Project 1- Read mapping and genome assembly

Fereshteh Noroozi

Spring 2023

Part A - Downloading E. Coli WGS data, preliminary analyses and quality controls.

1. Download SRR8185316 (short-read WGS of E.coli) and SRR10538956 (long-read WGS of E.coli) from SRA using the SRA Toolkit in Linux or Windows. (Hint: you may use fastq-dump [options] <accession>).

Answer:

Install SRA Toolkit in Linux:

```
sudo apt-get update
sudo apt-get install sra-toolkit
fastq-dump --version
```

Download above FASTQ files:

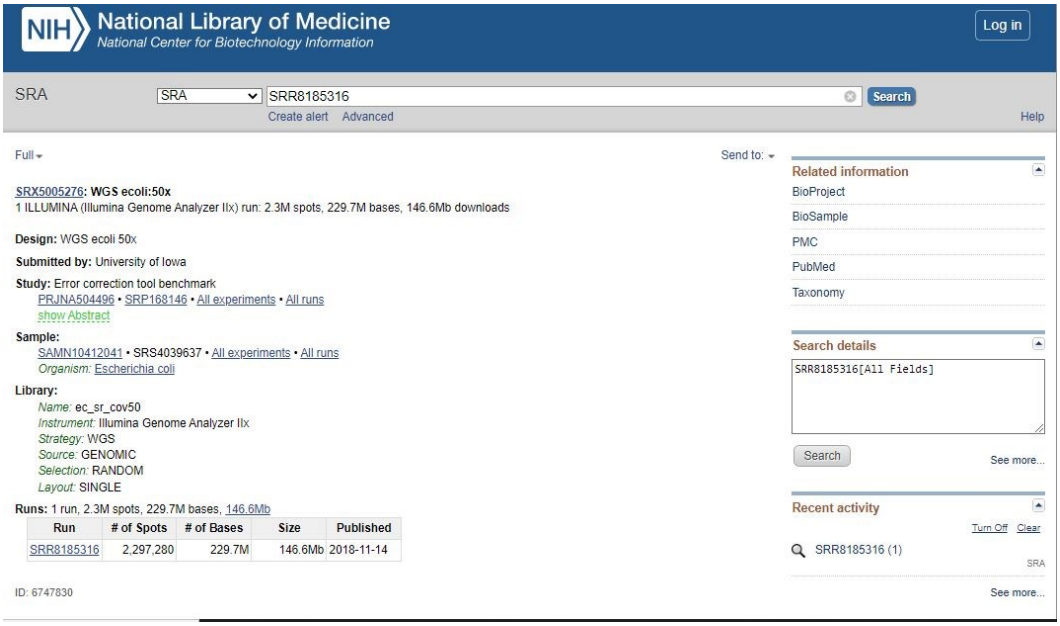
```
fastq-dump SRR8185316
```

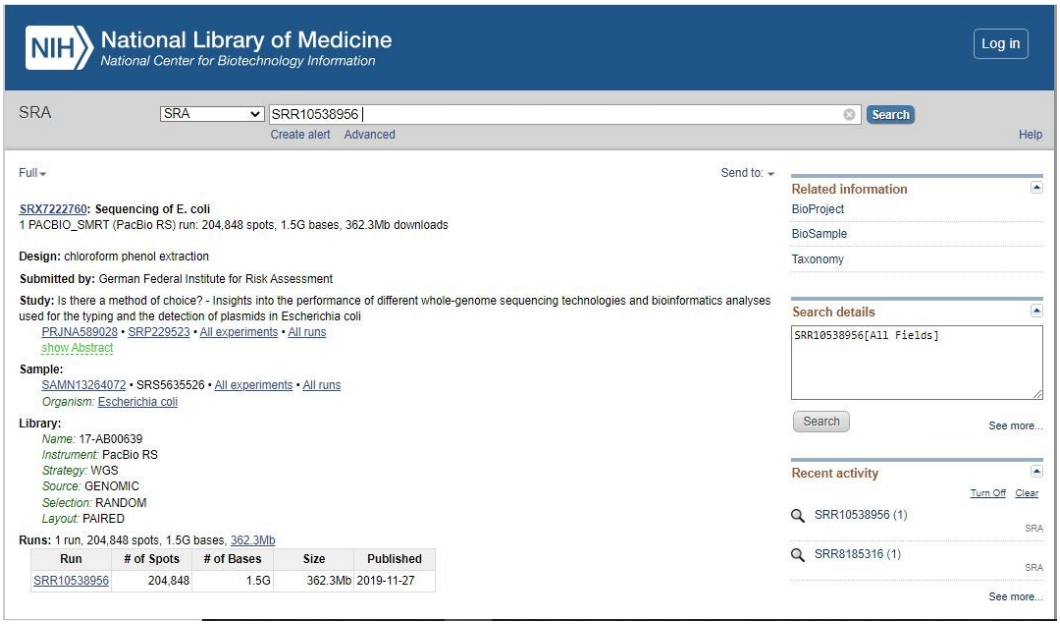
```
fastq-dump SRR10538956
```

2. In each file, find the strain of E.coli and the type of reads (single-end or paired-end) (refer to NCBI or EBI ENA (European Nucleotide Archive)). In addition, explain the difference between paired-end and interleaved paired-end files.

Answer:

I went to NCBI SRA database search page (https://www.ncbi.nlm.nih.gov/sra/). Then I entered "SRR8185316" in the search field and clicked on "Search". In library section, in layout field I found the answer. Also, repeat these steps for the other one.





