Finding structural variations in a genome after comparison with the reference genome *Lactobacillus casei*

Emma Roveroni 2058618

January 2023

Indice

1 Introduction

The aim of this project is to find structural variations in a genome of a bacterium named *Lactobacillus casei* after the comparison with a reference genome. More specifically, the objective is the "resequencing" of a smaller bacterium, called *lact.sp*, and finding out if it has genomic structural variations.

Structural variations are genomic alterations that can be involved in genetic conditions, such as disease or phenotypic variation. The structural variations that are expected to be found are deletions, insertions, and inversions: deletions involve the removal of a segment of DNA, insertions involve the addition of a segment of DNA and finally, inversion is a large section of DNA that is reversed in the subject genome compared to the reference genome. In particular, a small and a long structural variations, for each type mentioned before, are expected to be detected.

2 Softwares

The tools involved in the realization of this project are the following:

1. **BWA** - Burrow-Wheeler Aligner
2. **Samtools**
3. **IGV** - Integrative Genomics Viewer

The coding language program used is *Python v3*.

2.1 BWA

BWA is a software package for mapping genome sequences against a large reference 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 1Mbp. BWA-MEM and BWA-SW share similar features such as long-read support and split alignment, but BWA-MEM, which is the latest, is generally recommended for high-quality queries as it is faster and more accurate. BWA-MEM also has better performance than BWA-backtrack for 70-100bp Illumina reads. In this project, only the BWA-MEM algorithm has been used.

2.2 Samtools

Samtools is a program for interacting with high-throughput sequencing data. Samtools provide various utilities for manipulating alignments in the SAM format, including sorting, merging, indexing and generating alignments in a per-position format. SAM (which stands for Sequence Alignment/Map) format is a generic format for storing large nucleotide sequence alignments. 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.

In the SAM format, each alignment line typically represents the linear alignment of a segment, and each line consists of 11 or more TAB-separated fields, which are shown in the following table.

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.

- **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. Each bit is explained in the following table:

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. If @SQ header lines are present, RNAME (if not '*') must be present in one of the SQ-SN tag. 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**: 1-based leftmost mapping POSition of the first CIGAR operation that "consumes" a reference base. The first base in a reference sequence has coordinate 1. 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. It equals 10 log10 Prmapping position is wrong, rounded to the nearest integer. A value 255 indicates that the mapping quality is not available.

- **CIGAR**: CIGAR string. The CIGAR operations are given in the following table (set '*' if unavailable):

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

- **RNEXT**: Reference sequence name of the primary alignment of the NEXT read in the template. For the last read, the next read is the first read in the template.

- **PNEXT**: 1-based Position of the primary alignment of the NEXT read in the template. Set as 0 when the information is unavailable. This field equals POS at the primary line of the next read. If PNEXT is 0, 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 (same as the quality string in the Sanger FASTQ format). A base quality is the phred-scaled base error probability which equals 10 log10 Prbase is wrong. 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.

2.3 IGV

The Integrative Genomics Viewer (IGV) is a high-performance, easy-to-use, interactive tool for the visual exploration of genomic data. It supports flexible integration of all the common types of genomic data and metadata, investigator-generated or publicly available, loaded from local or cloud sources.

3 Primary Analysis

The first part of the project is centered on mapping and aligning the reads on the reference genome. Then an initial analysis of the results is performed through IGV.

3.1 Mapping and Alignment

For the realization of this objective, just BWA and Samtools are used. The first step is to use BWA to index the *Lactobacillus casei* fasta file: this allows to search more efficiently the genome during sequence alignment.

```
$ bwa index Lactobacillus_casei_genome.fasta
```

After that, there is the need to perform the alignment apllying BWA-MEM algorithm, using both reads file. The results is saved in a *.sam* file, named *lact.sam*.

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

In order to upload the results of the alignment on the IGV software, the *.sam* file need to be converted in a *.bam* file, which is the compressed binary version of a SAM file. To do this, samtools must be used.

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

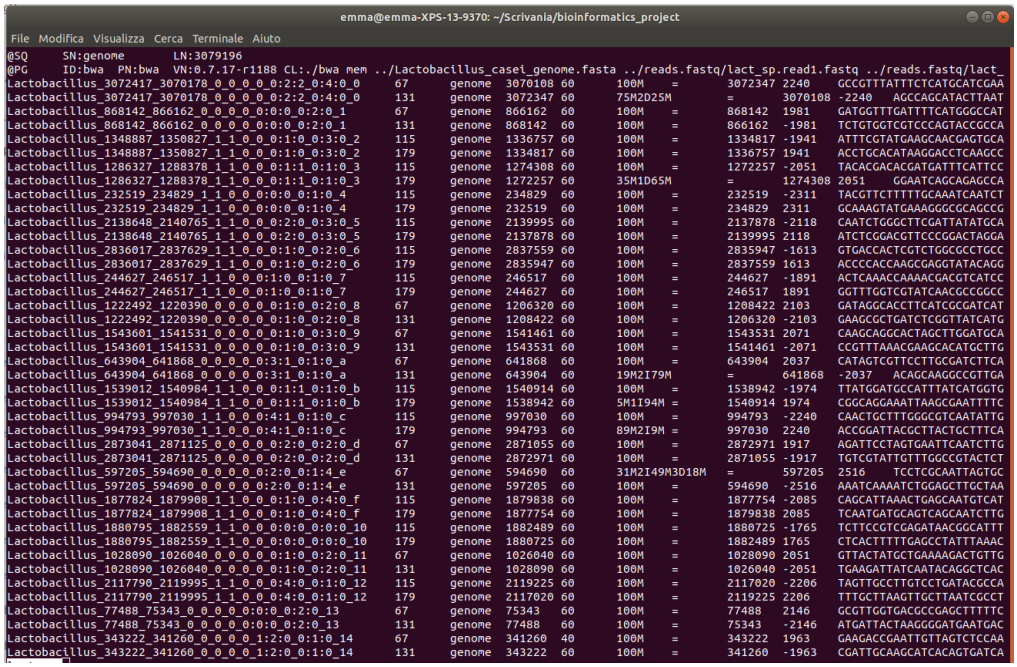
The output is the *lact.bam* file, which needs to be sorted. Once it is sorted by position, it is also more compressable, so the dimension is even lower.

