Finding structural variations in a genome after comparison with a reference genome: the lactobacillus casei

Fabiana Rapicavoli, 2045245
Bioinformatics
Agosto 2022

Contents

1 Introduction

The aim of this project was to find structural variations in a genome sequence of a bacterium called Lactobacillus casei, after the comparison with its reference genome.
In particular, I wanted to "resequence" of a similar bacterium, that we call lact.sp, using mate pairs reads.

Structural variations are in a genome are, for example, insertions, deletions or inversions of amino acids in a sequence.

Structural variations in the genome can directly or indirectly influence gene dosage through different mechanisms, and therefore influence phenotypic variation and disease.
It is important to control and study how to detect the structural variation of the genomes because its cataloguing and their frequencies in populations will be crucial for disease-mapping studies and for proper interpretation of clinical diagnostic-testing data. Feuk et al. (2006)

2 Softwares

In order to complete the task, I used three main softwares:

- BWA - Burrow-Wheeler Aligner to short-read alignment; Li and Durbin (2009)
- Samtools - Sequence Alignment/Map to format and store large nucleotide sequence alignments; Li et al. (2009)
- IGV - Integrative Genomics Viewer to an interactive genome visualization component. Thorvaldsdóttir et al. (2013)

2.1 BWA

BWA is a software package for mapping DNA sequences, against a large reference genome, such as the human genome.
It consists of three algorithms: BWA-backtrack, BWA-SW and BWA-MEM.

The first algorithm is designed for Illumina sequence reads up to 100bp, while the rest two for longer sequences ranged from 70bp to a few megabases. BWA-MEM and BWA-SW share similar features such as the support of long reads and chimeric alignment, but BWA-MEM, is generally recommended as it is faster and more accurate. BWA-MEM also has better performance than BWA-backtrack for 70-100bp Illumina reads.

For my project, I used BWA MEM.

2.2 Samtools

SAM aims to be a format that:

- Is flexible enough to store all the alignment information generated by various alignment programs;
- Is simple enough to be easily generated by alignment programs or converted from existing alignment formats;
- Is compact in file size;
- Allows most of operations on the alignment to work on a stream without loading the whole alignment into memory;
- Allows the file to be indexed by genomic position to efficiently retrieve all reads aligning to a locus.

Samtools provide various utilities for manipulating alignments in the SAM format, including sorting, merging, indexing and generating alignments in a per-position format.

3 Preparing and exploring the data

3.1 What is a read?

In next-generation sequencing, a read refers to a raw sequence, that comes off a sequencing machine.
A read may consist of multiple segments. For sequencing data, reads are indexed by the order in which they are sequenced.

3.2 Reads alignment

I used BWA index in the Lactobacillus casei fasta file to produce several files, useful for the alignment and BWA MEM for the actual alignment, using both the read files, provided.
Then I sorted by position the bam file, which is the compressed binary version of a SAM file, that is used to represent aligned sequences up to 128 Mb.
The SAM file is a specification, which aims to define a genetic nucleotide alignment format and describes the alignment of query sequences or sequencing reads to a reference sequence or assembly.
The algorithm, used for the sorting, had a complexity of O(logn) in the best case scenario.

In the SAM and BAM format, each alignment line typically represents the linear alignment of a segment.
Both the SAM and BAM files have this type of features:

Col	Field	Type	Regexp/Range	Brief description
1	QNAME	String	`[!-?A-~]{1,254}`	Query template NAME
2	FLAG	Int	$[0, 2^{16} - 1]$	bitwise FLAG
3	RNAME	String	`\*|[:rname:^*=][:rname:]*`	Reference sequence NAME[11]
4	POS	Int	$[0, 2^{31} - 1]$	1-based leftmost mapping POSition
5	MAPQ	Int	$[0, 2^{8} - 1]$	MAPping Quality
6	CIGAR	String	`\*|([0-9]+[MIDNSHPX=])+`	CIGAR string
7	RNEXT	String	`\*|=|[:rname:^*=][:rname:]*`	Reference name of the mate/next read
8	PNEXT	Int	$[0, 2^{31} - 1]$	Position of the mate/next read
9	TLEN	Int	$[-2^{31} + 1, 2^{31} - 1]$	observed Template LENgth
10	SEQ	String	`\*|[A-Za-z=.]+`	segment SEQuence
11	QUAL	String	`[!-~]+`	ASCII of Phred-scaled base QUALity+33

All mapped segments in alignment lines are represented on the forward genomic strand. For segments that have been mapped to the reverse strand, the recorded SEQ is reverse complemented from the original unmapped sequence and CIGAR, QUAL, and strand-sensitive optional fields are reversed and thus recorded consistently with the sequence bases as represented.

Let's study these features in detail:

- **QNAME**: Query template NAME.
  Reads/segments having identical QNAME are regarded to come from the

same template. A QNAME '*' indicates the information is unavailable. In a SAM file, a read may occupy multiple alignment lines, when its alignment is chimeric or when multiple mappings are given:

- **FLAG**: Combination of bitwise FLAGs.

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

- **RNAME**: Reference sequence NAME of the alignment.
  An unmapped segment without coordinate has a '*' at this field. However, an unmapped segment may also have an ordinary coordinate such that it can be placed at a desired position after sorting.
  If RNAME is '*', no assumptions can be made about POS and CIGAR;
- **POS**: first based leftmost mapping POSition of the first CIGAR operation that "consumes" a reference base. In other words, the first base in a reference sequence has coordinate 1, where the alignment begins.
  POS is set as 0 for an unmapped read without coordinate. If POS is 0, no assumptions can be made about RNAME and CIGAR;
- **MAPQ**: MAPping Quality.
  A value 255 indicates that the mapping quality is not available;
- **CIGAR**: CIGAR string, which can admits these type of operations:

Op	BAM	Description	Consumes query	Consumes reference
`M`	0	alignment match (can be a sequence match or mismatch)	yes	yes
`I`	1	insertion to the reference	yes	no
`D`	2	deletion from the reference	no	yes
`N`	3	skipped region from the reference	no	yes
`S`	4	soft clipping (clipped sequences present in SEQ)	yes	no
`H`	5	hard clipping (clipped sequences NOT present in SEQ)	no	no
`P`	6	padding (silent deletion from padded reference)	no	no
`=`	7	sequence match	yes	yes
`X`	8	sequence mismatch	yes	yes

"Consumes query" and "consumes reference" indicate whether the CIGAR operation causes the alignment to step along the query sequence and the reference sequence respectively;

- **RNEXT**: Reference sequence name of the primary alignment of the NEXT read in the template.
  This field is set as '*' when the information is unavailable, and set as '=' if RNEXT is identical RNAME;

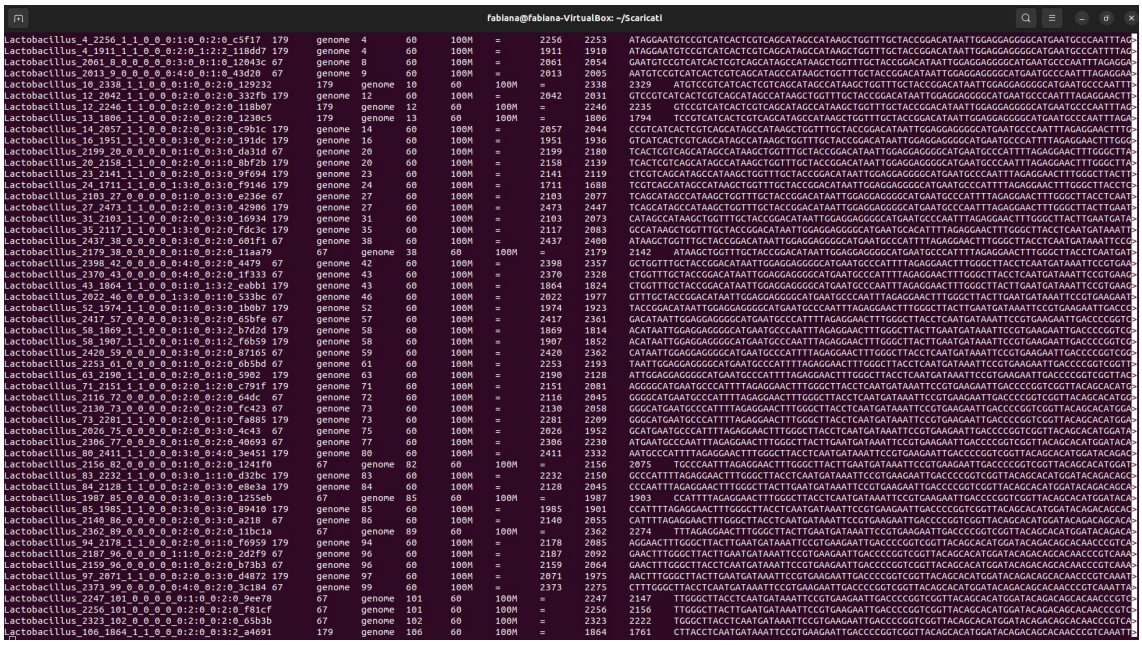
- **PNEXT**: first based position of the primary alignment of the NEXT read in the template.
  Set as 0 when the information is unavailable and no assumptions can be made on RNEXT and bit 0x20;

- **TLEN**: signed observed Template LENgth.
  For primary reads, where the primary alignments of all reads in the template are mapped to the same reference sequence, the absolute value of TLEN equals the distance between the mapped end of the template and the mapped start of the template, inclusively;

- **SEQ**: segment SEQuence.
  This field can be a '*' when the sequence is not stored.
  If not a '*', the length of the sequence must equal the sum of lengths of M/I/S/=/X operations in CIGAR.
  An '=' denotes the base is identical to the reference base.
  No assumptions can be made on the letter cases;

- **QUAL**: ASCII of base QUALity plus 33 because the first 33 elements are unprintable characters.
  This field can be a '*' when quality is not stored. If not a '*', SEQ must not be a '*' and the length of the quality string ought to equal the length of SEQ.

Below, you can find a sample of the BAM file, with the initial features described above:

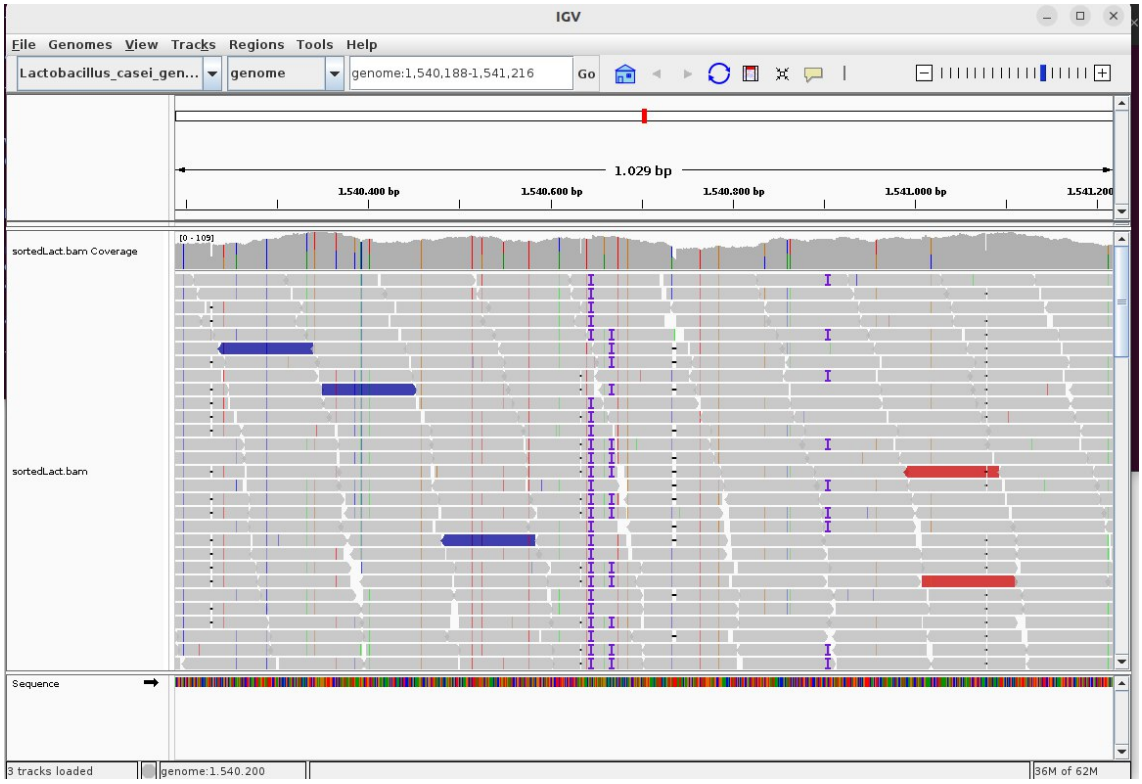


3.3 Visualize the coverage and mate pairs on the IGV

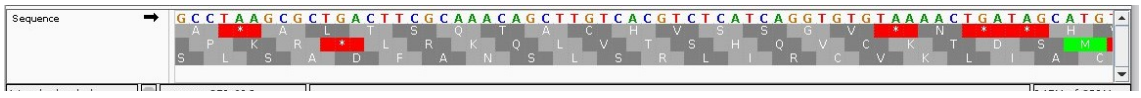
I used IGV software to visualise coverage and mate pairs, in order to have a graphical representation of them and have an immediate idea of the possible "anomalous" pair of mates.

3.3.1 Insert the files on IGV

I inserted the fasta file of Lactobacillus casei as genomes and the sorted BAM file as file.

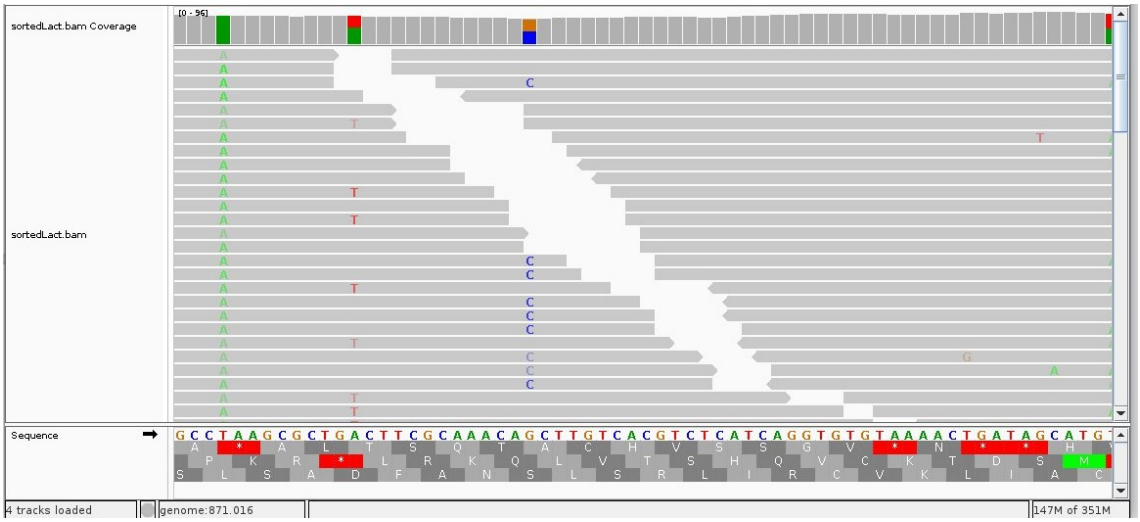


This picture is a portion of Lactobacillus genomes.
The upper bar shows the coverage of reads, in which every peaks show an overlaps of it. It allows us to understand if the read is well covered or not.
The bottom bar shows the reference genome and, if you zoom enough, you can see the bases of the genome, like adenine, cytosine, guanine or thymine:



The bar in the middle shows the actual reads.

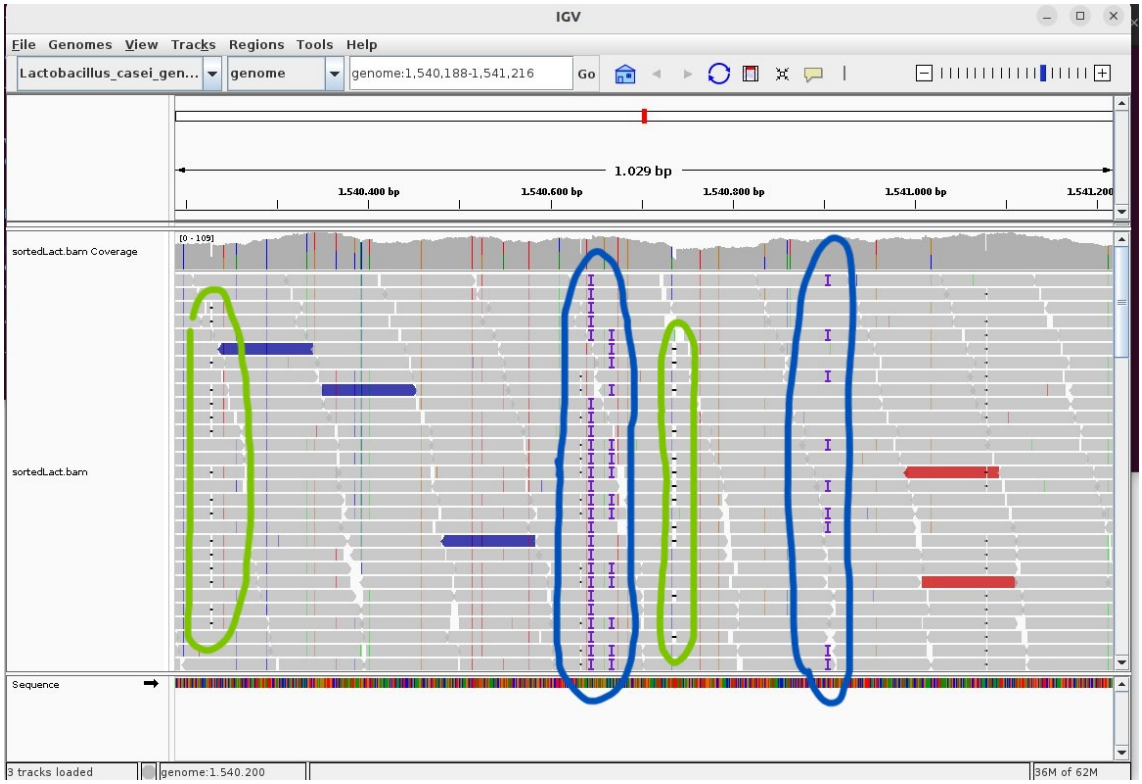
3.3.2 Anomaly 1: mismatch and unmapped genomes



