# Seminar 1 - Report

Katarina Gaćina

## 1 First tasks

E. coli ONT (https://www.ncbi.nlm.nih.gov/sra/?term=SRR12801740) and HiFi (https://www.ncbi.nlm.nih.gov/sra/?term=SRR11434954) datasets were downloaded from NCBI by using SRAtoolkit. Both, Pacific Biosciences (PacBio) Hifi and Oxford Nanopore Technologies (ONT) are long-read sequencing technologies [1].
Example of SRAtoolkit commands that were used: [2]

```
prefetch SRR11434954
fastq-dump SSRR11434954
```

Seqkit is a cross-platform and ultrafast toolkit for FASTA/Q file manipulation [3]. It was used for analyzing downloaded fastq files and subsampling them to about 50x.
Command stats was used for finding number of reads = num_seq and number of bases = sum_len [4]:

```
seqkit stats SRR12801740.fastq
seqkit stats SSRR11434954.fastq

file               format  type  num_seqs        sum_len  min_len  avg_len  max_len
SRR12801740.fastq  FASTQ   DNA     97,301  1,012,898,671      113   10,410  471,734

file               format  type   num_seqs          sum_len  min_len   avg_len  max_len
SRR11434954.fastq  FASTQ   DNA   1,789,131  23,122,913,014       46  12,924.1   26,294
```

Additionally, the reference https://ncbi.nlm.nih.gov/assembly/GCF_014117345.2 was downloaded from ReqSeq. It was used for calculating the coverages of downloaded datasets. Coverage is the factor of how much larger the set of reads is compared to the reference length [5]. It is desirable to have higher coverages because, for example, it reduces the impact of possible sequencing errors.
The used command for accessing the reference length was [4]:

```
seqkit fx2tab --name --length GCF_014117345.2_ASM1411734v2_genomic.fna
```

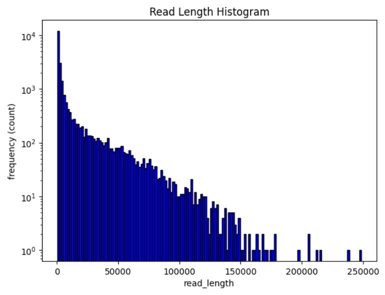
Calculated coverage for SRR12801740 (ONT) is 204.855x and for SRR11434954 (Hifi) is 4676.527601x.
Command sample was used for subsampling datasets to about 50x, instead of

suggested subseq, because subseq command is for getting specific subsequences by region, and sample for sampling sequences by number or proportion:

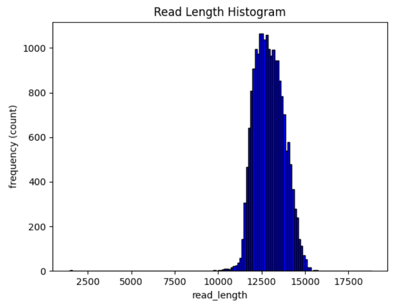
```
seqkit sample -p 0.244 -o downsampled_SRR12801740.fastq SRR12801740.fastq
seqkit sample -p 0.011 -o downsampled_SRR11434954.fastq SRR11434954.fastq
```

, where p is the wanted proportion of sequence to sample [4].

Read-length histograms were drawn in Python for downsampled datasets [6]. The code for it is shown in Figure 2. They show the distribution of read lengths. We want it to reflect the characteristics of the original dataset. In 1a we can see that the histogram is left-skewed, which means those higher values are infrequent. This histogram was drawn by using a log scale on the y-axis because of the large differences between frequencies. In 1b, the centre of the histogram is moved to the right and it looks normally distributed. The majority of read lengths for histogram 1b are in the range from 10000 to 15000, while for histogram 1a lengths go up to 250000. Therefore, the ONT dataset contains longer reads compared to the Hifi dataset.



