Bioinformatics

Practical Report of the course (2018/2019)

Università degli studi di Padova
Master Degree in Computer Science

Latobacillus casei - Genomic variations study

Studente: **Zecchin Giacomo** (1179034)
Docente: **Giorgio Valle**

July 2, 2019

Contents

1 Introduction **1**

2 Project Tasks **2**

3 Part I **3**
3.1 Align reads on reference 3
3.2 Obtein BAM file and sort it 3
3.3 Find anomalous pair of mates through IGV 4
3.3.1 Deletions 5
3.3.2 Insertions 6
3.3.3 Inversions 7

4 Part II **8**
4.1 Implementation choices 8
4.2 Physical Coverage 9
4.3 Sequence Coverage 10
4.4 Genomics insert length statistics 11
4.5 Average physical inserts coverage 12
4.6 Inserts 2-standard deviation distribution 13

5 Part III **14**
5.1 Identify Unique and Multiple mapping reads 14
5.2 Oriented Mates percentage 16
5.3 Single Mates percentage 17

6 Bioinformatics Tool **18**

7 References **19**

1 Introduction

In this report you could find the discussion about the Bioinformatics project, based on the study of a genome of a ***Lactobacillus casei*** that is **3,079,196bp** long.

The aim of the practical is the "re-sequencing" of a similar bacterium that we call **lact.sp**, using mate pairs reads. In particular we want to see if lact.sp has **genomic structural variations**. We suspect that the this genomic file may have small and large **deletions, insertions and inversions**, and we want to find where they are, through a graphical analysis (using appropriate tools).

Moreover, after this evaluation, follow a **statistical analysis** through some algorithms in a certain programming language for the production of results.

For these purposes 3 datasets are preparatory (provided by Professor through the Moodle of the course):

- **Lactobacillus_casei_genome.fasta:** the genome of the bacterium
- **lact_sp.read1.fastq:** the first reads of resequencing (from Illumina)
- **lact_sp.read2.fastq:** the second reads of resequencing (from Illumina)

These last two files contains the reads, the quality of each base of the reads and others useful information that allows the SAM file creation.

Besides this, we need to use some tools in order to align (map) the Illumina reads on the reference genome and then visualize them correctly, before stat. analysis:

- **BWA**: (Burrows-Wheeler Aligner) powerful software package for mapping low-divergent sequences against a large reference genome;
  `http://www.htslib.org/download/`
- **SAMtools**: light and fast tool for manipulation of alignments in the SAM format, including sorting, merging, indexing and generating alignments;
  `https://github.com/lh3/bwa`
- **IGV**: (Integrative Genomics Viewer) is a high-performance visualization tool for interactive exploration of large, integrated genomic datasets.
  `https://software.broadinstitute.org/software/igv/download`

2 Project Tasks

After this brief introduction, the report is divided in 3 PARTS, following the aims of the practical (Part II and III are one continuation of the other):

Part I

1. Use BWA to align the Illumina reads on the reference genome;
2. Process the SAM file obtained by the BWA analysis to make a BAM (binary) file, then sort and index it;
3. Visualize the coverage and mate pairs on the IGV;
4. Manually find and comment any “anomalous” pair of mates.

Part II

1. Implement in the language of our choice a program to create a wig file with the physical coverage track;
2. Create a wig track with sequence coverage and compare it with the IGV track;
3. Read the SAM file and for each mate pair calculate the length of the genomic insert, then calculate the mean and standard deviation of the inserts, possibly discarding those that are totally out of range. A plot of the length distribution may help to evaluate the sizes;
4. Create a track with the average length of the physical inserts covering the position of the genome;
5. Create a track with the percentage of inserts with a length exceeding n standard deviations (with n=2) above or below the mean. Compare it with track 12.

Part III

Be creative and make other tracks to identify other peculiarities associated to structural variations, in particular:

6. Create a track with the coverage of "unique" reads. Unique reads are those mapping on a single genomic position;
7. Create a track with the covarage of "multiple" reads. The multiple reads are those mapping in more than one genomic position;
8. Create a track with the percentage of oriented mates. The orientation can be found on "bit 16" of the "flag";
9. Create a track with percentage of single mates. Single mates are those where only one of the two reads is mapping.

3 Part I

3.1 Align reads on reference

In order to achieve this task, first I run the command (in the same folder of the Lactobacillus_casei_genome.fasta file):

```
bwa index Lactobacillus_casei_genome.fasta
```

that index the reference genome, using the default parameters.

Then, to align the Illumina reads on the reference genome I run the command:

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

This took quite a long time and the output was too long. At the end of this process, the alignments are stored in the **lact.sam** file.

3.2 Obtein BAM file and sort it

In order to process the SAM file obtained by the BWA analysis to make a BAM file, I used SAMtools. Running this command we transform sam file into a bam one (binary format)

```
samtools view -bS lact.sam > lact.bam
```

The option **-bS** is used to specify that we want as output a bam file starting from a sam file as input (-b for "output bam" and -S for "input sam").

In order to sort and index the bam file, first I run:

```
samtools sort lact.bam > lact_sorted.bam
```

This command sort the alignments **by genomic position**, saving the result in the **lact_sorted.bam** file. After that I run:

```
samtools index lact_sorted.bam
```

that produced a **lact_sorted.bam.bai** file that allow a fast random access to the bam file (It doesn't contain alignment informations, just an index to speed up the access).

