Project of Bioinformatic

Mahir SELEK
2041295

Academic year 2022/2023

University of Padua

1 Introduction

Given a sequence of a bacterium called ”Lactobacillius cosei” long 3.079.196 bases, resequence a similar bacterium called ”Loct.sp” using a mate pair reads. Mate pair reads is a technique that allows you to obtain paired-end reads with long inserts.

Mate pair sequencing involves generating long-insert paired-end DNA libraries useful for a number of sequencing applications, including:

- De novo sequencing
- Genome finishing
- Structural variant detection
- Identification of complex genomic rearrangements

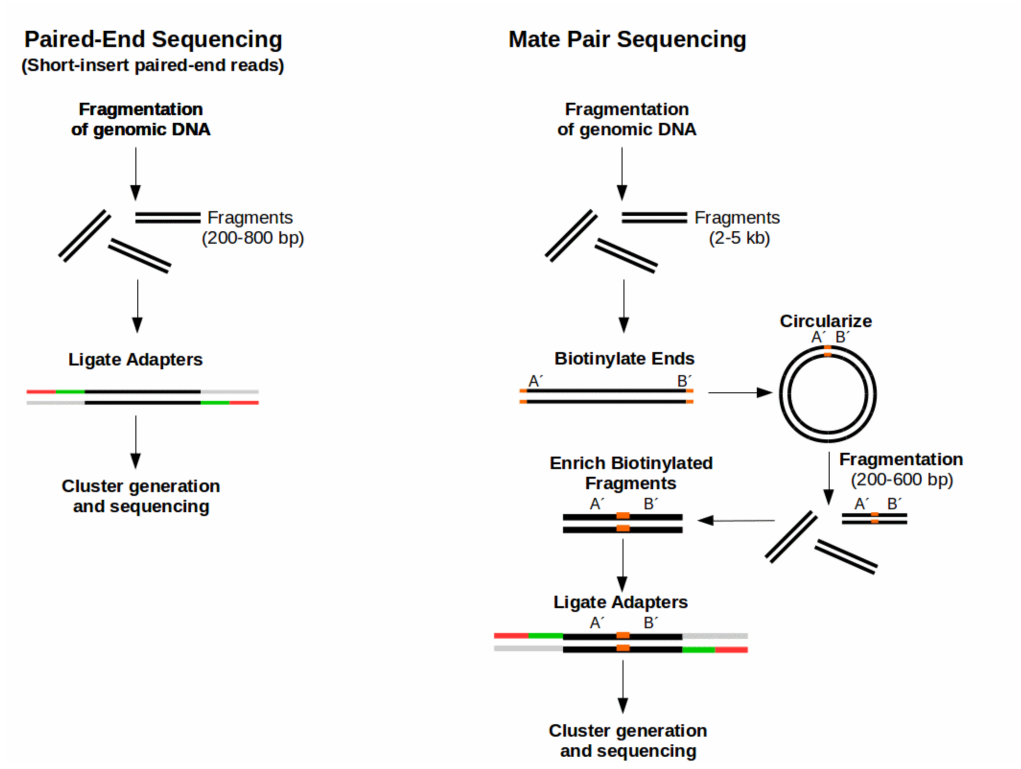

Paired-End Sequencing
(Short-insert paired-end reads)
Fragmentation of genomic DNA
Fragments (200-800 bp)
Ligate Adapters
Cluster generation and sequencing
Mate Pair Sequencing
Fragmentation of genomic DNA
Fragments (2-5 kb)
Biotinylate Ends
A´
B´
Circularize
A´ B´
Fragmentation (200-600 bp)
A´ B´
Enrich Biotinylated Fragments
A´ B´
Ligate Adapters
A´ B´
Cluster generation and sequencing

Figure 1: Mate pair sequencing

In this case we’ll use the mate pair to help us to detect in the sequence genome some structural variation.

2 Part 1 – Prerequisities & Visualizations

2.1 Step 1

Download the reference genome and the zip file with the fastq file. The format fastq represent the read and its quality in this way:

```
@SEQ ID
GATTTGGGGTTCAAAGCAGTATCGATCAAATAGTAAATCCATTTGTTCAACTCACAGTTT
+
!''*((((***+))%%%++)(%%%%).1***-+*''))**55CCF>>>>>>CCCCCCC65
```

- fist row: name of sequence
- second row: read
- third row: quality of the read (quality scores from 0 to 93 are encoded using ASCII 33 to 126)

2.2 Step 2

Installing BWA on my computer:

- First of all, I downloaded the file from terminal: sudo apt-get all install -y bwa
- I extracted the directory and inside this directory I typed:

  make

- At the end I executed

  bwa

2.3 Step 3

Use BWA to align the Illumina reads on the reference genome:

- I indexed the fasta file with the command:

  ```
  bwa index Lactobacillus_casei_genome.fasta
  ```

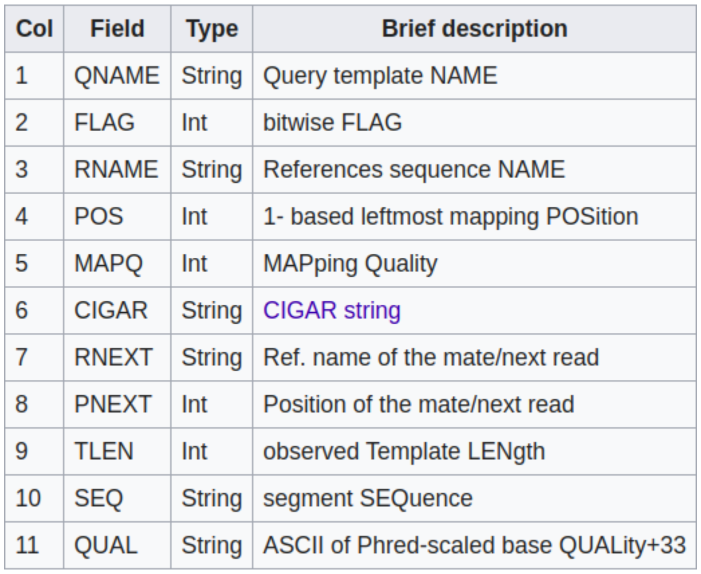
  A fasta file is text-based format for representing sequences of amino acid:

  ```
  >genome 3079196 bp
  TATATAGGAATGTCCGTCATCACTCGTCAGCATAGCTATAAGCTGGTTTGCTACCGGACATAA
  AGGGGCATGAATGCCCAATTTAGAGGAACTTTGGGCTTACCTGAATGATAAATTCCGTGAAGA
  CCGGTCGGTTACAGCACATGGATACAGACAGCAAAACCCGTCAAATTAACCAAAGATAAACTC
  AAGTCCCAGCTTCTTTACACAAAGCTTATTGGGAAAAGAATCTCGTGACCAAGGTAGTCGAAG
  CGAATTCGCGCAACTTGAAGTTGATCCCGTCATTATGACCAAGGATGAGTTGCAGCCAGCTCC
  GATCAACGTCCTGCGGTTGAAGAAGACGATCAAAATCTGACTTTTAAGGCAAAAACCCATTTA
  AATATACGTTTGACCGCTTCGTGATTGGTAAGGGCAACCAAATGGCCCACGCGGCTGCTTTGG
  TGAAGCCCCTGGGACAACATACAACCCTCTGTTTATTTATGGTGGCGTCGGTCTT........
  ```

- After I aligned the read using

```
bwa mem Lactobacillus_casei_genome.fasta lact_sp.read1.fastq
lact_sp.read2.fastq > lact.sam
```

I used the pipe for saving the output produced by the command in to the file: lact.sam. In the SAM format each reads have usually at least eleven fields that describe position, name, quality, length, etc. An example is present in Figure 2.