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

Finally, also the *.bam* file needs to be indexed, giving as output the *.bai* file.

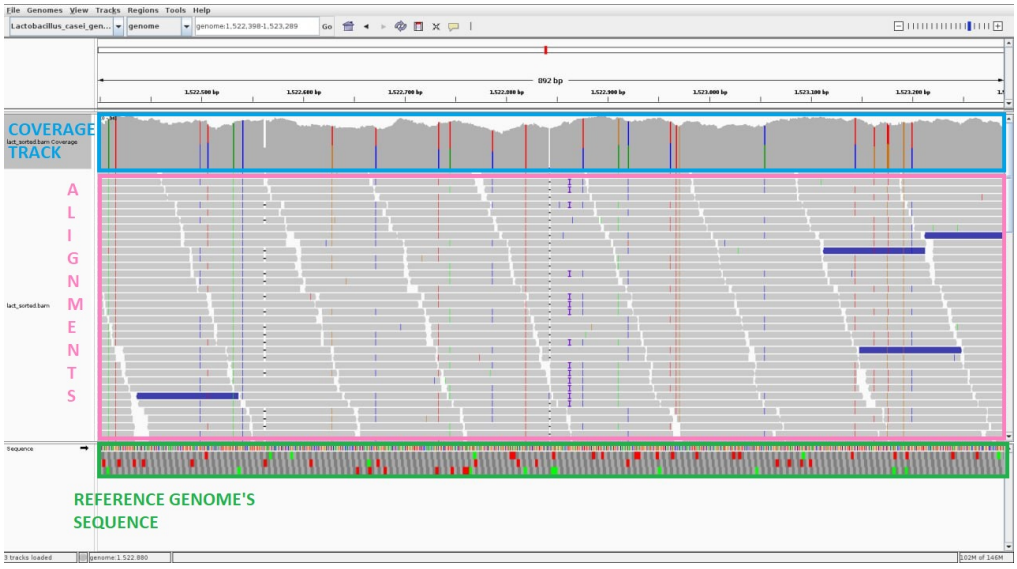
```
$ samtools index lact_sorted.bam
```

Here an example of the *lact.sam* file:



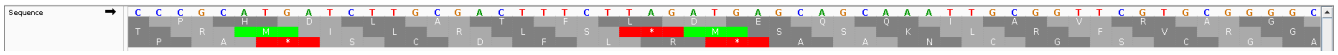
3.2 Vizualization on IGV and analysis

Both Lactobacillus casei and the *lact_sorted.bam* files are inserted on the IGV tools, the former as Genome and the latter as File. This is the view given by IGV:



The higher bar is the coverage track, which is the coverage of the reads that allows seeing how the reads are covered: each colored line is a mismatch while the peaks represent a large

number of overlaps. In the middle, there is the alignment track which shows the reads, and finally, there is the reference genome sequence, with the corresponding translations, as can be better seen in the next figure.



Furthermore, IGV highlights anomalous situations in the reads with different colors and symbols, like deletions and insertions. Now some examples of these events.

3.2.1 Insertions

In IGV insertions are shown with a capital I colored in purple, as the one that is shown in the figure below.

Another way to detect insertions with IGV is by using inferred insert size: an inferred insert size smaller than expected can be a possible evidence of insertion, and it's pointed out with a blue bar, like in the figure above. Along the reads, a lot of these sing of insertion can be found.

3.2.2 Deletions

In IGV deletions are shown with a black horizontal line in an unmapped region, as the one that is shown in the figure below.

Analogously from insertions, deletions can be detected also by using inferred insert size: an inferred insert size that is larger than expected can be a possible evidence of deletion, and it's pointed out with a red bar, like in the figure above.