3.3 Find anomalous pair of mates through IGV

To load the genome on IGV and visualize coverage and mate pairs, after opening IGV I clicked on *genomes* and then *Load genome from file*, selecting the ***Lactobacillus_casei_genome.fasta*** file. Then I loaded the ***lact_sorted.bam*** file clicking on *File* and then *Load from file...*

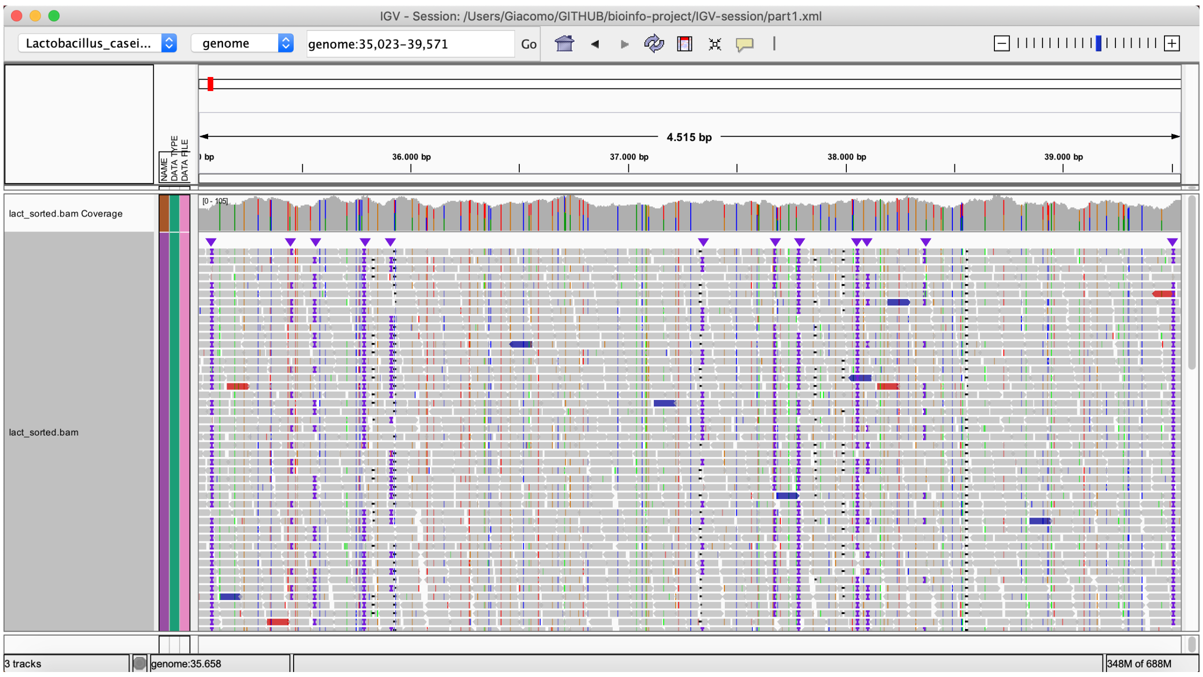


Figure 1: IGV view after genome load

According to IGV documentation[1] red reads suggest possible deletion, instead blue reads means that could be present an insertion.

[1] https://software.broadinstitute.org/software/igv/interpreting_insert_size

Anomalies found

In this paragraph I present some possible "anomalies" found in the genome browsing it using IGV. In order to make simpler to identify them I loaded in the program even the physical and the sequence coverage wig files of Part II.

3.3.1 Deletions

In figure 2 we could see a **long deletion**, instead in figure 3 we could see a **small** one.