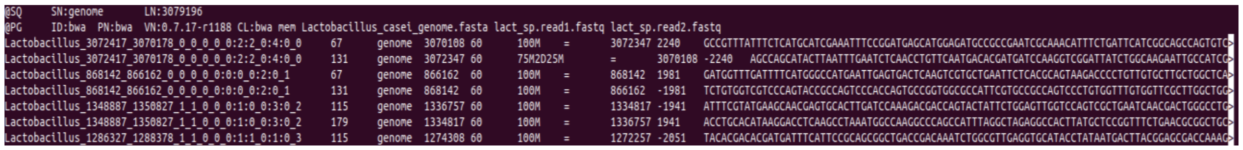
Col	Field	Type	Brief description
1	QNAME	String	Query template NAME
2	FLAG	Int	bitwise FLAG
3	RNAME	String	References sequence NAME
4	POS	Int	1- based leftmost mapping POSition
5	MAPQ	Int	MAPping Quality
6	CIGAR	String	CIGAR string
7	RNEXT	String	Ref. name of the mate/next read
8	PNEXT	Int	Position of the mate/next read
9	TLEN	Int	observed Template LENgth
10	SEQ	String	segment SEQuence
11	QUAL	String	ASCII of Phred-scaled base QUALity+33

Figure 2: Sequence alignment

For visualize the SAM file I can use:

```
less -S lact.sam
```

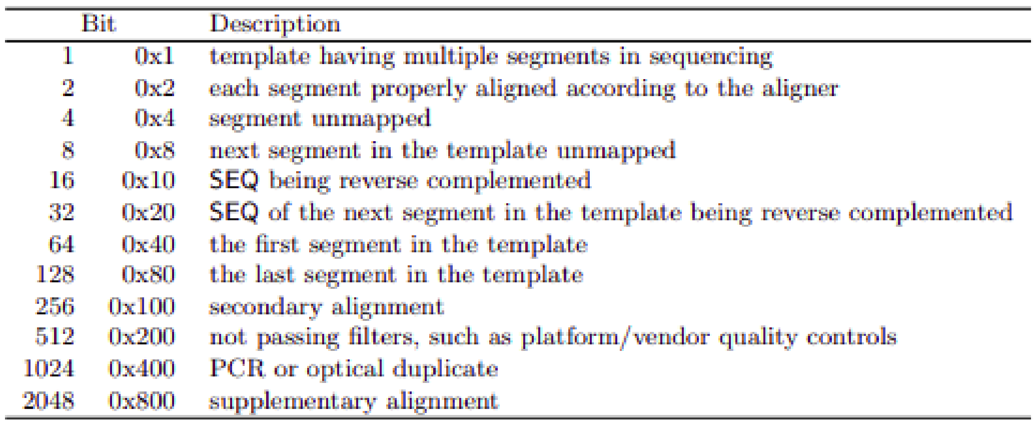
In Figure 3 is possible view the result.



```
@SQ     SN:genome       LN:3079196
@PG     ID:bwa  PN:bwa  VN:0.7.17-r1188 CL:bwa mem Lactobacillus_casei_genome.fasta lact_sp.read1.fastq lact_sp.read2.fastq
Lactobacillus_3072417_3070178_0_0_0_0_0:2:2_0:4:0_0     67      genome  3070108 60      100M    =       3072347 2240    GCCGTTTATTTCTCATGCATCGAAATTTCCGGATGAGCATGGAGATGCCGCCGAATCGCAAACATTTCTGATTCATCGGCAGCCAGTGTC>
Lactobacillus_3072417_3070178_0_0_0_0_0:2:2_0:4:0_0     131     genome  3072347 60      75M2D25M        =       3070108 -2240   AGCCAGCATACTTAATTTGAATCTCAACCTGTTCAATGACACGATGATCCAAGGTCGGATTATCTGGCAAGAATTGCCATCG>
Lactobacillus_868142_866162_0_0_0_0_0:0:0_0:2:0_1       67      genome  866162  60      100M    =       868142  1981    GATGGTTTGATTTTCATGGGCCATGAATTGAGTGACTCAAGTCGTGCTGAATTCTCACGCAGTAAGACCCCTGTTGTGCTTGCTGGCTCA>
Lactobacillus_868142_866162_0_0_0_0_0:0:0_0:2:0_1       131     genome  868142  60      100M    =       866162  -1981   TCTGTGGTCGTCCCAGTACCGCCAGTCCCACCAGTGCCGGTGGCGCCATTCGTGCCGCCAGTCCCTGTGGTTTGTGGTTCGCTTGGCTGG>
Lactobacillus_1348887_1350827_1_1_0_0_0:1:0_0:3:0_2     115     genome  1336757 60      100M    =       1334817 -1941   ATTTCGTATGAAGCAACGAGTGCACTTGATCCAAAGACGACCAGTACTATTCTGGAGTTGGTCCAGTCGCTGAATCAACGACTGGGCCTG>
Lactobacillus_1348887_1350827_1_1_0_0_0:1:0_0:3:0_2     179     genome  1334817 60      100M    =       1336757 1941    ACCTGCACATAAGGACCTCAAGCCTAAATGGCCAAGGCCCAGCCATTTAGGCTAGAGGCCACTTATGCTCCGGTTTCTGAACGCGGCTGC>
Lactobacillus_1286327_1288378_1_1_0_0_0:1:1_0:1:0_3     115     genome  1274308 60      100M    =       1272257 -2051   TACACGACACGATGATTTCATTCCGCAGCGGCTGACCGACAAATCTGGCGTTGAGGTGCATACCTATAATGACTTACGGAGCGACCAAAG>
```

Figure 3: Print of SAM file

One of the most important field is the flag, depending of bit that are set I can obtain a lot of information. A focus of the flag is on the Figure 4.



Bit		Description
1	0x1	template having multiple segments in sequencing
2	0x2	each segment properly aligned according to the aligner
4	0x4	segment unmapped
8	0x8	next segment in the template unmapped
16	0x10	SEQ being reverse complemented
32	0x20	SEQ of the next segment in the template being reverse complemented
64	0x40	the first segment in the template
128	0x80	the last segment in the template
256	0x100	secondary alignment
512	0x200	not passing filters, such as platform/vendor quality controls
1024	0x400	PCR or optical duplicate
2048	0x800	supplementary alignment

Figure 4: Flag field of SAM file

2.4 Step 4

Process the sam file obtained by the BWA analysis to make a bam file:

- For doing that I need to install the program Samtools, for doing that I can use the commands on terminal:

```
sudo apt-get all install -y samtools
make
```

- Now I can convert the file, BAM format is a binary version of SAM, more easy to handle for the programs.

```
Samtools view −S −b lact.sam > lact.bam
```

2.5 Step 5

Sort and index the bam file:
For doing that I typed:

- `samtools sort sample.bam −o lact.sorted.bam`

- `samtools index lact.sorted.bam`

2.6 Step 6

Visualize the coverage and mate pairs on the IGV:

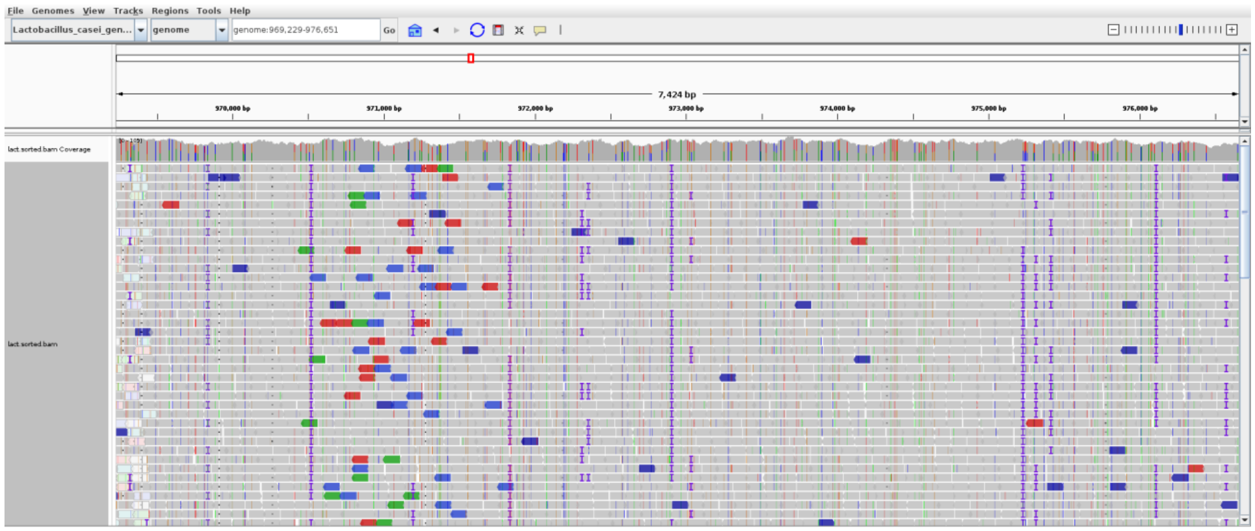
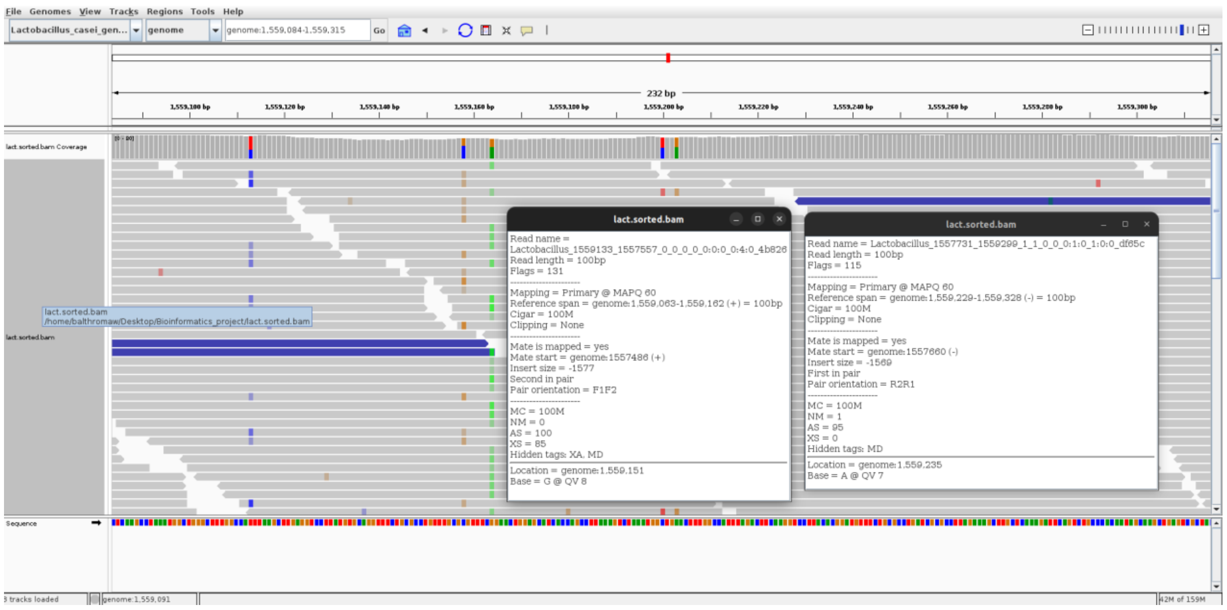