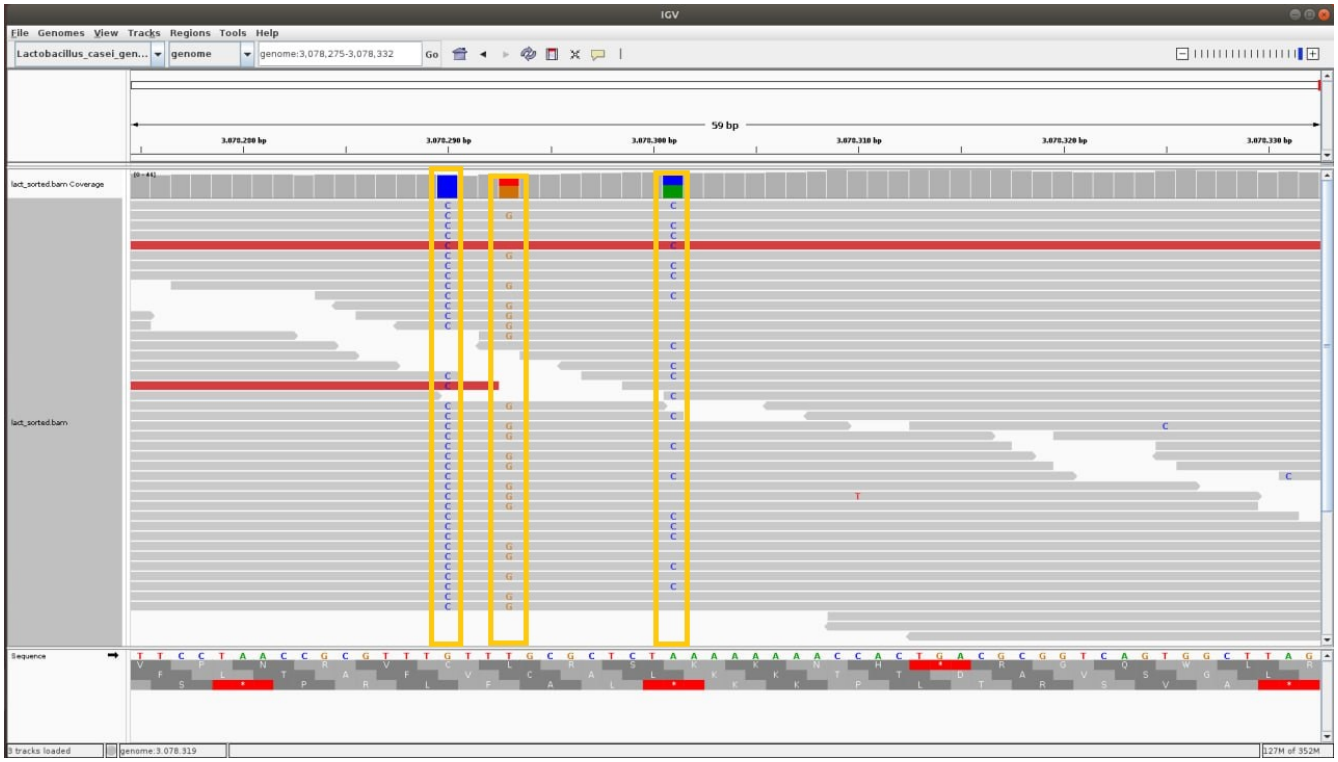
3.2.3 Inversions

Even inversions are detected by IGV with a teal bar like the ones shown in the picture below.

Another type of colored bar can be seen in the picture above, the green one. This kind of bar detects tandem duplications, which are when a large section of DNA is duplicated and inserted into the genome next to the original sequence.

3.2.4 Mismatches

Another thing that can be seen thanks to IGV are mismatches, like the ones shown in the picture below: in this example, in the first highlighted column, on the reference genome there is a guanine, but most of the reads have a cytosine. This event is very common along the reads.

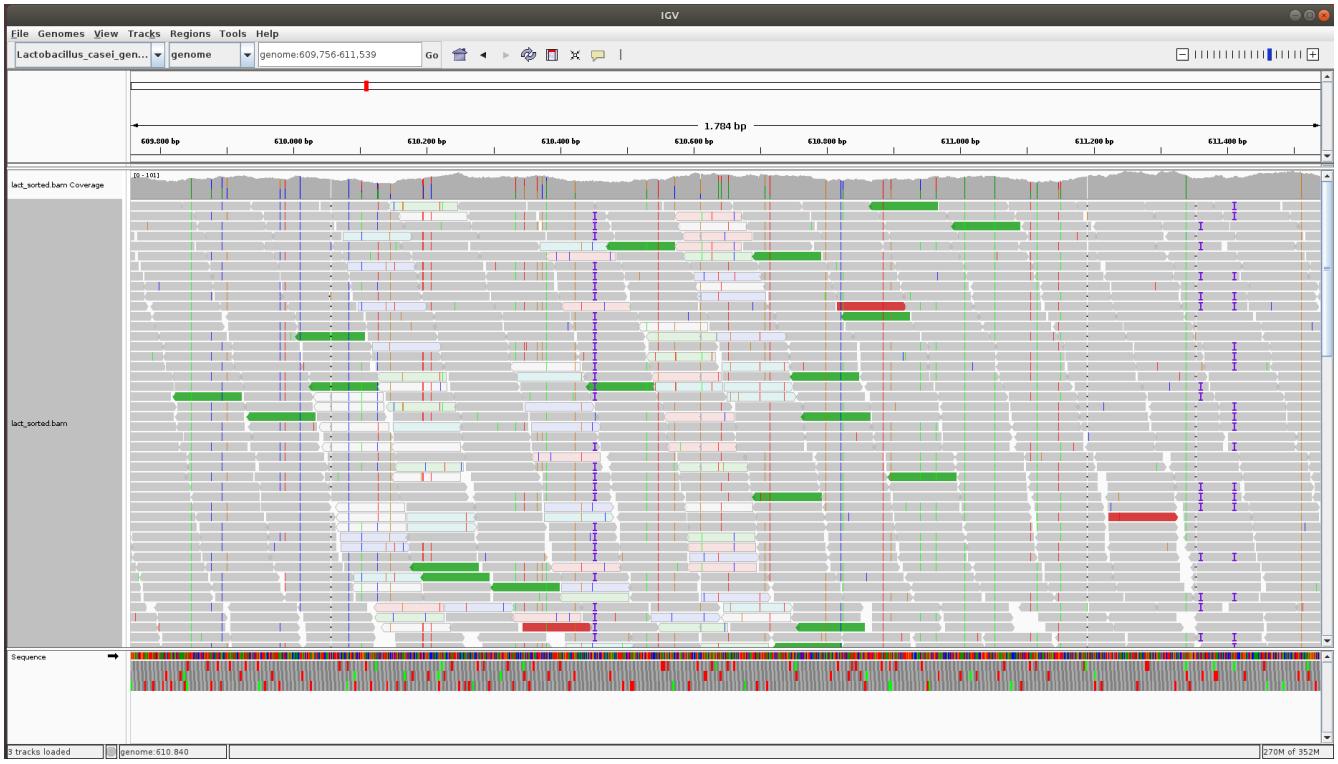


3.2.5 Unmapped regions

When a situation like the one shown in the following picture is encountered, it means that there is an unmapped region of the genome and so this point out some gaps in the reads.

