MScGG-BIA

Practicals Report

Mariana FAUSTINO

Exercise 02

— January 8, 2025—

UNIVERSITAT DE BARCELONA

Advanced Bionformatics 2023–24

MSc in Genetics & Genomics

Genetics, Microbiology & Statistics Dpt. – UB

Contents

List of Tables

List of Figures

1 Introduction

Different high-throughput sequencing experiments have been performed over genomic DNA samples of *Saccharomyces cerevisiae* and paired-end raw reads were provided. We were asked to assemble the reads obtained from at least two of the suggested datasets. Those sequence sets may have differences in sequencing methodology, but mainly they differ in whole-genome coverage. We can evaluate if differences in coverage can have an impact on the final assembled genome as we already have a reference genome.

1.1 Objectives

- To check qualities and other properties of sequencing reads.
- To run an assembly protocol: cleaning raw reads with `trimmomatic` and assembling the contigs to reconstruct a small genome using `SOAPdenovo`.
- To map the reads back to the assembly so we can visualize the coverage across the sequences and estimate the insert size.
- To compare our assembly against a reference genome, so we can assess the performance of the assembler and the protocol.
- To check completeness of our assembly with `BUSCO`.
- We introduce some LaTeX examples for citing paper references as footnotes.

1.2 Pre-requisites

The commands listed are compatible with a `Debian`-based linux distribution (Linux 6.8.0-48-generic #48~22.04.1-Ubuntu).

- Perl (v5.32.0)
- Pandoc 2.9.2.1
- Latex pdfTeX 3.141592653-2.6-1.40.22
- EMACS or Geany
- EMBOSS:6.6.0.0
- NCBI SRA Toolkit
- fastqc 0.11.9+dfsg-5
- trimmomatic 0.39+dfsg-2
- samtools 1.13-4
- bamtools 2.5.1+dfsg-10build1
- picard-tools 2.26.10+dfsg-1
- gawk 1:5.1.0-1ubuntu0.1
- bwa 0.7.17-6
- bowtie2 2.4.4-1
- soapdenovo2 242+dfsg-2
- gnuplot 5.4 patchlevel 2
- BUSCO 5.2.2
- gunzip (gzip) 1.10
- Python 3.10.12
- R version 4.1.2
- GNU Wget 1.21.2

To install `pandoc`:

```
sudo apt-get install pandoc                      \
                     texlive-latex-recommended \
                     texlive-latex-extra       \
                     texlive-fonts-recommended \
                     texlive-fonts-extra
```

To install LaTeX:

```
sudo apt-get install texlive-full
```

Installing optional packages, such a text editor with programming facilities and extensions, like `emacs` or `geany`:

```
sudo apt-get install emacs geany vim
```

If experiencing LaTeXor `pandoc` formatting errors, then install the latest `pandoc` version. Following the instructions from: https://pandoc.org/installing.html

```
# On a Debian/Ubuntu/Mint box, visit
# pandoc's releases page, to get the latest version from repository:
#     https://github.com/jgm/pandoc/releases/latest
# Then, run those two commands:
wget https://github.com/jgm/pandoc/releases/download/3.1.11.1/pandoc-3.1.11.1-1-amd64.deb

sudo dpkg -i pandoc-3.1.11.1-1-amd64.deb
```

1.3 Installing Bioinformatics software

Below are examples on how to install a software tool from different repositories or systems (`emboss` must be installed).

```
##################################
# emboss - European molecular biology open software suite

# on a debian/ubuntu/mint linux system (DEBs)
apt-cache search emboss     # to check if there is such a package
sudo apt-get install emboss # to install such a package
```

The instructions for the installion of the remaning toolkits are detailed below.

```
##################################
# NCBI SRA Toolkit https://github.com/ncbi/sra-tools/wiki/01.-Downloading-SRA-Toolkit
wget https://ftp-trace.ncbi.nlm.nih.gov/sra/sdk/3.0.10/sratoolkit.3.0.10-ubuntu64.tar.gz \
     -O $WDR/bin/sratoolkit.3.0.10-ubuntu64.tar.gz
pushd $WDR/bin
tar -zxvf sratoolkit.3.0.10-ubuntu64.tar.gz
cd sratoolkit.3.0.10-ubuntu64
popd

export PATH=$WDR/bin/sratoolkit.3.0.0-ubuntu64/bin:$PATH
# NOTE: added to the projectvars.sh file

# seqtk - sampling, trimming, fastq2fasta, subsequence, reverse complement
sudo apt-get install seqtk

# jellyfish - count k-mers in DNA sequences
sudo apt-get install jellyfish

# fastqc - quality control for NGS sequence data
sudo apt-get install fastqc

# trimmomatic - flexible read trimming tool for Illumina NGS data
sudo apt-get install trimmomatic

# samtools - processing sequence alignments in SAM and BAM formats
# bamtools - toolkit for manipulating BAM (genome alignment) files
# picard-tools - Command line tools to manipulate SAM and BAM files
sudo apt-get install samtools bamtools picard-tools

# Downloading the latest version of picard
# see https://github.com/broadinstitute/picard/releases/latest
wget https://github.com/broadinstitute/picard/releases/download/2.27.5/picard.jar \
     -O $BIN/picard.jar

# bwa - Burrows-Wheeler Aligner
# bowtie2 - ultrafast memory-efficient short read aligner
sudo apt-get install bwa bowtie2

# soapdenovo2 - short-read assembly method to build de novo draft assembly
sudo apt-get install soapdenovo2

# igv - Integrative Genomics Viewer
sudo apt-get install igv

# ncbi-blast+ - next generation suite of BLAST sequence search tools
sudo apt-get install ncbi-blast+
#
# gnuplot-qt - a portable command-line driven interactive data and function plotting utility
#              It is required for mummerplot later on...
sudo apt-get install gnuplot-qt
sudo ln -vfs /usr/bin/gnuplot-qt /usr/bin/gnuplot;
#
# graphicsmagick - collection of image processing tools (replacement for imagemagick)
sudo apt-get install graphicsmagick
#
# mummer - Efficient sequence alignment of full genomes
sudo apt-get install mummer
#
# Potential errors and problem solving:
#   + just in case mummerplot returns error: Can't use 'defined(%hash)'
sudo sed -i 's/defined (%/(%/
            ' /usr/bin/mummerplot
#
#   + gnuplot fails because some instruction was implemented in later versions
sudo sed -i 's/^\(.*set mouse.*\)$/#\1/
            ' /usr/bin/mummerplot
#
#   + mummerplot cannot find gnuplot:
sudo sed -i 's/system (\"gnuplot --version\")/system (\"\/usr\/bin\/gnuplot --version\")/
            ' /usr/bin/mummerplot
sudo sed -i 's/my \$cmd = \"gnuplot\";/my \$cmd = \"\/usr\/bin\/gnuplot\";/
            ' /usr/bin/mummerplot
#
#   + just in case mummerplot returns error: Inappropriate ioctl for device
sudo ln -vfs /usr/bin/gnuplot-qt /etc/alternatives/gnuplot;
#
#
# BUSCO - estimating the completeness and redundancy of processed genomic data
#         based on universal single-copy orthologs.
#
# cd $BIN/
# git clone https://gitlab.com/ezlab/busco.git
# cd busco/
# sudo python3 setup.py install
# cd $WDR
#
# Alternatively, download the source code and install it with the commands below:
#
# wget https://gitlab.com/ezlab/busco/-/archive/5.4.3/busco-5.4.3.tar.gz \
 #     -O busco-5.4.3.tar.gz
# tar -zxvf busco-5.4.3.tar.gz
# cd busco-5.4.3
# python3 setup.py install --user
# cd $WDR
# ln -vs ./busco-5.4.3/bin/busco $BIN/busco
```

1.4 Datasets

The budding yeast *Saccharomyces cerevisiae* is one of the major model organisms for understanding cellular and molecular processes in eukaryotes. This single-celled organism is also important in industry, where it is applied to make bread, beer, wine, enzymes, and pharmaceuticals. The *S. cerevisiae* genome has approximately 12 Mbp, distributed in 16 chromosomes. Here, we are going to work on raw reads from one of selected four different sequencing experiments that have been already submitted to the SHORT READS ARCHIVE (SRA) and we will compare the resulting assemblies against the reference *S. cerevisiae* (strain S288C). Table 1 summarizes the information about those sets.

Sequence set	SRA accession	Sequencer	Run info	Run ID
Original WT strain	SRX3242873	Illumina HiSeq 4000	2.8M spots, 559.2M bases, 221.2Mb sra	SRR6130428
S288c	SRX1746300	Illumina HiSeq 2000	5.3M spots, 1.1G bases, 733.6Mb sra	SRR3481383
MiSeq PE-sequencing of SAMD00065885	DRX070537	Illumina MiSeq	5.6M spots, 3.3G bases, 1.6Gb sra	DRR076693
S288c genomic DNA library	SRX4414623	Illumina HiSeq 2500	43.1M spots, 8.6G bases, 3.2Gb sra	SRR7548448

Table 1: **Summary of raw reads datasets that we can use from `SRA` database.** Only the SRR6130428 and SRR3481383 sets were used for the exercise.

We will get first the reference genome[1], *S. cerevisiae* (strain S288C) (assembly `R64.4.1`). This genome version has 16 chromosome sequences, including the mitochondrion genome, totaling 12 157 105 bp (see Table 2).

```
mkdir -vp $WDR/seqs

URL="https://downloads.yeastgenome.org/sequence/S288C_reference/genome_releases"
wget $URL/S288C_reference_genome_Current_Release.tgz \
     -O $WDR/seqs/S288C_reference_genome_Current_Release.tgz

pushd $WDR/seqs/
tar -zxvf S288C_reference_genome_Current_Release.tgz
popd

#also on projectvars
export REFDIR="S288C_reference_genome_R64-4-1_20230830"
export REFGEN="S288C_reference_sequence_R64-4-1_20230830"

zcat $WDR/seqs/$REFDIR/$REFGEN.fsa.gz | \
  infoseq -only -length -noheading -sequence fasta::stdin  2> /dev/null | \
  gawk '{ s+=$1 } END{ printf "# Yeast genome size (v.R64.4.1): %dbp\n", s }'
# Yeast genome size (v.R64.4.1): 12157105bp

#Producing a LaTeX table with the chromosome sizes and GC content.

zcat $WDR/seqs/$REFDIR/$REFGEN.fsa.gz | \
  infoseq -noheading -sequence fasta::stdin  2> /dev/null | \
    gawk 'BEGIN {
            printf "%15s & %10s & %12s & %10s \\\\\n",
                   "Chromosome", "GenBank ID", "Length (bp)", "GC content";
          }
          $0 != /^[ \t]*$/ {
            L=$0;
            sub(/^.*\[(chromosome|location)=/,"",L);
            sub(/\].*$/,"",L);
            sub(/_/,"\\_",$3);
            printf "%15s & %10s & %12d & %8.2f\\%% \\\\\n",
                   L, $3, $6, $7;
          }' > $WDR/docs/chromosomes_info.tex;
```

The smaller dataset, SRR6130428 (221.2Mb SHORT READS ARCHIVE file), was the chosen dataset for further analysis due to hardware limitations.

```
## The reads dataset: SRR6130428
```

[1]Engel et al. "The Reference Genome Sequence of *Saccharomyces cerevisiae*: Then and Now". G3 (Bethesda), g3.113.008995v1, 2013 (PMID:24374639).

Chromosome	GenBank ID	Length (bp)	GC content
I	NC_001133	230218	39.27%
II	NC_001134	813184	38.34%
III	NC_001135	316620	38.53%
IV	NC_001136	1531933	37.91%
V	NC_001137	576874	38.51%
VI	NC_001138	270161	38.73%
VII	NC_001139	1090940	38.06%
VIII	NC_001140	562643	38.50%
IX	NC_001141	439888	38.90%
X	NC_001142	745751	38.37%
XI	NC_001143	666816	38.07%
XII	NC_001144	1078177	38.48%
XIII	NC_001145	924431	38.20%
XIV	NC_001146	784333	38.64%
XV	NC_001147	1091291	38.16%
XVI	NC_001148	948066	38.06%
mitochondrion	NC_001224	85779	17.11%

Table 2: **Reference *Saccharomyces cerevisiae* chromosome summary.** This table displays information about length and GC content for each of the chromosomes of the *S. cerevisiae* reference genome.

```
# 221.2Mb downloads
mkdir -vp $WDR/seqs/SRR6130428
prefetch -v SRR6130428  -O $WDR/seqs
#
# The SRA file is stored as $WDR/seqs/SRR6130428/SRR6130428.sra,

fastq-dump -X 2 -Z $WDR/seqs/SRR6130428/SRR6130428.sra

# Read 2 spots for seqs/SRR6130428/SRR6130428.sra
# Written 2 spots for seqs/SRR6130428/SRR6130428.sra
# @SRR6130428.1 1 length=202
# NGACCTTGGCGTTGGTGTAGGTCACCACTCCGATTTTGAGCTAGAATATGAAATTCATCACTGGAATAAGTTTCATAAGGACAAAGATCCAGACGTTAAAGNAT
# +SRR6130428.1 1 length=202
# #AAFFJJJJJJJJJJJJJJJJJJJJJJJJJJFJJJJJJJJJJJJJJJJJJJJJJJJJFJJJJFJJJJJJJJJJFFJJJJJJJJJJJJJJJJJJJJJ#AA
# @SRR6130428.2 2 length=202
# NTGCTACTCTCATGGTCTCAATACTGCCGCCGACATTCTGTCCCACATACTAAATCTCTTCCCGTCATTATCGCCCGCATCCGGTGCCGTAAATGCAAAACNGC
# +SRR6130428.2 2 length=202
# #AAAAFFAFFJJJJJJJJJJJJJJJJJFJJJFJJJJJJJFJJJJAJJJFJ<JJFA<FFAJJFFJJ<JFJJJJJJJFFJ7JJJJFJJJJJFFJAAFJFFJJ#AA

SQSET=SRR6130428
fastq-dump -I --split-files $WDR/seqs/${SQSET}/${SQSET}.sra \
           --gzip --outdir $WDR/seqs/${SQSET}/
#    Read 2768518 spots for seqs/SRR6130428/SRR6130428.sra
# Written 2768518 spots for seqs/SRR6130428/SRR6130428.sra
```

2 The Assembly Protocol

2.1 Exploratory data analysis of the raw reads

On this initial step, `fastQC` was run over `fastq` files for the forward (R1) and reverse (R2) raw reads sets. Then those two sets were compared from three of the resulting plots: the quality distribution per base position (boxplots), the base content per position (lineplots), and the read sequences GC content distribution (lineplot). The program generated an `HTML` summary page, as well as a `zip` file containing all the figures in a folder and some results in tabular format.