Figure 5: lact.sorted.bam on IGV

2.7 Step 7

Manually find and comment any “anomalous” pair of mates:

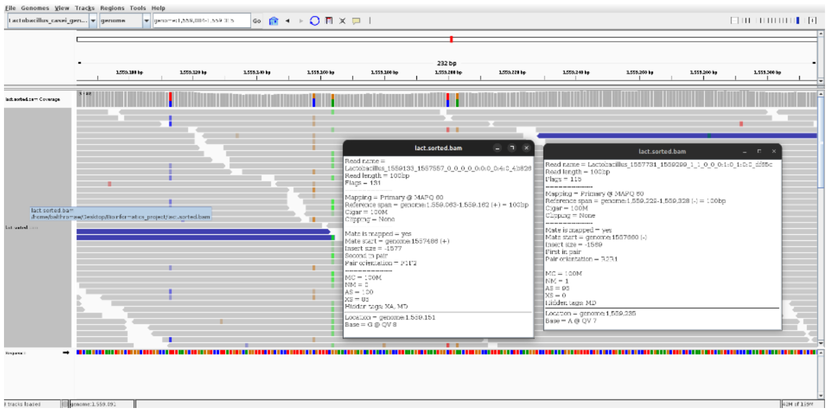

File Genomes View Tracks Regions Tools Help
Lactobacillus_casei_gen...
genome
genome:1,559,084-1,559,315
Go
232 bp
lact.sorted.bam Coverage
lact.sorted.bam
lact.sorted.bam
/home/balthromaw/Desktop/Bioinformatics_project/lact.sorted.bam
Sequence
3 tracks loaded
genome:1,559,091
42M of 159M

```
lact.sorted.bam
Read name =
Lactobacillus_1559133_1557557_0_0_0_0_0:0:0_0:4:0_4b826
Read length = 100bp
Flags = 131
----------------------
Mapping = Primary @ MAPQ 60
Reference span = genome:1,559,063-1,559,162 (+) = 100bp
Cigar = 100M
Clipping = None
----------------------
Mate is mapped = yes
Mate start = genome:1557486 (+)
Insert size = -1577
Second in pair
Pair orientation = F1F2
----------------------
MC = 100M
NM = 0
AS = 100
XS = 85
Hidden tags: XA, MD
Location = genome:1,559,151
Base = G @ QV 8
```

```
lact.sorted.bam
Read name = Lactobacillus_1557731_1559299_1_1_0_0_0:1:0_1:0:0_df65c
Read length = 100bp
Flags = 115
----------------------
Mapping = Primary @ MAPQ 60
Reference span = genome:1,559,229-1,559,328 (-) = 100bp
Cigar = 100M
Clipping = None
----------------------
Mate is mapped = yes
Mate start = genome:1557660 (-)
Insert size = -1569
First in pair
Pair orientation = R2R1
----------------------
MC = 100M
NM = 1
AS = 95
XS = 0
Hidden tags: MD
Location = genome:1,559,235
Base = A @ QV 7
```

Figure 6: lact.sorted.bam on IGV

- The color blue indicates the mate pair with less length than expected. In this case length is equal to 1352, so less then 200



lact.sorted.bam

Read name = Lactobacillus_1559133_1557557_0_0_0_0_0:0:0_0:4:0_4b828
Read length = 100bp
Flags = 131

Mapping = Primary @ MAPQ 60
Reference span = genome:1,559,063-1,559,162 (+) = 100bp
Cigar = 100M
Clipping = None

Mate is mapped = yes
Mate start = genome:1557488 (+)
Insert size = -1577
Second in pair
Pair orientation = F1F2

MC = 100M
NM = 0
AS = 100
XS = 85
Hidden tags: XA, MD
Location = genome:1,559,151
Base = G @ QV 8

lact.sorted.bam

Read name = Lactobacillus_1557731_1559299_1_1_0_0_0:1:0_1:0:0_df85c
Read length = 100bp
Flags = 115

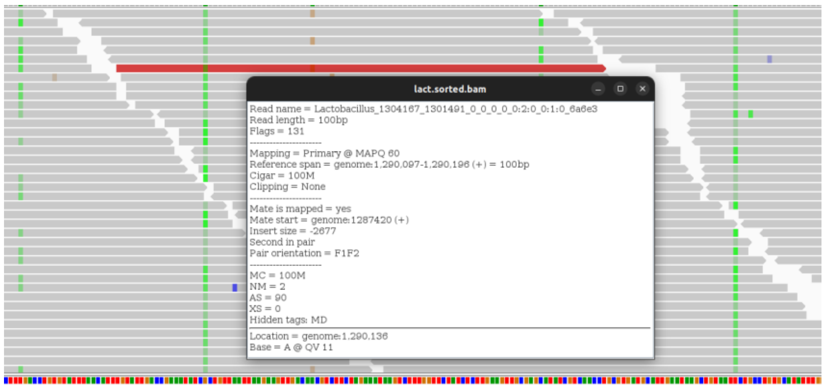
Mapping = Primary @ MAPQ 60
Reference span = genome:1,559,229-1,559,328 (-) = 100bp
Cigar = 100M
Clipping = None

Mate is mapped = yes
Mate start = genome:1557660 (-)
Insert size = -1569
First in pair
Pair orientation = R2R1

MC = 100M
NM = 1
AS = 95
XS = 0
Hidden tags: MD
Location = genome:1,559,235
Base = A @ QV 7

Figure 7: Short reads on IGV

- The color red indicate the mate pair with more length than expected. In this case length is equal to 1352, so less then 2764.



lact.sorted.bam

Read name = Lactobacillus_1304167_1301491_0_0_0_0_0:2:0_0:1:0_6a6e3
Read length = 100bp
Flags = 131

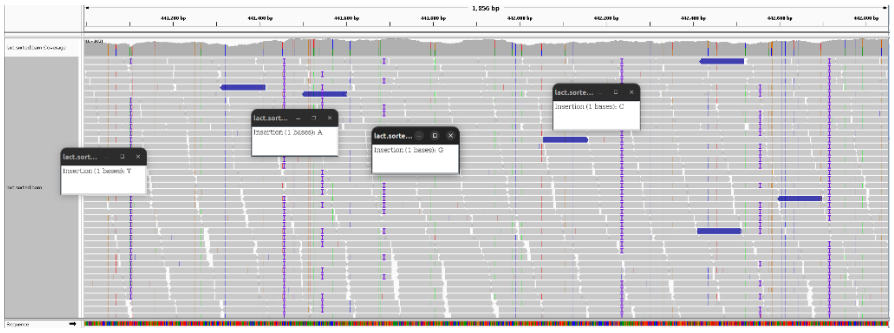
Mapping = Primary @ MAPQ 60
Reference span = genome:1,290,097-1,290,196 (+) = 100bp
Cigar = 100M
Clipping = None

Mate is mapped = yes
Mate start = genome:1287420 (+)
Insert size = -2677
Second in pair
Pair orientation = F1F2

MC = 100M
NM = 2
AS = 90
XS = 0
Hidden tags: MD
Location = genome:1,290,136
Base = A @ QV 11

Figure 8: Long reads on IGV

- The "I" represent the point that had an insertion, in this case we can see all different 4 bases A-T-G-C



Insertion (1 bases): T

Insertion (1 bases): A

Insertion (1 bases): G

Insertion (1 bases): C

Figure 9: Insertions on IGV

- The ”-” represent the point that had a deletion.