Therefore, SRR8185316 is single-end sequencing data and SRR10538956 is paired -end sequencing data.

Paired-end sequencing generates two separate reads for each DNA fragment, while interleaved paired-end sequencing generates a single file containing both the forward and reverse reads of each fragment interleaved together. Paired-end sequencing provides more information about the fragment and is easier to process and analyze, while interleaved paired-end sequencing can make the data easier to manage but can be larger and more

difficult to analyze. The choice between the two depends on the platform, application, and research question.

3. Answer the following questions about the short reads fastq file (use existing R packages such as ShortRead to answer the following questions):

How many reads are in the fastq file?

Answer:

```
fastq_file <- file("SRR8185316.fastq", "r")
lines_per_read <- count.fields(fastq_file, sep = "\n")
num_reads <- length(lines_per_read) / 4
cat(paste0("The fastq file has ", num_reads, " reads."))
```

```
## The fastq file has 2297273 reads.
```

This code reads a FASTQ file and uses the `count.fields()` function to count the number of lines per read. The `sep` argument is set to "\n" to indicate that each line in the file represents a new record. The total number of reads is calculated by dividing the total number of lines by 4, since each read in a FASTQ file consists of four lines. Finally, the `cat()` function is used to print a message to the console indicating the total number of reads in the file.

Print the identifier, quality, and sequence of the first read of the fastq file.

Answer:

```
# Open fastq file connection
fastq_filecon <- file("SRR8185316.fastq", "r")

# Read the first line to get the identifier
first_line <- readLines(fastq_filecon, n = 1)
first_read_id <- first_line

# Read the second line to get the sequence
second_line <- readLines(fastq_filecon, n = 1)
first_read_seq <- second_line

# Skip the third line (the "+") and read the fourth line to get the quality scores
readLines(fastq_filecon, n = 1)
```

```
## [1] "+SRR8185316.1 ERR022075.10741970 length=100"
```

```
fourth_line <- readLines(fastq_filecon, n = 1)
first_read_quality <- fourth_line

# Close the fastq file connection
close(fastq_filecon)

# Print the identifier, quality, and sequence of the first read
cat("Identifier:", first_read_id, "\n")
```

```
## Identifier: @SRR8185316.1 ERR022075.10741970 length=100
```

```
cat("Quality:", first_read_quality, "\n")
```

```
## Quality: IIGIIIIIIIIIIIIHIIIIIIIIIIIIIIIII@IHHEHIIIIIIIIIIIHIIHIIIIIIIIGIHIIHFIIEIIGIHHGH@BE3BB>@>>2?@8?>?A@1
```

```
cat("Sequence:", first_read_seq, "\n")
```

```
## Sequence: AGCGGTACACATTATGGGTCTGCTCTCCGCAGGCGGCGTACACAGCCACGAAGATCACATCATGGCGATGGTAGAACTGGCAGCTGAACGCGGCGCAGAA
```

The readLines() function is used to read the lines of the file, with the n argument specifying how many lines to read. The cat() function is used to print the extracted information to the console. After reading the information, the file connection is closed using the close() function.

How many times does the TTAAATGGAA subsequence appear in the file?

Answer:

```
data <- readLines("SRR8185316.fastq")

count <- 0

 for (i in 1:length(data)) {
  if (grepl("TTAAATGGAA", data[i])) {
    count <- count + 1
  }
}


cat("The subsequence TTAAATGGAA appears", count, "times in the fastq file.")
```

```
## The subsequence TTAAATGGAA appears 179 times in the fastq file.
```

First, the readLines() function is used to read the FASTQ file (SRR8185316.fastq) line by line and store the contents of the file in the data variable. Next, a counter variable called count is initialized to 0.Then, a loop is used to iterate through each line in the data variable. For each line, the grepl() function is used to search for the specific subsequence "TTAAATGGAA". If the subsequence is found in the line, the count variable is incremented by 1. Finally, the cat() function is used to print the total count of the subsequence occurrences in the FASTQ file to the console.

Extract the first 1000 sequences of the fastq files (4000 lines).

Answer:

```
con <- file("SRR8185316.fastq", "r")

first1000 <- character(4000)

for (i in 1:4000) {
  first1000[i] <- readLines(con, n = 1)
}

close(con)

writeLines(first1000, "first1000.fastq")
```

First, the file() function is used to open the FASTQ file in read mode and the resulting connection is stored in the con variable. Next, a character vector called first1000 is initialized with a length of 4000. This vector will be used to store the first 4000 lines of the FASTQ file. Then, a loop is used to iterate through the first 4000 lines of the file. For each line, the readLines() function is used to read a single line from the file connection (con) and store it in the corresponding element of the first1000 vector. After all 4000 lines have been read and stored in first1000, the close() function is used to close the file connection (con). Finally, the writeLines() function is used to write the contents of first1000 to a new FASTQ file called first1000.fastq.