For all the code blocks following the variable 'SQSET' defined was particular to each dataset.

```
SQSET=SRR6130428    #either was exported at a time
```

```
SQSET=SRR3481383

mkdir -vp $WDR/seqs/${SQSET}/${SQSET}.QC

for READSET in 1 2;
  do {
    echo "# Running fastQC on $SQSET R${READSET}..." 1>&2;
    fastqc -t 8 --format fastq                                \
           --contaminants $BIN/fastqc_conf/contaminant_list.txt \
           --adapters     $BIN/fastqc_conf/adapter_list.txt     \
           --limits       $BIN/fastqc_conf/limits.txt           \
           -o $WDR/seqs/${SQSET}/${SQSET}.QC                    \
           $WDR/seqs/${SQSET}/${SQSET}_${READSET}.fastq.gz      \
           2> $WDR/seqs/${SQSET}/${SQSET}.QC/fastQC_${SQSET}_${READSET}.log 1>&2;
  }; done
```

Calculating the sequence sizes in the SRR6130428 dataset FASTQ files (R1 and R2).

```
zcat seqs/SRR6130428/SRR6130428_2.fastq.gz | awk 'NR % 4 == 2 { seqlen += length($0) } NR % 4 == 0 { print
2768518 101
zcat seqs/SRR6130428/SRR6130428_1.fastq.gz | awk 'NR % 4 == 2 { seqlen += length($0) } NR % 4 == 0 { print
2768518 101
```

Each sequence is 101 nucleotides long for the SRR6130428 dataset.

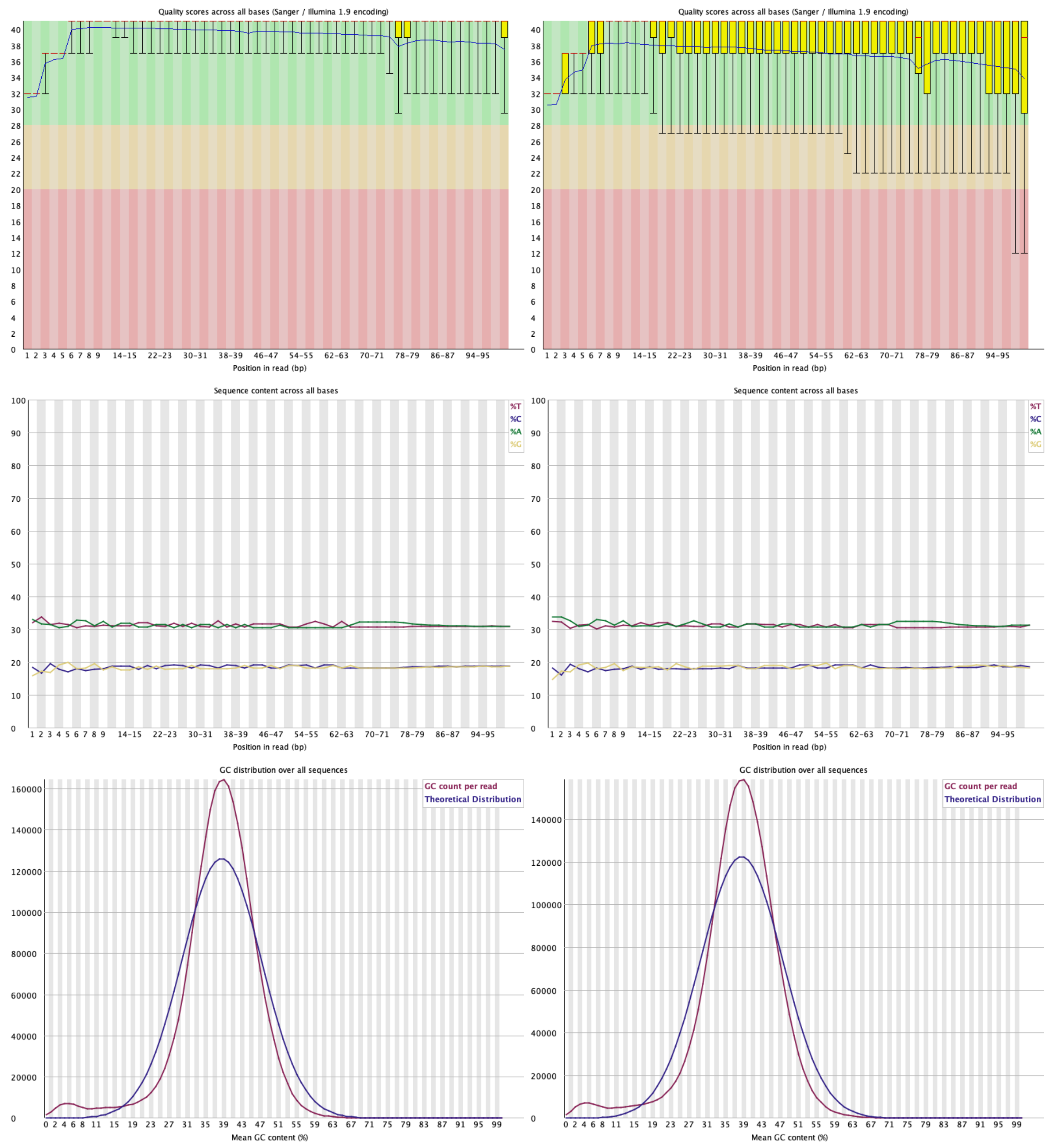

Figure 1: **Raw reads basic sequence analyses for `SRR6130428`.** Left column shows results for the forward reads (R1), right column for the reverse reads (R2). On top panels we can observe phred scores distribution per base position, mid panels correspond to the base composition per position, while the bottom ones show the GC content distribution across reads.

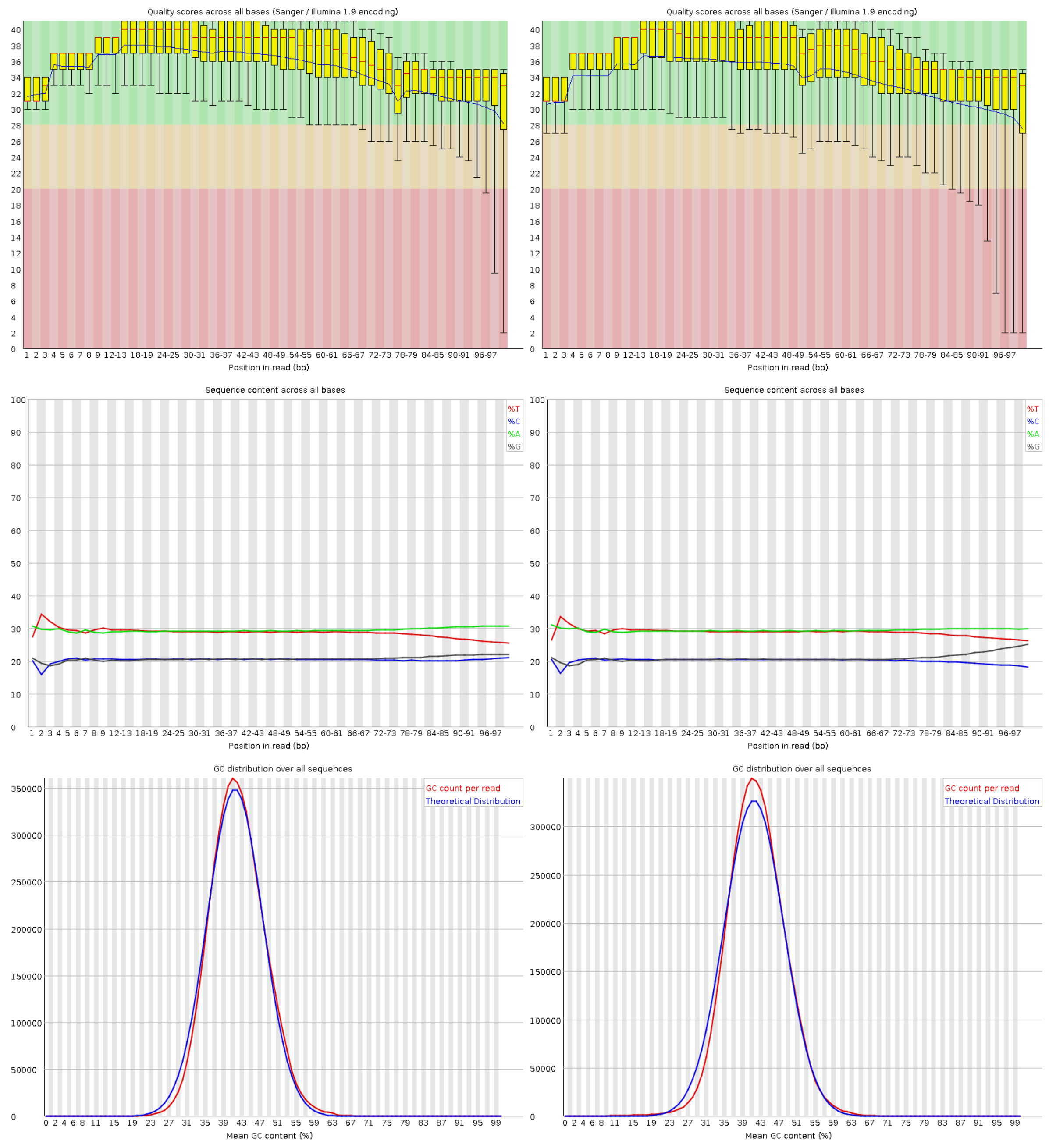

Figure 2: **Raw reads basic sequence analyses for `SRR3481383`.** Left column shows results for the forward reads (R1), right column for the reverse reads (R2). On top panels we can observe phred scores distribution per base position, mid panels correspond to the base composition per position, while the bottom ones show the GC content distribution across reads.