3.2.6 Mapping quality

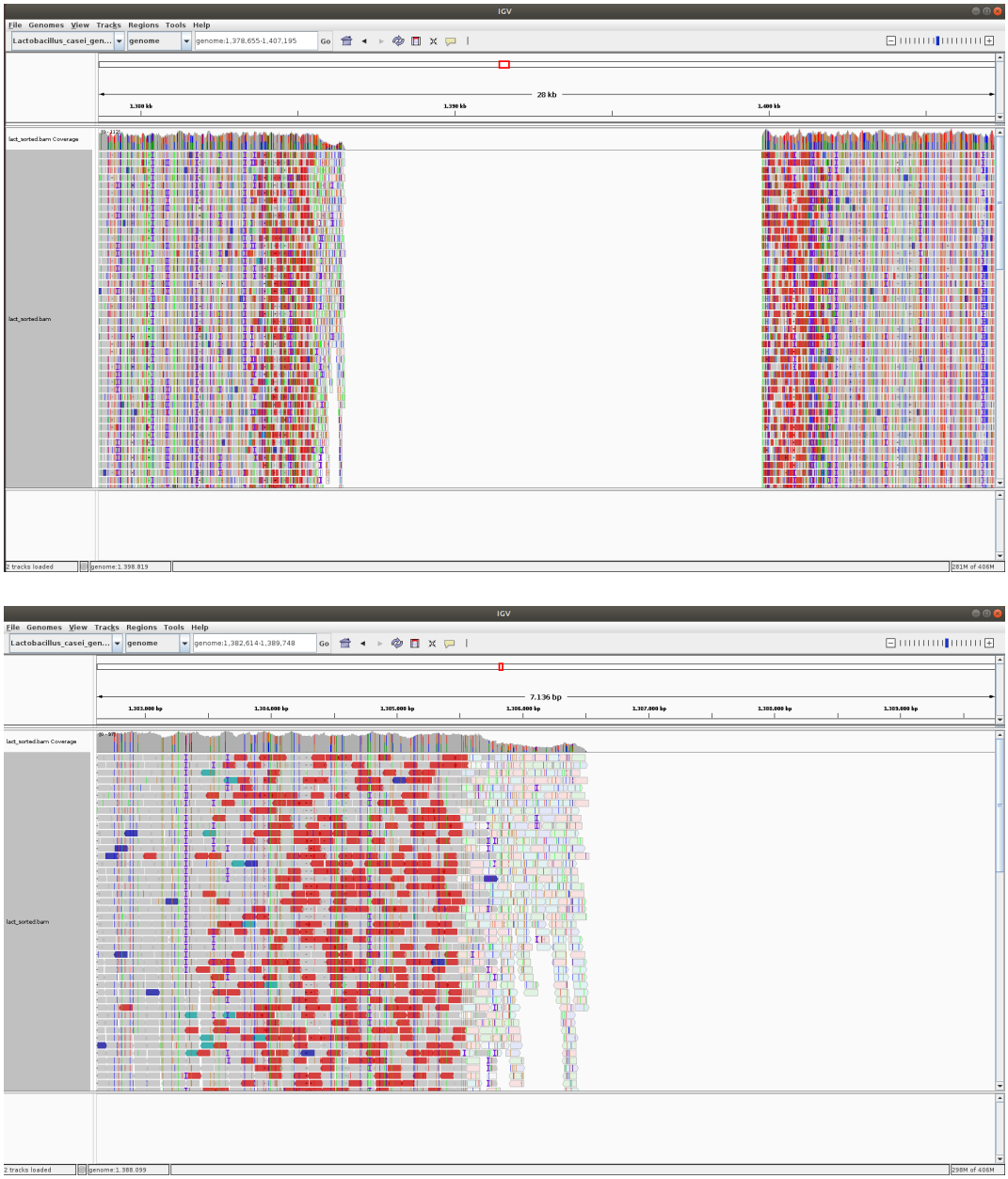
Some of the reads are hollow instead of gray, this is because not all reads have the same mapping quality: the reads that have a low mapping quality are marked as hollow. An example of this event can be seen in the following picture.



3.2.7 Observations

By searching along the reads with IGV, some observations can be made:

- Even if not many, reads of low quality are present and they are typically found together;
- also low quality reads are marked as structural variations;
- structural variations are present in the reads. More specifically insertions, deletions, inversions, and tandem repeats can be found (as shown in the previous figures);
- structural variations are most of the time not by oneself;
- from position 1.386.526 to position 1.399.789, the reference genome is not covered at all. This indicates a long deletion. What's more, it's interesting to notice that before this long deletions, there is a region rich in deletions inserts, followed by a region rich in low-quality reads. Even after the long deletion, a region rich in deletions inserts is present. This situation is going to be supported by the following pictures.



The first one is the long deletion, then there a zoom in the regions before the long deletion.

