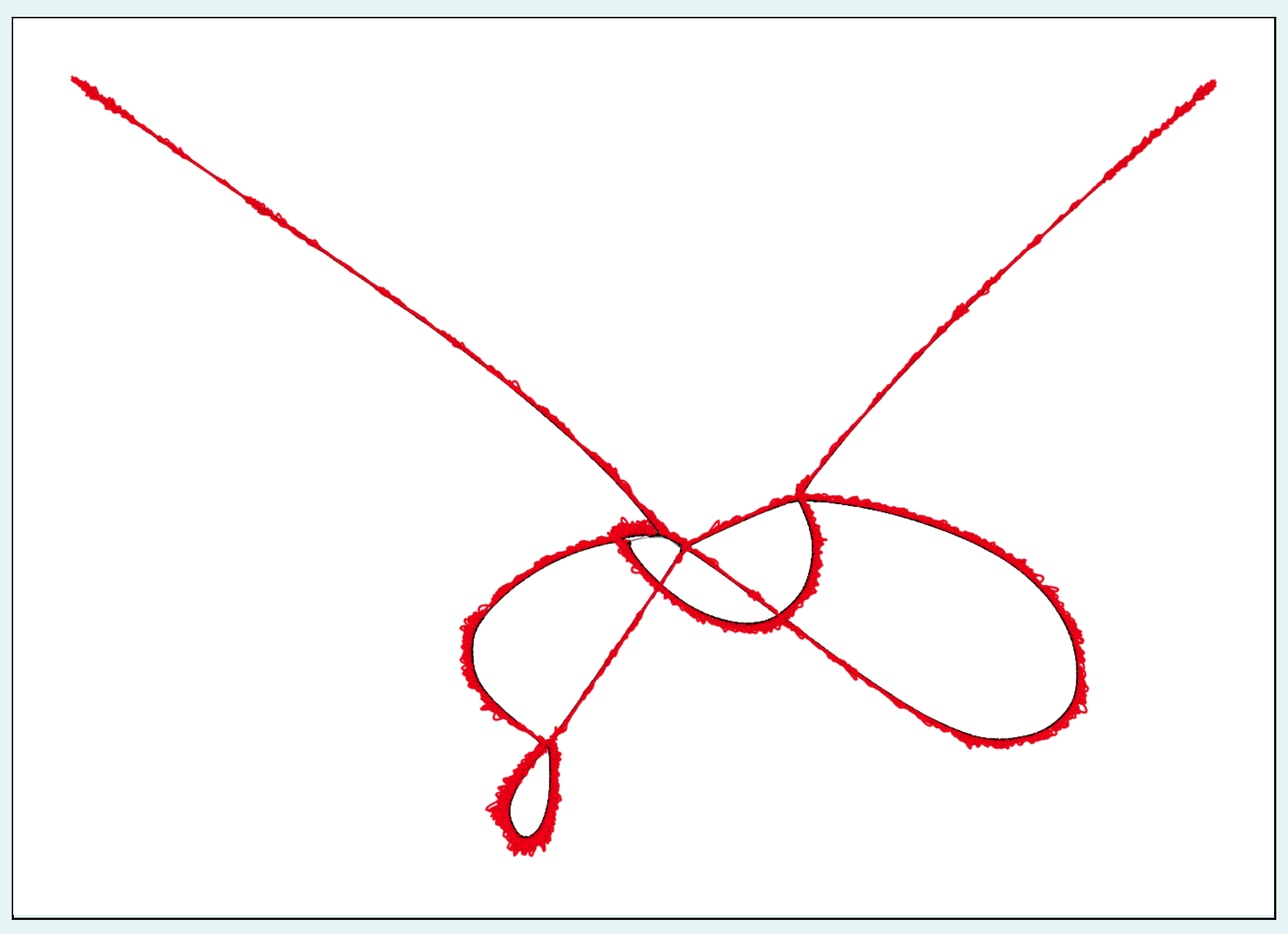


The **minimap** [3] software also creates a **PAF** file:
i. it computes a hash table that associates minimal kmers to each read
ii. two reads overlap if they share some number of minimal kmers
iii. **minimap** use this overlap to generate this **PAF**