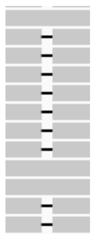


Figure 10: Deletion on IGV

- The vertical colored lines represents the point where there is a wrong base

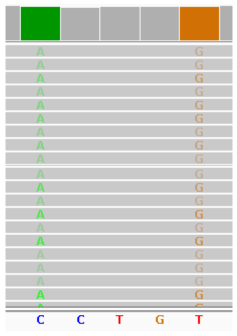


Figure 11: Error alignment on IGV

3 Part 2 – Sequence Coverage

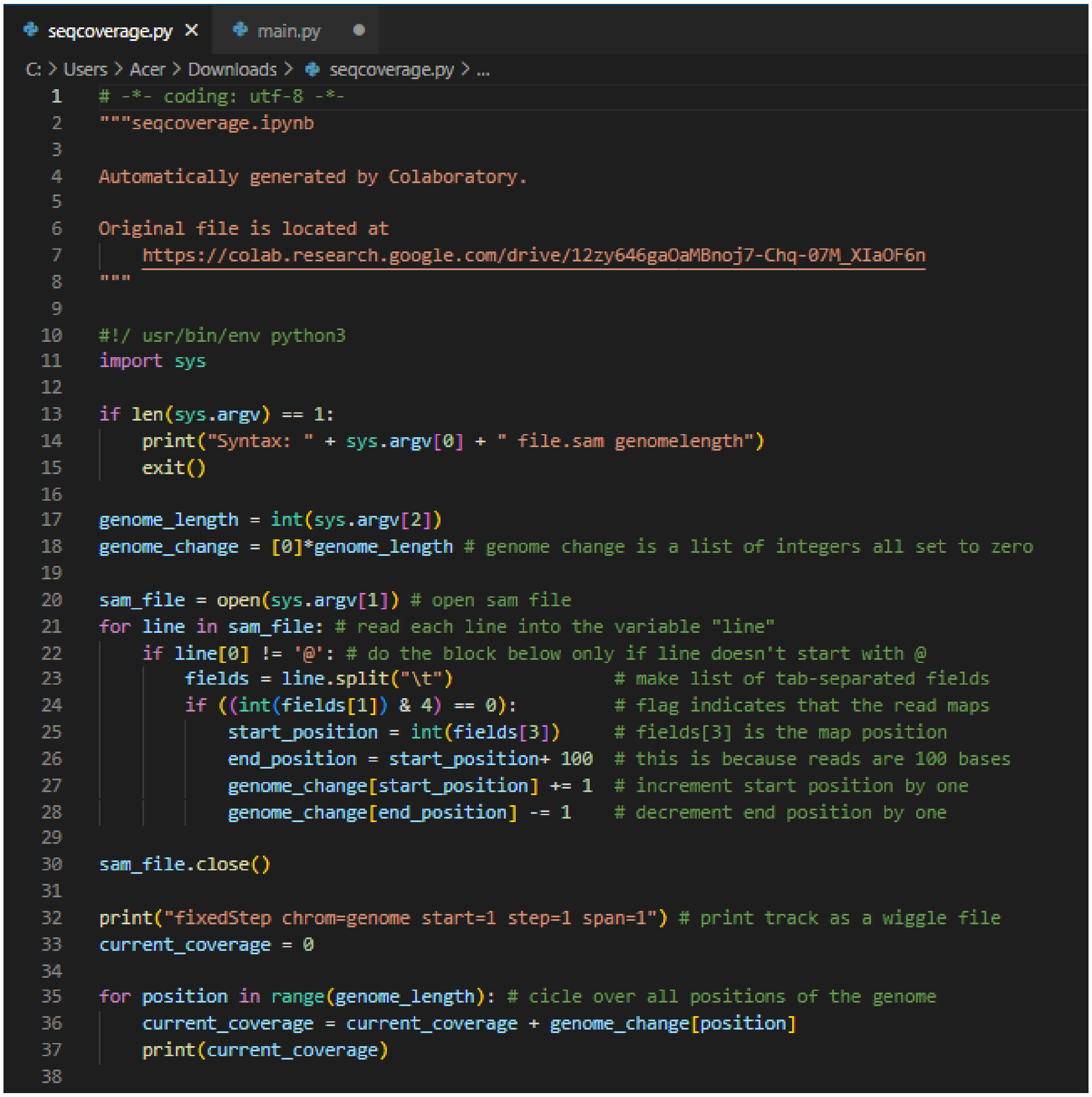
3.1 Track1

Implement in the language of your choice a program to create a wig file with the sequence coverage. In the Moodle platform you can find C implementation (SeqCoverage.c) and a Python implementation (seqcoverage.py). Run the program to create a seqcov.wig file. Then load the file on IGV. Load on IGV also the lact.sorted.bam file and compare the sequence coverage.

3.1.1 Solution

I choose to use the Python implementation and i downloaded the file seqcoverage.py. This python code creates a track that is wrote in a wig file that represents the sequence coverage. The sequence is defined by the number of sample read bases that align to a specific locus on the reference genome.

I downloaded the file, that’s the one present in the Figure12.



seqcoverage.py × main.py ●

C: > Users > Acer > Downloads > seqcoverage.py > ...

```python
# -*- coding: utf-8 -*-
"""seqcoverage.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/12zy646gaOaMBnoj7-Chq-07M_XIaOF6n
"""

#!/ usr/bin/env python3
import sys

if len(sys.argv) == 1:
    print("Syntax: " + sys.argv[0] + " file.sam genomelength")
    exit()

genome_length = int(sys.argv[2])
genome_change = [0]*genome_length # genome change is a list of integers all set to zero

sam_file = open(sys.argv[1]) # open sam file
for line in sam_file: # read each line into the variable "line"
    if line[0] != '@': # do the block below only if line doesn't start with @
        fields = line.split("\t")                # make list of tab-separated fields
        if ((int(fields[1]) & 4) == 0):          # flag indicates that the read maps
            start_position = int(fields[3])      # fields[3] is the map position
            end_position = start_position+ 100   # this is because reads are 100 bases
            genome_change[start_position] += 1   # increment start position by one
            genome_change[end_position] -= 1     # decrement end position by one

sam_file.close()

print("fixedStep chrom=genome start=1 step=1 span=1") # print track as a wiggle file
current_coverage = 0

for position in range(genome_length): # cicle over all positions of the genome
    current_coverage = current_coverage + genome_change[position]
    print(current_coverage)

```

Figure 12: seqcoverage.py

For executing it I typed:

```
python3 seqcoverage.py lact.sam 30791969 > seqcoverage.wig
```

- Input (two parameters):

  - The SAM file: lact.bam
  - Genome length: 30791969.

- Output:
  - The output of the code is printed in a wig file, using the pipe.

Structure of the program:

- The first part checks if the parameter needed are passed correctly and it creates a vector of the size of genome, for saving in each position the value of the track.

```
#!/ usr/bin/env python3
import sys

if len(sys.argv) == 1:
    print("Syntax: " + sys.argv[0] + " file.sam genomelength")
    exit()

genome_length = int(sys.argv[2])
genome_change = [0]*genome_length # genome change is a list of integers all set to zero
```

- The program opens the SAM file and checks all the reads present on the file ignoring the one that starts with @. It puts the reads in vector, and checks the flag's field. If the bit 0x04 is unset that means the reads is mapped and the program increments of 1 the field of vector corresponding of the start position and decrements of 1 the field of the end position.

```
sam_file = open(sys.argv[1]) # open sam file
for line in sam_file: # read each line into the variable "line"
    if line[0] != '@': # do the block below only if line doesn't start with @
        fields = line.split("\t")                # make list of tab-separated fields
        if ((int(fields[1]) & 4) == 0):          # flag indicates that the read maps
            start_position = int(fields[3])      # fields[3] is the map position
            end_position = start_position+ 100   # this is because reads are 100 bases
            genome_change[start_position] += 1   # increment start position by one
            genome_change[end_position] -= 1     # decrement end position by one

sam_file.close()
```

- At the end it prints for each position the sum of all reads that are present.

```
print("fixedStep chrom=genome start=1 step=1 span=1") # print track as a wiggle file
current_coverage = 0

for position in range(genome_length): # cicle over all positions of the genome
    current_coverage = current_coverage + genome_change[position]
    print(current_coverage)
```

I load the wig file on the program IGV for comparing the track with the lact.sorted.bam (the BAM file is a binary version of SAM file).