2.2 Cleaning and trimming reads with `trimmomatic`

A crucial step before starting the assembly is to remove any contaminant sequences, like the sequencing adapters, and low quality segments. For this purpose, `trimmomatic` was used to perform the main analysis, and the results were compared with those of `cutadapt`, another raw reads cleaner.

```
#
##  We took the first 100000 sequences with the "head" command for the SRR3481383 dataset.
#
 gunzip -c $WDR/seqs/${SQSET}/${SQSET}_1.fastq.gz | head -n 400000 | \
        gzip -c9 - > $WDR/seqs/${SQSET}/${SQSET}_1.subset100k.fastq.gz;
 gunzip -c $WDR/seqs/${SQSET}/${SQSET}_2.fastq.gz | head -n 400000 | \
        gzip -c9 - > $WDR/seqs/${SQSET}/${SQSET}_2.subset100k.fastq.gz;
        #The subset100k were renamed in the work through of this report
        #to SRR3481383_1.fastq.gz/SRR3481383_2.fastq.gz  for simplicity
#
##  Alternatively, using the "seqtk" program (if installed), as follows:
#
# seqtk sample -s11 $WDR/seqs/${SQSET}/${SQSET}_1.fastq.gz 0.1 | \
#         gzip -c9 - > $WDR/seqs/${SQSET}/${SQSET}_1.subset10pct.fastq.gz;
# seqtk sample -s11 $WDR/seqs/${SQSET}/${SQSET}_2.fastq.gz 0.1 | \
#         gzip -c9 - > $WDR/seqs/${SQSET}/${SQSET}_2.subset10pct.fastq.gz;
#
##  The -s option specifies the seed value for the random number generator,
#   it needs to be the same value for file1 and file2 to keep the paired reads
#   on the sampled output. 0.1 argument sets the percent of seqs to sample (10%).
#

# to find where trimmomatic was installed in the system
find /usr -name trimmomatic.jar

# Taking the result of the previous command as the folder to set on TMC:
export TMC=/usr/share/java/trimmomatic.jar
# Set this variable to suit the system installation:
export TMA=/usr/share/trimmomatic/
#
# The trimmomatic parameters documentation is available at:
# http://www.usadellab.org/cms/uploads/supplementary/Trimmomatic/TrimmomaticManual_V0.32.pdf
#
export TRMPAR="LEADING:15 TRAILING:15 SLIDINGWINDOW:4:15 MINLEN:30 TOPHRED33";
export TRMPECLP="ILLUMINACLIP:$TMA/TruSeq2-PE.fa:2:30:10";

java -jar $TMC  PE                            \
     $WDR/seqs/${SQSET}/${SQSET}_1.fastq.gz \
     $WDR/seqs/${SQSET}/${SQSET}_2.fastq.gz \
     $WDR/seqs/${SQSET}/${SQSET}_1.trimmo_pe.fastq.gz \
     $WDR/seqs/${SQSET}/${SQSET}_1.trimmo_sg.fastq.gz \
     $WDR/seqs/${SQSET}/${SQSET}_2.trimmo_pe.fastq.gz \
     $WDR/seqs/${SQSET}/${SQSET}_2.trimmo_sg.fastq.gz \
     $TRMPECLP $TRMPAR                        \
  2> $WDR/seqs/${SQSET}/${SQSET}.trimmo.log 1>&2 ;

tail -n 2 $WDR/seqs/${SQSET}/${SQSET}.trimmo.log
#SQSET=SRR6130428
#  Input Read Pairs:         2768518
#            Both Surviving: 2536084 (91.60%)
#    Forward Only Surviving:  176365 ( 6.37%)
#    Reverse Only Surviving:    4082 ( 0.15%)
#                   Dropped:   51987 ( 1.88%)
# TrimmomaticPE: Completed successfully

tail -n 2 $WDR/seqs/${SQSET}/${SQSET}.trimmo.log
#SQSET=SRR3481383
#  Input Read Pairs:         100000
#        Both Surviving: 81468 (81.47%)
#    Forward Only Surviving: 17918 (17.92%)
#    Reverse Only Surviving:    343 (0.34%)
#                   Dropped:    271 (0.27%)
# TrimmomaticPE: Completed successfully


#Using cutadapt for trimming reads
#R1
cutadapt \
    --quality-cutoff 15,15 \
    --overlap 30 \
    --quality-base 33 \
   $WDR/seqs/${SQSET}/${SQSET}_1.fastq.gz \
    --output $WDR/seqs/${SQSET}/${SQSET}_1.cutadapt.fastq.gz
#R2
cutadapt \
    --quality-cutoff 15,15 \
    --overlap 30 \
    --quality-base 33 \
   $WDR/seqs/${SQSET}/${SQSET}_2.fastq.gz \
    -o $WDR/seqs/${SQSET}/${SQSET}_2.cutadapt.fastq.gz

#SQSET=SRR6130428 cutadapt output for R1
=== Summary ===

Total reads processed:               5,346,334
Reads written (passing filters):     5,346,334 (100.0%)

Total basepairs processed:   539,979,734 bp
Quality-trimmed:              26,063,936 bp (4.8%)
Total written (filtered):    513,915,798 bp (95.2%)

#SQSET=SRR6130428 cutadapt output for R2
=== Summary ===

Total reads processed:               2,768,518
Reads written (passing filters):     2,768,518 (100.0%)

Total basepairs processed:   279,620,318 bp
Quality-trimmed:               1,364,499 bp (0.5%)
Total written (filtered):    278,255,819 bp (99.5%)

#SQSET=SRR3481383 cutadapt output for R1
=== Summary ===

Total reads processed:                 100,000
Reads written (passing filters):       100,000 (100.0%)

Total basepairs processed:    10,100,000 bp
Quality-trimmed:                 171,591 bp (1.7%)
Total written (filtered):      9,928,409 bp (98.3%)

#SQSET=SRR3481383 cutadapt output for R2
=== Summary ===

Total reads processed:                 100,000
Reads written (passing filters):       100,000 (100.0%)

Total basepairs processed:    10,100,000 bp
Quality-trimmed:                 447,684 bp (4.4%)
Total written (filtered):      9,652,316 bp (95.6%)
```

Checking the new reads filtered by running `fastqc` over the pair-end reads from `trimmomatic` and `cutadapt`.

```
#SQSET=SRR6130428
#SQSET=SRR3481383
mkdir -vp $WDR/seqs/${SQSET}/${SQSET}.trimmo.QC
mkdir -vp $WDR/seqs/${SQSET}/${SQSET}.cutadapt.QC

#for trimmomatic pair-end reads
for READSET in 1 2;
  do {
    echo "# Running fastQC on trimmed PE reads $SQSET R${READSET}..." 1>&2;
    fastqc -t 8 --format fastq                                    \
           --contaminants $BIN/fastqc_conf/contaminant_list.txt \
           --adapters     $BIN/fastqc_conf/adapter_list.txt     \
           --limits       $BIN/fastqc_conf/limits.txt           \
           -o $WDR/seqs/${SQSET}/${SQSET}.trimmo.QC             \
              $WDR/seqs/${SQSET}/${SQSET}_${READSET}.trimmo_pe.fastq.gz \
           2> $WDR/seqs/${SQSET}/${SQSET}.trimmo.QC/fastQC_${SQSET}_${READSET}.log 1>&2;
  }; done

#for cutadapt pair-end reads
for READSET in 1 2;
  do {
    echo "# Running fastQC on cutadapt PE reads $SQSET R${READSET}..." 1>&2;
    fastqc -t 8 --format fastq                                  \
           --contaminants $BIN/fastqc_conf/contaminant_list.txt \
           --adapters     $BIN/fastqc_conf/adapter_list.txt     \
           --limits       $BIN/fastqc_conf/limits.txt           \
           -o $WDR/seqs/${SQSET}/${SQSET}.cutadapt.QC             \
              $WDR/seqs/${SQSET}/${SQSET}_${READSET}.cutadapt.fastq.gz \
           2> $WDR/seqs/${SQSET}/${SQSET}.cutadapt.QC/fastQC_${SQSET}_${READSET}.cutadapt.log 1>&2;
           }; done
```

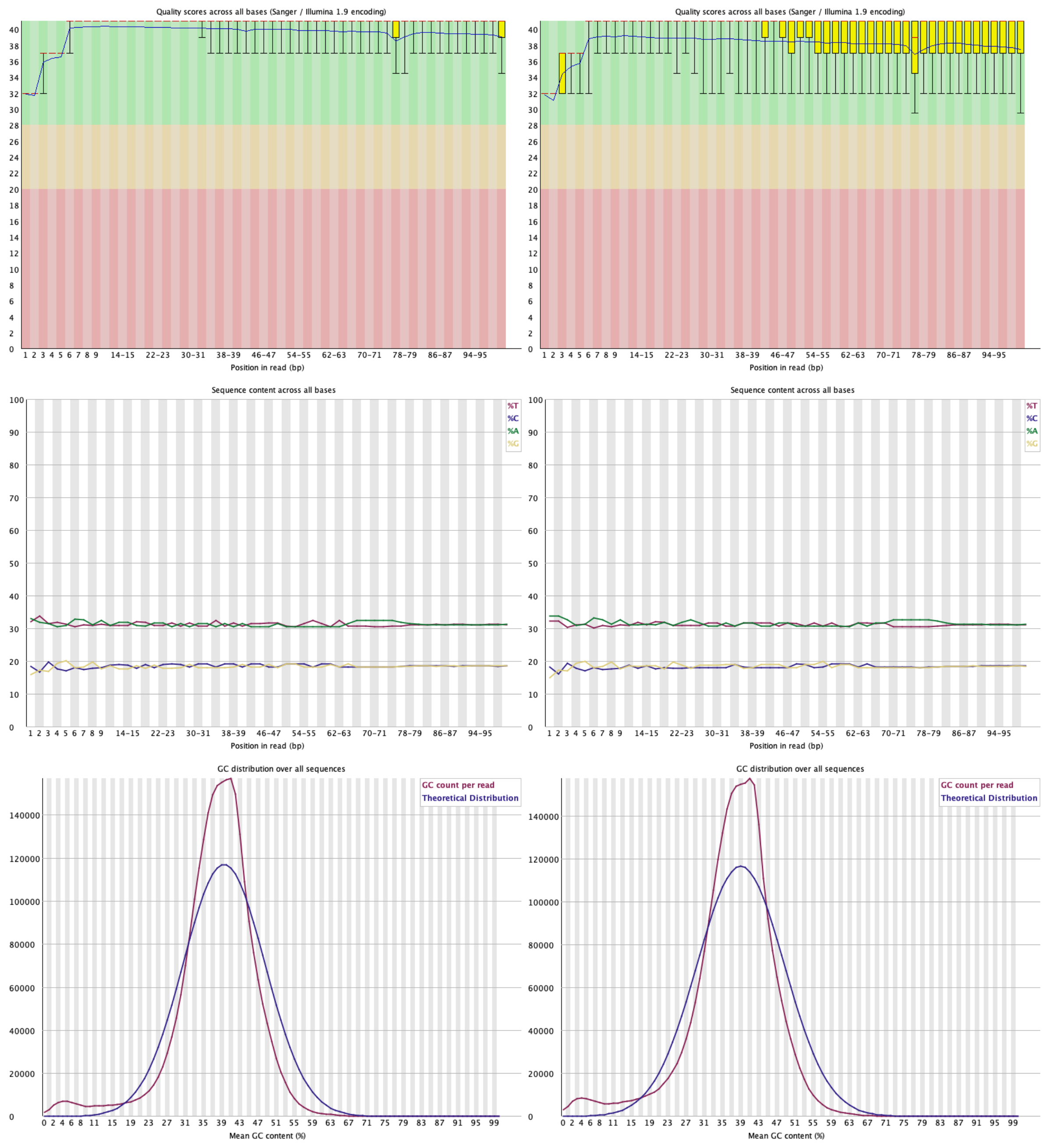

Figure 3: **Cleaned reads basic sequence analyses for `SRR6130428` using 'trimmomatic'.** Left column shows results for the forward reads (R1), right column for the reverse reads (R2). On top panels we can observe phred scores distribution per base position, mid panels correspond to the base composition per position, while the bottom ones show the GC content distribution across reads.

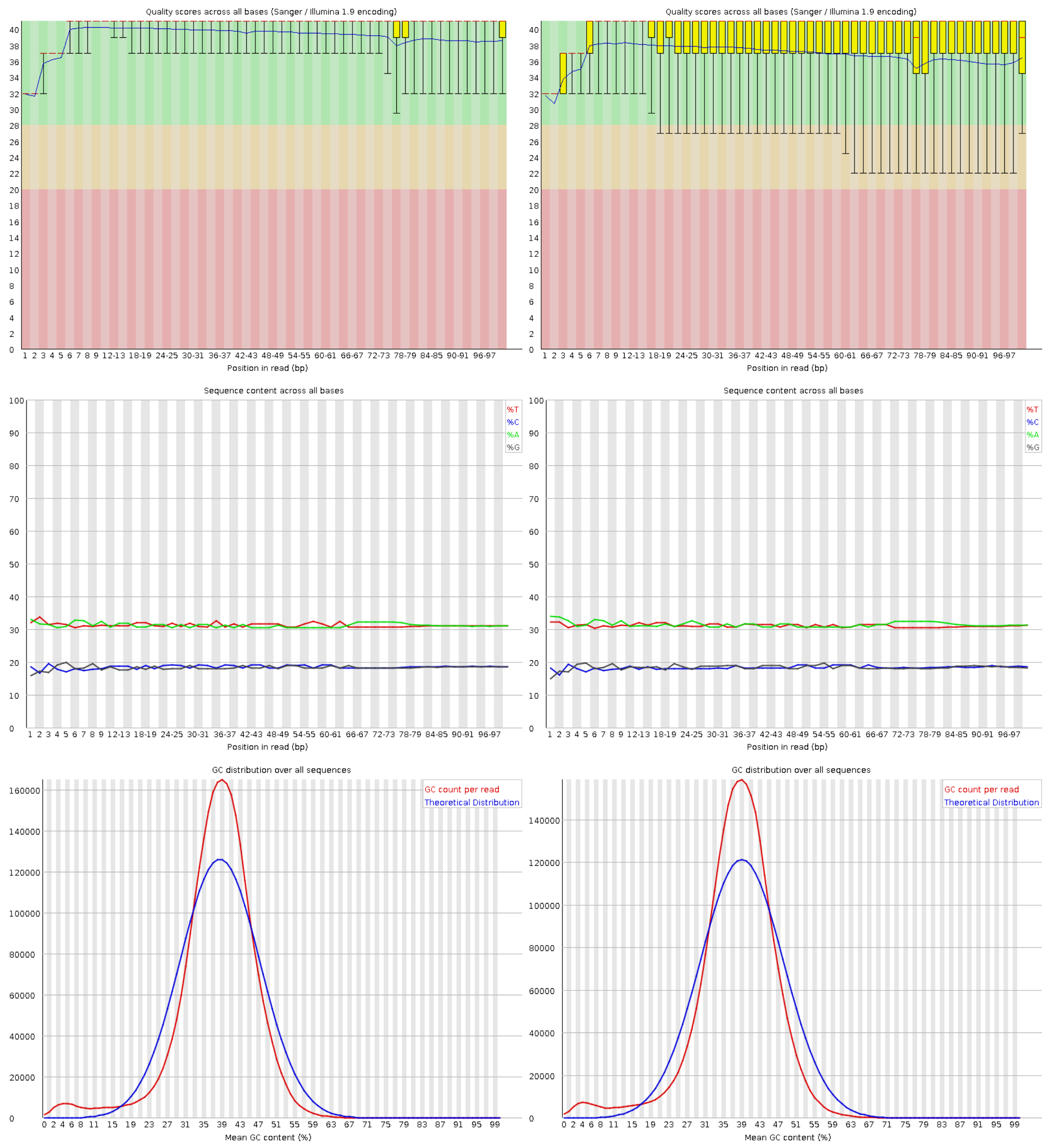

Figure 4: **Cleaned reads basic sequence analyses for `SRR6130428` using 'cutadapt'.** Left column shows results for the forward reads (R1), right column for the reverse reads (R2). On top panels we can observe phred scores distribution per base position, mid panels correspond to the base composition per position, while the bottom ones show the GC content distribution across reads.