4 Coverage Tracks

In order to help with the detection of structural variations, different coverage tracks have been developed. The coverage tracks built are:

1. sequence coverage;
2. physical coverage;
3. multimappers coverage;
4. clipped reads coverage;
5. average length coverage;
6. orientation coverage.

The programming language used to develop these tracks is *Python*. The structure of the code is similar for all the tracks, with just some variations. The code is generally divided into three parts:

1. gets the argument from the command line and sets up the main variables;
2. reads the entire SAM file and computes the coverage at every single base;
3. outputs the wig file.

The aim of the code is to compute a value for each position of the genome; in this case, since the genome has a length of 3079196, it is necessary to compute 3079196 values. The genome length is one of the input terms of the program, together with the SAM file. Firstly, a list of 3079196 "counters", one for each position of the genome, is set:

```
genome_change = [0]*genome_length
```

The other input, as mentioned before, is the SAM file, which is going to be scanned, one alignment at a time, through a *for* cycle. In this cycle, the lines starting with '@' are going to be ignored, while all the others, which are the actual alignments, are going to be split into fields: in this way each field can be independently accessible. The next step is to check through these fields some conditions that are related to the type of coverage track that is performed. In particular, the bits of the **FLAG** field are used. Finally, there is another *for* cycle in order to create the wig file, so the coverage track, by computing the sum of every value in genome change and printing this number for every position of the genome.

4.1 Sequence coverage

The sequence coverage objective is to quantify the number of reads of *lact.sp* that map for each position of the *Lactobacillus casei* genome.

4.1.1 Code

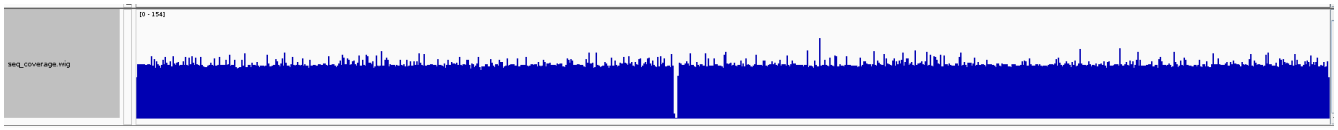
The condition that needs to be checked to realize the sequence coverage is if the segment is mapped or not. In order to do that the **FLAG** field of the SAM file using bitwise. In this case, the bit to check is the third bit of the flag, which has a value of 4 and indicates "segment unmapped". Therefore it must be 0 in order to be mapped

```
if((int(fields[1]) & 4) == 0)
```

If the genome is mapped, the starting position of every read (POS features) is memorized 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 (end of the reads). After that, there is the creation of the wig file, as explained before.

4.1.2 Results and observations

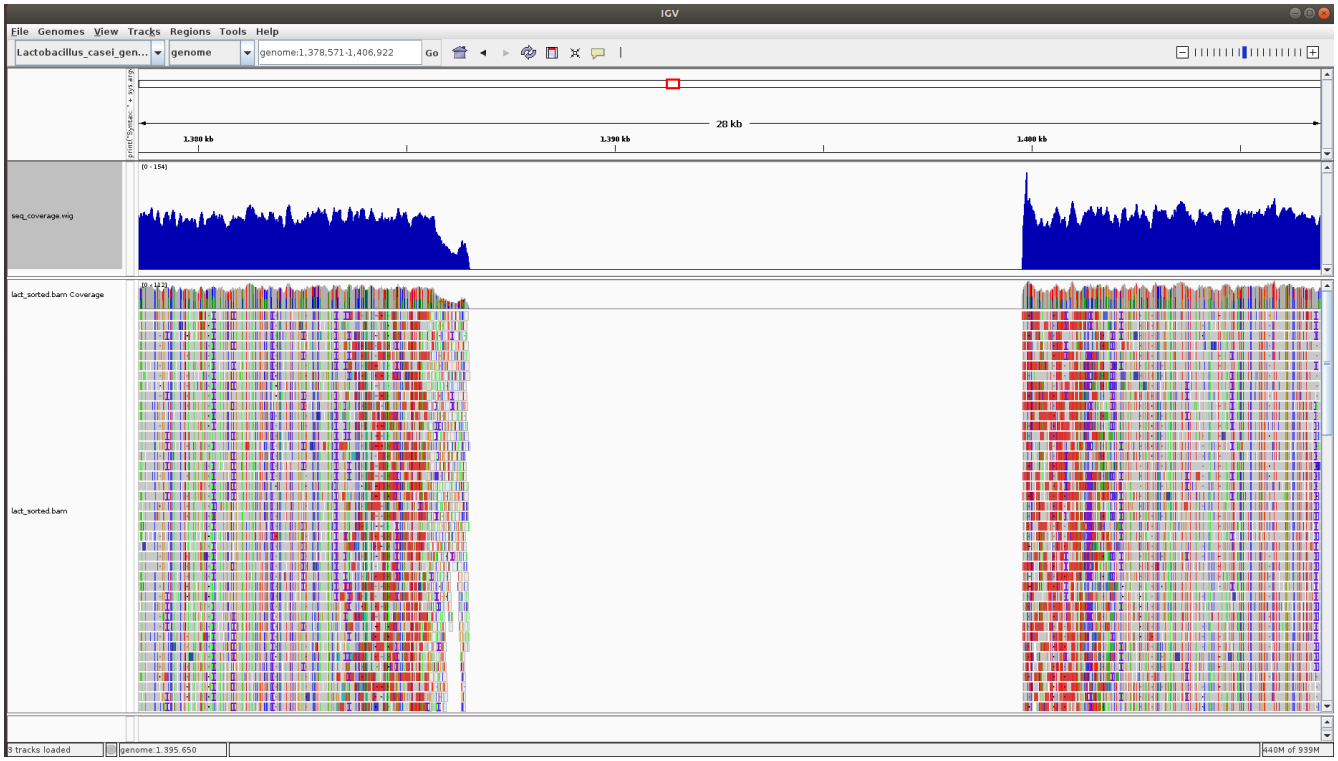
The sequence coverage track is the one shown in the following picture:



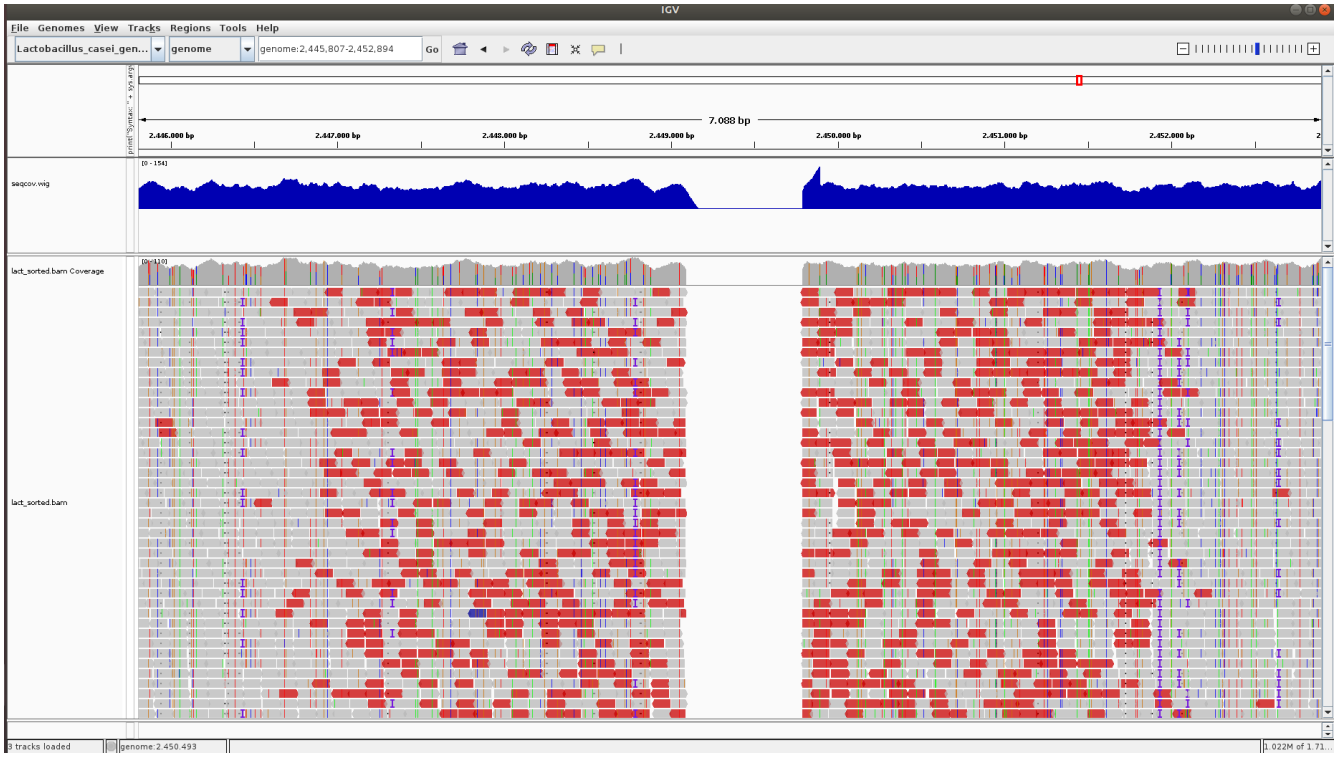
The behavior of the sequence coverage track is very similar to the one of the coverage track (the one in gray, which is automatically created when loading the *.bam* file on IGV), as can be seen in the next image.

The difference lies in the range in which the number of reads mapped at each position are. This is probably due to the fact that in the sequence coverage track reads 100 bases long are considered.

From the picture below we can see that negative peaks indicate long deletions:



Here, another deletion, but smaller than the previous one, can be found in the correspondence with a negative peak.



4.2 Physical coverage

The physical coverage is given by the number of fragments of *lact.sp* that map for each position of the *Lactobacillus casei* genome. For this reason, it is necessary to check for both the read and its mate. Furthermore, it was necessary to look just for the read that was going in one direction (left to right).

4.2.1 Code

A new input value is added: the maximum length of a fragment. Apart from that, the code is similar to the one of the sequence coverage. The only things that have been changed are related to the conditions that must be checked when analyzing the alignments in the SAM file. First is verified that the read (bit 4) and its mate (bit 8) are mapped (both bits set to 0), using again the FLAG field:

```
if((int(fields[1]) & 12) == 0)
```