The Figure 13 show this comparison, where we can clearly see that the value assumed by the track in each point of genome is proportional to the number of read present on the alignment, like for definition. The track is practically the same compared to the coverage given by the program, but is more easy to manage and permits the global view of all genome.

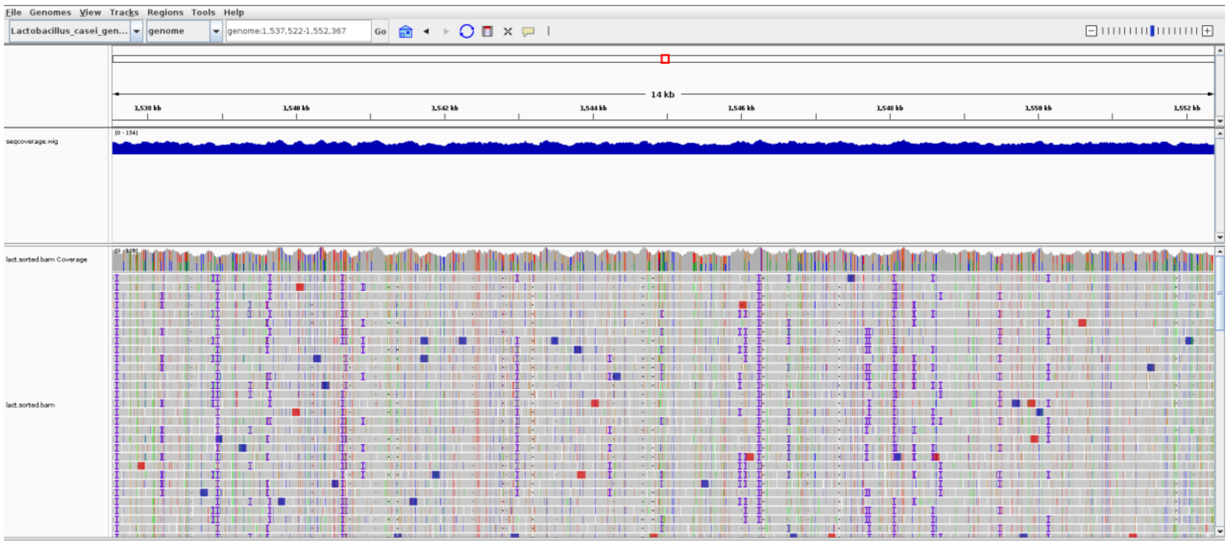


Figure 13: Sequence coverage and Mate-Pairs Alignment

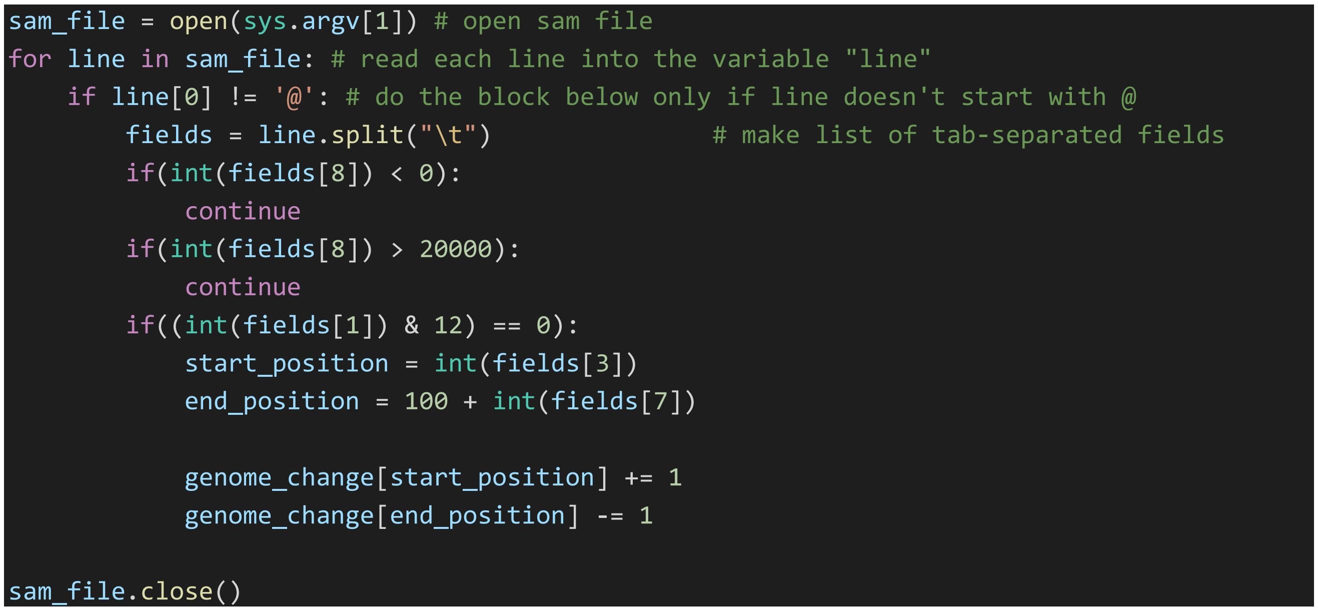
4 Part 3 - TRACK

4.1 Physical Coverage

Modify the program to make a track with the physical coverage. Remember: the physical coverage of a genomic position is given by the number of fragment laying on that position rather than the number of reads. You should consider that some reads may map on multiple positions and that the fragments of DNA where selected to be relatively homogeneous in size. So if a fragment has an implausible length it should not be considered.

4.1.1 Solution

In this case the definition of physical coverage is already given. The program that calculates the track of physical coverage is on Figure 14.



```python
sam_file = open(sys.argv[1]) # open sam file
for line in sam_file: # read each line into the variable "line"
    if line[0] != '@': # do the block below only if line doesn't start with @
        fields = line.split("\t")                # make list of tab-separated fields
        if(int(fields[8]) < 0):
            continue
        if(int(fields[8]) > 20000):
            continue
        if((int(fields[1]) & 12) == 0):
            start_position = int(fields[3])
            end_position = 100 + int(fields[7])

            genome_change[start_position] += 1
            genome_change[end_position] -= 1

sam_file.close()
```

Figure 14: physical_coverage.py

The program phycoverage.py is like the previous, but have some difference:

- We need to found the segment for this reason I didn’t take in consideration one of two reads, but only the one that have positive length. (First if block)

- It ignores the segment with a length greater then 20000. (Second if block)

- In this case the program verifies if the bit 0x4 and 0x8 is unset for checking if the fragment is mapped. (Third if block)

For execution the code needs the same parameter of previous. After that i load the wig file on IGV an in Figure 16 i can see the result.