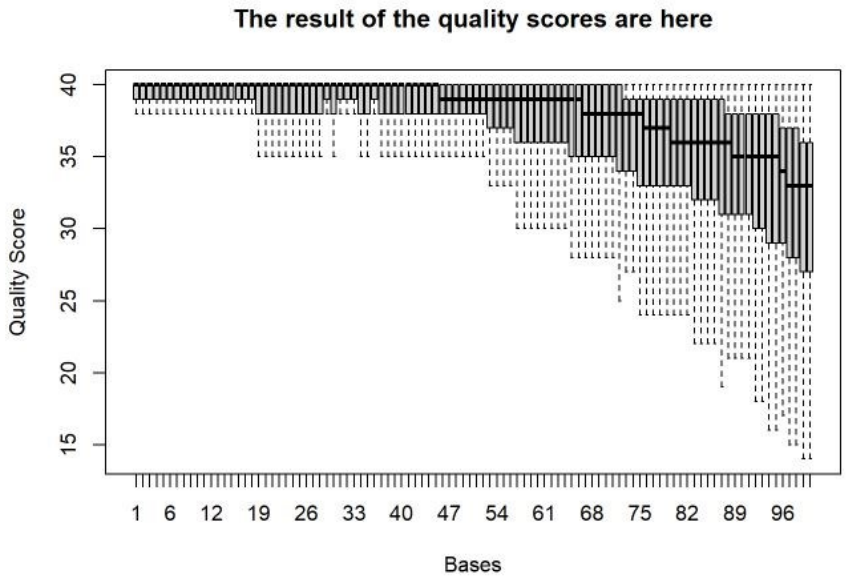
Note: The first1000.fastq is available in PartA-results folder.

Plot the quality of the reads in the fastq file using a box plot.

Answer:

```
file_con <- file("SRR8185316.fastq", "r")
#file_con <- file(fastq_file, "r")

scores_list <- list()
count = 1

while (length(line <- readLines(file_con, n = 4)) == 4) {
  quality_scores <- as.integer(charToRaw(line[4])) - 33

  scores_list[[length(scores_list) + 1]] <- quality_scores
  #scores_list = lapply (scores_list,quality_scores)
  count = count+1

  print (count)

  if (count>100000){
    break
  }
}
```

```
mat <- do.call(rbind, scores_list)

boxplot(mat, main = "The result of the quality scores are here", xlab = "Bases", ylab = "Quality Score",outline =
FALSE)
```



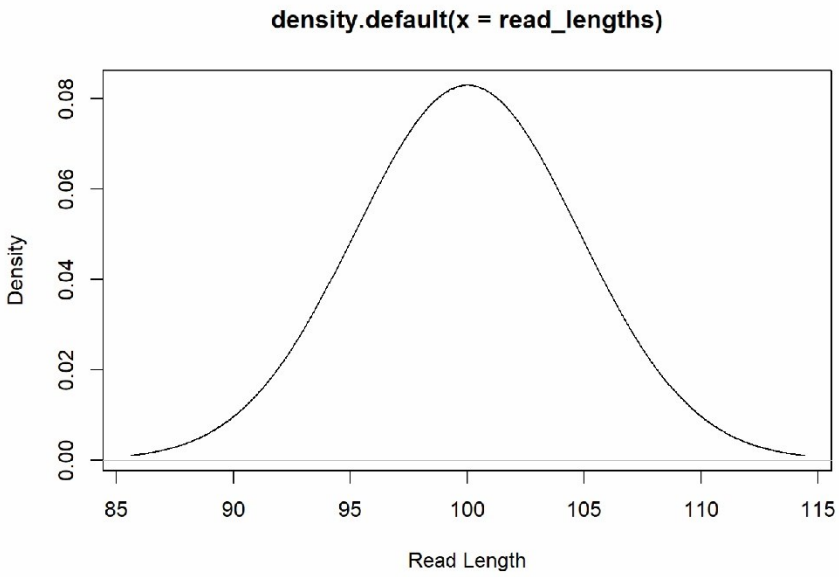

```
close(file_con)
```

In the box plot, the x-axis is representing the position of the base in the read, and the y-axis is representing the quality score. The box plot has a box for each position in the read, and the box showes the distribution of quality scores for that position.The box has a line in the middle representing the median, and the box should extend to the 25th and 75th percentiles of the quality scores. The whiskers of the box plot should extend to the most extreme data

point that is within 1.5 times the interquartile range (IQR) of the lower or upper quartile. Outliers beyond the whiskers should be shown as points.

Show the distribution of read lengths using a density plot.