We can see from the picture that some genomes are mismatched in their base. Also, in the same image, we can notice that there are some unmapped genome segments locally in some points and this forms some gap in the reads.

3.3.3 Anomaly 2: insertions and deletions

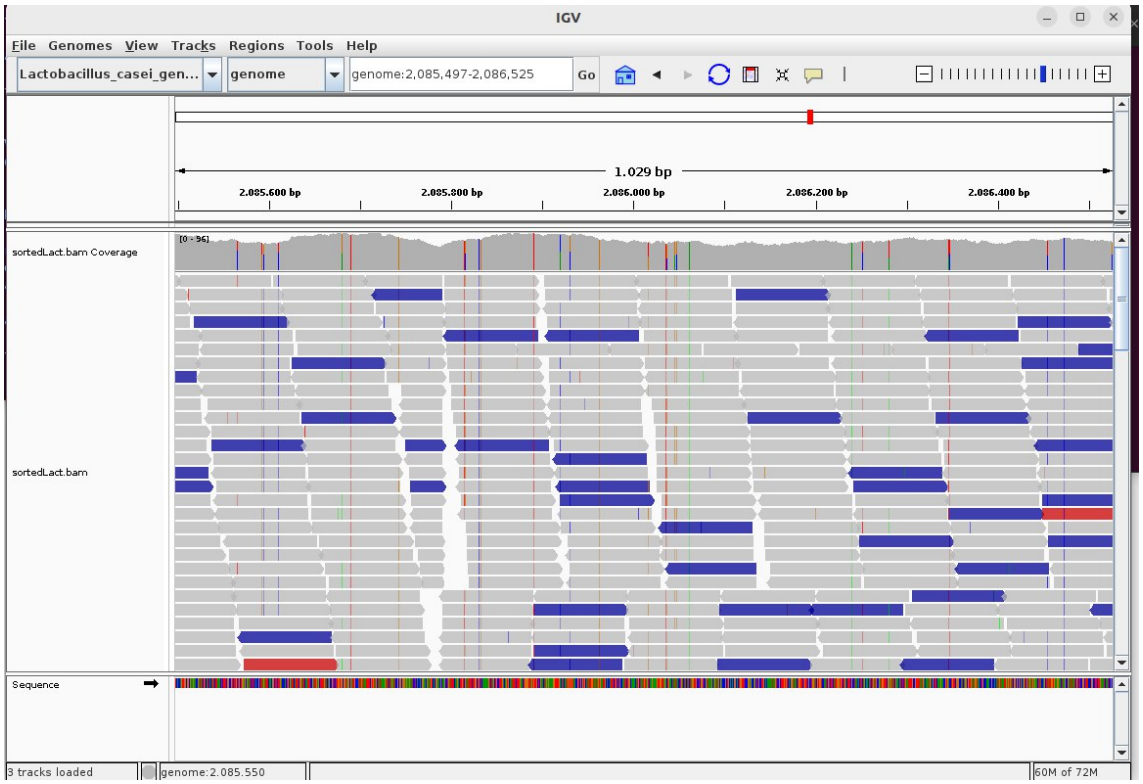
IGV reports insertions with a little vertical blue line and deletions with a little unmapped region and with a horizontal black line, here an example of both phenomena:



Two examples of insertion are circled on blue and other two examples of deletions are circled on green.
After a quick observation, I noticed more insertions, than deletions.

There is another way to detect insertions and deletions, by using the inferred insert size.
An inferred insert size larger than expected could be a possible evidence of a deletion, reported as a big red horizontal line.
On the other hand, an inferred insert size smaller than expected could be a possible evidence of an insertion, reported as a big blue horizontal line.

We can notice this phenomena in the previous pictures, but, especially in this image below, we can see, again, that there are more insertions, than deletions.


IGV
File Genomes View Tracks Regions Tools Help
Lactobacillus_casei_gen...
genome
genome:2,085,497-2,086,525
Go
1.029 bp
2,085,600 bp
2,085,800 bp
2,086,000 bp
2,086,200 bp
2,086,400 bp
sortedLact.bam Coverage
[0 - 96]
sortedLact.bam
Sequence
3 tracks loaded
genome:2,085,550
60M of 72M

3.3.4 Tandem duplications and invertions

As deletions and insertions, we can detect also tandem duplications and invertions with big horizontal line colored by green and aqua green, respectively.

4 Creating tracks

After analyzing the possible anomalies and structural variations, I created a track to better understand the reasons of these phenomena.

4.1 Sequence coverage

I downloaded the python file seqcoverage.py, in order to quantify the number of reads, occurred for each position of the Lactobacillus genomes.