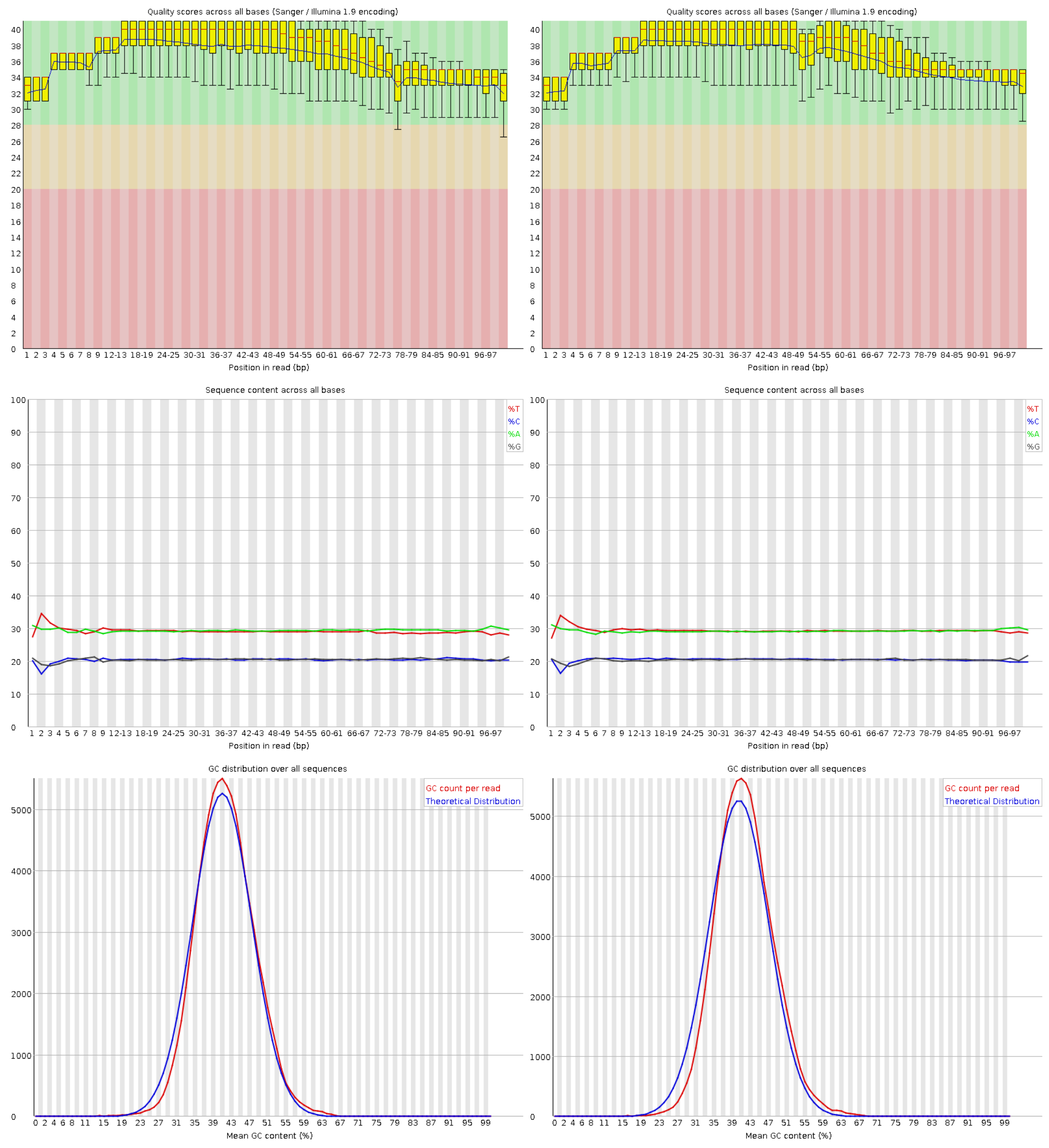

Figure 5: **Cleaned reads basic sequence analyses for `SRR3481383` using 'trimmomatic'.** Left column shows results for the forward reads (R1), right column for the reverse reads (R2). On top panels we can observe phred scores distribution per base position, mid panels correspond to the base composition per position, while the bottom ones show the GC content distribution across reads.

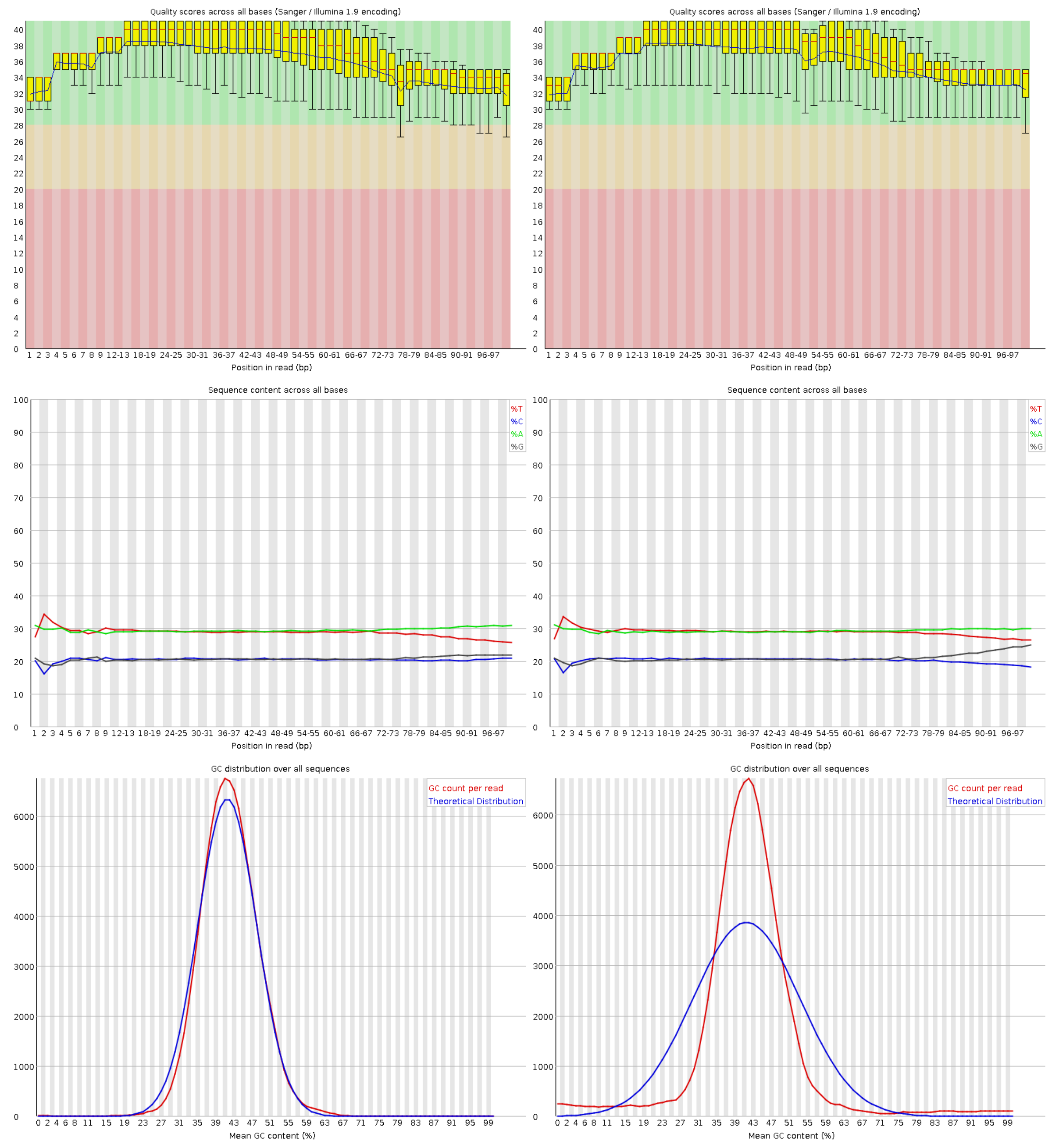

Figure 6: **Cleaned reads basic sequence analyses for `SRR3481383` using 'cutadapt'.** Left column shows results for the forward reads (R1), right column for the reverse reads (R2). On top panels we can observe phred scores distribution per base position, mid panels correspond to the base composition per position, while the bottom ones show the GC content distribution across reads.

We can now estimate the sequencing coverage, as we already have the size of the reference genome, and the total amount of nucleotides generated by the sequencing projects.

```
#SQSET=SRR6130428
#SQSET=SRR3481383
# raw PE-reads
gunzip -c $WDR/seqs/${SQSET}/${SQSET}_[12].fastq.gz | \
  infoseq -sequence fastq::stdin -only -length -noheading | \
    gawk '{ s+=$1; n++ }
          END{ printf "# Total %d sequences and %d nucleotides\n", n, s }'
#SQSET=SRR6130428
# Total 5,537,036 sequences and 559,240,636 nucleotides
#
#SQSET=SRR3481383
# Total 10,692,668 sequences and 1,079,959,468 nucleotides

# cleaned PE-reads
gunzip -c $WDR/seqs/${SQSET}/${SQSET}_[12].trimmo_pe.fastq.gz | \
  infoseq -sequence fastq::stdin -only -length -noheading | \
    gawk '{ s+=$1; n++ }
          END{ printf "# Total %d sequences and %d nucleotides\n", n, s }'
#SQSET=SRR6130428
# Total 5,072,168 sequences and 499,530,830 nucleotides
#
#SQSET=SRR3481383
# Total 8,636,178 sequences and 839,065,950 nucleotides
```

For raw and cleaned reads of `SRR6130428` we can estimate a sequencing coverage of $559\,240\,636/12\,157\,105 = 46.00$X and $499\,530\,830/12\,157\,105 = 41.09$X respectively. For the complete set of raw and cleaned reads of `SRR3481383` we have estimated a sequencing coverage of $1\,079\,959\,468/12\,157\,105 = 88.83$X and $839\,065\,950/12\,157\,105 = 69.02$X respectively.

2.3 Assembling reads with `SOAPdenovo`

`SOAPdenovo`[2] is easy to install and to configure. It has been reported to perform like other more complex tools, generating in some tests less chimeric contigs and missasemblies.

```
# SQSET=SRR6130428
# SQSET=SRR3481383
export SPD=$WDR/soapdenovo/$SQSET
mkdir -vp $SPD

export GENOMESIZE=12157105;

# Generating the configuration file for SRR6130428
cat > $SPD/SRR6130428_soap.conf <<EOF
# Raw reads from SRA experiment SRR6130428
# Description: Original WT strain, Illumina HiSeq 4000 (2x100bp)
# Insert size was not described on the SRA project, using 450bp
# ---
# maximal read length
max_rd_len=100
#
[LIB]
# average insert size
avg_ins=450
# if sequence needs to be reversed (0 for short insert PE libs)
reverse_seq=0
# in which part(s) the reads are used (3 for both contig and scaffold assembly)
asm_flags=3
# use only first 100 bps of each read
```

[2]Luo et al. "SOAPdenovo2: an empirically improved memory-efficient short-read de novo assembler". *GigaScience*, 1:18, 2012.

```
rd_len_cutoff=100
# in which order the reads are used while scaffolding
rank=1
# cutoff of pair number for a reliable connection (at least 3 for short insert size)
pair_num_cutoff=3
# minimum aligned length to contigs for a reliable read location (at least 32 for short insert size)
map_len=32
# a pair of fastq file, read 1 file should always be followed by read 2 file
# using pair-end reads after filtering with trimmomatic
q1=$WDR/seqs/${SQSET}/${SQSET}_1.trimmo_pe.fastq.gz
q2=$WDR/seqs/${SQSET}/${SQSET}_2.trimmo_pe.fastq.gz
#
EOF

# Generating the configuration file for SRR3481383
cat > $SPD/SRR3481383_soap.conf <<EOF
# Raw reads from SRA experiment SRR3481383
# Description: Original WT strain, Illumina HiSeq 2000
# Insert size was not described on the SRA project, using 450bp
# ---
# maximal read length
max_rd_len=100
#
[LIB]
# average insert size
avg_ins=450
# if sequence needs to be reversed (0 for short insert PE libs)
reverse_seq=0
# in which part(s) the reads are used (3 for both contig and scaffold assembly)
asm_flags=3
# use only first 100 bps of each read
rd_len_cutoff=100
# in which order the reads are used while scaffolding
rank=1
# cutoff of pair number for a reliable connection (at least 3 for short insert size)
pair_num_cutoff=3
# minimum aligned length to contigs for a reliable read location (at least 32 for short insert size)
map_len=32
# a pair of fastq file, read 1 file should always be followed by read 2 file
# using pair-end reads after filtering with trimmomatic
q1=$WDR/seqs/${SQSET}/SRR3481383_1.subset100k.fastq.gz
q2=$WDR/seqs/${SQSET}/SRR3481383_2.subset100k.fastq.gz
#
EOF

# Building the k-mers graph
soapdenovo2-63mer pregraph -s $SPD/${SQSET}_soap.conf -K 63 -R -p 8 \
                           -o $SPD/${SQSET}_k63_graph               \
                           2> $SPD/${SQSET}_k63_pregraph.log 1>&2;

# Contiging stage
soapdenovo2-63mer contig   -g $SPD/${SQSET}_k63_graph -R -p 8 \
                           2> $SPD/${SQSET}_k63_contig.log 1>&2;

# Mapping reads back over the contigs
soapdenovo2-63mer map      -s $SPD/${SQSET}_soap.conf -p 8 \
                           -g $SPD/${SQSET}_k63_graph      \
                           2> $SPD/${SQSET}_k63_map.log 1>&2;

# Scaffolding contigs if we have long range reads (i.e. a 2kbp mate-pairs run)
# in this case, as we only have a single pair-ends library, we will get a result
# that will be similar or the same as what we have obtained in the contiging stage
#
```

```
soapdenovo2-63mer scaff    -F -p 8 -N $GENOMESIZE            \
                           -g $SPD/${SQSET}_k63_graph_prefix \
                           2> $SPD/${SQSET}_k63_scaff.log 1>&2;

# just by looking at the $SPD/${SQSET}_k63_contig.log file,
# we can retrieve info about contigs assembly, such as N50
tail $SPD/${SQSET}_k63_contig.log

# SQSET=SRR6130428
# There are 3156 contig(s) longer than 100, sum up 11779878 bp, with average length 3732.
# The longest length is 69946 bp, contig N50 is 14294 bp, contig N90 is 3279 bp.
# 4867 contig(s) longer than 64 output.
#
# SQSET=SRR3481383
# There are 463 contig(s) longer than 100, sum up 127222 bp, with average length 274.
# The longest length is 7366 bp, contig N50 is 231 bp,contig N90 is 193 bp.
# 494 contig(s) longer than 64 output.


# Computing some extra stats from the assemblies,

export PERL5LIB=$BIN;

chmod +x $WRD/bin/assemblathon_stats.pl
 zcat $WDR/seqs/$REFDIR/$REFGEN.fsa.gz | \
 $BIN/assemblathon_stats.pl \
            -csv -genome_size 12160000 \
            $WDR/soapdenovo/${SQSET}/${SQSET}_k63_graph.contig \
          > $WDR/stats/assembly_stats_${SQSET}_soapdenovo_k63_graph.contig.txt
```

The files `*.contig` and `*.scafSeq` were created for the fasta sequences assembled contigs and scaffolds, respectively.