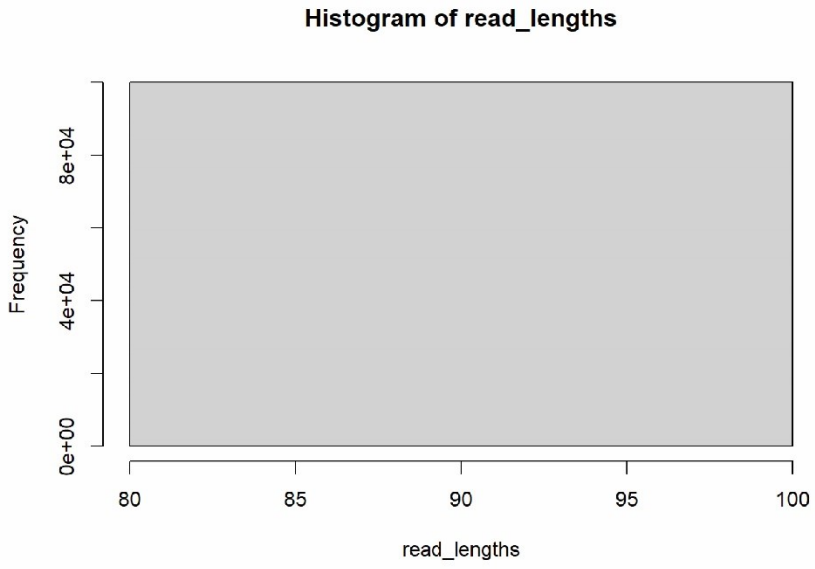
```r
# open file connection
con <- file("SRR8185316.fastq", "r")

# initialize variables
read_lengths <- c()

# loop through every 4 lines (reads) in the file
while(length(line1 <- readLines(con, n = 1)) > 0) {
  # read second line (sequence)
  line2 <- readLines(con, n = 1)
  # extract read length
  len <- nchar(line2)
  # add read length to vector
  read_lengths <- c(read_lengths, len)
  # skip next two lines (quality scores)
  readLines(con, n = 2)
}

# close file connection
close(con)
dens <- density(read_lengths)
# plot density plot
plot(dens, type = "l", xlab = "Read Length", ylab = "Density")
```



```r
fastq_file <- file("SRR8185316.fastq", "r")
read_lengths <- vector("integer", length = 0)

count = 1

while (TRUE) {

  header <- readLines(fastq_file, n = 1)
  if (length(header) == 0) {
    break
  }

  sequence <- readLines(fastq_file, n = 1)
  spacer <- readLines(fastq_file, n = 1)
  quality <- readLines(fastq_file, n = 1)
  read_lengths <- c(read_lengths, nchar(sequence))
  count = count +1
  if (count>100000){
    break
  }
}

close(fastq_file)
head(read_lengths, 10)
```

```
##  [1] 100 100 100 100 100 100 100 100 100 100
```

```r
#plot(density(read_lengths))
hist(read_lengths,)
```



4. Perform quality control of the reads using FastQC and interpret the results. Complete

this task using both the command-line function as well as the FastQC graphical interface. (you may download FastQC from here.)

Answer:

Install SRA FastQC in Linux:

```
sudo apt-get install default-jdk
java -version
wget https://www.bioinformatics.babraham.ac.uk/projects/fastqc/fastqc_v0.11.9.zip
unzip fastqc_v0.11.9.zip
```

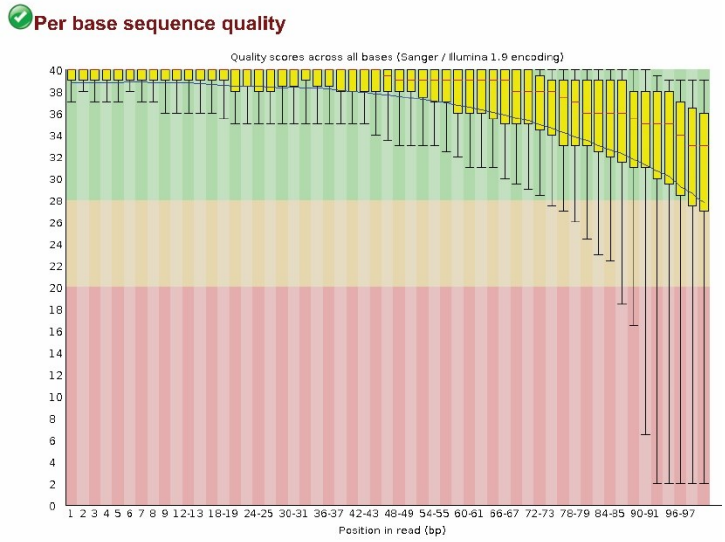
To run the FastQC:

```
cd FastQC
./fastqc
```

Command-line function:

```
fastqc /home/Fereshteh/Project1-algorithim/inputs/SRR8185316.fastq
-o /home/Fereshteh/Project1-algorithim
```

Both the command-line function as well as the FastQC graphical interface had html outputs that are available in PartA-results folder.



Part B - De novo genome assembly

Install SPAdes, Canu, Quast, BWA, Samtools, Pilon, or any other alternative software of your choice. Afterwards, follow the steps provided below.

Answer:

Install SPAdes in Linux:

```
sudo apt-get install -y build-essential g++ cmake python3-dev zlib1g
-dev libbz2-dev liblzma-dev libncurses5-dev libffi-dev libboost-all-
dev
wget http://cab.spbu.ru/files/release3.15.3/SPAdes-3.15.3-Linux.tar.
gz
tar -xzf SPAdes-3.15.3-Linux.tar.gz
cd SPAdes-3.15.3-Linux
PREFIX=$(pwd) make install
export PATH=$PATH:/home/Fereshteh/SPAdes-3.15.3-Linux/bin
sudo apt install spades
```

Install Canu in Linux:

```
git clone https://github.com/marbl/canu.git
sudo apt install git
cd canu/src
sudo apt install canu
```

Install Quast in Linux:

```
pip install quast
sudo apt install linuxbrew-wrapper
brew install quast
conda install -c bioconda quast
```

Install BWA in Linux:

```
sudo apt-get install build-essential zlib1g-dev
sudo apt autoremove
wget https://github.com/lh3/bwa/releases/download/v0.7.17/bwa-0.7.1
7.tar.bz2
tar xjf bwa-0.7.17.tar.bz2
cd bwa-0.7.17
make
sudo apt-get install bwa
```

Install Samtools in Linux:

```
sudo apt-get install build-essential zlib1g-dev libncurses5-dev libb
z2-dev liblzma-dev
wget https://github.com/samtools/samtools/releases/download/1.13/sam
tools-1.13.tar.bz2
tar xjf samtools-1.13.tar.bz2
cd samtools-1.13
./configure --prefix=/usr/local
make
sudo make install
```

Install Samtools in Linux:

```
sudo apt-get update
sudo apt-get -y install pilon
```

1. Run SPAdes to generate draft genome assemblies from short reads.

Answer:

```
spades.py -o /home/Fereshteh/Project1-algorithim/outputs-spades -s /
home/Fereshteh/Project1-algorithim/inputs/SRR8185316.fastq
```

The outputs-spades is the output of this above command, which is available in PartB-results folder. The output of a genome assembly analysis using SPAdes or any other tool typically includes several files that provide information about the quality and characteristics of the assembled genome. Here are some of the key output files that you might expect to see:

Contigs and scaffolds: These are the primary output of the assembly process. Contigs are the initial contiguous sequences of DNA that are assembled from the input reads, while

scaffolds are the ordered and oriented contigs that are connected by gaps. The final scaffolds represent the draft genome assembly.