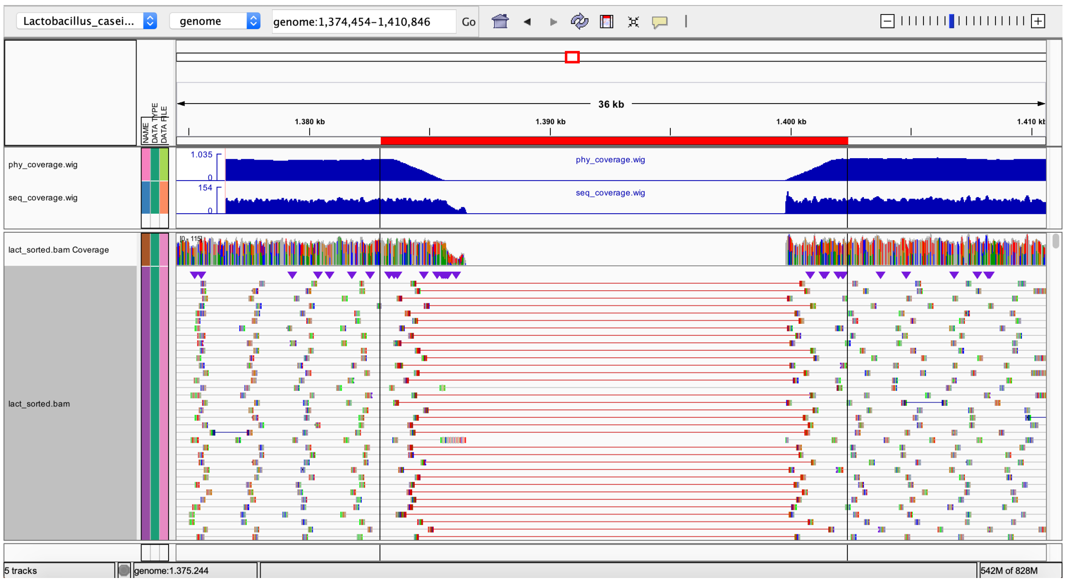


Figure 2: Long deletion (> 13.500 bp) - 36 kb zoom scale

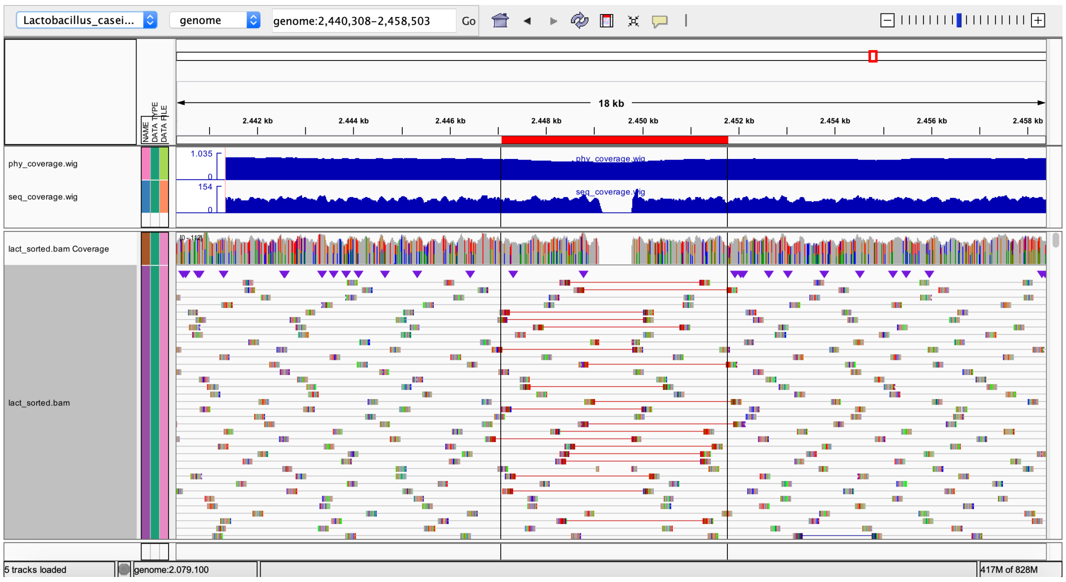


Figure 3: Small deletion (< 1000 bp) - 18 kp zoom scale

We can notice that, in case of a long deletion, both the physical coverage and the sequence coverage show a "hole". Instead, when a small deletion is present (fig. 3), this can be recognized only by analyzing the sequence coverage; the physical coverage seems to be unmodified. The reason for this difference is that, while sequence coverage takes into account the average number of times a base is read, physical coverage is the average number of times a base is spanned by mate paired reads.

3.3.2 Insertions

In figure 4 we could see a **long insertion**, instead in figure 5 we could see a **small insertion**.