Open questions arise:

- Is it possible to calculate the contig's lengths, and produce a plot distribution?
- What would happen if using a larger coverage reads set?
- Would knowing the real insert size improve the assembly?

In an attempt to answer these questions, the assembly was compared with another assembly of the suggested raw-reads set.

2.4 Estimating insert size with picard

Firstly, we checked whether the estimated insert size was an educated guess by aligning the PE reads against the reference genome or the assembly, and then taking the alignments in `bam` format to estimate insert size with `picard` tool. This was only performed for the SRR6130428 dataset, as we have previosuly trimmed the SRR3481383 dataset.

```
# SQSET=SRR6130428;
# SQSET=SRR3481383
export BWT=$WDR/bowtie;

mkdir -vp $BWT/$SQSET;
mkdir $WDR/tmpsort

# by linking contigs file to one file with a fasta suffix,
# it facilitates using those sequences on IGV.
ln -vs ./${SQSET}_k63_graph.contig \
       $WDR/soapdenovo/${SQSET}/${SQSET}_k63_graph.contig.fa;

# preparing reference sequence databases for bowtie
FSAD=$WDR/seqs/$REFDIR;
bowtie2-build --large-index -o 2 \
              $FSAD/$REFGEN.fsa.gz \
              $BWT/scer_refgenome.bowtiedb \
           2> $BWT/scer_refgenome.bowtiedb.log 1>&2;

bowtie2-build --large-index -o 2 \
              $WDR/soapdenovo/${SQSET}/${SQSET}_k63_graph.contig \
              $BWT/${SQSET}/${SQSET}_k63_graph.contig.bowtiedb \
           2> $BWT/${SQSET}/${SQSET}_k63_graph.contig.bowtiedb.log 1>&2;

# mapping pe reads over reference sequences
TMP=$WDR/tmpsort;
PEfileR1=$WDR/seqs/${SQSET}/${SQSET}_1.trimmo_pe.fastq.gz;
PEfileR2=$WDR/seqs/${SQSET}/${SQSET}_2.trimmo_pe.fastq.gz;

BWTBF=$BWT/${SQSET}/${SQSET}-x-scer_refgen.bowtie;
TMPBF=$TMP/${SQSET}-x-scer_refgen.bowtie;

bowtie2 -q --threads 8 -k 5 -L 12                    \
        --local --sensitive-local --no-unal --met 60 \
        --met-file $BWTBF.metrics                    \
        -x         $BWT/scer_refgenome.bowtiedb      \
        -1         $PEfileR1  \
        -2         $PEfileR2  \
        -S         $TMPBF.sam \
        2>         $BWTBF.log 1>&2;

( samtools view -Sb -o ${TMPBF}.bam \
                       ${TMPBF}.sam;
  samtools sort ${TMPBF}.bam   \
             -o ${TMPBF}.sorted.bam;
  mv -v ${TMPBF}.sorted.bam ${BWTBF}.sorted.bam;
  rm -v ${TMPBF}.sam ${TMPBF}.bam
  ) 2> ${BWTBF}.bowtie2sortedbam.log 1>&2;

samtools index $BWTBF.sorted.bam;

java -jar $BIN/picard.jar CollectInsertSizeMetrics   \
          HISTOGRAM_FILE=$BWTBF.insertsize.hist.pdf  \
                   INPUT=$BWTBF.sorted.bam           \
                  OUTPUT=$BWTBF.insertsize_stats.txt \
           ASSUME_SORTED=true \
              DEVIATIONS=25   \
                      2> $BWTBF.insertsize_stats.log;


# checking the assembled contigs

BWTBF=$BWT/${SQSET}/${SQSET}-x-soapk63ctgs.bowtie;
TMPBF=$TMP/${SQSET}-x-soapk63ctgs.bowtie;

bowtie2 -q --threads 8 -k 5 -L 12                    \
        --local --sensitive-local --no-unal --met 60 \
        --met-file $BWTBF.metrics                    \
        -x         $BWT/scer_refgenome.bowtiedb      \
        -1         $PEfileR1  \
        -2         $PEfileR2  \
        -S         $TMPBF.sam \
        2>         $BWTBF.log 1>&2;

( samtools view -Sb -o $TMPBF.bam \
                       $TMPBF.sam;
  samtools sort $TMPBF.bam   \
             -o $TMPBF.sorted.bam;
  mv -v $TMPBF.sorted.bam $BWTBF.sorted.bam;
  rm -v $TMPBF.sam $TMPBF.bam
  ) 2> $BWTBF.bowtie2sortedbam.log 1>&2;

samtools index $BWTBF.sorted.bam;

java -jar $BIN/picard.jar CollectInsertSizeMetrics   \
          HISTOGRAM_FILE=$BWTBF.insertsize.hist.pdf  \
                   INPUT=$BWTBF.sorted.bam           \
                  OUTPUT=$BWTBF.insertsize_stats.txt \
           ASSUME_SORTED=true \
              DEVIATIONS=25   \
                      2> $BWTBF.insertsize_stats.log;
```

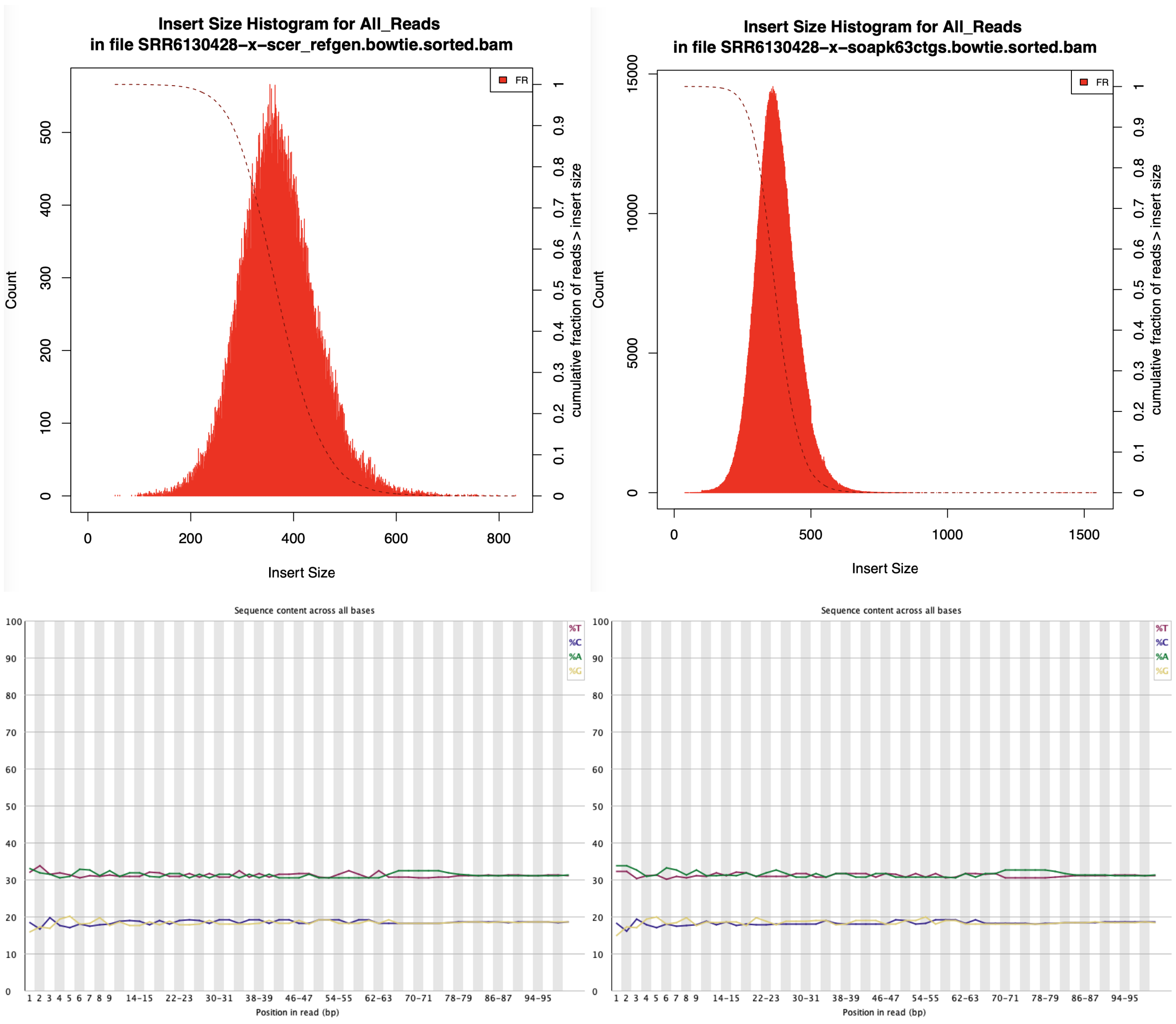

Figure 7: **Histogram outputs by *picard* of the SRR6130428 reads alignment over the reference genome and the *soapdenovo* assemblies.**

3 Exploring the assemblies

3.1 Filter out contigs mapping to reference chromosomes

On this section we are going to run `dnadiff` from the `MUMmer` package[3] to compare assembled contigs against the reference or between them. In order to speed up the example, we will first use `NCBI-BLAST`[4] to project all the assembled contigs into the chosen reference chromosomes, and reduce all the downstream calculations. Thus, a database was created for each assembly sequence sets which will be queried by the reference selected chromosomes.

```
SQSET=SRR6130428

mkdir -vp $WDR/blast/dbs

makeblastdb -in $WDR/soapdenovo/$SQSET/${SQSET}_k63_graph.contig.fa \
            -dbtype nucl \
        -title "${SQSET}_SOAPdenovo_k63_contigs" \
        -out $WDR/blast/dbs/${SQSET}_SOAPdenovo_k63_contigs \
          2> $WDR/blast/dbs/${SQSET}_SOAPdenovo_k63_contigs.log 1>&2;
```

[3] S. Kurtz, A. Phillippy, A.L. Delcher, M. Smoot, M. Shumway, C. Antonescu, and S.L. Salzberg. "Versatile and open software for comparing large genomes." *Genome Biology*, 5:R12, 2004.
[4] C. Camacho, G. Coulouris, V. Avagyan, N. Ma, J. Papadopoulos, K. Bealer, and T.L. Madden. "BLAST+: architecture and applications." *BMC Bioinformatics*, 10:421, 2008.