(a) Histogram for downsampled SRR12801740

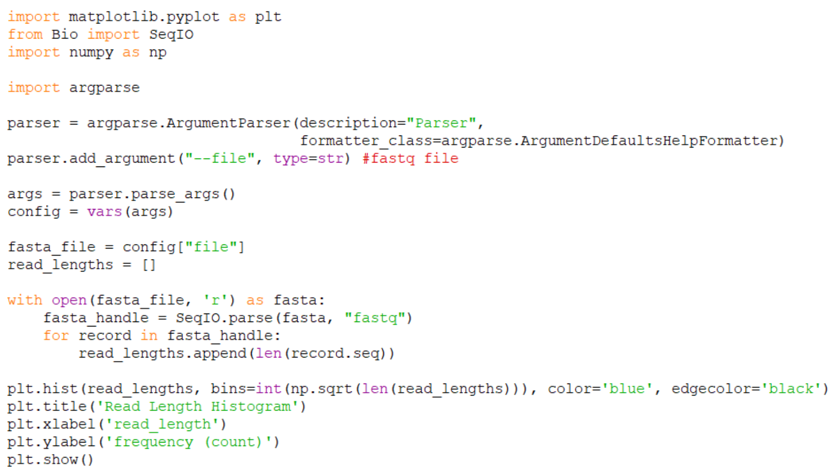


(b) Histogram for downsampled SSRR11434954

Figure 1: Read-length histograms

## 2 Second tasks

Hifiasm, LJA, Raven and Flye are de novo assemblers. Hifiasm and LJA were used for creating assemblies of the previously subsampled Hifi dataset, and Raven and Flye were used for creating assemblies of the previously subsampled ONT dataset.
Note: I had a small issue when installing LJA; although all dependencies were met, I had to add $\#include < optional >$ to LJA/src/tools/graplite/serialize.hpp for it to work.



```python
import matplotlib.pyplot as plt
from Bio import SeqIO
import numpy as np

import argparse

parser = argparse.ArgumentParser(description="Parser",
                                 formatter_class=argparse.ArgumentDefaultsHelpFormatter)
parser.add_argument("--file", type=str) #fastq file

args = parser.parse_args()
config = vars(args)

fasta_file = config["file"]
read_lengths = []

with open(fasta_file, 'r') as fasta:
    fasta_handle = SeqIO.parse(fasta, "fastq")
    for record in fasta_handle:
        read_lengths.append(len(record.seq))

plt.hist(read_lengths, bins=int(np.sqrt(len(read_lengths))), color='blue', edgecolor='black')
plt.title('Read Length Histogram')
plt.xlabel('read_length')
plt.ylabel('frequency (count)')
plt.show()
```

Figure 2: Read-length histogram code in Python

## 2.1 Overlap-Layout-Consensus paradigm vs de Bruijn paradigm

Hifiasm and Raven follow the Overlap-Layout-Consensus paradigm of de novo assembly. Flye and LJA follow the de Bruijn paradigm. The Overlap-Layout-Consensus (OLC) starts with an Overlap phase in which reads are overlapped between each other, all-versus-all, in order to determine relations between them [7]. It builds a directed graph where reads are nodes and overlaps between reads are edges [7]. In the Layout phase, the given graph is simplified. It is wanted to find a Hamiltonian path through a graph which would correspond to sequence, but due to errors, it is not always possible [7]. Therefore, assemblers use a heuristic approach to iteratively remove nodes and edges assumed unnecessary, such as transitive edges, dead-ends and bubbles, to create a chain [7]. Fragmentation occurs in situations where the assembler cannot find a unique solution for some complex region [7]. In the Consensus phase, the assembly is cleaned of per-base errors by mapping all the reads onto the assembly [7], which means that the most likely nucleotide sequence for each contig is chosen. On the other hand, the de Brujin paradigm has just a layout phase. Nodes are (k-1)-mers (a substring of length k-1) and edges represent reads between adjacent nodes in the constructed directed multigraph [8]. In the de Bruijn graph the nodes represent overlaps [8]. It is wanted to find the Eulerian cycle [9]. With perfect sequencing, the procedure of constructing de Bruijn graph always results in an Eulerian graph, but, for example, gaps or differences in coverage can lead to a disconnected graph [10]. The graph is simplified, with procedures such as chain merging or bubble removal. Assembler follows the chains of nodes through the graph and reads the bases to create the contigs [8]. If there is an ambiguous divergence (multiple paths from a single node) or convergence (multiple paths converge into a single node), the new contig is created [8]. The multiplex de Brujin graph, which was used in LJA, uses vertices with varying k-mer-sizes which improves the contiguity of HiFi assemblies [11]. In low-coverage regions, it is preferred to reduce k-mer size [11]. Shorter lengths and therefore increased number of vertices allow for more connections between reads, which is a mechanism for reducing fragmentation. On the other hand, in high-coverage regions, it is preferred to increase k-mer sizes to improve repeat resolution, since longer reads have a greater chance of including repetitive sequences [11].