This file, first of all, initializes an array of zeros, called genome_change, with length equal to the number of genomes, given in input.
Then, for every line in the SAM file not starting with @, it makes a list with all the separated features, exploiting the tabulation. It checks if the bitwise of the second feature (FLAG) is mapped and, if the genome is mapped, the file memorizes the starting position of every read (POS feature) and increment by one the value of genome_change at position POS and decrement by one the value of genome_length at position POS+100.
Lastly, a variable called current_coverage is made, which computes the sum of every values in gemone_change in a for cycle and it prints this number for every position of the genome.

I used this script for creating a wig file and I added it in IGV as a file.

The behavior of the previous phenomena is very similar as before, even if it counts the number of reads in Lactobacillus position and it has only positive

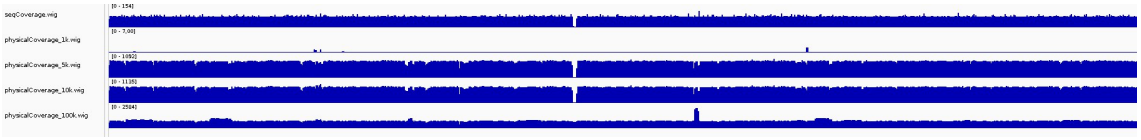
values.
The only difference is that before the coverage was from 0 to 112 and now it is from 0 to 154, probably due to the fact that I considered every read 100 bases long.

4.2 Physical coverage

Physical coverage of a genomic position is given by the number of fragments, laying on that position, rather than the number of reads. Some reads may map on multiple positions (chimeric alignment) 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.

For that reason, I made a copy of the python script of before (sequence coverage) and modified it a bit.
In particular, I added as input a value, which indicates the maximum length of a fragment.
Instead of checking just if the bitwise of the second feature (FLAG) is mapped, I checked also if the next segment (its mate) is.
Then, I also checked if the TLEN feature is positive and less than the maximum size, given in input.
If all these conditions was true, the programme would calculate the variable current_coverage as before.

I runned the code four times, with different maximum length: 1.000, 5.000, 10.000 and 100.000.
Again, I created the wig files and uploaded them in IGV as a file. This is the result:



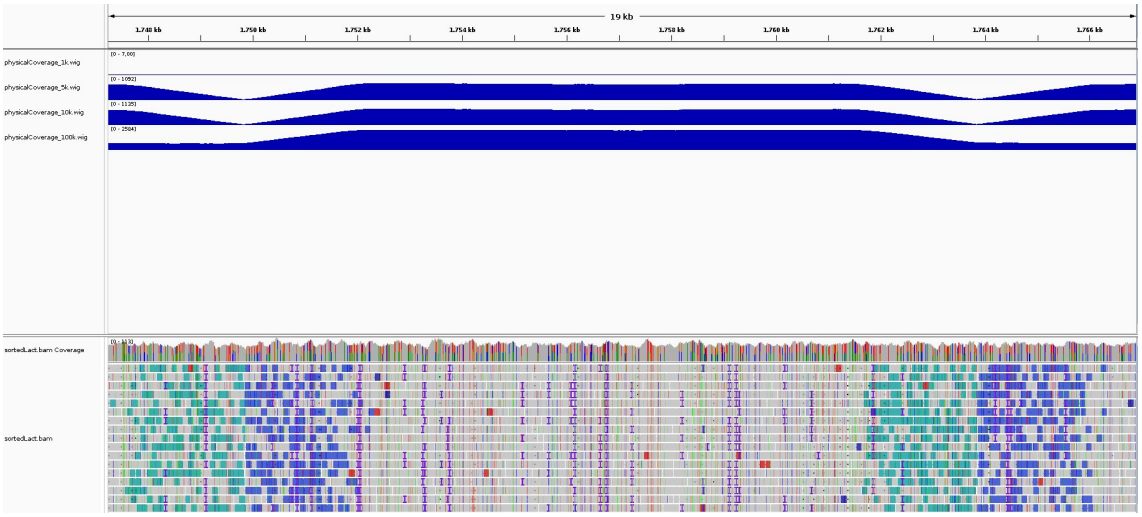
This picture shows that there are many overlapping, caused by the fact that many parts of the Lactobacillus genome are mapped by multiple misaligned reads.

It is possible to notice that the range of values with 1.000 as maximum length goes from 0 to 7 fragments counted, which is quite low. In fact, the programme calculated very few genomes and the line is almost flat on zero.
The range of values with 5.000 as maximum length goes from 0 to 1.092 fragments counted, which is a big improvement, than before. Now, the program counts many genomes, with a single exception of an empty spot, which indicates that, in that area, the length of the segments exceed 5.000.

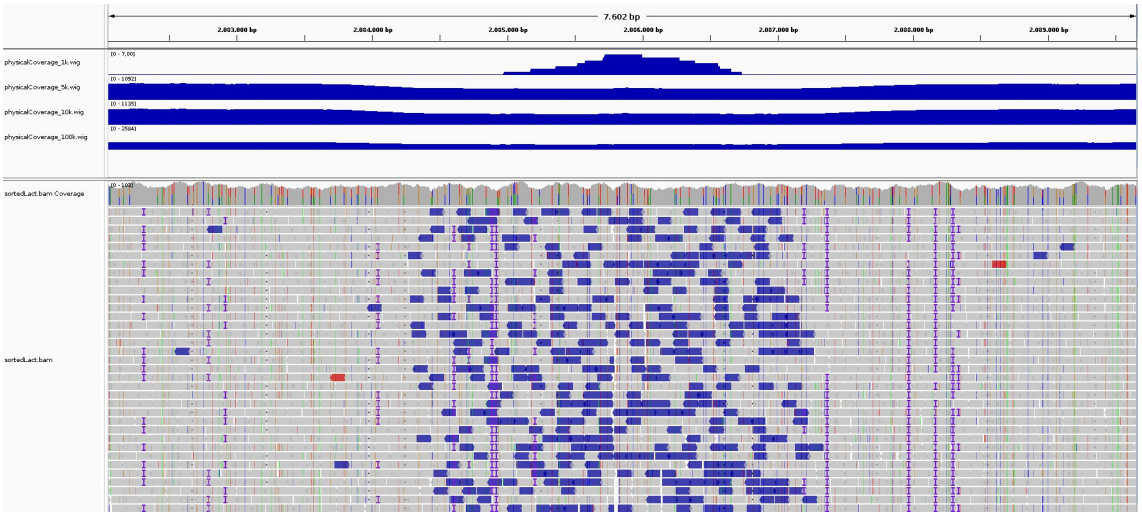
The range of values with 10.000 as maximum length goes from 0 to 1.135 fragments counted. Even if I increased a lot the maximum length of the segments, neither the range of values, nor the line changed much, which is almost the same of the previous one.
The range of values with 100.000 as maximum length goes from 0 to 2.584 fragments counted. Now, the result has changed considerably and there aren't any empty spots.