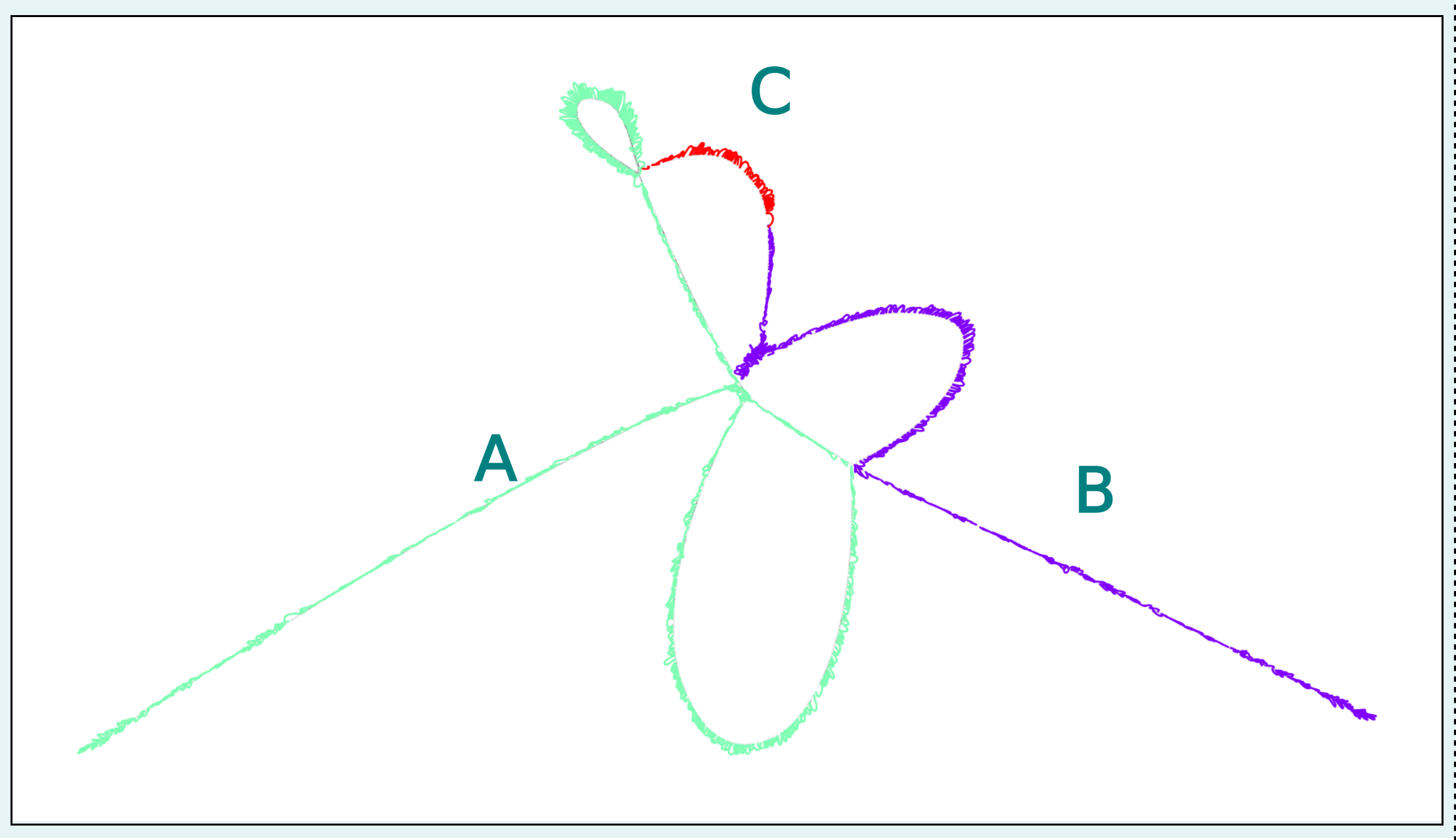
Minimap

Number of reads	11381
Number of overlaps	232627

PAF



Canu contigs on minimap PAF

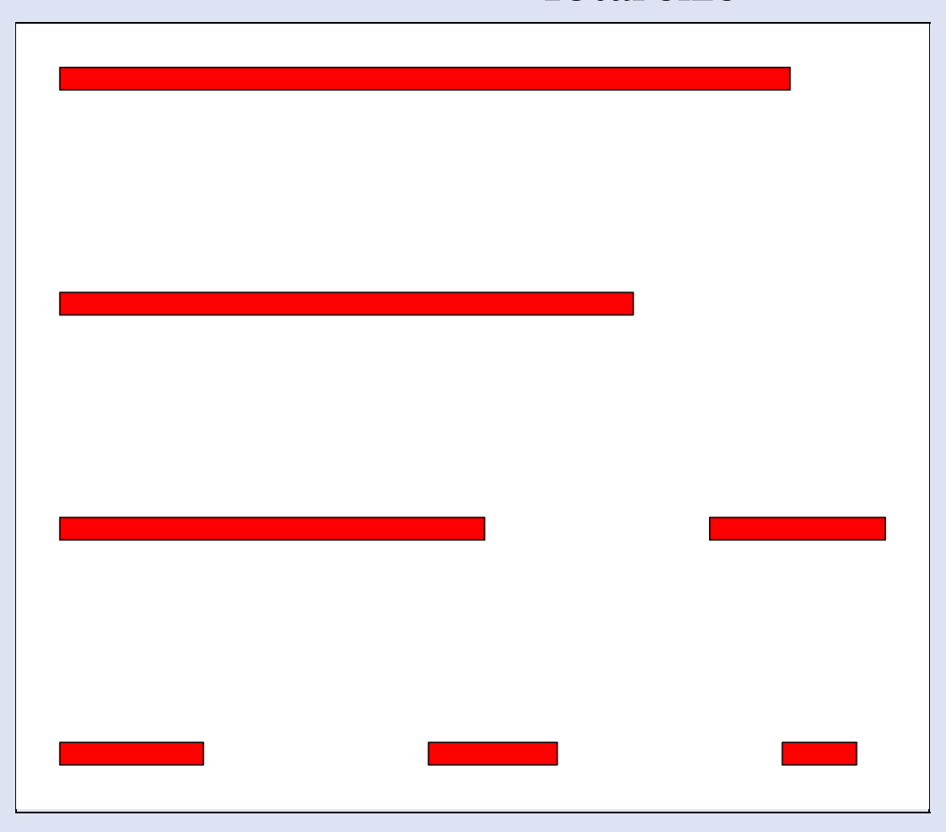


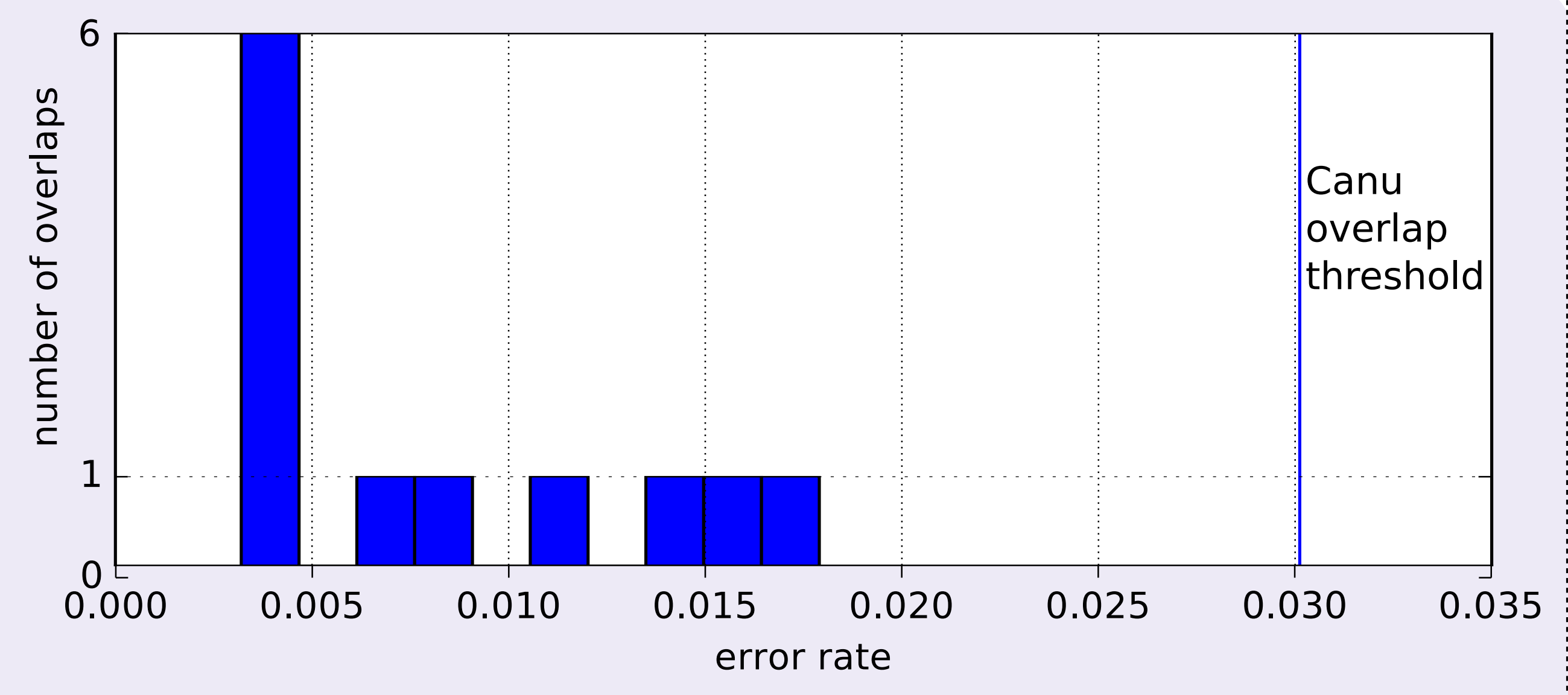
Graph projection
i. *idea:* compare **Canu** and **minimap** assembly graphs on the same graph
ii. we include a read if it is present in both assembly graphs
iii. we assign a different color to each connected component according to the **Canu BOG**

miniasm uses the **minimap PAF** to assemble contigs:
i. for each read, bases that are not covered by at least one other read are removed
ii. assembly graph is created from the PAF, and transitive edges, small bubbles and tips are removed
iii. contigs are non-branching paths