## 2.2 Assembly analysis

Commands for creating assemblies by:

- using Hifiasm [12, 13]:

  ```
  ./hifiasm/hifiasm -o ./hifiasm_result -f0 ./downsampled_SRR11434954.fastq
  awk '/^S/{print ">"$2;print $3}' hifiasm_result.p_ctg.gfa > hifiasm_assembly.fa
  ```

- using LJA [14]:

  ```
  ./LJA/bin/lja -o ./lja_result --reads ./downsampled_SRR11434954.fastq
  ```

- using Raven [15]:

  ```
  ./raven/build/bin/raven ./downsampled_SRR12801740.fastq > ./raven_result
  ```

- using Flye [16]:

  ```
  flye --nano-raw ./downsampled_SRR12801740.fastq --out-dir ./flye_result
  ```

The only additional option that was used is: -f0 for Hifiasm, to disable the initial bloom filter which takes 16GB of memory at the beginning (for small genomes) [12].

NG50 is a metric used for measuring the contiguity of the assembly [17]. It is calculated by first sorting the contigs from longest to shortest and then summing the length of the contigs until 50% of the total assembly size is reached [18]. The NG50 is the shortest contig for which longer and equal length contigs cover at least 50% of the assembly [17]. NGA50 is calculated similarly, but it takes the reference genome size instead of the assembly size, which makes it more comparable [19]. So, the higher the NG50 or NGA50 metric number, the more contiguous the assembly is. Calculation process [20]:

```
minigraph -xasm -K1.9g --show-unmap=yes <ref.fa> <asm.fa> > <asm.paf>
samtools faidx <ref.fa>
paftools.js asmstat <ref.fa.fai> <asm.paf>
```

Table 1: NG50 and NGA50 scores

| | **Hifiasm** | **LJA** | **Raven** | **Flye** |
|---|---|---|---|---|
| **NG50** | 4944461 | 4944451 | 4919089 | 4927360 |
| **NGA50** | 4247191 | 2675395 | 4079911 | 3265328 |

In Table 1, it is shown that the Hifiasm assembly has the highest and the LJA assembly has the lowest NGA50 among all tested assemblers. Also, LJA and Flye, which follow the Brujin paradigm, have smaller NGA50 scores compared to Hifiasm and Raven.

QUAST (Quality Assessment Tool for Genome Assemblies) is used for genome assemblies evaluation and comparison [21]. QUAST calculated the number of mismatches and indels (per 100 kbp). Indels are the number of insertions and deletions in the assembled genome compared to a reference genome. Naturally, we want these metrics to have lower values. Used command [22]:

```
quast assembly.fasta -r ref.fasta -o quast_results
```

In Table 2, it is shown that Raven and Flye, which were used on the ONT dataset, have higher values of mismatches and indels per 100 kbp than Hifiasm and LJA, which were used on the Hifi dataset.