Assembly statistics: SPAdes and other assembly tools typically generate summary statistics that describe the quality and characteristics of the assembled genome. These might include metrics such as the total number of contigs and scaffolds, the N50 (the length of the shortest scaffold that covers 50% of the genome), the total assembly size, and the number of gaps in the final assembly.

Assembly graphs: Assembly graphs are visual representations of the connections between contigs and scaffolds in the final assembly. These can be used to visualize the structure of the genome and identify potential misassemblies or regions of low quality.

Annotation files: Once a draft genome assembly has been generated, it can be annotated to identify genes, regulatory elements, and other functional elements. Annotation files might include predicted gene sequences, gene locations, functional annotations, and other information.

Overall, the output of a genome assembly analysis using SPAdes or any other tool can provide valuable insights into the structure and function of a genome. However, it's important to keep in mind that genome assembly is a complex and often imperfect process, and the quality of the final assembly will depend on a variety of factors, including the quality and quantity of the input reads, the specific assembly parameters and algorithms used, and the characteristics of the genome itself.

2. Run Canu to generate draft genome assemblies from long reads.

Answer:

```
./canu -p ecoli -d ecoli-pacbio genomeSize=4.8m -pacbio SRR10538956.
fastq &
```

This command runs the Canu assembly pipeline to generate a draft genome assembly from PacBio long-read sequencing data. It takes several parameters and options that control various aspects of the assembly process.

- `./canu`: This is the command to run the Canu assembly pipeline. `./` specifies that the command should be executed from the current directory, and `canu` is the name of the Canu executable file.

- `-p ecoli`: This option specifies the prefix to use for the output files generated by Canu. In this case, the prefix is set to `ecoli`.

- `-d ecoli-pacbio`: This option specifies the output directory where Canu should save the output files. In this case, the output directory is set to `ecoli-pacbio`.

- `genomeSize=4.8m`: This parameter specifies the expected size of the genome being assembled, in this case, 4.8 million base pairs.

- `-pacbio SRR10538956.fastq`: This option specifies the input PacBio long-read sequencing data file in FASTQ format, with the file name `SRR10538956.fastq`. Canu will use this input data to generate the draft genome assembly.

- `&`: This character at the end of the command runs the Canu assembly pipeline in the background, allowing you to continue using the terminal while the assembly process is running.

Note: The canu folder is output of above command, which is available in PartB-results folder.

The output files is included various intermediate files generated during the assembly process, as well as the final draft genome assembly in FASTA format. The quality and completeness of the draft genome assembly will depend on many factors, such as the quality and quantity of the input sequencing data, the expected genome size, and the specific parameters and algorithms used in the Canu assembly pipeline. It is important to carefully evaluate the quality of the draft genome assembly using tools such as QUAST or BUSCO, which can provide metrics on genome completeness, accuracy, and fragmentation, before using it for downstream analyses.

3. Assess the quality of the draft genome assembly (long reads) using Quast and compare it to the reference genome (Download the reference genome from here)

Answer:

```
quast.py /home/Fereshteh/Project1-algorithim/canu/ecoli.contigs.fas
ta -r /home/Fereshteh/Project1-algorithim/referencegenome.fasta -o
/home/Fereshteh/Project1-algorithim/quast_result.fastq
```

Overall, this command is evaluating the quality of a genome assembly by comparing it to a reference genome using the quast.py tool, and saving the results to an output directory. This type of analysis is commonly used in genome assembly projects to assess the quality of the assembly and identify areas for improvement.

Overall, the QUAST output files provide a comprehensive evaluation of the quality of the draft genome assembly, including metrics of contiguity, completeness, accuracy, and alignment to a reference genome. These metrics can be used to identify potential issues with the assembly, such as misassemblies, gaps, or regions of low quality, and to guide further refinement or validation of the assembly.

Note: The quast_result.fastq is output of above command, which is available in PartB-results folder. In addition, for comparing it to the reference genome, report.pdf is available.

4. Long-read technology may produce various types of error, leading to low accuracy in genome assemblies. To address this challenge, Pilon can be used, a tool that detects and corrects errors in genome assemblies, including single nucleotide polymorphisms (SNPs), insertions, and deletions (indels). The goal of this part is to enhance the draft genome assembly by using the short-read data. To this end, map the Illumina short-read data to the PacBio assembly using BWA, and then use Pilon to identify and correct assembly errors.

Answer:

Index the PacBio assembly using BWA:

```
bwa index /home/Fereshteh/Project1-algorithim/canu/ecoli.contigs.fa
sta
```

This command is creating an index file for a genome assembly file using the bwa index tool. The resulting index file is used for read alignment with bwa mem or bwa aln commands. The index file is a critical step for efficient alignment of reads to the genome assembly.