I analyzed the anomalies, that I found:

In the zone, where there is a gap with maximum values 5.000 and 10.000 as segment length, we can notice a long deletion.
There is not the gap, when the maximum size is 20.000, probably because the segments are so big that they don't permit to the program to notice the real gap.



In the zone, where there is the maximum peak with maximum value of 20.000 as maximum length of the segments, we can notice an invertion.



In the zone, where there is the peak with maximum value of 1.000 as maximum length of the segments, we can notice a long insertion.

4.3 Single mates

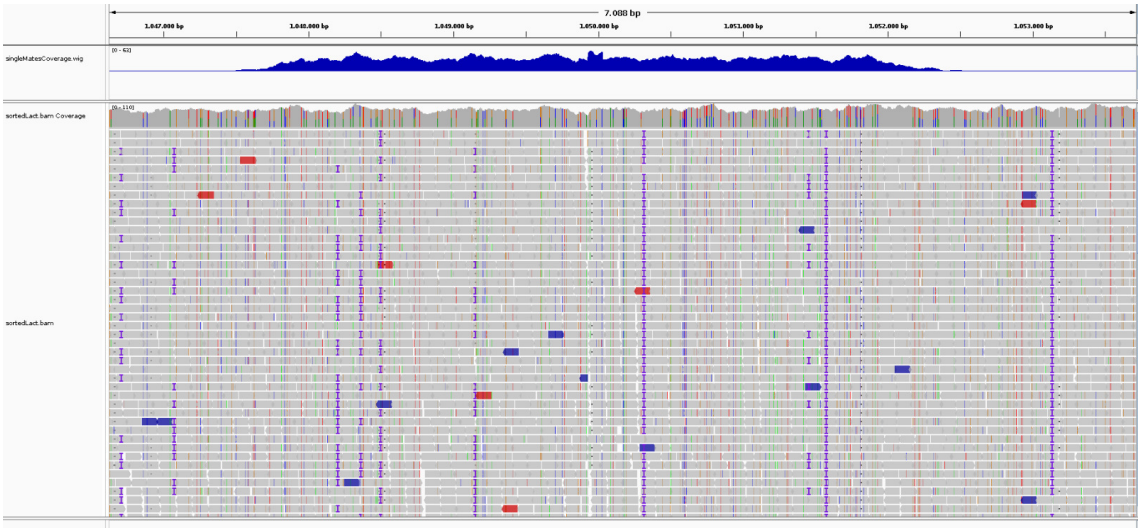
Single mates are those, where only one of the two reads is mapping.
I modified the previous pyhton script, in order to calculate the percentage of the single reads.

So I simply deleted the controls about the TLEN measurement and, regarding the FLAG bitwise, I check if the current read was mapped, but the mate wasn't. Apart from these modifications, I computed the variable od current_coverage as before.

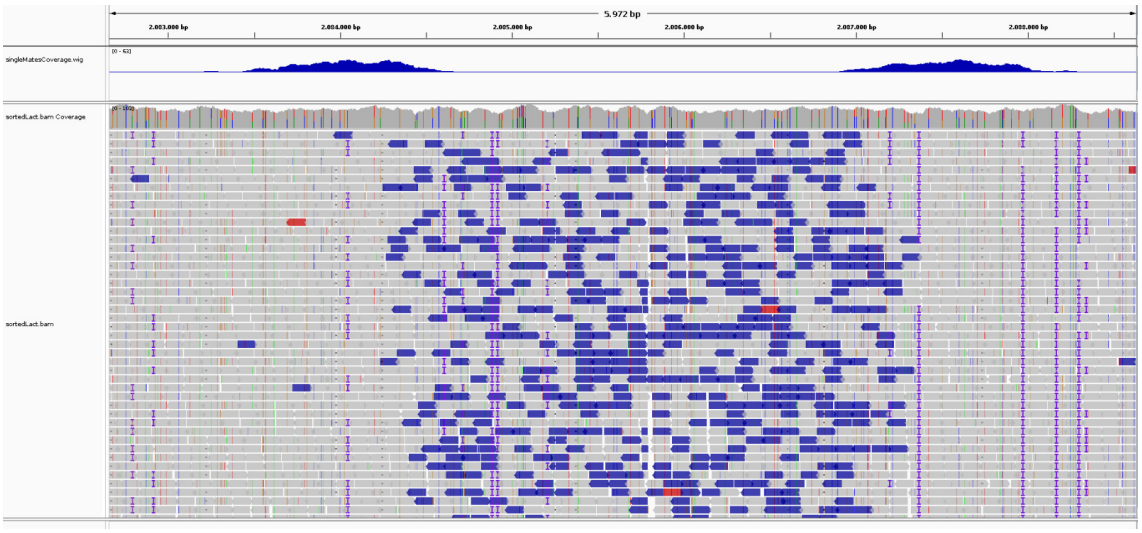
I uploaded the usual wig file on IGV and this is the result:

The picture shows thare are only two picks, a bigger one aroud 1.000 kb and another one smaller aroud 2.000 kb base. This could mean that all the single mates are grouped and sparse.