Table 2: Mismatches and indels (per 100 kbp)

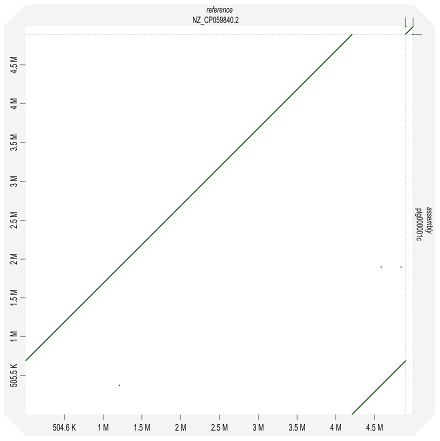
| | **Hifiasm** | **LJA** | **Raven** | **Flye** |
|---|---|---|---|---|
| **Mismatches** | 1.62 | 1.07 | 376.81 | 217.90 |
| **Indels** | 0.18 | 0.54 | 647.79 | 400.53 |

These metrics (mismatches and indels per 100 kbp) give insight into the local structure of the assembly as opposed to the percentage of single-copy and duplicate genes which give insight into the global structure. Single Copy Complete Genes are the BUSCO genes that can be entirely aligned in the assembly and with only one copy present, while Duplicated Complete Genes are the BUSCO genes that can be completely aligned in the assembly, with more than one copy present [23]. Those metrics were not calculated due to problems with Compleasm. I had a problem with parameter lineage, which I was not able to resolve even with direct download of a specific lineage. I managed to download and use only lineage eucaryota_odb10, but that was not useful for this task. Used command for Compleasm [23, 24]:

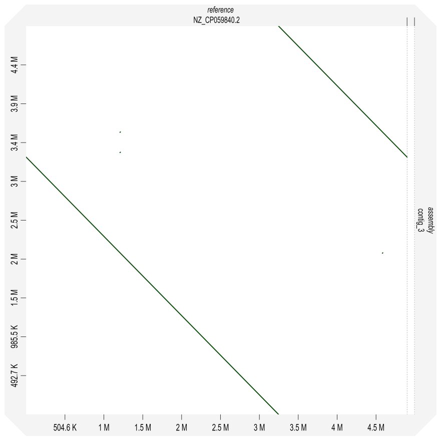
```
compleasm run -a assembly.fasta -o results -l enterobacterales_odb10

Success download from https://busco-data.ezlab.org/v5/data/lineages/
enterobacterales_odb10.2021-02-23.tar.gz
No refseq_db.faa.gz in lineage enterobacterales_odb10,
this lineage cannot be used in compleasm!
Lineage file has been deleted.
```

The assembly vs reference mapping was visualized with D-Genies, a tool used for creating dot plots for large genomes [25]. The dot plot for reference vs Hifiasm assembly can be seen in Figure 3a. Dots on the plot represent the similarity between a position in the reference genome and a position in the assembled genome [26]. The lines in the graph represent a high similarity match [26]. Dots outside of the lines indicate short regions of similarity between the reference and assembled genome sequence. The dot plot for reference vs Flye assembly can be seen in Figure 3b. Since matches from dot plot 3a and 3b are oriented differently, it indicates that the sequences in the assemblies are oriented differently relative to one another [27].



(a) D-Genies dot plot: Hifiasm assembly vs reference



(b) D-Genies dot plot: Flye assembly vs reference

Figure 3: Comparison of assemblies using D-Genies dot plot

## References

[1] Dandan Lang et al. "Comparison of the two up-to-date sequencing technologies for genome assembly: HiFi reads of Pacific Biosciences Sequel II system and ultralong reads of Oxford Nanopore". In: *GigaScience* 9.12 (Dec. 2020), giaa123. ISSN: 2047-217X. DOI: 10.1093/gigascience/giaa123. eprint: https://academic.oup.com/gigascience/article-pdf/9/12/giaa123/34906708/giaa123.pdf. URL: https://doi.org/10.1093/gigascience/giaa123.

[2] *SRAtoolkit prefetch and fasterq dump.* URL: https://github.com/ncbi/sra-tools/wiki/08.-prefetch-and-fasterq-dump.