Map the Illumina short-read data to the PacBio assembly using BWA:

```
bwa mem /home/Fereshteh/Project1-algorithim/canu/ecoli.contigs.fast
a /home/Fereshteh/Project1-algorithim/inputs/SRR8185316.fastq -o /h
ome/Fereshteh/Project1-algorithim/canu/SRR.sam
```

This command is performing a read alignment using bwa mem and saving the alignment results to a SAM file. The resulting SAM file can be further processed, converted to BAM format, and sorted using tools such as samtools.

Convert the SAM file to BAM format and sort the alignments:

```
samtools view -S -b /home/Fereshteh/Project1-algorithim/canu/SRR.sa
m | samtools sort -o /home/Fereshteh/Project1-algorithim/canu/SRR.b
am
```

This command is converting a SAM file to a sorted BAM file using samtools. Sorted BAM files are commonly used in downstream analyses such as variant calling, transcript quantification, and ChIP-seq analysis.

Sort and index the BAM file:

```
samtools sort /home/Fereshteh/Project1-algorithim/canu/SRR.bam -o /
home/Fereshteh/Project1-algorithim/outputs/SRR.sorted.bam
samtools index /home/Fereshteh/Project1-algorithim/outputs/SRR.sort
ed.bam
```

This command is performing a sorting operation on a BAM file using samtools and creating an index file for the resulting sorted BAM file. The index file is necessary for efficient retrieval of data from the sorted BAM file and is commonly used in downstream analyses such as variant calling and transcript quantification.

Use Pilon to identify and correct assembly errors:

```
java -Xmx16G -jar pilon-1.24.jar --genome /home/Fereshteh/Project1-
algorithim/canu/ecoli.contigs.fasta --unpaired /home/Fereshteh/Proj
ect1-algorithim/canu/SRR.sorted.bam --output corrected_assembly
```

This command is performing genome polishing using pilon and using a sorted BAM file containing unpaired read alignments to correct errors in the initial genome assembly. Genome polishing is a critical step in improving the quality of a genome assembly, particularly in cases where the initial assembly contains errors or is incomplete.

Note: All outputs of this part are available in PartB-4 of PartB-results.

Part C - Read mapping and variant calling

1. Print the head of the obtained SAM file in part B, question 4. Explain what you see for the first hit (you can do this step either in Linux or R).

Answer:

```
head /home/Fereshteh/Project1-algorithim/SRR.Sam
```

```
@SQ     SN:tig00000001  LN:4774248
@SQ     SN:tig00000002  LN:22995
@SQ     SN:tig00000003  LN:168715
@SQ     SN:tig00000004  LN:122462
@SQ     SN:tig00000005  LN:127866
@SQ     SN:tig00000007  LN:4059
@SQ     SN:tig00000008  LN:2644
@SQ     SN:tig00000009  LN:66245
@SQ     SN:tig00000010  LN:19760
@SQ     SN:tig00000011  LN:19891
```

SAM files are commonly used to store the results of sequence alignment performed by bioinformatics tools such as Bowtie, BWA, or STAR. The header section provides important metadata about the reference sequences and the alignment process, which can be used for downstream analysis and interpretation of the alignment results.

Indicates that the SAM file being examined contains 10 reference sequences (contigs) and their corresponding lengths. The @SQ lines provide metadata information about the reference sequences and are part of the SAM file header. Each reference sequence is identified by a unique name, specified by the SN: field, and has a length specified by the LN: field. For example, the first reference sequence has the name tig00000001 and a length of 4,774,248 base pairs, while the second reference sequence has the name tig00000002 and a length of 22,995 base pairs.

2. Convert the SAM file to an indexed BAM file. Hint: use samtools view, samtools sort, samtools index.

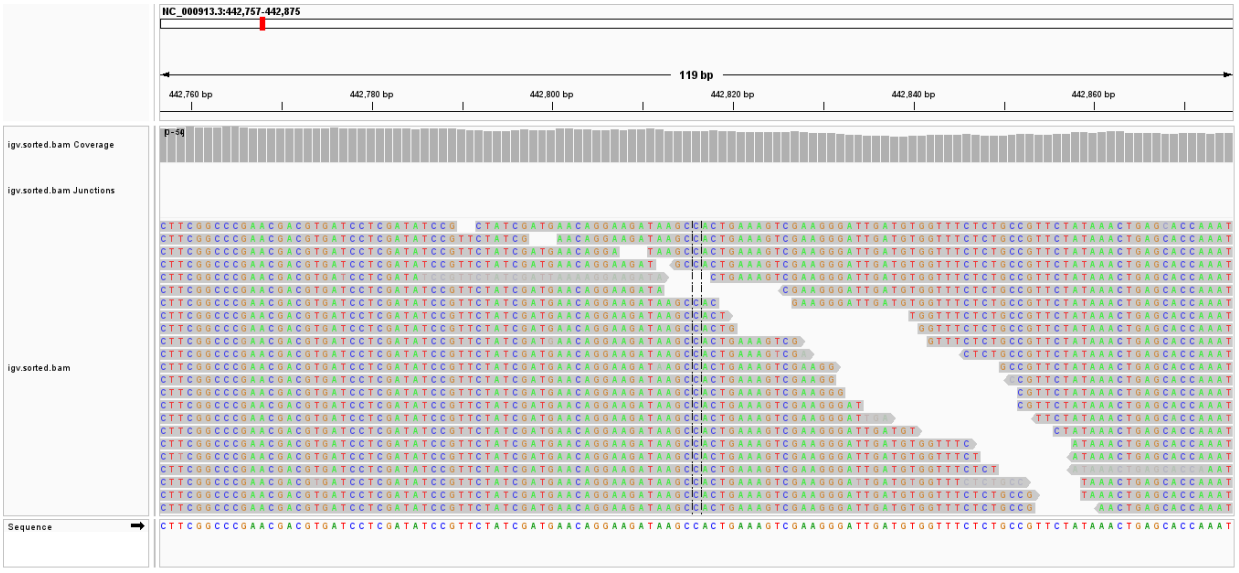
Answer:

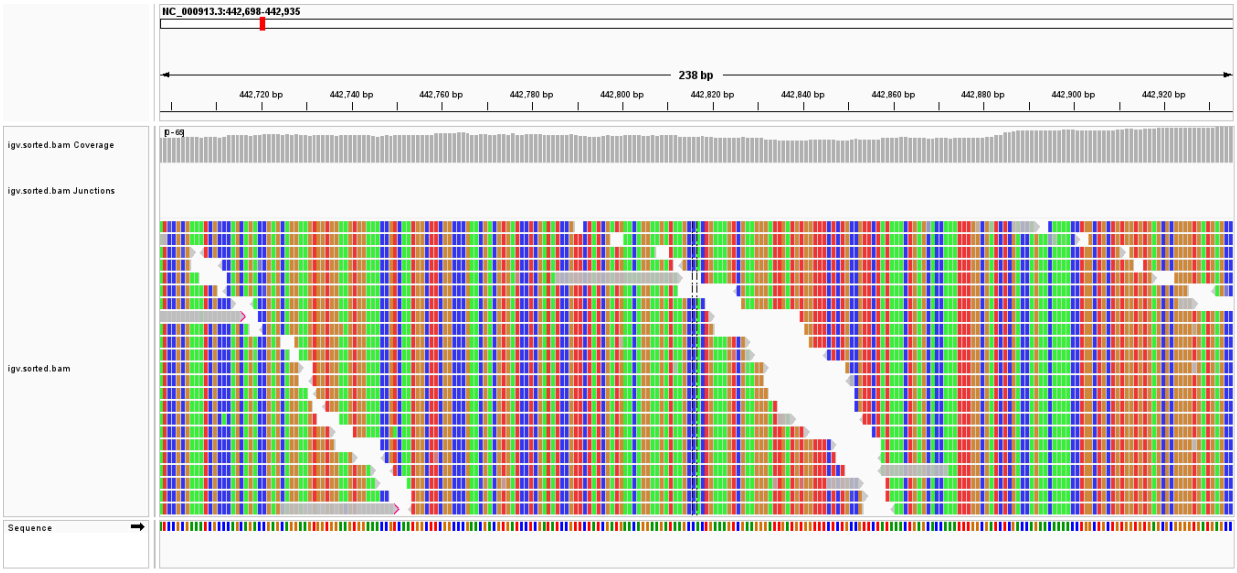
```
samtools view -bS /home/Fereshteh/Project1-algorithim/SRR.sam | samt
ools sort -o /home/Fereshteh/Project1-algorithim/SRR.bam -
samtools index /home/Fereshteh/Project1-algorithim/SRR.bam
```

These commands use samtools to convert a SAM file (/home/Fereshteh/Project1-algorithim/SRR.sam) to a sorted binary BAM file (samtools view and samtools sort), and then create an index file for the BAM file (samtools index). The resulting BAM file (/home/Fereshteh/Project1-algorithim/SRR.bam) can be used for downstream analysis and visualization of the sequence alignments.

3. Use the Integrative Genomics Viewer (IGV) to visualize the mapped reads in a 200-b genomic region of your choice. Select the reference genome fasta and GTF file (GTF is optional).

Answer:





Note: All outputs of this part are available in PartC-results.

4. Determine the percentage of short reads that are mapped to the assembled genome from the long reads as well as to the reference genome. Hint: use samtools flagstat.

Answer:

```
samtools flagstat /home/Fereshteh/Project1-algorithim/SRR.sorted.bam
```

```
2299401 + 0 in total (QC-passed reads + QC-failed reads)
0 + 0 secondary
2121 + 0 supplementary
0 + 0 duplicates
2188862 + 0 mapped (95.19% : N/A)
0 + 0 paired in sequencing
0 + 0 read1
0 + 0 read2
0 + 0 properly paired (N/A : N/A)
0 + 0 with itself and mate mapped
0 + 0 singletons (N/A : N/A)
0 + 0 with mate mapped to a different chr
0 + 0 with mate mapped to a different chr (mapQ>=5)
```

2299401: The total number of reads, including both QC-passed and QC-failed reads.

0 + 0 secondary: The number of reads that are secondary (e.g. alternative mapping locations). 2121 + 0 supplementary: The number of reads that are supplementary (e.g. chimeric alignments). 0 + 0 duplicates: The number of reads that are PCR or optical duplicates. 2188862 + 0 mapped (95.19% : N/A): The number and percentage of reads that are mapped to a reference genome. 0 + 0 paired in sequencing: The number of reads that are paired in sequencing. 0 + 0 read1 and 0 + 0 read2: The number of reads that are mate 1 and mate 2 in a paired-end sequencing experiment. 0 + 0 properly paired (N/A : N/A): The number and percentage of reads that are properly paired (i.e. both mates align to the same chromosome and with the expected orientation and insert size). 0 + 0 with itself and mate mapped: The number of reads that are mapped to the same chromosome and with the expected orientation and insert size. 0 + 0 singletons (N/A : N/A): The number and percentage of reads that are not part of a properly paired or self-aligned pair. 0 + 0 with mate mapped to a different chr: The number of reads that are mapped to a different chromosome than their mate. 0 + 0 with mate mapped to a different chr (mapQ>=5): The number of reads that are mapped to a different chromosome than their mate with a mapping quality score of at least 5.