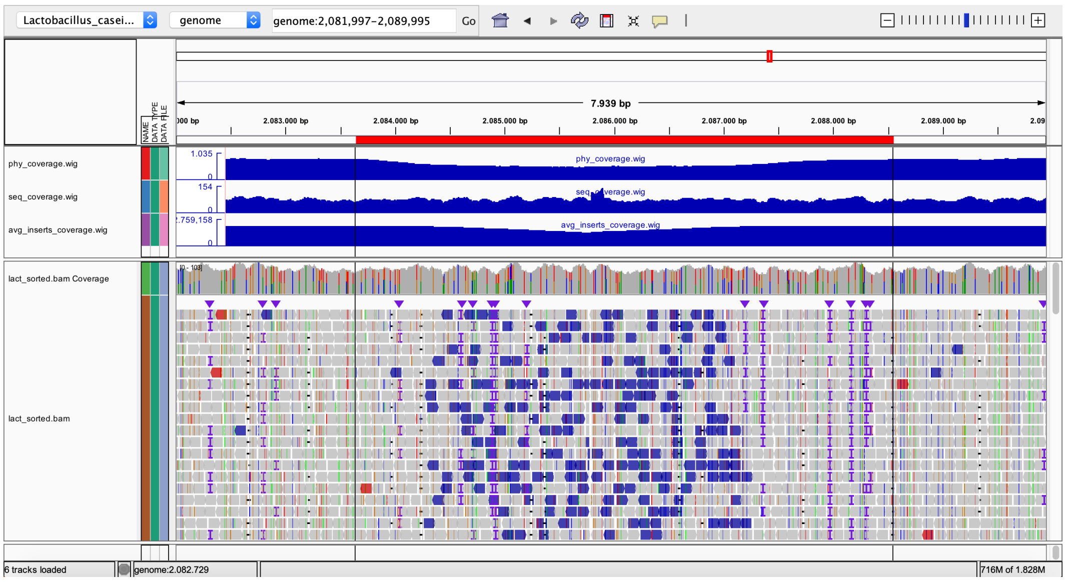


Figure 4: Small insertion - 8000 bp zoom scale

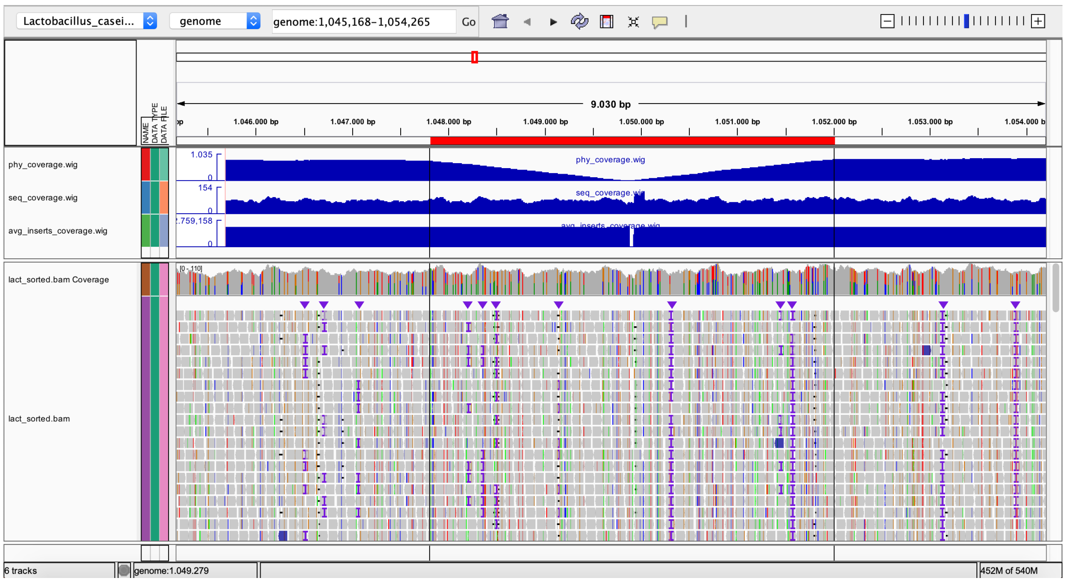


Figure 5: Small insertion - 9000 bp zoom scale

Notice that, in order to emphasize this insertion, I also load the average coverage of inserts (avg_inserts_coverage.wig) of Part II.4.

3.3.3 Inversions

In figure 6 we could see a **long inversion**, instead in figure 7 we could see a **small** one.

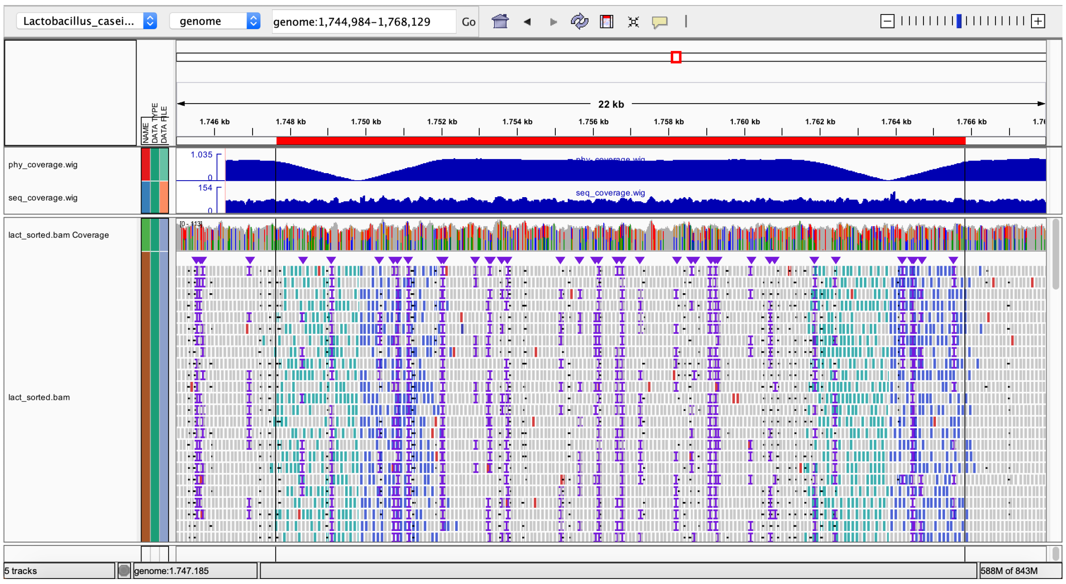


Figure 6: Long inversion - 22 kp zoom scale

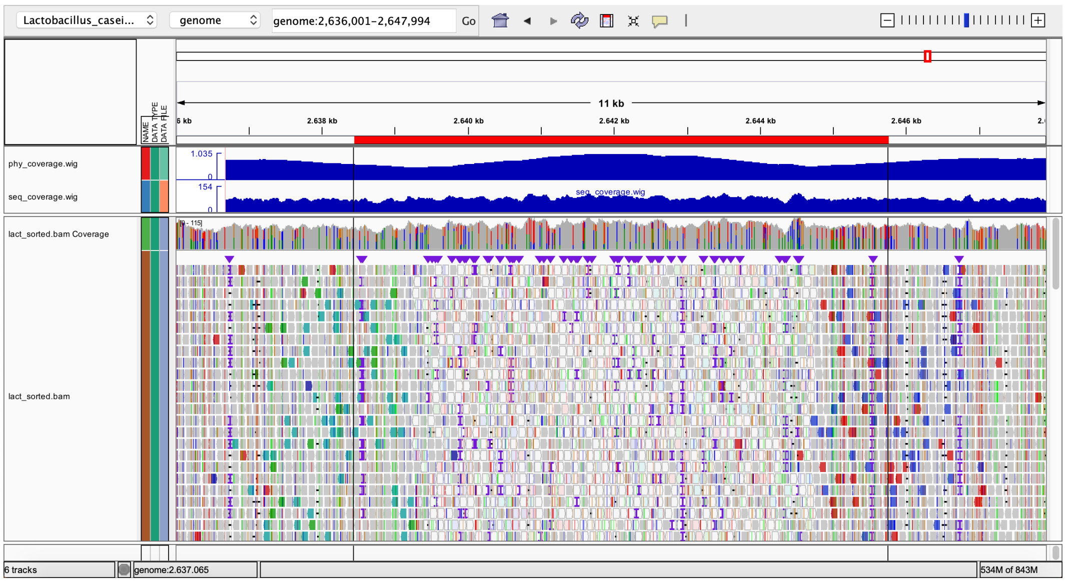


Figure 7: Small inversion - 11 kp zoom scale

4 Part II

4.1 Implementation choices

For this and next Part, I have chosen to implement all the scripts/algorithms using the language **Python 3** and take advantage of a Python library called ***pybam***[2] for loading and parsing the BAM file *lact_sorted.bam* produced after the BWA mapping in Subsection 3.2.