I tried to analyze the reason of the presence of those peaks:



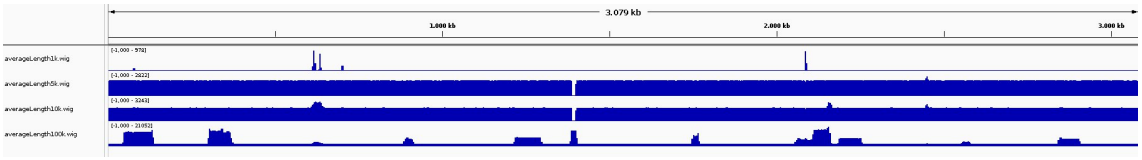
Around the bigger peak, the picture shows a long insertion, long approximately 5.000 bp.



Zooming in the lowest peak, it is obvious that, in reality, there are two peaks, which represent two short insertions, long approximately 1.000 bp.

4.4 Average length of the fragments

For this part of the project I created a track with the average length of the fragments, covering each genomic position.
I started from the physical coverage program and I instantiated an array of zeros sum_fragment_change with dimension equal to the length of the number of genomes.
If the pair of mates are both mapped and if the TLEN feature is included between zero to the maximum length, given in input, the program memorizes the starting position of every read (POS feature), increments by one the value of genome_change in that position and decrement by one the value of genome_length at the position of its mate (PNEXT) + 100.
In addition to all the other programs, it increments by TLEN the value of sum_fragment_change in position POS and, likewise, decrements by TLEN the value of sum_fragment_length in position of its mate (PNEXT) + 100.
Then it closes the SAM file and, through a for cycle, it computes the variable current_coverage as always and it also computes the variable coverage_sum, which is the comulative sum of all the values in sum_fragment_change.
At the end, if current_coverage is positive, it will be printed, divided by current_coverage. Otherwise, it will printed -1.



The only negative values we can observe is -1 and it indicate a lack of coverage.
As before, a maximum value of 1.000 is too low for making proper considerations.
When the maximum value is 5.000 or 10.000, the results are very similar and we can make the same analysis.

As it happened for the physical coverage, there is a delection, which wasn't noticed when the maximum size is too big.
The more increase the size of the fragments, the more the overlaps become more frequent.

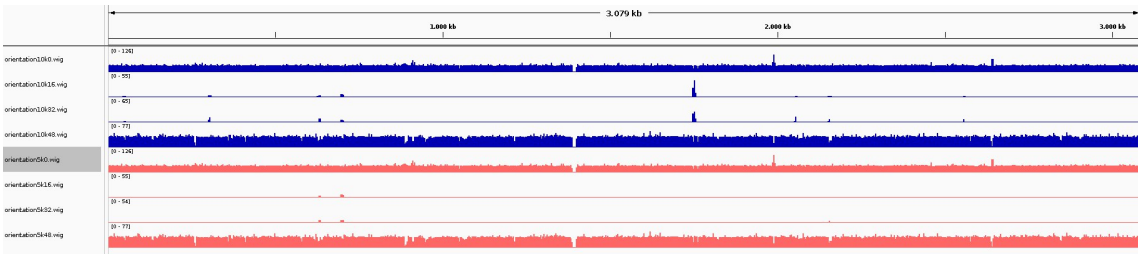
4.5 Orientation

As a final analysis, I considered the orientation of the reads for every genomic position from the bits 16 and 32 of the flag.

In order to do that, I modified the physical coverage file and, for every line in the SAM file not starting with @, it makes a list with all the separated features, expoliting the tabulation.
Then, I checked if the pair of mates are both mapped, if the TLEN feature is included between zero to the maximum length, given in input, and if the orientation is correct, through the FLAG bitwise.

I computed the variable od current_coverage of the reads from left to rigth as before.

For this track, I tried this program eight times: four times with maximum value 100.000 and orientation FLAG equal to 0, 16, 32, 48 (in blue), and other four times with the same orientation FLAGs, but maximum value 5.000 (in pink).

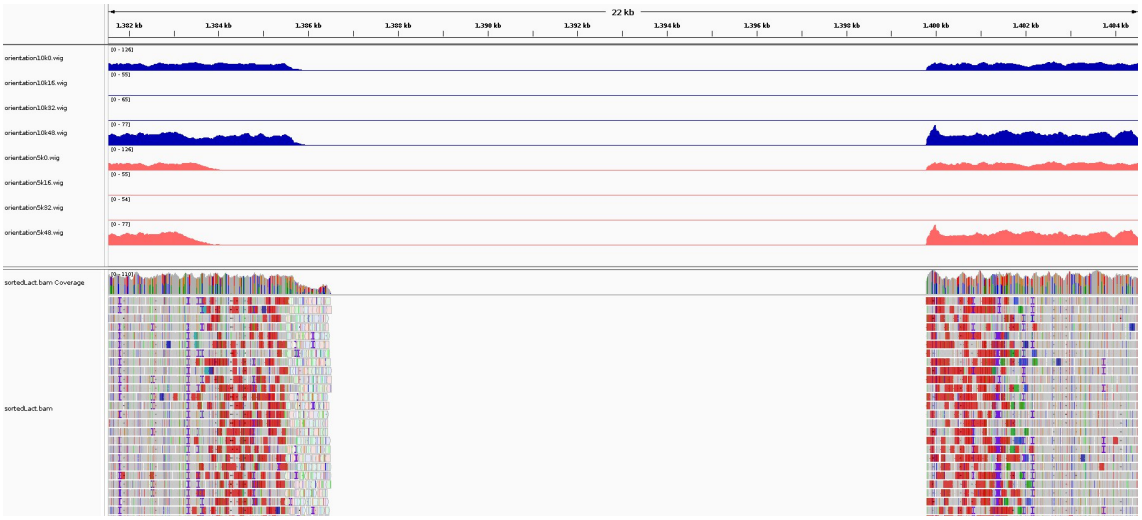


From the picture, we can do some considerations:

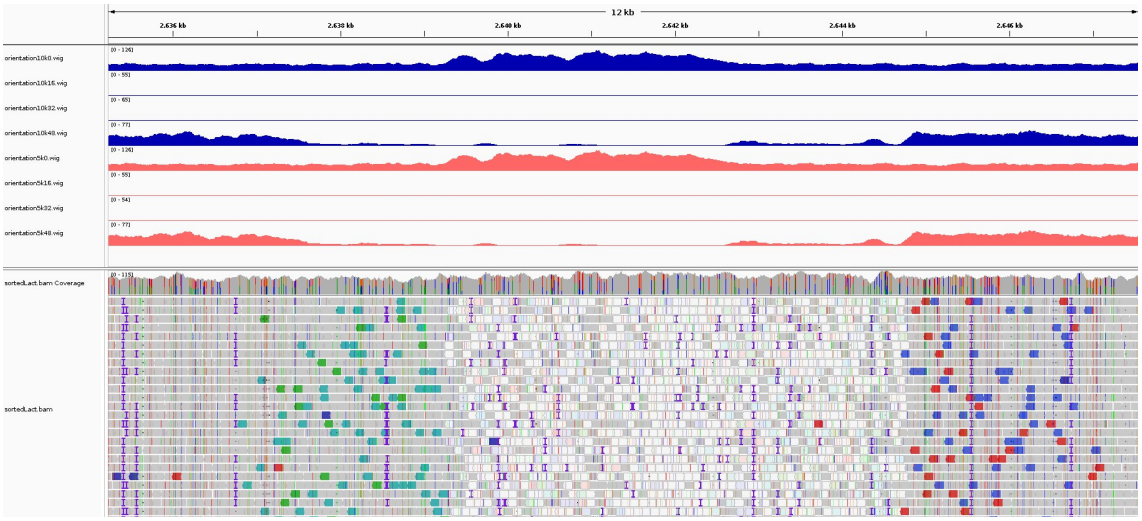
- The maximum value, in this track, is not important, because the results are very similar from each other;
- The results with orientation 16 and 32, so the intermediate values, are extremely similar from each other and their line is almost flat to zero;
- The results with orientation 0 and 48, so the values in the extremes, are different from each other and they reach the zero value just in a small spot.

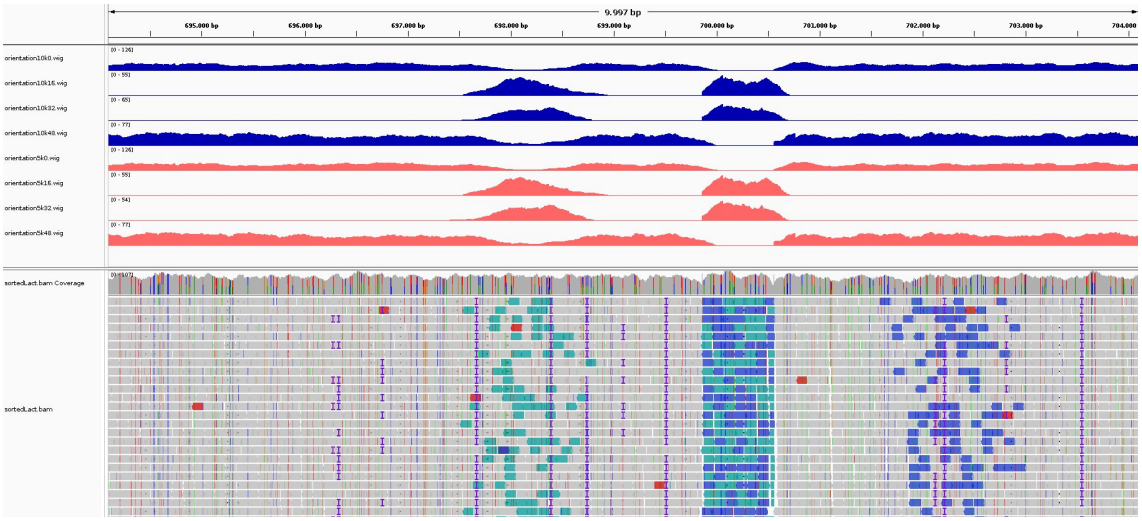
Normally the orientation is 0 or 48, so forward-forward or reverse-reverse, respectively.
There are some peaks also in the 12 and 38, this means that something is in the wrong orientation.

The zero spot in the 0 and 48 orientation, instead, denotes a long deletion:



When all the peaks are low, even in the 0 and 48 orientations, or when there are big peaks in the 16 and 32 ones, it is an indication of tandem duplications and long inversion reads.





5 Conclusion

We have just seen that file fasta file of Lactobacillus casei was many mismatchs, unmapped regions and structural variations.

- we can noticed the majority of the deletions are close to unmapped regions;
- the majority of the insertions are close to overlaps of genomes and when just one of the pair of mates are mapped;
- the majority of the tandem duplications and invertions when the FLAG bitwise with the orientation information is or 16 or 32.

References

Feuk, L., Carson, A. R., and Scherer, S. W. (2006). Structural variation in the human genome. *Nature Reviews Genetics*, 7(2):85–97.

Li, H. and Durbin, R. (2009). Fast and accurate short read alignment with burrows–wheeler transform. *bioinformatics*, 25(14):1754–1760.

Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., Marth, G., Abecasis, G., Durbin, R., Subgroup, . G. P. D. P., et al. (2009). The sequence alignment/map (sam) format and samtools. *Bioinformatics*, 25(16):2078–2079.

Thorvaldsdóttir, H., Robinson, J. T., and Mesirov, J. P. (2013). Integrative genomics viewer (igv): high-performance genomics data visualization and exploration. *Briefings in bioinformatics*, 14(2):178–192.