5. Get the read depth for the sorted BAM file at all positions of the assembled genome and report the mean of all reads. Hint: use samtools depth.

Answer:

```
samtools depth /home/Fereshteh/Project1-algorithim/SRR.sorted.bam |
awk '{sum+=$3} END {print "Mean read depth:",sum/NR}'
```

Mean read depth: 48.6027

The result "48.6027" is the calculated mean read depth based on the above command .This means that the average number of reads covering each base in the BAM file is approximately 49. This metric can be useful for assessing the overall quality of the sequencing data and determining whether the coverage is sufficient for downstream analyses.

The entire command is a pipeline that combines these two programs to calculate the mean read depth of a BAM file.

6. Make yourself familiar with the CIGAR format. How do you interpret the following expressions?

Answer:

“29S21=1X25=”

29 soft-clipped bases (not included in the alignment) - 21 matched bases - 1 base mismatch (substitution) - 25 matched bases- 1 base deletion - 1 base matched

"29S": The first 29 bases of the read are soft-clipped, which means they are present in the read but not aligned to the reference genome. Soft-clipping can occur when the read extends beyond the end of the reference genome or contains low-quality bases at the beginning or end. "21=": The next 21 bases of the read match exactly with the reference genome. "1X": The next base of the read differs from the reference genome by a single nucleotide substitution (X)."25=": The next 25 bases of the read match exactly with the reference genome.

“20M2I1M1D10M”

20 matched bases - 2 inserted bases - 1 matched base - 1 deleted base - 10 matched bases

"20M": The first 20 bases of the read match exactly with the reference genome."2I": The next two bases of the read are insertions (I) that are not present in the reference genome."1M": The next base of the read matches exactly with the reference genome."1D": The next base of the reference genome is deleted (D) in the read."10M": The next 10 bases of the read match exactly with the reference genome.

"5M10N25M"

5 matched bases- 10 skipped bases (not included in the alignment) - 25 matched bases

"5M": The first 5 bases of the read match exactly with the reference genome."10N": The next 10 bases of the reference genome are skipped (N) in the read. This represents a gap or a splice junction in the alignment where the read spans an intron in the reference genome. "25M": The next 25 bases of the read match exactly with the reference genome.

7. Perform variant calling on the obtained BAM file using the reference genome or refined assembled genome from the long reads and save the output as a VCF file. Hint: you may need the following commands/options: samtools mpileup, bcftools, multiallelic-caller algorithm

Answer:

```
samtools mpileup -uf referencegenome.fasta SRR.sorted.bam | bcftool
s call -mv --ploidy 1 | bcftools filter --threads 4 -e '%QUAL<30 ||
DP<15' | bcftools view -Ov -M2 -o variant-calling.vcf
```

samtools mpileup -uf referencegenome.fasta SRR.sorted.bam: Creates a pileup of reads against the reference genome, using the referencegenome.fasta file as a reference and the SRR.sorted.bam file as input. bcftools call -mv --ploidy 1: Calls variants using the multiallelic-caller algorithm (-m) and identifies variant sites (-v). The --ploidy 1 option specifies to call variants as if the sample is haploid. bcftools filter --threads 4 -e '%QUAL<30 || DP<15': Filters variants based on quality and read depth. Variants with a quality score (QUAL) below 30 or a depth of coverage (DP) below 15 are excluded from the output. bcftools view -Ov -M2 -o variant-calling.vcf: Converts the output to VCF format (-Ov) and saves the result to the variant-calling.vcf file. The -M2 option specifies to output only biallelic variants.

Overall, this command pipeline performs basic variant calling and filtering on the input BAM file, using the reference genome to identify variant sites and filtering based on quality and read depth. Note that the specific filtering and calling options used may vary depending on the characteristics of the data and the analysis goals.

A Variant Call Format (VCF) file is a text file that contains information about genetic variants identified in a sample or population. The VCF format is widely used in genomics research and is designed to be human-readable and easy to parse by computational tools.

A VCF file typically contains information about the following aspects of each variant:

Chromosome and position: The chromosome and genomic position where the variant is located.

ID: A unique identifier for the variant, if available.

Reference and alternate alleles: The nucleotide(s) that are present in the reference genome at the variant position, as well as the nucleotide(s) that are observed in the sample at that position.

Quality score: A measure of the confidence in the variant call, typically based on factors such as sequencing depth, base quality, and mapping quality.

Filter status: A flag indicating whether the variant passed or failed certain quality control filters.

Allele frequency: The frequency of the alternate allele in the sample or population, if available.

Annotation: Additional information about the variant, such as its functional consequences (e.g. missense, frameshift), predicted impact on protein structure or function, or overlap with known genetic features (e.g. exons, regulatory regions).

By analyzing the information in a VCF file, researchers can gain insights into the genetic variation present in a sample or population, as well as its potential functional and phenotypic consequences. This information can be used to identify disease-causing variants, study population genetics, and inform personalized medicine approaches.

Note: The output of this part is available in PartC-results.