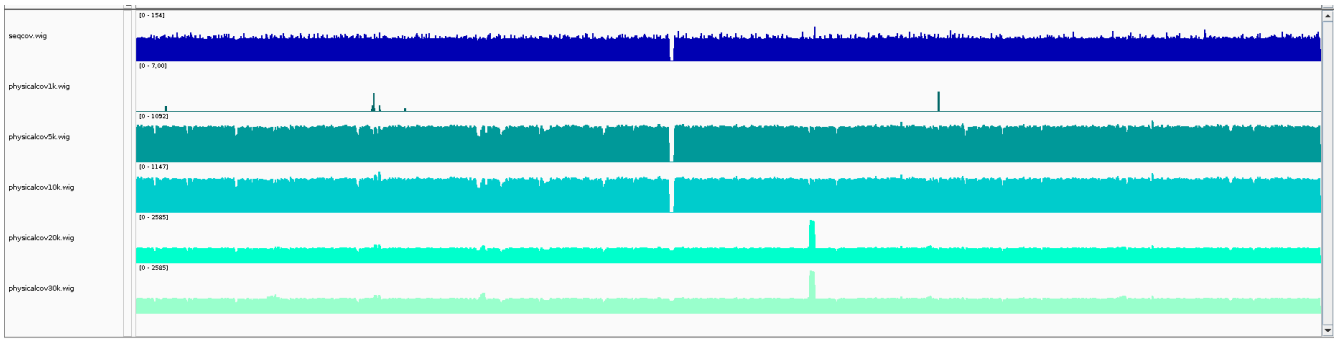
Then the field TLEN is checked: this value must be positive and lower than the maximum length passed as input:

```
if ((int(fields[8]) > 0) and (int(fields[8]) < max_fragment_size))
```

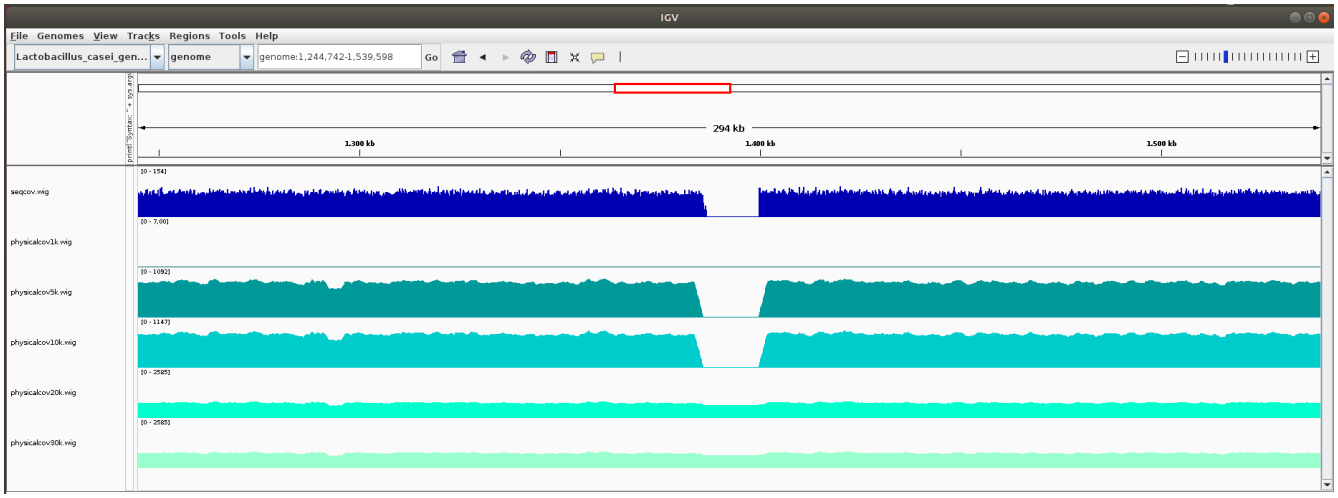
If all these conditions are true, then the program behaves like before and computes the coverage. Different maximum lengths have been tried: 1000, 5000, 10000, 20000, 30000.

4.2.2 Results and observations

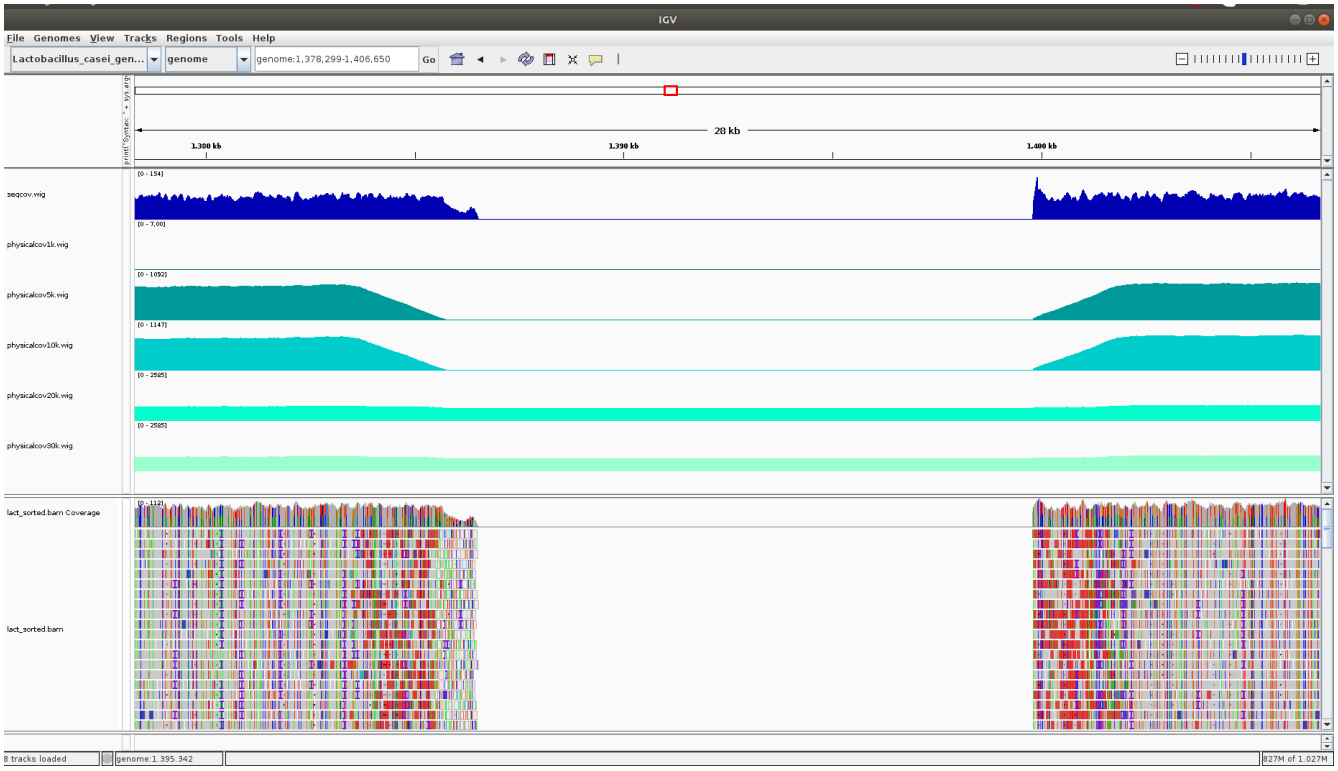
Here it is possible to see the five physical coverage tracks: the first one is the sequence coverage, then the physical coverage with 1k max length, right after the physical coverage with 5k max length, then the physical coverage with 10k max length, the physical coverage with 20k max length and finally the physical coverage with 30k max length.