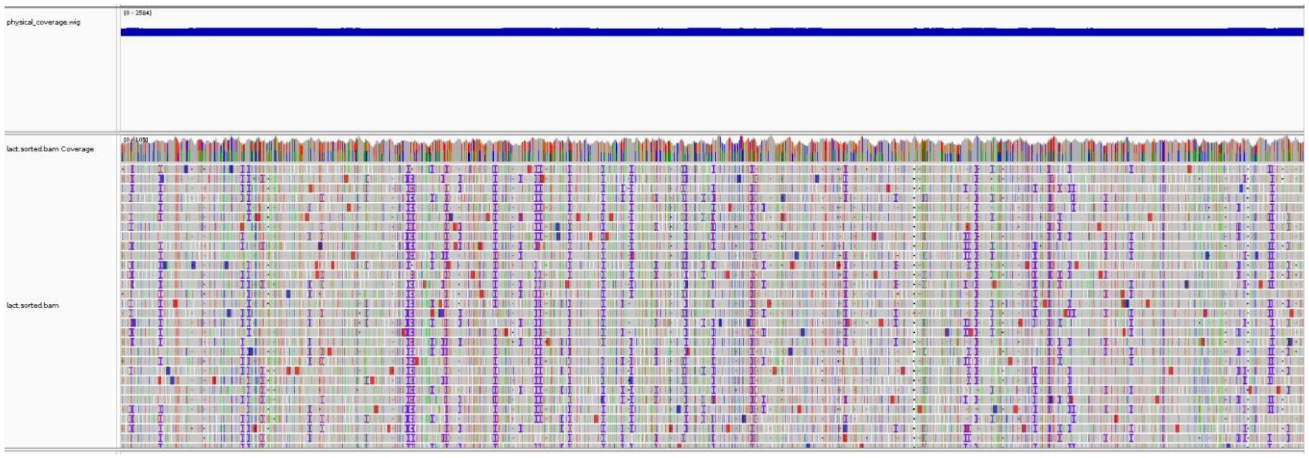
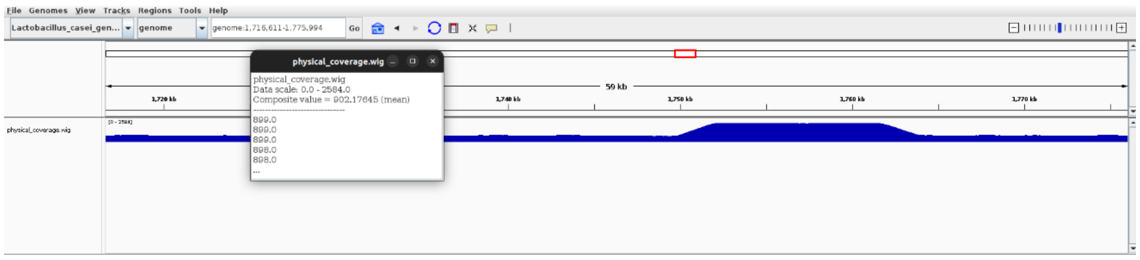


Figure 15: Physical coverage and Lact.sorted.bam

I can see clearly that the physical coverage is so different from the sequence coverage, the values that print are more stable than the values assumed by the coverage. But in this case we are considering a too short number of bases for find some information. The problem is 'lact.sorted.bam' is too big for supporting all the bases in a single view, but I can use the track in a global way comparing the track with the other, that is obtain by the previous step.

Also there is peak as we can see below. This means that maybe there is a deletion in my reference genome. And the value is around 1000 as we expected.


File Genomes View Tracks Regions Tools Help
Lactobacillus_casei_gen...
genome
genome:1,716,611-1,775,994
Go
59 kb
1,720 kb
1,740 kb
1,750 kb
1,760 kb
1,770 kb
physical_coverage.wig
[0 - 2584]

```
physical_coverage.wig
Data scale: 0.0 - 2584.0
Composite value = 902.17645 (mean)
-------------------------------
899.0
899.0
899.0
898.0
898.0
...
```

Figure 16: Physical coverage

4.2 Average Length

Create a track with the average length of the fragments covering each genomic position. You can start from the physical coverage program and for every position you should also calculate the sum of the fragment length. Then when you print the track, for every position you should divide the sum of the fragment lengths by the physical coverage.

4.2.1 Solution

In this case I follow the instructions given by the track, the result code is that:

```
genome_length = int(sys.argv[2])
genome_change = [0]*genome_length # genome change is a list of integers all set to zero
genome_sum = [0]*genome_length

sam_file = open(sys.argv[1]) # open sam file
for line in sam_file: # read each line into the variable "line"
    if line[0] != '@': # do the block below only if line doesn't start with @
        fields = line.split("\t")                # make list of tab-separated fields
        if(int(fields[8]) < 0):
            continue
        if(int(fields[8]) > 20000):
            continue
        if((int(fields[1]) & 12) == 0):  #this means both will be mapping
            start_position = int(fields[3])
            end_position = 100 + int(fields[7])

            genome_sum[start_position] += abs(int(fields[8]))
            genome_sum[end_position] -= abs(int(fields[8]))

            genome_change[start_position] += 1
            genome_change[end_position] -= 1

sam_file.close()
```

The result product by this code is available at Figure 18. We can see clearly that the average fragment length in most of the case is constant, but in some point is grow, like when we have a point where the sequence coverage has value 0.

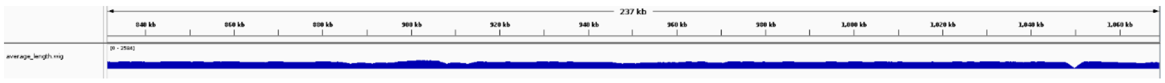


Figure 17: Average length of the fragment

It’s possible to notice that the average length of the fragment increases and decreases when are present some structural variation, like: insert, deletion or inversion.

4.3 Indicating Repeated Sequences

Create a track with the coverage of ”multiple” reads. The multiple reads are those mapping in more than one genomic position. This information can be found in the sam file.

4.3.1 Solution

The definition of multiple reads is already given in the track request, fist of all we need to do some modification, BWA uses a special annotation with the label XA that indicates the read that are aligned more that one time, for selecting only the read that have present the XA symbol i can use the command:

```
grep ”XA:” lact.sam | > lactXA.sam
```

The pipe is used for saving the new SAM file. After that i worked on the new SAM file, the fields[16] has data that represent the number of time the read is mapped in the position. The data contained has the following form:

```
XA:Z:genome,+638549,100M,2;genome,+411509,100M,2;
```

This read informs that there are two extra alignments on genome, at position 638549 and 411509, both with a cigar 100M and NM 2. I use this information for producing the track, consider some exception in case of IndexError.

```python
sam_file = open(sys.argv[1]) # open sam file
for line in sam_file: # read each line into the variable "line"
    if line[0] != '@': # do the block below only if line doesn't start with @
        continue

    fields = line.split("\t")               # make list of tab-separated fields
    try:
        multiple = fields[16]
    except IndexError:
        continue
    mul = multiple.split(";")

    for i in range(len(mul) - 1):
        array = mul[i].split(",")
        genome_change[abs(int(array[1]))] += 1
        genome_change[abs(int(array[1])) + 100] -= 1

sam_file.close()
```

The resulting track of the code is available at Figure 18.

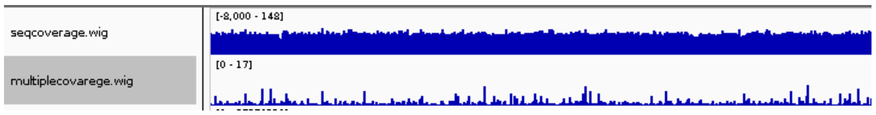


Figure 18: Multiple reads coverage

4.4 How to find insertions and deletion?

I can create a track reporting the presence of “H” or “S” in the CIGAR field. I can have a look at the sam specifications. Deletions are easy to find because there will be a missing coverage on the reference genome.

4.4.1 Solution

When it’s present ”S” that’s indicate a soft clipping, otherwise if is present ”H” that’s mean that is present a hard clipping. The soft clipping is when bases in 5’ and 3’ of the read are not part of the alignment. In the hard clipping bases in 5’ and 3’ of the read are NOT part of the alignment AND those bases have been removed from the read sequence in the BAM file.

I produced the track using this code below, I use the function find to search the symbol ”H” or ”S”

```
sam_file = open(sys.argv[1]) # open sam file
for line in sam_file: # read each line into the variable "line"
    if line[0] != '@': # do the block below only if line doesn't start with @
        continue
    fields = line.split("\t")                # make list of tab-separated fields

    if ((int(fields[1]) & 4) == 0):
        CIGAR = (fields[5])
        if ((CIGAR).find("S") == 1) or ((CIGAR).find("H") == 1):
            start_position = int(fields[3])
            end_position = 100 + int(fields[3])

            genome_change[start_position] += 1
            genome_change[end_position] -= 1

sam_file.close()
```

The result produced by the code is represent on Figure 19

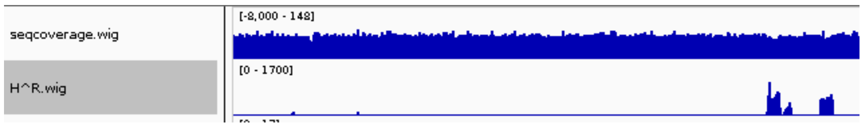


Figure 19: Track of soft and hard clipping

4.5 Detecting Structural Variables – Insertion/Deletion

First, I should define/calculate the relative orientation of reads for every genomic position. To do that I need to find on "bit 16" and “bit 32” of the "flag" which is mentioned in the description of part 15. It will be necessary to detect structural variations (Insertion, Deletion, Inversion). Also, I will use 4 different tracks in the next part (13).