Miniasm

Number of contigs	7
Total size	5007205

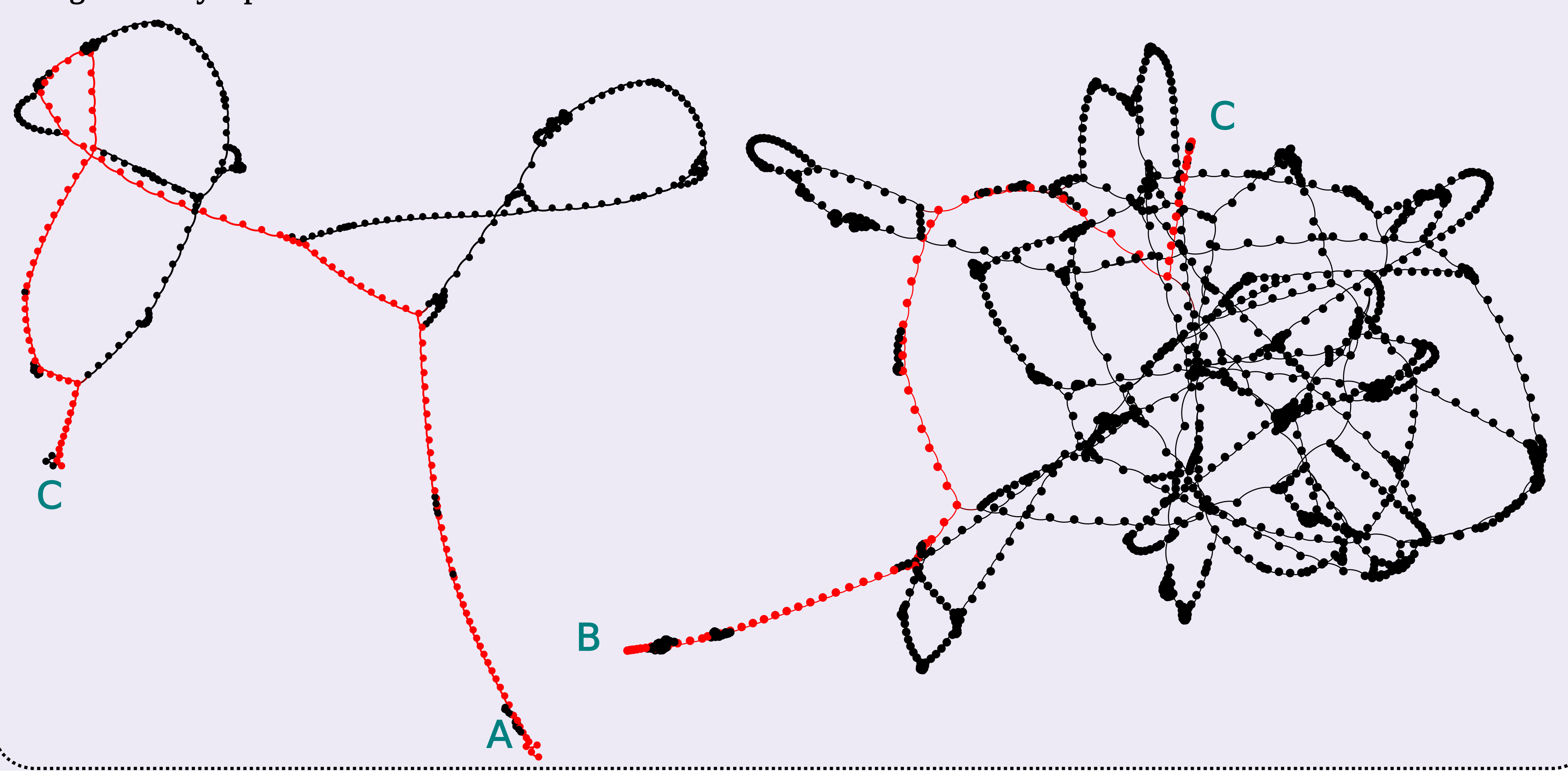




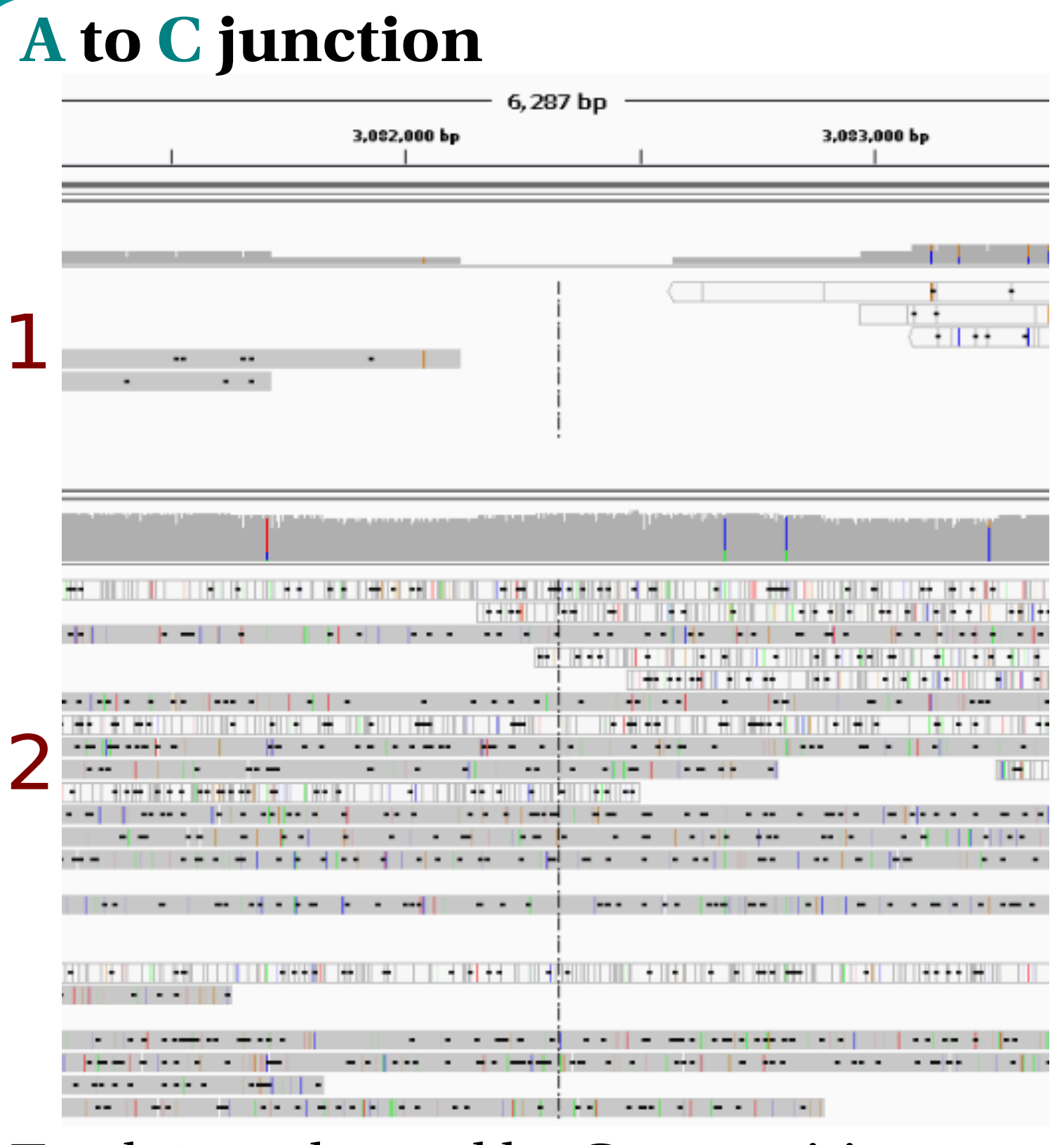
Overlap read error rate histogram at the extremities of **Canu** contigs.
Overlaps with a high error rate induce the end of a contig. But here, no overlap has higher error rate than a threshold determined by **Canu**.

We went further and examined the assembly graph around the extremities of contigs by going back to the **Canu PAF**.
To generate the graphs on the right, we search for the shortest path of overlaps between contig read extremities. We also generated a DAG using depth-first search, and took the intersection of the two DAGs from both contig extremities.