[3] *SeqKit Github repository.* URL: https://github.com/shenwei356/seqkit.

[4] *SeqKit - Usage and Examples.* URL: https://bioinf.shenwei.me/seqkit/usage/.

[5] *How to calculate coverage?* URL: https://www.biostars.org/p/90626/.

[6] *plotReadLengths Github repository.* URL: https://github.com/MCorentin/plotReadLengths.

[7] Lovro Vrček et al. *Learning to Untangle Genome Assembly with Graph Convolutional Networks.* 2022. arXiv: 2206.00668 [q-bio.GN].

[8] *Deeper look into Genome Assembly algorithms.* URL: https://training.galaxyproject.org/training-material/topics/assembly/tutorials/algorithms-introduction/slides-plain.html.

[9] Tesler G. Compeau PE Pevzner PA. "How to apply de Bruijn graphs to genome assembly". In: *Nat Biotechnol.* (Nov. 2011). DOI: 10.1038/nbt.2023.

[10] *De Bruijn Graph assembly.* URL: https://www.cs.jhu.edu/~langmea/resources/lecture_notes/assembly_dbg.pdf.

[11] Anton Bankevich et al. "LJA: Assembling Long and Accurate Reads Using Multiplex de Bruijn Graphs". In: *bioRxiv* (2021). DOI: 10.1101/2020.12.10.420448. eprint: https://www.biorxiv.org/content/early/2021/10/09/2020.12.10.420448.full.pdf. URL: https://www.biorxiv.org/content/early/2021/10/09/2020.12.10.420448.

[12] *Hifiasm Github repository.* URL: https://github.com/chhylp123/hifiasm.

[13] *Hifiasm FAQ.* URL: https://hifiasm.readthedocs.io/en/latest/faq.html.

[14] *LJA Github repository.* URL: https://github.com/AntonBankevich/LJA.

[15] *Raven Github repository.* URL: https://github.com/lbcb-sci/raven.

[16] *Flye Github repository.* URL: https://github.com/fenderglass/Flye.

[17] Hind Alhakami, Hamid Mirebrahim, and Stefano Lonardi. "A comparative evaluation of genome assembly reconciliation tools". In: *Genome Biology* 18 (May 2017). DOI: 10.1186/s13059-017-1213-3.

[18] *Genome Assembly Quality Assessment.* URL: https://manual.omicsbox.biobam.com/user-manual/omicsbox-modules/module-genome-analysis/genome-assembly-quality-assessment/.

[19] Xiao Luo, Xiongbin Kang, and Alexander Schönhuth. "Enhancing Long-Read-Based Strain-Aware Metagenome Assembly". In: *Frontiers in Genetics* 13 (2022). ISSN: 1664-8021. DOI: 10.3389/fgene.2022.868280. URL: https://www.frontiersin.org/journals/genetics/articles/10.3389/fgene.2022.868280.

[20] Haoyu Cheng et al. "Haplotype-resolved de novo assembly using phased assembly graphs with hifiasm". In: (2021). URL: `https://doi.org/10.1038/s41592-020-01056-5`.

[21] *QUAST Github repository.* URL: `https://github.com/ablab/quast`.

[22] *QUAST Manual.* URL: `https://quast.sourceforge.net/docs/manual.html`.

[23] *Compleasm Github Repository.* URL: `https://github.com/huangnengCSU/compleasm`.

[24] *Carbapenem-resistant Enterobacterales (CRE).* URL: `https://www.cdc.gov/hai/organisms/cre/index.html`.

[25] *D-GENIES.* URL: `https://dgenies.toulouse.inra.fr/`.

[26] *D-GENIES Dot plot.* URL: `https://dgenies.toulouse.inra.fr/documentation/dotplot`.

[27] *Tumačenje dot plot-bioinformatike s primjerom.* URL: `https://omicstutorials.com/interpreting-dot-plot-bioinformatics-with-an-example/`.