The advantage of Python is that this scripting programming language is designed to manipulate large statistics calculation, in easy and linear way. The only inefficiency is the slow speed of execution (especially in cycles), but thanks to the *pybam* external library combined with the use of the dataset in BAM format (lighter, smaller size and more machine readable) instead of a SAM, this defect can be compensated.

So, the starting point of all algorithms created, is take as an input argument a bam type file (our *lact_sorted.bam* data), created in the `main function` with the *pybam library* help through this command:

```
sorted_bam = pybam.read('../data/lact_sorted.bam')
```

Project setup
The entire programming section (Part II and III) of the Practical was set up to create a **usable, simple, extensible and user-oriented tool**.
In this way I created a project and developed scripts with a powerful Python oriented IDE called **PyCharm** (available at https://www.jetbrains.com/pycharm/download/).

This is the structure of the PyCharm project:

```
bioinfo-project
    |
    |-- data                    <--- (genomic datasets)
    |__ scripts
        |-- bam_parser.py       <--- (TOOL)
        |__ pybam.py            <--- (bam-parser library)
    |-- venv                    <--- (IDE virtual environment)
    |__ wig-tracks              <--- (IGV tracks folder)
    ..
    ..
```

After this topic overview, we could see now how I solved the required tasks of Part II and III...

[2] https://github.com/luidale/pybam

4.2 Physical Coverage

The function used to calculate the **Physical Coverage** is called `phy_coverage(bam_f)` and require, like all others functions from here on, a BAM file as input variable. Informations extracted from BAM file needed for this function are:

- **flag**: [2nd column in SAM] The FLAG number of the alignment;
- **starting_mate_position**: The 0-based position of the alignment;
- **template_length**: [9th column in SAM] The TLEN value.

This function, after having initialized the `genome_change` vector to a length equal to the reference genome (3079196 bp), search, for all alignment in the BAM input, if the binary flag ends with '11' (Before discards also outliers 0bp > inserts > 3000bp tlen). This end-part of the flag meanings read paired and read paired in proper pair, so we consider the entire genomic inserts that mapping in the reference genome. Then, inside this if, increment by one at the texttttstarting_mate_position index of genome_change while decrements by one at
`genome_change[starting_mate_position + template_length] position` (end position of the insert).

Finally, with genome_change full, cycle over all positions of the genome with this `for` construct, for add the correct inserts occurencies at the corresponding genomic position writing physical coverage in the related wig file:

```

  if 0 < template_length <= 3000:

    # if (interesting_flag == '11') and (mate_length > 0)
    #   11 -> read paired and read paired in proper pair
    if flag.endswith('11'):
      # increment start position by one
      genome_change[starting_mate_position] += 1

      # decrement end position by one
      genome_change[starting_mate_position + template_length]
          -= 1

current_coverage = 0
# cycle over all positions of the genome
for position in range(genome_length):
  current_coverage += genome_change[position]
  f.write(str(current_coverage) + '\n')
```

The output of this function it's a wig file called **phy_coverage.wig** put in wig-tracks folder of the project.

4.3 Sequence Coverage

The function used to calculate the **Sequence Coverage** is called `sequence_coverage(bam_f)`. Informations need from BAM file are the same of the previous function, the differences with physical coverage algorithm are that it don't filter about inserts length, the read length is fixed to 100 and it consider only alignment with 3rd bit of binary flag equal to 0, so it takes **mapped reads** and finally consider his coverage, coherently with sequence definition.

```
for alignment in bam_f:

  # get start position and fix read length
  starting_mate_position = alignment.sam_pos1  # 4th column
  read_length = 100

  if alignment.sam_flag & 4 == 0:  # take all mapped reads:
     [3rd bit] in bam format == 0
    genome_change[starting_mate_position] += 1

    # decrement end position by one
    genome_change[starting_mate_position + read_length] -= 1
```

The output of this function it's a wig file called **seq_coverage.wig** put in wig-tracks folder of the project and, as we can see in figure below, it's really similar to the IGV track.

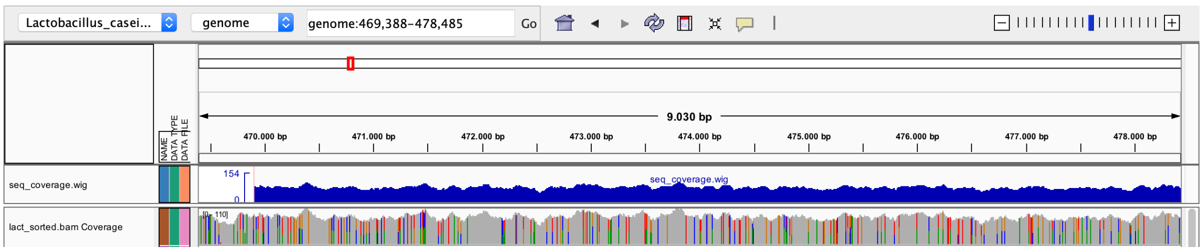


Figure 8: seq_coverage.wig track similar to IGV one

4.4 Genomics insert length statistics

With function called **get_genome_stats(bam_f)** I calculate, for each mate pair, the length of the genomics insert; then calculate the mean and standard deviation of the inserts, discarding those that are totally out of range (> 3000 bp):

```python
for alignment in bam_f:
  # get tlen value
  template_length = alignment.sam_tlen  # 9th column SAM

  if 0 < template_length <= 3000:  # manual filter discarding OUTLIERS

    count_values += 1
    if template_length > max_tlen:
      max_tlen = template_length

    if template_length < min_tlen:
      min_tlen = template_length

    total_sum += template_length
    tlen_list.append(template_length)

print("done..Calculating statistics..\n\n")
# Standard deviation calc
average = total_sum / count_values
stdev = statistics.stdev(tlen_list) # use basic stat module
```