For practical reasons, we focused on a couple of reference genome's chromosomes, `chrI` and `chrM` (mitochondrion genome). In order to obtain their sequences, they may be downloaded from the genome repository or filtered out from the whole genome fasta file as shown below:

```
mkdir -vp $WDR/seqs/chrs;

for SQ in chrI:NC_001133 chrM:NC_001224;
  do {
    SQN=${SQ%%:*}; # get the chr from SQ string
    SQI=${SQ##*:}; # get the refseq id from SQ string
    echo "# Filtering $SQN [$SQI] from whole genome fasta file..." 1>&2;
    samtools faidx \
             $WDR/seqs/S288C_reference_genome_R64-4-1_20230830/S288C_reference_sequence_R64-4-1_20230830.f
         'ref|'$SQI'|' | \
      sed 's/^>.*$/>Scer_'$SQN'/;' > $WDR/seqs/chrs/$SQN.fa
  }; done
```

Subsequently, the reference sequences may be `BLAST`-ed against the newly created database; using the `megablast` option to compare sequences of the same species, since this `BLAST` program is optimal for the desired genomic searches.

```
# here is defined a custom BLAST tabular output format
BLASTOUTFORMAT='6 qseqid qlen sseqid slen qstart qend sstart send length';
BLASTOUTFORMAT=$BLASTOUTFORMAT' score evalue bitscore pident nident ppos positive';
BLASTOUTFORMAT=$BLASTOUTFORMAT' mismatch gapopen gaps qframe sframe';
export BLASTOUTFORMAT;

#fixing permission/ownership issues with the blast/ directory
ls -l blast/
#   total 12
#   drwxr-xr-x 2 root root 4096 Mar 20 11:48 chrI-x-SRR6130428
#   drwxr-xr-x 2 root root 4096 Mar 20 11:48 chrM-x-SRR6130428
#   drwxr-xr-x 2 root root 4096 Mar 20 11:25 dbs

sudo chown -R mariana /home/mariana/uni/exercise_02/blast/
chmod u+rwx blast/
ls -l blast/
#   total 12
#   drwxr-xr-x 2 mariana root 4096 Mar 20 11:48 chrI-x-SRR6130428
#   drwxr-xr-x 2 mariana root 4096 Mar 20 11:48 chrM-x-SRR6130428
#   drwxr-xr-x 2 mariana root 4096 Mar 20 11:25 dbs

for SQ in chrI chrM;
  do {
    echo "# Running MEGABLAST: $SQSET x $SQ ..." 1>&2;
    mkdir -vp $WDR/blast/${SQ}-x-${SQSET};
    blastn -task megablast -num_threads 8                       \
           -db    $WDR/blast/dbs/${SQSET}_SOAPdenovo_k63_contigs \
           -outfmt "$BLASTOUTFORMAT"                            \
           -query $WDR/seqs/chrs/$SQ.fa                         \
           -out   $WDR/blast/${SQ}-x-${SQSET}/${SQ}-x-${SQSET}_megablast.out \
             2>   $WDR/blast/${SQ}-x-${SQSET}/${SQ}-x-${SQSET}_megablast.log;
  }; done
```

Once contigs have been matched, they are filtered from the whole assembly fasta file.

```
for SQ in chrI chrM;
  do {
    echo "# Running MEGABLAST: $SQSET x $SQ ..." 1>&2;
    OFBN="$WDR/blast/${SQ}-x-${SQSET}/${SQ}-x-${SQSET}_megablast";
    # this is a check to ensure we start with an empty contigs fasta file
    if [ -e "$WDR/blast/${SQ}-x-${SQSET}/${SQ}-x-${SQSET}_megablast.fa" ];
      then
```

```
        printf '' > "$WDR/blast/${SQ}-x-${SQSET}/${SQ}-x-${SQSET}_megablast.fa"; # rm can be also used her
      fi;
    # get the contig IDs from third column and filter out sequences
    gawk '{ print $3 }' "$WDR/blast/${SQ}-x-${SQSET}/${SQ}-x-${SQSET}_megablast.out" | sed 's/^>//' | sort
      while read SQID;
        do {
           samtools faidx \
                    $WDR/soapdenovo/$SQSET/${SQSET}_k63_graph.contig.fa \
                    "$SQID";
        }; done >> "$WDR/blast/${SQ}-x-${SQSET}/${SQ}-x-${SQSET}_megablast.fa" \
                2> "$WDR/blast/${SQ}-x-${SQSET}/${SQ}-x-${SQSET}_megablast.fa.log";
  }; done
```

The filtered contigs are then subjected to a sequence comparison procedure based on `dnadiff`:

```
SQSET=SRR6130428

for SQ in chrI chrM;
  do {
    printf "# Running DNADIFF protocol: $SQSET x $SQ ..." 1>&2;
    IFBN="$WDR/blast/${SQ}-x-${SQSET}/${SQ}-x-${SQSET}_megablast";
    OFBD="$WDR/blast/${SQ}-x-${SQSET}/${SQ}-x-${SQSET}_megablast.dnadiff";
    #
    dnadiff -p $WDR/blast/${SQ}-x-${SQSET}/${SQ}-x-${SQSET}_megablast.dnadiff                \
            $WDR/seqs/chrs/${SQ}.fa \
            $WDR/blast/${SQ}-x-${SQSET}/${SQ}-x-${SQSET}_megablast.fa      \
         2> $WDR/blast/${SQ}-x-${SQSET}/${SQ}-x-${SQSET}_megablast.dnadiff.log;
    printf " DNAdiff..." 1>&2;
    mummerplot --large --layout --fat --postscript \
               -t "Alignment Plot: ${SQ}-x-${SQSET}" \
               -p $WDR/blast/${SQ}-x-${SQSET}/${SQ}-x-${SQSET}_megablast.dnadiff          \
                  $WDR/blast/${SQ}-x-${SQSET}/${SQ}-x-${SQSET}_megablast.dnadiff.1delta    \
               2> $WDR/blast/${SQ}-x-${SQSET}/${SQ}-x-${SQSET}_megablast.dnadiff.alnplot.log;
    gm convert \
         $WDR/blast/${SQ}-x-${SQSET}/${SQ}-x-${SQSET}_megablast.dnadiff.ps \
         -background white
         $WDR/blast/${SQ}-x-${SQSET}/${SQ}-x-${SQSET}_megablast.dnadiff.png;
               # add this to convert PostScript image to PNG
    printf " ALNplot..." 1>&2;
    mummerplot --large --layout --fat --postscript \
               -t "Coverage Plot: ${SQ}-x-${SQSET}" \
               -p $WDR/blast/${SQ}-x-${SQSET}/${SQ}-x-${SQSET}_megablast.dnadiff.covg     \
                  $WDR/blast/${SQ}-x-${SQSET}/${SQ}-x-${SQSET}_megablast.dnadiff.1delta    \
               2> $WDR/blast/${SQ}-x-${SQSET}/${SQ}-x-${SQSET}_megablast.dnadiff.cvgplot.log;
    convert-im6 -verbose \
        -background white \
  -contrast-stretch 0x70% \
    -quality 600 \
        $WDR/blast/${SQ}-x-${SQSET}/${SQ}-x-${SQSET}_megablast.dnadiff.covg.ps \
        $WDR/blast/${SQ}-x-${SQSET}/${SQ}-x-${SQSET}_megablast.dnadiff.covg.png;
              # add this to convert PostScript image to PNG
    printf " CVGplot..." 1>&2;j
    ( grep "TotalBases"   $WDR/blast/${SQ}-x-${SQSET}/${SQ}-x-${SQSET}_megablast.dnadiff.report;
      grep "AlignedBases" $WDR/blast/${SQ}-x-${SQSET}/${SQ}-x-${SQSET}_megablast.dnadiff.report;
      grep "AvgIdentity"  $WDR/blast/${SQ}-x-${SQSET}/${SQ}-x-${SQSET}_megablast.dnadiff.report
      ) > $WDR/blast/${SQ}-x-${SQSET}/${SQ}-x-${SQSET}_megablast.dnadiff.shortsummary;
    printf " DONE\n" 1>&2;
  }; done
```

For chrI and chrM assemblies, the estimated aligned bases coverage is 297 103/230 218 = 1.29X and 85 749/85 779 = 0.99X, respectively.

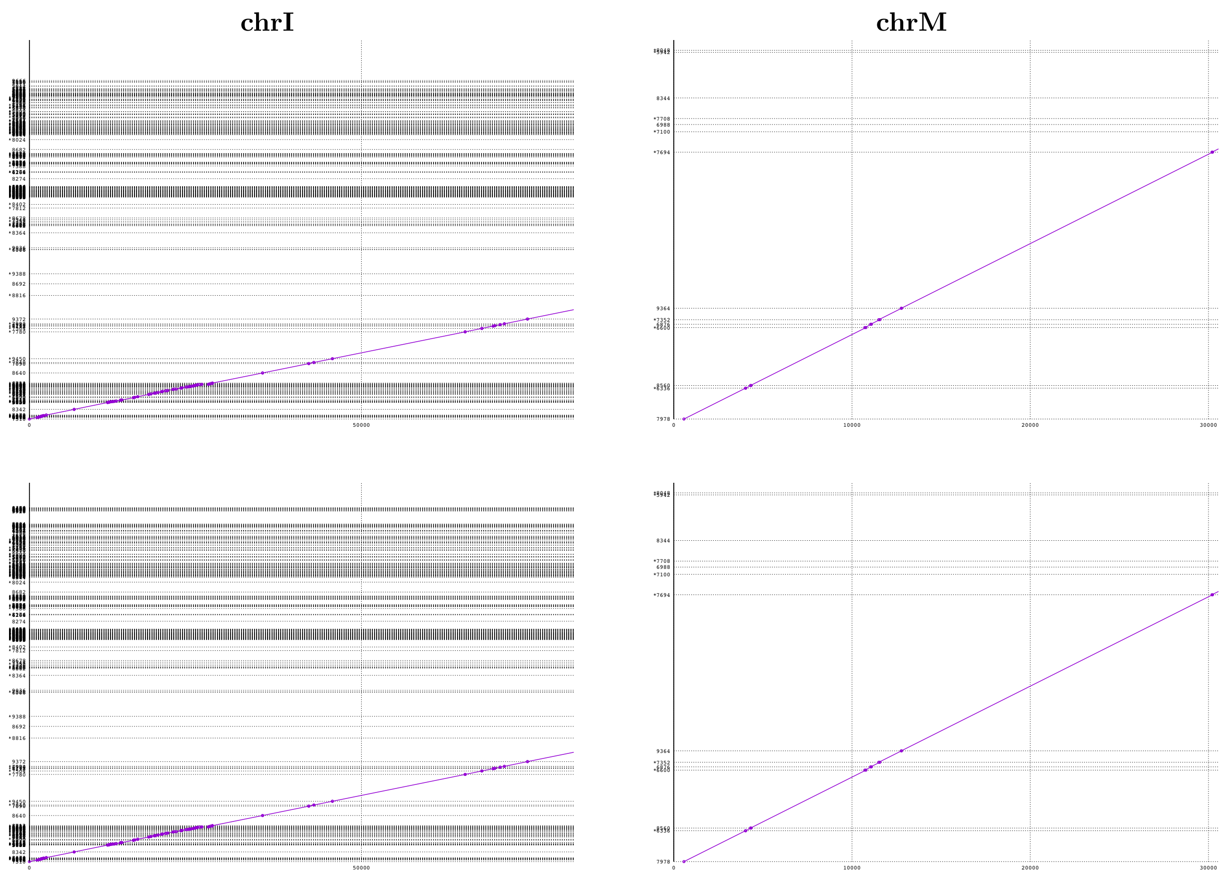

Figure 8: **`dnadiff` comparison between two reference chromosomes and `SRR6130428` contigs.** Top panels show the alignment plots of contigs from `SRR6130428` assembly mapped over two reference *Saccharomyces cerevisiae* chromosomes, chrI and chrM on left and right panels respectively. Bottom panels show the chrI and chrM alignment coverage. Contigs aligning to chrM have better contiguity and higher coverage than the nuclear chromosomes (chrI), as the plot displays a more diagonal and longer line.

3.2 Assessment of genome completeness with `BUSCO`

`BUSCO`[5] estimates the completeness and redundancy of processed genomic data based on universal single-copy orthologs. First of all, we need to check if there is a clade-specific parameters set that fits with our organism, *Saccharomyces cerevisiae* belong to the *Saccharomycetes* class, within the *Ascomycota* phylum in the Fungi kingdom (see NCBI Taxonomy browser species card).

```
busco --list-datasets
# 2022-10-24 19:06:49 INFO: Downloading information on latest versions of BUSCO data...
# 2022-10-24 19:06:52 INFO: Downloading file 'https://busco-data.ezlab.org/v5/data/information/lineages_li
# 2022-10-24 19:06:53 INFO: Decompressing file '/home/lopep/SANDBOX/BScCG2324/exercise_02/busco_downloads/
#
# ################################################
#
# Datasets available to be used with BUSCO v4 and v5:
#
#  bacteria_odb10
#      - ...
#  archaea_odb10
#      - ...
#  eukaryota_odb10
#      - ...
```

[5]M. Manni, M.R. Berkeley, M. Seppey, F.A. Simão, and E.M. Zdobnov.
"`BUSCO` Update: Novel and Streamlined Workflows along with Broader and Deeper Phylogenetic Coverage for Scoring of Eukaryotic, Prokaryotic, and Viral Genomes." *Molecular Biology and Evolution*, 38(10)4647-–4654, 2021.

```
#      - fungi_odb10
#          - ascomycota_odb10
#              - ...
#              - saccharomycetes_odb10
#              - ...
#          - ...
#      - ...
#  viruses (no root dataset)
#      - ...
#
```

There is a class-specific set to evaluate the completeness of the genome assembly `saccharomycetes_odb10`. However, it may be informative to use a more general parameters set, such as `fungi_odb10` or even `eukaryota_odb10` (or both), and compare the corresponding results. This can be useful to extrapolate the assessment to other species' genomes.

```
mkdir -vp $WDR/busco/$SQSET

# fix PATH to point bin folders where you installed bbmap and busco programs
export BBMAP=$BIN/bbmap:$BIN/busco/bin:$BBMAP
export BUSCO=$BIN/bin/busco/bin:$BUSCO

export OUT=$WDR/busco/$SQSET/${SQSET}_k63_contigs:$OUT

# cd $WDR
busco -m genome \
      -i $WDR/soapdenovo/$SQSET/${SQSET}_k63_graph.contig.fa \
      -o $WDR/busco/$SQSET/${SQSET}_k63_contigs \
      -l saccharomycetes_odb10
      -f
## if you get an error "A run with the name xxx already exists"
## you should add command-line option "-f" to force overwriting those files
#    --------------------------------------------------
#    |Results from dataset saccharomycetes_odb10       |
#    --------------------------------------------------
#    |  C:98.8%[S:96.7%,D:2.1%],F:0.5%,M:0.7%,n:2137   |
#    |2111  Complete BUSCOs (C)                        |
#    |2067  Complete and single-copy BUSCOs (S)        |
#    |44    Complete and duplicated BUSCOs (D)         |
#    |11    Fragmented BUSCOs (F)                      |
#    |15    Missing BUSCOs (M)                         |
#    |2137  Total BUSCO groups searched                |
#    --------------------------------------------------
# 2022-10-26 19:55:26 INFO: BUSCO analysis done. Total running time: 440 seconds
```