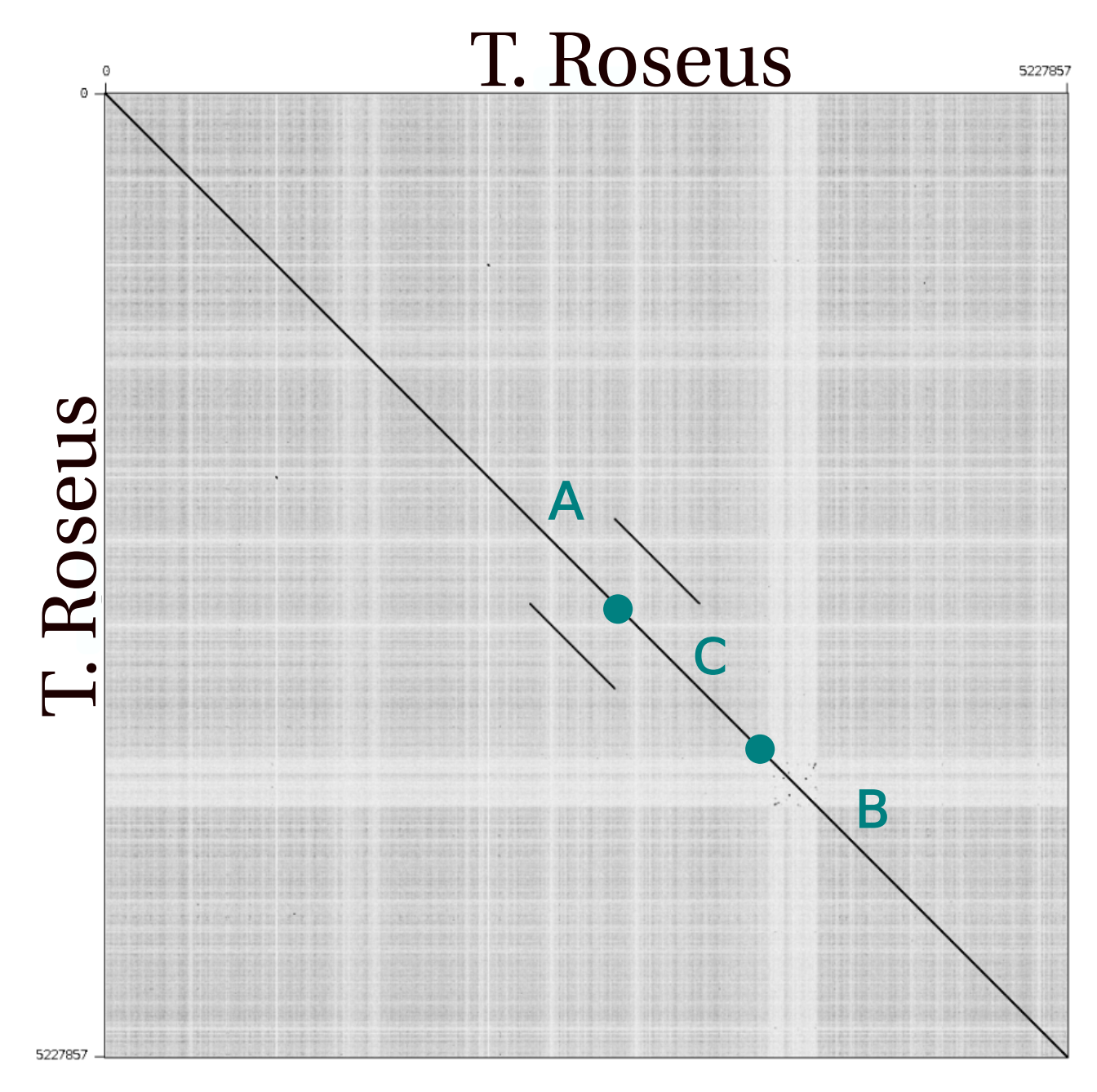
Contig extremity report



A to C junction



Track 1 reads used by **Canu** unitigger
Track 2 raws reads



To understand why 3rd generation assembly tools create fragmented assemblies:
i. We analyze the quality and the length of the overlaps of the reads at contig extremities.
ii. We examine overlaps with good scores (with respect to Canu thresholds), but that were discarded by Canu.
iii. We build a string graph (following Myers [4]) of reads around contig extremities (using the overlap information computed by Canu), and then we search for a path between contig extremities.

Our pipeline provides us with insights as to why and where an assembly failed. From those first observations, we will improve and automate the extraction of graphs between reads at contig extremities, with further annotations. Finally, we hope to be able to propose alternative assemblies.

[1] Sergey Koren and Adam M Phillippy. One chromosome, one contig: complete microbial genomes from long-read sequencing and assembly. Current Opinion in Microbiology, 23:110–120, 2015.
[2] Sergey Koren, and al. Canu: scalable and accurate long-read assembly via adaptive k-mer weighting and repeat separation. Genome Research, page gr.215087.116, 2017.
[3] Heng Li. Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences. Bioinformatics, 32(14):2103–2110, 2016.
[4] Eugene W Myers. The fragment assembly string graph. Bioinformatics, 21 (suppl_2): ii79-ii85, 2005.