If we run this function, the resulting output in the IDE console is:

```
--------------------------------------
Standard Deviation: 201.66577043621606

Total Reads: 1202194
Max Length: 3000
Min Length: 445
Average: 2101.0225496051385
Std.error: 0.18392675631490737
--------------------------------------
```

4.5 Average physical inserts coverage

The function called **avg_inserts_coverage(bam_f)** calculate the average length of the physical inserts that cover the position of the genome.

So, after the same filter to discard the outliers inserts, this function use same reasoning like physical coverage calculation but take advantage of a second vector where it stores reads coverage, useful values to callculate the average length of physical inserts:

```
# ......
# ......
        # if (interesting_flag == '11') and (mate_length > 0)
        #   11 -> read paired and read paired in proper pair
        if flag.endswith('11'):
          # increment start position by one
          genome_change[starting_mate_position] += 1
          genome_reads[starting_mate_position] += template_
             length

          # decrement end position by one
          genome_change[starting_mate_position + template_
             length] -= 1
          genome_reads[starting_mate_position + template_
             length] -= template_length
# .....
# .....
for position in range(genome_length):
  current_coverage += genome_change[position]
  current_sum += genome_reads[position]

  if current_coverage > 0:
    f.write(str(current_sum / current_coverage) + '\n')
  else:
    f.write('0\n')
```

This script produce in output the **avg_inserts_coverage.wig** track.

4.6 Inserts 2-standard deviation distribution

The function used for calculate the percentage of inserts with a length exceeding 2-standard deviations (above or below the mean) it's called **two_distribution_percentage(bam_f, average, stdev)** and need the usal BAM file, as well as the mean and the standard deviation founded previously in the subsection 4.4.

```
# .....
# .....
    if flag.endswith('11') and (template_length > 0) and \
    ((template_length < average - stdev * 2) or (template_length > average + stdev * 2)):

    # if (interesting_flag == '11') and (mate_length > 0)
    #   11 -> read paired and read paired in proper pair
    if flag.endswith('11'):
      # increment start position by one
      genome_change[starting_mate_position] += 1

      # decrement end position by one
      genome_change[starting_mate_position + template_length] -= 1
# .....
print("calculating percentage coverage distribution in 'std_%distribution.wig'...\n")
get_percent_match("phy_coverage.wig", "std_distribution(temp).wig", "std_%distribution.wig")

print("Removing 'std_distribution(temp)' wig file..\n")
os.remove("../wig-tracks/std_distribution(temp).wig")
```

After creation of temporary coverage track of distribution (**std_distribution(temp).wig**), it compare this temp. file with the physical coverage wig track calculated at the beginning to make the percentage distribution **std_%distribution.wig** track file.
The comparison with the previous average inserts track is showed below through IGV:

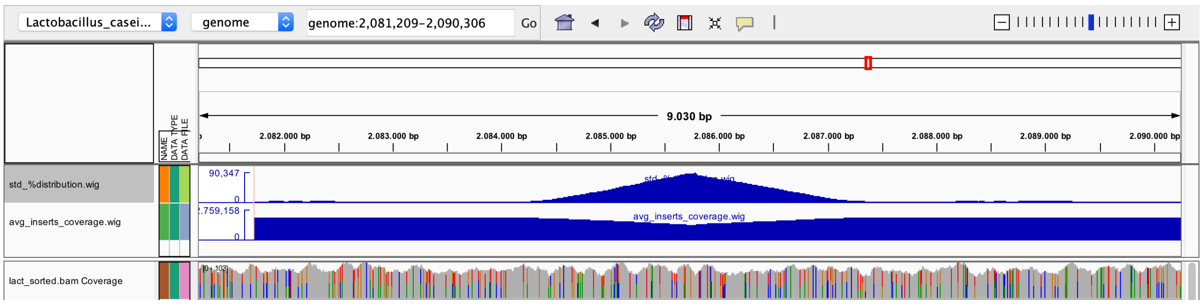


Figure 9: Comparison between *avg_inserts_coverage.wig* and *std_%distribution.wig* tracks - long insertion genome section

5 Part III

I present now the Third and last part of statistical analysis with other creative tracks, useful for further conclusions about structural variations in the reference genome.

5.1 Identify Unique and Multiple mapping reads

Uniquely Mapped Reads

To identify uniquely mapped reads, or those reads that are mapped into a single genomic location, we must first discard several elements to get to filter only the searched reads. To do this we exploit the **SAMtool** software and through it we ignore the following elements from the original bam (*lact_sorted.bam*):

- **-F 4**: not mapped reads;
- **-F 256**: not primary aligned;
- **-q 1**: reads with MAPQ quality less than 1.

After that, we perform a further filtering by selecting the correct tag that identifies these reads according to the current documentation of the `bwa -mem command`. So, the complete inline command to find the uniquely mapped reads and create a new BAM file from original one is:

```
samtools view -h -q 1 -F 4 -F 256 DATA/lact_sorted.bam | grep
    -v XA:Z | grep -v SA:Z | samtools view -b - > DATA/unique
   .bam
```

where **XA** to report alternative sites reads and **SA** to filter for unique ones[3].