Note: The `generate_plot.py` script that is provided under the scripts folder of the `busco` installation may be used for plotting the resulting completeness values.

```
python3 $BIN/busco/scripts/generate_plot.py \
    -wd ./busco/$SQSET/${SQSET}_k63_contigs
```

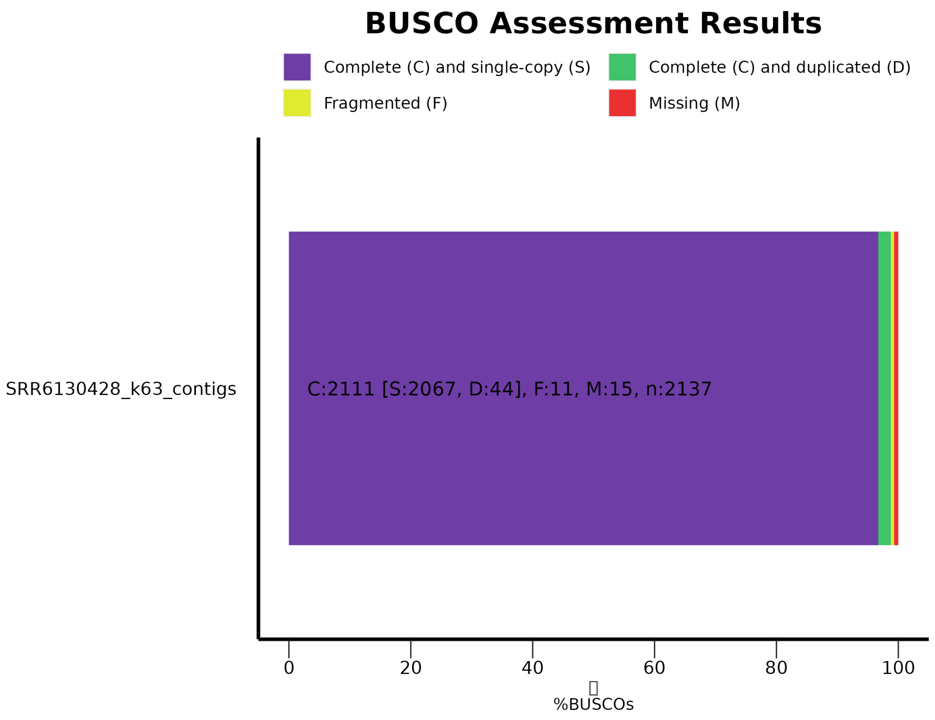

Figure 9: **`busco`'s quantitative assessment of the genetic completeness of the SRR6130428 contigs from *Saccharomyces cerevisiae*.**

4 Discussion

Due to hardware limitations, this report will solely discuss the complete analysis results of the Illumina HiSeq 4000 sequencing dataset SRR6130428 (SRX3242873 Short Reads Archive, 221.2Mb) of the *S. cerevisiae* genome. It will, however, also include some preliminary analysis of another dataset of the *S. cerevisiae* genome, for comparison of data quality between the different Illumina technologies (SRR3481383). From this SRA file, the two extracted spots were split into a forward (R1) and reverse (R2) fastq files of the raw sequencing data. The quality control of the raw sequencing data was performed to filter out contaminant sequences, adapter sequences, and any other sequences that do not meet the default quality thresholds defined by `fastqc`. As seen in Figure2, the mean phred score distribution per base position is high for R1 and R2 and the contigs present with an uniform GC content distribution.

The quality of the sequencing data was further improved by `trimmomatic` and `cutadapt` tools which removed the Illumina library adpater sequences and performed qualiy threshold filtration of the reads. Overall, `trimmomatic` and `cutadapt` removed 8.40% and 3.34% of the SRR6130428 total reads, and 18.53% and 3.06% of the SRR3481383 total reads. The filtered and trimmed dataset is represented in Figures **??**, **??** for the SRR6130428 dataset and **??** and **??** for the SRR3481383 dataset, representing 499,530,830 nucleotides at 41.09X coverage and 839,065,950 nucleotides at 69.02X coverage, respectively.

After continging, mapping and scaffolding the reads, `SOAPdenovo` assessed the SRR3481383 assembly and reported an average contig length of 3732 nucleotides, and an N50 of 14294 base pairs. Considering the estimated genome size (12,157,105 bp) of *S. cerevisiae* and its relative low level of complexity, the N50 statistic is reflecting that a substancial portion of the genome is represented in long contigs (14,294) and, thus, suggesting a contiguous assembly.

Chromosome 1's assembly sequence matched to 92.80% of that of the reference sequence, and the mitochondrion assembled genome matched to that of 77.78% of the reference sequence of *S. cerevisiae*. These results are reflected on the `mummerplot` output plots of Figure8, where the query and reference sequences are plotted based on similarity and identity at each nucleotide position. The alignment plots for chrI and chrM are indicative of a high degree of similarity between the assembled sequences and the reference genome of *S. cerevisiae*, with low number of tandem repeats, duplications or translocation events. Similarly, the coverage plots produced by `mummerplot` suggest that, for both chromosomal sequences, the query sequences were very similar to the references. As no deviation points from a diagonal line were observed, the assembly data has equivalent coverage of each nucleotide as that of the reference sequence. However, the chrM assembly stood out for its high coverage of aligned reads to the reference and alignment as it presented with the most proportional diagonal plot output for either measure. Ultimately, both chromosomal assemblies are seemingly complete and accurate in comparison with the reference.

To fully assess the completeness of the assemblies, the `BUSCO` tool was applied to both assembly datasets using the *S. cerevisiae*-specific parameters. This assay confirmed the hypothesis that the SRR3481383 dataset is a fairly complete and single dataset representative of the *S. cerevisiae* genome.

However, the SRR3481383 is only one of the datasets avalable on the Short Reads Archive depositorie. Using a larger dataset could potentially increase the sequence depth and contig length therefore producing a higher coverage, more contiguous genomic assembly better representative of the genome, inherently containing lower errors.

5 Appendices

5.1 Supplementary files

5.1.1 conda environment dependencies for the exercise

environment.yml

```
#
## #########################################################################################
##
##   environment.yml
##
##   Defining conda/mamba software dependencies to run MScGG-BIA practical exercises.
##
## #########################################################################################
##
##                 CopyLeft 2023/24 (CC:BY-NC-SA) --- Josep F Abril
##
##   This file should be considered under the Creative Commons BY-NC-SA License
##   (Attribution-Noncommercial-ShareAlike). The material is provided "AS IS",
##   mainly for teaching purposes, and is distributed in the hope that it will
##   be useful, but WITHOUT ANY WARRANTY; without even the implied warranty
##   of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
##
## #########################################################################################
#
# To install software for the exercise use the following command:
#
#    conda env create --file environment.yml
#
# then run the command below to activate the conda environment:
#
#    conda activate MScGG-BIA2324_exercises
#
name: MScGG-BIA2324_exercises
channels:
  - bioconda
  - conda-forge
  - defaults
dependencies:
  - htop
  - vim
  - emacs
  - gawk
  - perl
  - python
  - biopython
  - wget
  - curl
  - gzip
  - texlive-core
  - pandoc
  - pandocfilters
  - emboss
  - jellyfish
  - sra-tools
  - seqtk
  - fastqc
  - trimmomatic
  - samtools
  - bamtools
  - picard
  - bwa
  - bowtie2
  - soapdenovo2
  - igv
  - blast
  - gnuplot
  - graphicsmagick
  - mummer
  - hmmer
  - bbmap
  - busco
  # R-packages
  - r-ggplot2
  - r-reshape2
```