4.5.1 Different Tracks

In this case I need to write 4 different tracks for the 4 kind of orientation that the reads can have:

- **Forward-Forward: The two reads are forward:**

```
fields = line.split("\t")  # make list of tab-separated fields

    #Forward - Forward
    if ((int(fields[1]) & 12) == 0):

            if((int(fields[1]) & 48) == 0): #The both bit are not set, so that mean it will produce the track
for FF.

                start_position = int(fields[3])
                end_position = 100 + int(fields[3])

                genome_change[start_position] += 1
                genome_change[end_position] -= 1
```

- **Forward-Reverse: The read is forward his mate pair is reverse:**

```
fields = line.split("\t")  # make list of tab-separated fields

    #Forward - Reverse
    if ((int(fields[1]) & 12) == 0):

            if((int(fields[1]) & 48) == 32): #The fifth bit is set and forth no, it's the condition for FR.

                start_position = int(fields[3])
                end_position = 100 + int(fields[3])

                genome_change[start_position] += 1
                genome_change[end_position] -= 1
```

- **Reverse-Forward: The read is reverse his mate pair forward:**

```
fields = line.split("\t")  # make list of tab-separated fields

#Reverse - Forward
    if ((int(fields[1]) & 12) == 0):

            if((int(fields[1]) & 48) == 16):
                #The fifth bit is not set and forth yes, in this case the condition will consider only the segment RF.
                start_position = int(fields[3])
                end_position = 100 + int(fields[3])

                genome_change[start_position] += 1
                genome_change[end_position] -= 1
```

- **Reverse-Reverse: The two reads are reverse:**

```
fields = line.split("\t")  # make list of tab-separated fields

#Reverse - Reverse
    if ((int(fields[1]) & 12) == 0):

            if((int(fields[1]) & 48) == 48): #The both bit are set, it'll produce the RR segment.

                start_position = int(fields[3])
                end_position = 100 + int(fields[3])

                genome_change[start_position] += 1
                genome_change[end_position] -= 1
```

In the code I can see that I used like base the code for the sequence coverage. But I added a condition using the information present on the flag. The bit 4 and 5 show the direction of the read in the mate pair. If the fourth bit is set that means the read is reverse, if is not set the read is forward, the same for the fifth but considering that it refers to the direction of the second read of mate pair. If we combine this information as we can use it to get track that I need

The results of this four code can be observe in the Figure 20

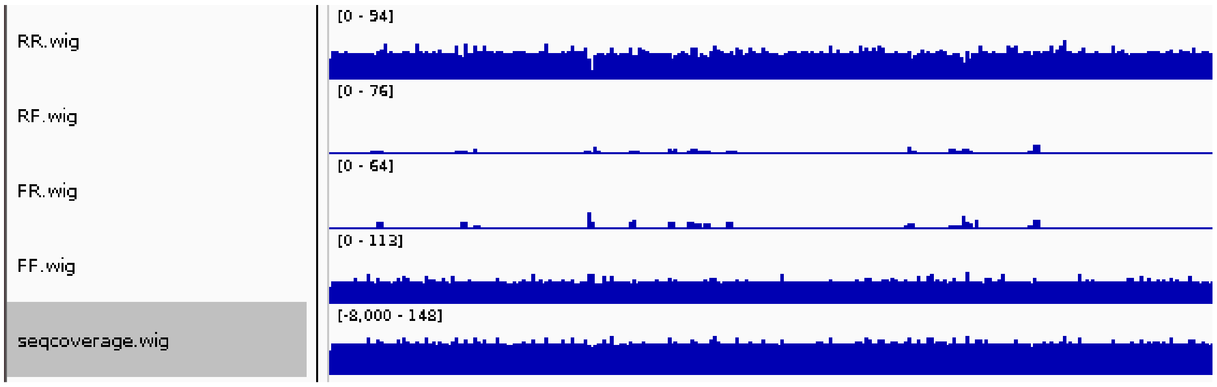


Figure 20: Orientation of fragment and Sequence coverage

4.5.2 Insertion:

There two type of insertion the short one or the long one, in the genome are present both of them:

- Short Insertion:

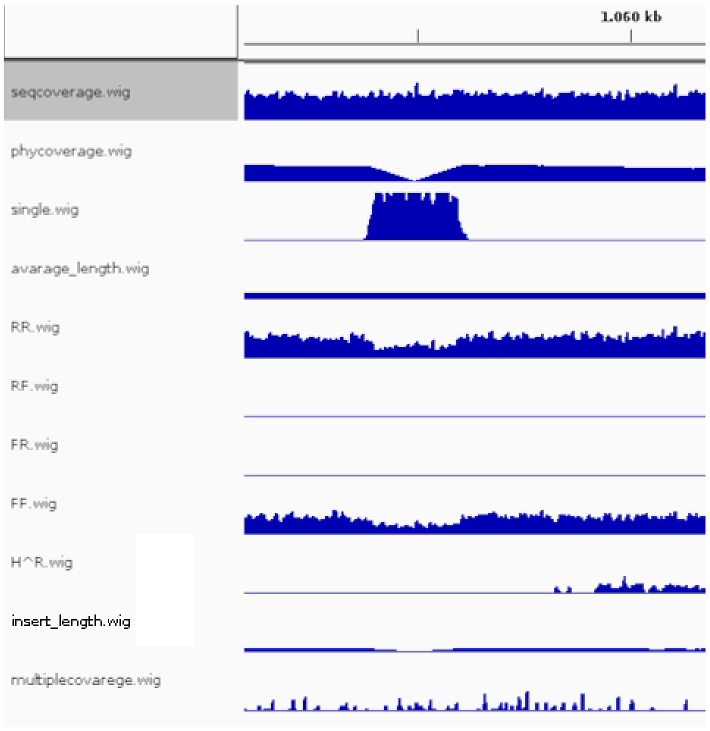


Figure 21: Sample of short insertion

- Long Insertion:

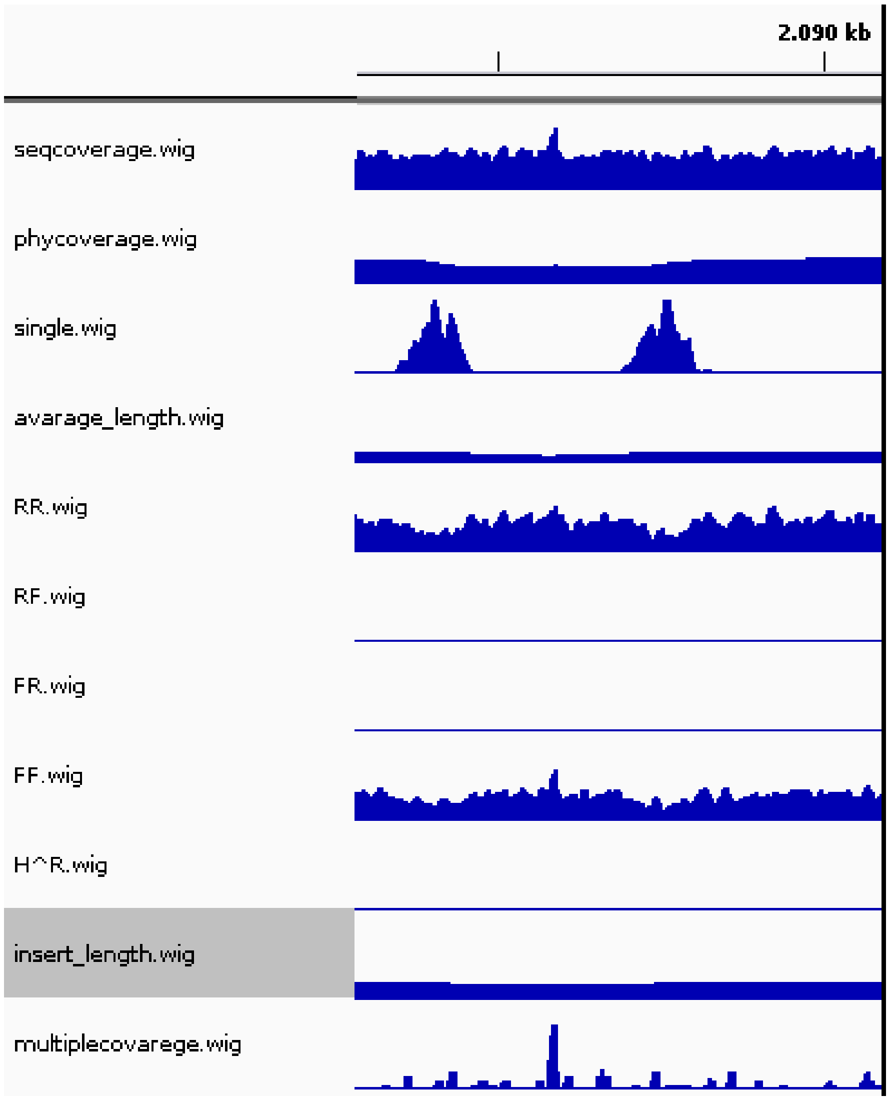


Figure 22: Sample of long insertion

4.5.3 Deletion:

Like for the insertion, for deletion are present two subcategory:

- Short deletion:

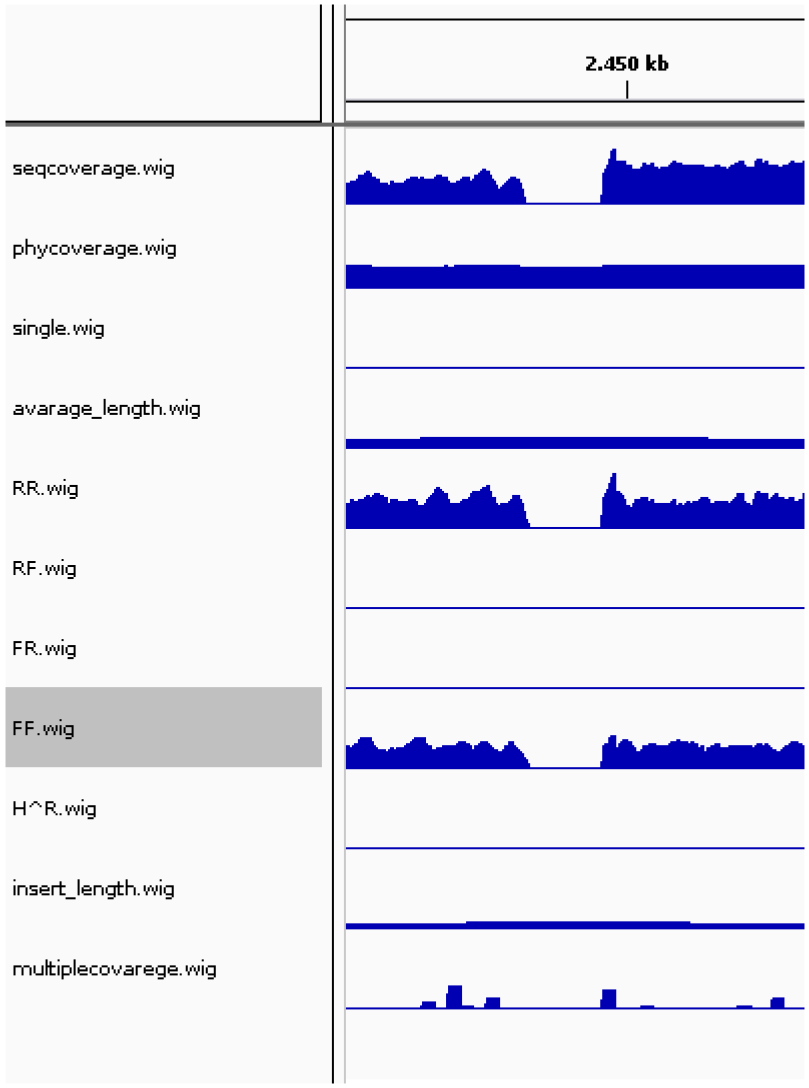


Figure 23: Sample of short deletion

- Long deletion:

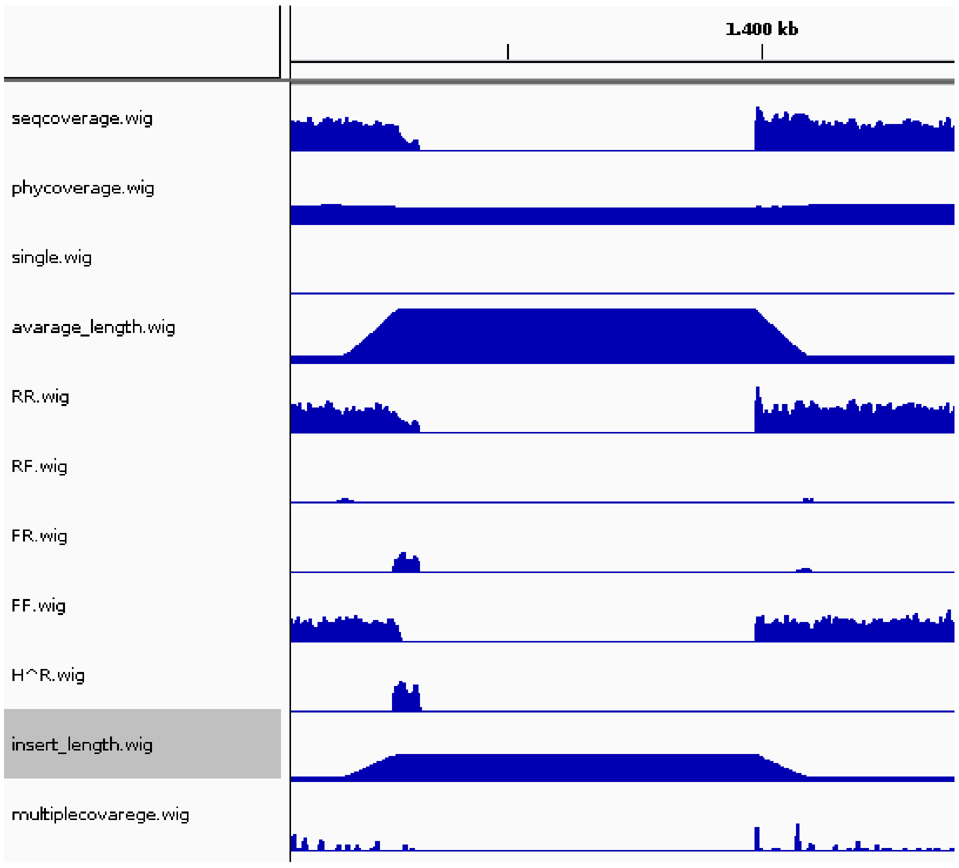


Figure 24: Sample of long deletion

4.6 Detecting Structural Variables – Inversion

Inversion is the last structural variation that we found on the genome.

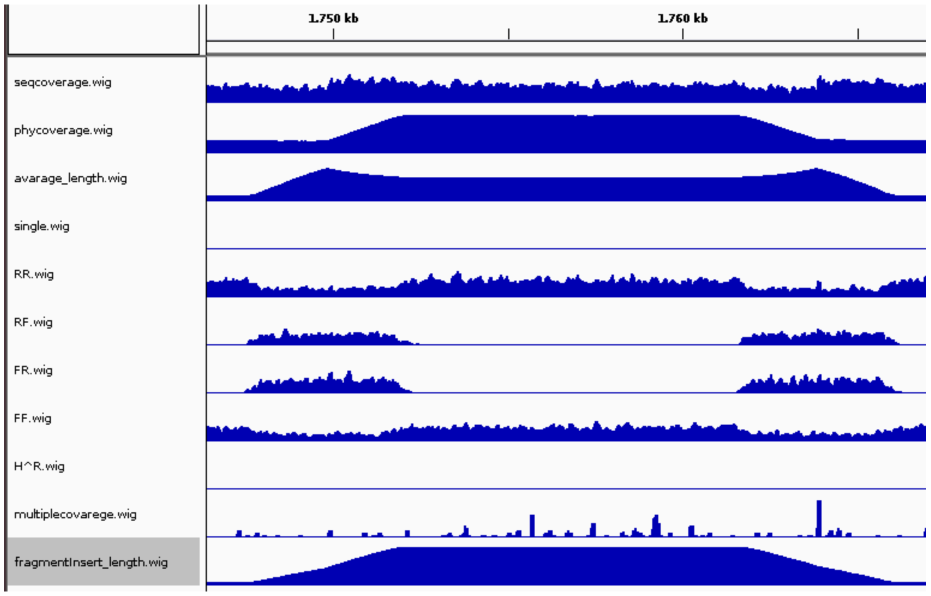


Figure 25: Sample of Inversion

5 CONCLUSION

The structural variations are the assumption for the biology evolution, the transmission genetic information is done with the DNA replication, this process is so accurate but there is a little probability that append an error. This probability is on the order of 10^{-8}. This structural variation are product by an accidental mutation, caused either by external factor. This mutation to be considered evolutionarily relevant need to be fixed, that's mean that are present not only in an individual but in entire population, for this reason for be transmit the mutation need to be relate to a germinal cell. That usually doesn't appendand when in a population can coexist different mutation, this is the definition of Polymorphism (it need to be with a frequency at leaf of 1%, if it's less is defined mutation). The possibly structural variations are called: insertion, deletion and inversion.