Finally the function **unique_reads_coverage(bam_f)** calculate the coverage of this reads obviously taking as a parameter bam for data reading, the one created by the previous command (called ***unique.reads.bam***) producing **unique_coverage.wig** IGV track.

Multiple Mapped Reads

Instead, to identify multi-mapped reads (those mapping in more than one genomic position), the prescribed function is **multiple_reads_coverage(bam_f)**, taking as input file the usual *lact_sorted.bam* dataset.

To discriminate the multiple mapped reads, as the SAM documentation suggests, we need to check the **9th bit of the binary flag** (dec: 256 bin: 0x100), which indicates whether this alignment is considered as **secondary alignment**. So, in order to calculate the coverage, this function perform the following part of code:

[3]wabi-wiki/Filter+uniquely+mapped+reads+from+a+BAM+file

```
# .....
    multi_flag = flag[-9:] # take last 9 digit of bam flag

    # get start position and tlen value
    starting_mate_position = alignment.sam_pos1  # 4th column
    template_length = alignment.sam_tlen  # 9th column

    if template_length <= 3000 and multi_flag.startswith('1')
       : # is secondary align
      #   11 -> read paired and read paired in proper pair
      if flag.endswith('11'):
        # increment start position by one
        genome_change[starting_mate_position] += 1
        # decrement end position by one
        genome_change[starting_mate_position + template_
           length] -= 1
# .....
```

The resulting coverage file it's called **multiple_reads_coverage.wig**.

Observations about Multimapping coverage

After doing this calculation, the **file appears empty** which means that no reads is mapped several times in some part of the genome, very strange.
Searching through the **bwa documentation** I found that with the -a option (Professor suggestion) the resulting output following the alignment is: "..all found alignments for single-end or unpaired paired-end reads. These alignments will be flagged as secondary alignments".
Deductively then, you could filter the resulting SAM file again (or converted to BAM) and search, based on the name, which alignments appear only 2 times (unique) and which ones from 3 upwards (multiple).

For reasons of time I did not take this second possibility for the task resolution (also because *bwa* commands change constantly and check this is the right procedure isn't so easy).

5.2 Oriented Mates percentage

Since The orientation can be found on **bit 16** of the SAM flag **(bin:0x10 -> SEQ being reverse complemented)**, the function **oriented_mates_percentage(bam_f)** take the last 6 bits of the binary flag and then discriminate cases according to the 3 orientation groups, like could be notice in the code below:

```
# ...
    if tlen <= 3000 and flag.endswith('1'):  # filter + paired

    ## case: <-- <-- & --> --> as reads are WRONG MAPPED,
    if flag.startswith('11') or flag.startswith('00'):
      if tlen > 0:
        genome_wrong[start_pos] += 1
        genome_wrong[start_pos + tlen] -= 1
      else:
        genome_wrong[start_pos + tlen + 1] += 1
        genome_wrong[start_pos + 1] -= 1


    ## case: <-- --> positive strand and positive length(10 & l>0),
    elif flag.startswith('10') and tlen > 0:
      genome_out[start_pos] += 1
      genome_out[start_pos + tlen] -= 1

    ## case: <-- --> negative strand and negative length(01 & l<0),
    elif flag.startswith('01') and tlen < 0:
      genome_out[start_pos + tlen + 1] += 1
      genome_out[start_pos + 1] -= 1


    ## case: --> <-- positive strand but negative length(10 & l<0)
    elif flag.startswith('01') and tlen > 0:
      genome_in[start_pos] += 1
      genome_in[start_pos + tlen] -= 1

    ## case: --> <-- negative strand but positive length(01 & l>0)
    elif flag.startswith('10') and tlen < 0:
      genome_in[start_pos + tlen + 1] += 1
      genome_in[start_pos + 1] -= 1
# ...
```

.

Finally produce 3 percentage coverage files using **total_phy_coverage()** function for the comparison (it produce physical coverage without tlen size filter).
Those files are:

- **out_%oriented.wig**: for <— —> orientation verse;
- **in_%oriented.wig**: for —> <— orientation verse;
- **wrong_%oriented.wig**: for —> —> or <— <— orientation verse;

5.3 Single Mates percentage

Single mates are those where only one of the two reads is mapping. The related flag is the 4th bit: 0x4 which stands for **segment unmapped**. The function **single_mates_percentage(bam_** check if this bit is 1 and also if his complementary is 0:

```
for alignment in bam_f:
  # conversion of flag from integer to binary
  single_flag = bin(alignment.sam_flag)[-4:]  # 4th bit: 0
     x4 stand for SEGMENT UNMAPPED
  start_pos = alignment.sam_pos1  # 4th column: start mate
     pos

  if single_flag.startswith('01'):  # not mapped reads(1)
     but complementary is(0)
    #   TLEN value is set as 0 for single-segment template
       ...
    genome_change[start_pos] += 1
```

According to SAM specification[4], TLEN value is set as 0 for single-segment template, so it increment only the starting position of the mate-pair.