5.1.2 Project specific scripts

assemblathon_stats.pl

```
#!/usr/bin/perl
#
# assemblathon_stats.pl
#
# A script to calculate a basic set of metrics from a genome assembly
#
# Author: Keith Bradnam, Genome Center, UC Davis
# This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License.
#
# Last updated by: $Author: keith $
# Last updated on: $Date: 2011/10/13 00:07:00 $

use strict;
use warnings;
use FAlite;
use Getopt::Long;
use List::Util qw(sum max min);

###############################################
#
#  C o m m a n d   l i n e   o p t i o n s
#
###############################################

my $limit;       # limit processing of data to first $limit sequences (for quick testing)
my $graph;       # produce some output ready for Excel or R
my $csv;         # produce CSV output file of results
my $n_limit;     # how many N characters should be used to split scaffolds into contigs
my $genome_size; # estimated or known genome size (will be used for some stats)

GetOptions ("limit=i"       => \$limit,
                        "csv"           => \$csv,
                        "graph"         => \$graph,
                        "n=i"           => \$n_limit,
                        "genome_size=i" => \$genome_size);

# set defaults
$limit = 1000000000 if (!$limit);
$n_limit = 25       if (!$n_limit);


# check we have a suitable input file
my $usage = "Usage: assemblathon_stats.pl <assembly_scaffolds_file>
options:
        -limit <int> limit analysis to first <int> sequences (useful for testing)
        -csv         produce a CSV output file of all results
        -graph       produce a CSV output file of NG(X) values (NG1 through to NG99), suitable for graphing
        -n <int>     specify how many consecutive N characters should be used to split scaffolds into contigs
        -genome_size <int> estimated or known genome size
";

die "$usage" unless (@ARGV == 1);
my ($file) = @ARGV;

###############################################
#
#  S o m e   G l o b a l   v a r i a b l e s
#
###############################################

my $scaffolded_contigs = 0;                # how many contigs that are part of scaffolds
                                        # (sequences must have $n_limit consecutive Ns)
my $scaffolded_contig_length = 0;        # total length of all scaffolded contigs
my $unscaffolded_contigs = 0;                # how many 'orphan' contigs, not part of a scaffold
my $unscaffolded_contig_length = 0;        # total length of all contigs not part of scaffold
my $w = 60;                                # formatting width for output
my %data;                                # data structure to hold all sequence info key is
                                        # either 'scaffold', 'contig' or intermediate',
                                        # values are seqs & length arrays
my (@results, @headers);                # arrays to store results (for use with -csv option)
```

```
# make first loop through file, capture some basic info and add sequences to arrays
process_FASTA($file);

print "\n---------------- Information for assembly \'$file\' ----------------\n\n";

if(defined($genome_size)){
        my $mbp_size = sprintf("%.2f", $genome_size / 1000000);
        printf "%${w}s %10s\n", "Assumed genome size (Mbp)", $mbp_size;
}

# produce scaffold statistics
sequence_statistics('scaffold');

# produce a couple of intermediate statistics based on scaffolded contigs vs unscaffolded contigs
sequence_statistics('intermediate');

# finish with contig stats
sequence_statistics('contig');

# produce CSV output if required
write_csv($file) if ($csv);

exit(0);



##########################################
#
#
#    S  U  B  R  O  U  T  I  N  E  S
#
#
##########################################


##########################################
#    M A I N  loop through FASTA file
##########################################

sub process_FASTA{

        my ($seqs) = @_;

        my $input;

        # if dealing with gzip file, treat differently
        if($seqs =~ m/\.gz$/){
                open($input, "gunzip -c $seqs |") or die "Can't open a pipe to $seqs\n";
        } else{
                open($input, "<", "$seqs") or die "Can't open $seqs\n";
        }

        my $fasta = new FAlite(\*$input);

        # want to keep track of various contig + scaffold counts
        my $seq_count = 0;

        while(my $entry = $fasta->nextEntry){
            my $seq = uc($entry->seq);
                my $length = length($seq);
                $seq_count++;

                # everything gets pushed to scaffolds array
                push(@{$data{scaffold}{seqs}},$seq);
                push(@{$data{scaffold}{lengths}},$length);

                # if there are not at least 25 consecutive Ns in the sequence we need to split it into contigs
                # otherwise the sequence must be a contig itself and it still needs to be put in @contigs array
                if ($seq =~ m/N{$n_limit}/){

                        # add length to $scaffolded_contig_length
                        $scaffolded_contig_length += $length;

                        # loop through all contigs that comprise the scaffold
                        foreach my $contig (split(/N{25,}/, $seq)){
                                $scaffolded_contigs++;
                                my $length = length($contig);
                                push(@{$data{contig}{seqs}},$contig);
                                push(@{$data{contig}{lengths}},$length);
                        }
                } else {
                        # must be here if the scaffold is actually just a contig (or is a scaffold with < 25 Ns)
                        $unscaffolded_contigs++;
                        $unscaffolded_contig_length += $length;
                        push(@{$data{contig}{seqs}},$seq);
                        push(@{$data{contig}{lengths}},$length);
                }
                # for testing, just use a few sequences
                last if ($seq_count >= $limit);

        }
        close($input);
}


##########################################
#    Calculate basic assembly metrics
##########################################

sub sequence_statistics{
        my ($type) = @_;

        print "\n";

        # need descriptions of each result
        my $desc;

        # there are just a couple of intermediate level statistics to print
        if($type eq 'intermediate'){
                my $total_size = sum(@{$data{scaffold}{lengths}});

                # now calculate percentage of assembly that is accounted for by scaffolded contigs
                my $percent = sprintf("%.1f",($scaffolded_contig_length / $total_size) * 100);
                $desc = "Percentage of assembly in scaffolded contigs";
                printf "%${w}s %10s\n", $desc, "$percent%";
                store_results($desc, $percent) if ($csv);

                # now calculate percentage of assembly that is accounted for by unscaffolded contigs
                $percent = sprintf("%.1f",($unscaffolded_contig_length / $total_size) * 100);
                $desc = "Percentage of assembly in unscaffolded contigs";
                printf "%${w}s %10s\n", $desc, "$percent%";
                store_results($desc, $percent) if ($csv);


                # statistics that describe N regions that join contigs in scaffolds

                # get number of breaks
                my $contig_count = scalar(@{$data{contig}{lengths}});
                my $scaffold_count = scalar(@{$data{scaffold}{lengths}});
                my $average_contigs_per_scaffold = sprintf("%.1f",$contig_count / $scaffold_count);
                $desc = "Average number of contigs per scaffold";
                printf "%${w}s %10s\n", $desc, $average_contigs_per_scaffold;
                store_results($desc, $average_contigs_per_scaffold) if ($csv);

                # now calculate average length of break between contigs
                # just find all runs of Ns in scaffolds (>=25) and calculate average length
                my @contig_breaks;
                foreach my $scaffold (@{$data{scaffold}{seqs}}){
                        while($scaffold =~ m/(N{25,})/g){
                                push(@contig_breaks, length($1));
                        }
                }
                # set break size to zero if there are no Ns in scaffolds
                my $average_break_length;

                if(@contig_breaks == 0){
                        $average_break_length = 0;
                } else{
                        $average_break_length = sum(@contig_breaks) / @contig_breaks;
                }
                $desc = "Average length of break (>25 Ns) between contigs in scaffold";
                printf "%${w}s %10d\n", $desc, $average_break_length;
                store_results($desc, $average_break_length) if ($csv);
                return();
        }
        # n
        my $count = scalar(@{$data{$type}{lengths}});
        $desc = "Number of ${type}s";
        printf "%${w}s %10d\n", $desc, $count;
        store_results($desc, $count) if ($csv);



        # more contig details (only for contigs)
        if ($type eq 'contig'){
                $desc = "Number of contigs in scaffolds";
                printf "%${w}s %10d\n",$desc, $scaffolded_contigs;
                store_results($desc, $scaffolded_contigs) if ($csv);

                $desc = "Number of contigs not in scaffolds";
                printf "%${w}s %10d\n", $desc,$unscaffolded_contigs;
                store_results($desc, $unscaffolded_contigs) if ($csv);
        }


        # total size of sequences
        my $total_size = sum(@{$data{$type}{lengths}});
        $desc = "Total size of ${type}s";
        printf "%${w}s %10d\n", $desc, $total_size;
        store_results($desc, $total_size) if ($csv);


        # For scaffold data only, can caluclate the percentage of known genome size
        if ($type eq 'scaffold' && defined($genome_size)){
                my $percent = sprintf("%.1f",($total_size / $genome_size) * 100);
                $desc = "Total scaffold length as percentage of assumed genome size";
                printf "%${w}s %10s\n", $desc, "$percent%";
                store_results($desc, $percent) if ($csv);
        }


        # longest and shortest sequences
        my $max = max(@{$data{$type}{lengths}});
        $desc = "Longest $type";
        printf "%${w}s %10d\n", $desc, $max;
        store_results($desc, $max) if ($csv);

        my $min = min(@{$data{$type}{lengths}});
        $desc = "Shortest $type";
        printf "%${w}s %10d\n", $desc, $min;
        store_results($desc, $min) if ($csv);


        # find number of sequences above certain sizes
        my %sizes_to_shorthand = (1000     => '1K',
                                                  10000    => '10K',
                                                  100000   => '100K',
                                                  1000000  => '1M',
                                                  10000000 => '10M');

        foreach my $size (qw(1000 10000 100000 1000000 10000000)) {
                my $matches = grep { $_ > $size } @{$data{$type}{lengths}};
                my $percent = sprintf("%.1f", ($matches / $count) * 100);

                $desc = "Number of ${type}s > $sizes_to_shorthand{$size} nt";
                printf "%${w}s %10d %5s%%\n", $desc, $matches, $percent;
                store_results($desc, $matches)  if ($csv);

                $desc = "Percentage of ${type}s > $sizes_to_shorthand{$size} nt";
                store_results($desc, $percent)  if ($csv);
        };


        # mean sequence size
        my $mean = sprintf("%.0f",$total_size / $count);
        $desc = "Mean $type size";
        printf "%${w}s %10d\n", $desc, $mean;
        store_results($desc, $mean) if ($csv);

        # median sequence size
    my $median = (sort{$a <=> $b} @{$data{$type}{lengths}})[$count/2];
        $desc = "Median $type size";
        printf "%${w}s %10d\n", $desc, $median;
```

```
store_results($desc, $median) if ($csv);



#####################################################################################
 #
# N50 values
#
# Includes N(x) values, NG(x) (using assumed genome size)
# and L(x) values (number of sequences larger than or equal to N50 sequence size)
#####################################################################################

# keep track of cumulative assembly size (starting from smallest seq)
my $running_total = 0;

# want to store all N50-style values from N1..N100. First target size to pass is N1
my $n_index = 1;
my @n_values;
my $n50_length = 0;

my $i = 0;

my $x = $total_size * 0.5;
# start with longest lengths scaffold/contig
foreach my $length (reverse sort{$a <=> $b} @{$data{$type}{lengths}}){
        $i++;
        $running_total += $length;

        # check the current sequence and all sequences shorter than current one
        # to see if they exceed the current NX value
        while($running_total > int (($n_index / 100) * $total_size)){
                if ($n_index == 50){
                        $n50_length = $length;
                        $desc = "N50 $type length";
                        printf "%${w}s %10d\n", $desc, $length;
                        store_results($desc, $length) if ($csv);

                        # L50 = number of scaffolds/contigs that are longer than or equal to the N50 size
                        $desc = "L50 $type count";
                        printf "%${w}s %10d\n","L50 $type count", $i;
                        store_results($desc, $i) if ($csv);
                }
                $n_values[$n_index] = $length;
                $n_index++;
        }
}

my @ng_values;

# do we have an estimated/known genome size to work with?
if(defined($genome_size)){
        my $ng_index = 1;
        my $ng50_length = 0;

        $running_total = 0;
        $i = 0;

        foreach my $length (reverse sort{$a <=> $b} @{$data{$type}{lengths}}){
                $i++;
                $running_total += $length;

                # now do the same for NG values, using assumed genome size
                while($running_total > int (($ng_index / 100) * $genome_size)){
                        if ($ng_index == 50){
                                $ng50_length = $length;
                                $desc = "NG50 $type length";
                                printf "%${w}s %10d\n", $desc, $length;
                                store_results($desc, $length) if ($csv);

                                $desc = "LG50 $type count";
                                printf "%${w}s %10d\n", $desc, $i;
                                store_results($desc, $i) if ($csv);
                        }
                        $ng_values[$ng_index] = $length;
                        $ng_index++;
                }
        }
```

```
                my $n50_diff = abs($ng50_length - $n50_length);
                $desc = "N50 $type - NG50 $type length difference";
                printf "%${w}s %10d\n", $desc, $n50_diff;
                store_results($desc, $n50_diff) if ($csv);
        }
        # add final value to @n_values and @ng_values which will just be the shortest sequence
#        $n_values[100] = $min;
#        $ng_values[100] = $min;


        # base frequencies
        my %bases;

    my $seq = join('',@{$data{$type}{seqs}});
        my $length = length($seq);

    # count mononucleotide frequencies
    $bases{A} = ($seq =~ tr/A/A/);
    $bases{C} = ($seq =~ tr/C/C/);
    $bases{G} = ($seq =~ tr/G/G/);
    $bases{T} = ($seq =~ tr/T/T/);
    $bases{N} = ($seq =~ tr/N/N/);

        my $base_count = 0;
        foreach my $base (qw(A C G T N)) {
                my $percent = sprintf("%.2f", ($bases{$base} / $length) * 100);
                $desc = "$type %$base";
                printf "%${w}s %10s\n", $desc, $percent;
                store_results($desc, $percent) if ($csv);
                $base_count += $bases{$base};
        };

    # calculate remainder ('other) in case there are other characters present
        my $other = $length - $base_count;
        my $percent = sprintf("%.2f", ($other / $length) * 100);
        $desc = "$type %non-ACGTN";
        printf "%${w}s %10s\n",$desc, $percent;
        store_results($desc, $percent) if ($csv);

        $desc = "Number of $type non-ACGTN nt";
        printf "%${w}s %10d\n",$desc, $other;
        store_results($desc, $other) if ($csv);


        # anything to dump for graphing?
        if($graph){

                # create new output file name
                my $file_name = $file;
                $file_name =~ s/\.gz$//;
                $file_name =~ s/\.(fa|fasta)$//;
                $file_name .= ".${type}.NG50.csv";

                open(my $out, ">", "$file_name") or die "Can't create $file_name\n";
                print $out join (',',"Assembly",1..99), "\n";

                # make some guesses of what might constitute the unique assembly ID
                my $assembly_ID = $file;
                ($assembly_ID) = $file =~ m/^([A-Z]\d{1,2})_/ if ($file =~ m/^[A-Z]\d{1,2}_/);
                ($assembly_ID) = $file =~ m/^((bird|snake|fish)_\d+(C|E))_/ if ($file =~ m/^(bird|snake|fish)_\d+C|E_/);

                # CSV file, with filename in first column
                print $out "$assembly_ID";

                for (my $i = 1; $i < 100; $i++){
                        # higher NG values might not be present if assembly is poor
                        if (defined $ng_values[$i]){
                                print $out ",$ng_values[$i]";
                        } else{
                                print $out ",0";
                        }
                }
                print $out "\n";
                close($out);
        }
}

# simple routine to add results to a pair of arrays that will be used for printing results later on
# if -csv option is used
```

```
sub store_results{
        my ($desc, $result) = @_;

        push(@headers,$desc);
        push(@results,$result);
}

sub write_csv{
        my ($file) = @_;

        # create new output file name
        my $output = $file;
        $output =~ s/\.gz$//;
        $output =~ s/\.(fa|fasta)$//;
        $output .= ".csv";

        # make some guesses of what might constitute the unique assembly ID
        my $assembly_ID = $file;
        ($assembly_ID) = $file =~ m/^([A-Z]\d{1,2})_/ if ($file =~ m/^[A-Z]\d{1,2}_/);
        ($assembly_ID) = $file =~ m/^((bird|snake|fish)_\d+(C|E))_/ if ($file =~ m/^(bird|snake|fish)_\d+C|E_/);

        open(my $out, ">", $output) or die "Can't create $output\n";

        print $out "Assembly,";
        foreach my $header (@headers){
                print $out "$header,";
        }
        print $out "\n";

        print $out "$assembly_ID,";
        foreach my $result (@results){
                print $out "$result,";
        }
        print $out "\n";


        close($out);
}
```

5.1.3 Shell global vars and settings for this project

projectvars.sh

```
## ######################################################################################
##
##   projectvars.sh
##
##   A BASH initialization file for MScGG-BIA practical exercise folders
##
## ######################################################################################
##
##                 CopyLeft 2023/24 (CC:BY-NC-SA) --- Josep F Abril
##
##   This file should be considered under the Creative Commons BY-NC-SA License
##   (Attribution-Noncommercial-ShareAlike). The material is provided "AS IS",
##   mainly for teaching purposes, and is distributed in the hope that it will
##   be useful, but WITHOUT ANY WARRANTY; without even the implied warranty
##   of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
##
## ######################################################################################

#
# Base dir
export WDR=$PWD;
export BIN=$WDR/bin;
export DOC=$WDR/docs;

#
# Formating chars
export TAB=$'\t';
export RET=$'\n';
export LC_ALL="en_US.UTF-8";


export PATH=$WDR/bin/sratoolkit.3.0.0-ubuntu64/bin:$PATH

export SPD=$WDR/soapdenovo/$SQSET
```

```
  export BUSCO_CONFIG_FILE=$WDR/bin/busco/config/"myconfig.ini"

#
# pandoc's vars
NM="FAUSTINO_MARIANA";
RB="README_MScGG-BIA2324_exercise02";
RD="${RB}_${NM}";
PDOCFLGS='markdown+pipe_tables+header_attributes';
PDOCFLGS=$PDOCFLGS'+raw_tex+latex_macros+tex_math_dollars';
PDOCFLGS=$PDOCFLGS'+citations+yaml_metadata_block';
PDOCTPL=$DOC/MScGG_BIA_template.tex;
export RD PDOCFLGS PDOCTPL;

function ltx2pdf () {
    RF=$1;
    pdflatex $RF.tex;
    bibtex $RF;
    pdflatex $RF.tex;
    pdflatex $RF.tex;
}

function runpandoc () {
  pandoc -f $PDOCFLGS               \
         --template=$PDOCTPL        \
         -t latex --natbib          \
         --number-sections          \
         --highlight-style pygments \
         -o $RD.tex $RD.md;
  ltx2pdf $RD;
}

#
# add your bash defs/aliases/functions below...
alias dir='/bin/ls -alFhrt --color=auto'

export REFDIR="S288C_reference_genome_R64-4-1_20230830"
export REFGEN="S288C_reference_sequence_R64-4-1_20230830"

export DT=$WDR/data;

export TMA=/usr/share/trimmomatic/

export PERL5LIB=$BIN;
```

5.2 About this document

This document was be compiled into a PDF using `pandoc` (see `projectvars.sh` from previous subsection) and some `LaTeX` packages installed in this linux system.