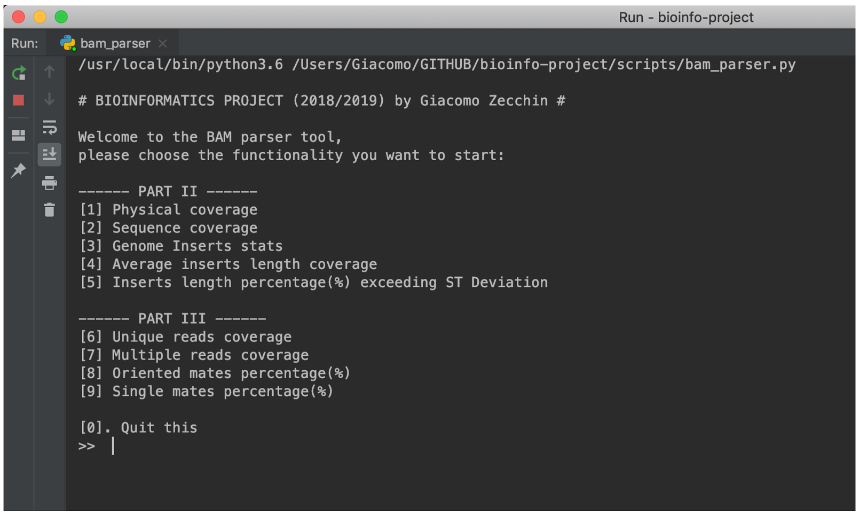
The output track is called **single_mates_%coverage.wig**, a percentage coverage calculated using total physical coverage as comparison.

[4]https://samtools.github.io/hts-specs/SAMv1.pdf

6 Bioinformatics Tool

In this conclusive section I'll briefly present how the developed tool inside the PyCharm project is presented, with the aim of using all the algorithms described in section 4(Part II) and 5(Part III).

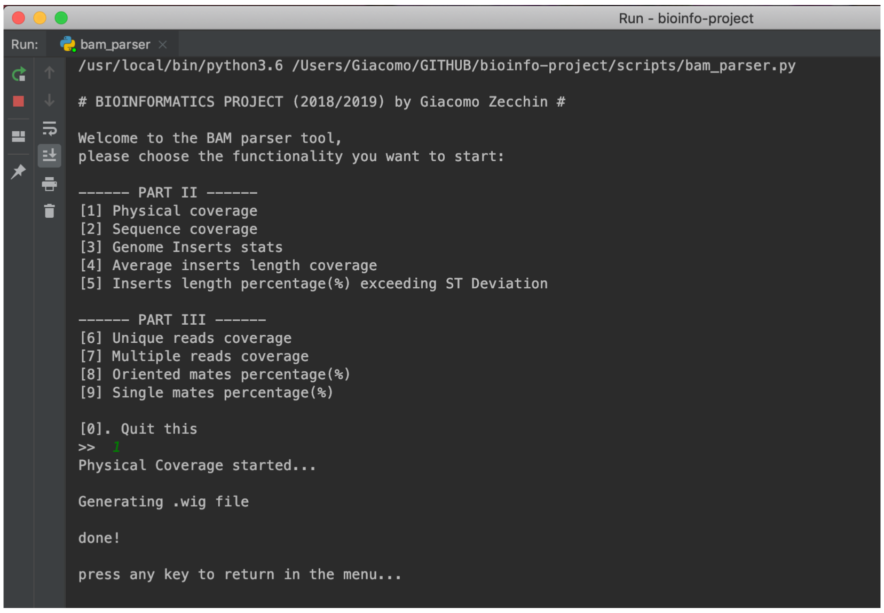
The **Welcome Window** of the Tool is presented in figure below:



```
/usr/local/bin/python3.6 /Users/Giacomo/GITHUB/bioinfo-project/scripts/bam_parser.py

# BIOINFORMATICS PROJECT (2018/2019) by Giacomo Zecchin #

Welcome to the BAM parser tool,
please choose the functionality you want to start:

------ PART II ------
[1] Physical coverage
[2] Sequence coverage
[3] Genome Inserts stats
[4] Average inserts length coverage
[5] Inserts length percentage(%) exceeding ST Deviation

------ PART III ------
[6] Unique reads coverage
[7] Multiple reads coverage
[8] Oriented mates percentage(%)
[9] Single mates percentage(%)

[0]. Quit this
>> 
```

Figure 10: Welcome view after [>run] of Tool - bam_parser.py

As we can see, the menu offers to the user the preferred functionality that, based on the input received, start the related function/algorithm. Exception and invalid input are controlled!
In conclusion, an example after selecting the first algorithm:



```
/usr/local/bin/python3.6 /Users/Giacomo/GITHUB/bioinfo-project/scripts/bam_parser.py

# BIOINFORMATICS PROJECT (2018/2019) by Giacomo Zecchin #

Welcome to the BAM parser tool,
please choose the functionality you want to start:

------ PART II ------
[1] Physical coverage
[2] Sequence coverage
[3] Genome Inserts stats
[4] Average inserts length coverage
[5] Inserts length percentage(%) exceeding ST Deviation

------ PART III ------
[6] Unique reads coverage
[7] Multiple reads coverage
[8] Oriented mates percentage(%)
[9] Single mates percentage(%)

[0]. Quit this
>> 1
Physical Coverage started...

Generating .wig file

done!

press any key to return in the menu...
```

Figure 11: Example choice of Tool - 1: Physical Coverage algorithm start

7 References

- http://www.htslib.org/download/: SAMtools sw
- https://github.com/lh3/bwa: BWA sw
- https://software.broadinstitute.org/software/igv/download: IGV sw
- https://software.broadinstitute.org/software/igv/interpreting_insert_size
- https://www.jetbrains.com/pycharm/download/#section=mac: Pycharm download
- https://github.com/luidale/pybam: pybam Python library
- **Find Uniquely mapped reads link**
- http://bio-bwa.sourceforge.net/bwa.shtml: bwa commands spec.
- https://samtools.github.io/hts-specs/SAMv1.pdf: SAM spec.
- https://www.samformat.info/sam-format-flag: SAM flag explanation