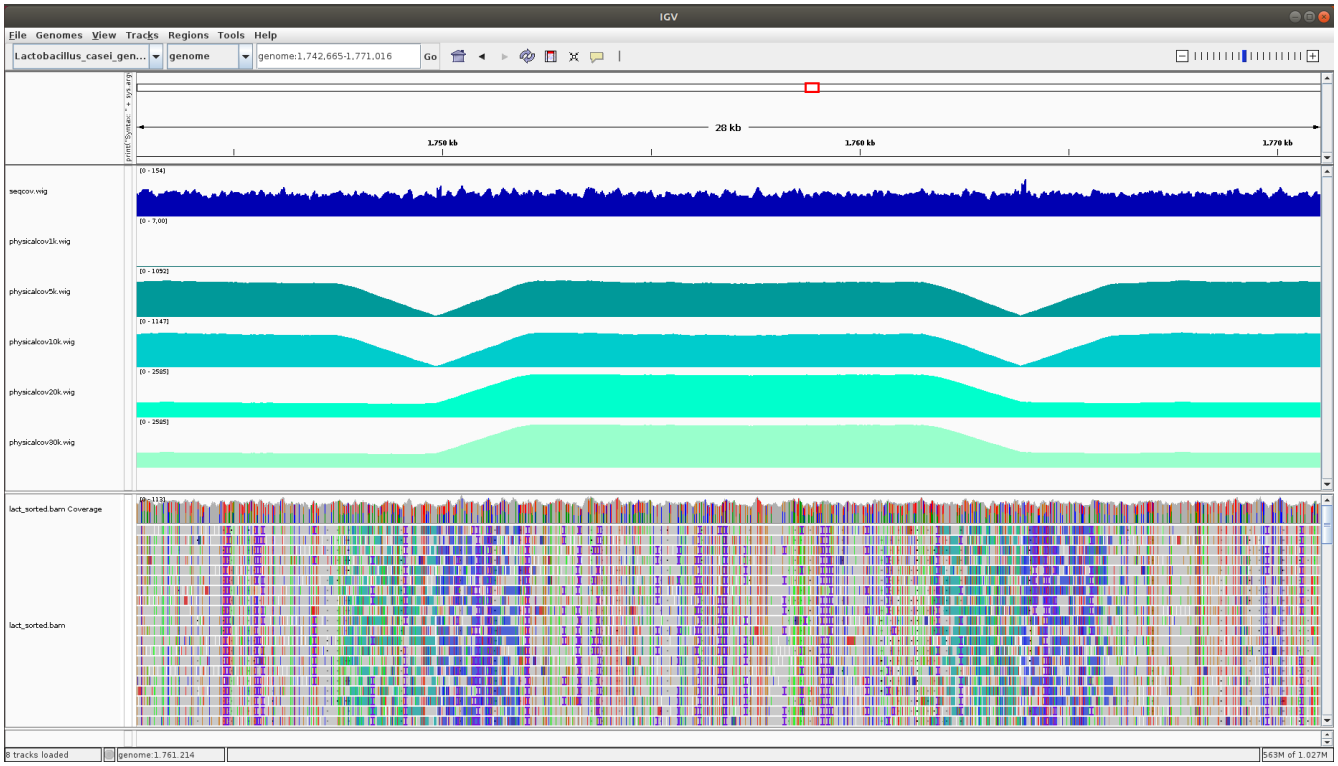
For what regards the range of the 1k max length track, it's possible to see that is very low: it goes from 0 to 7. In fact, the track is almost everywhere equal to zero. The range of the others tracks, improved a lot. The behavior related to the 5k and 10k tracks is very similar, and the same goes for the tracks that have 20k and 30k as max lengths. What can be noticed regarding the 5k and 10k physical coverage is that there is in both an empty spot, indicating that in that area the length of the segments overcomes the max length of both 5000 and 10000. Instead, by looking at the 20k and 30k cases, it's possible to notice that the range is the same for the two tracks (from 0 to 2585) meaning that, independently from the max fragment length, if it is larger than 20k, for each base it's possible to count 2585 fragments at maximum. What's more in these two tracks there isn't the empty spot identified by the other two tracks, as it's possible to see better in the next image.



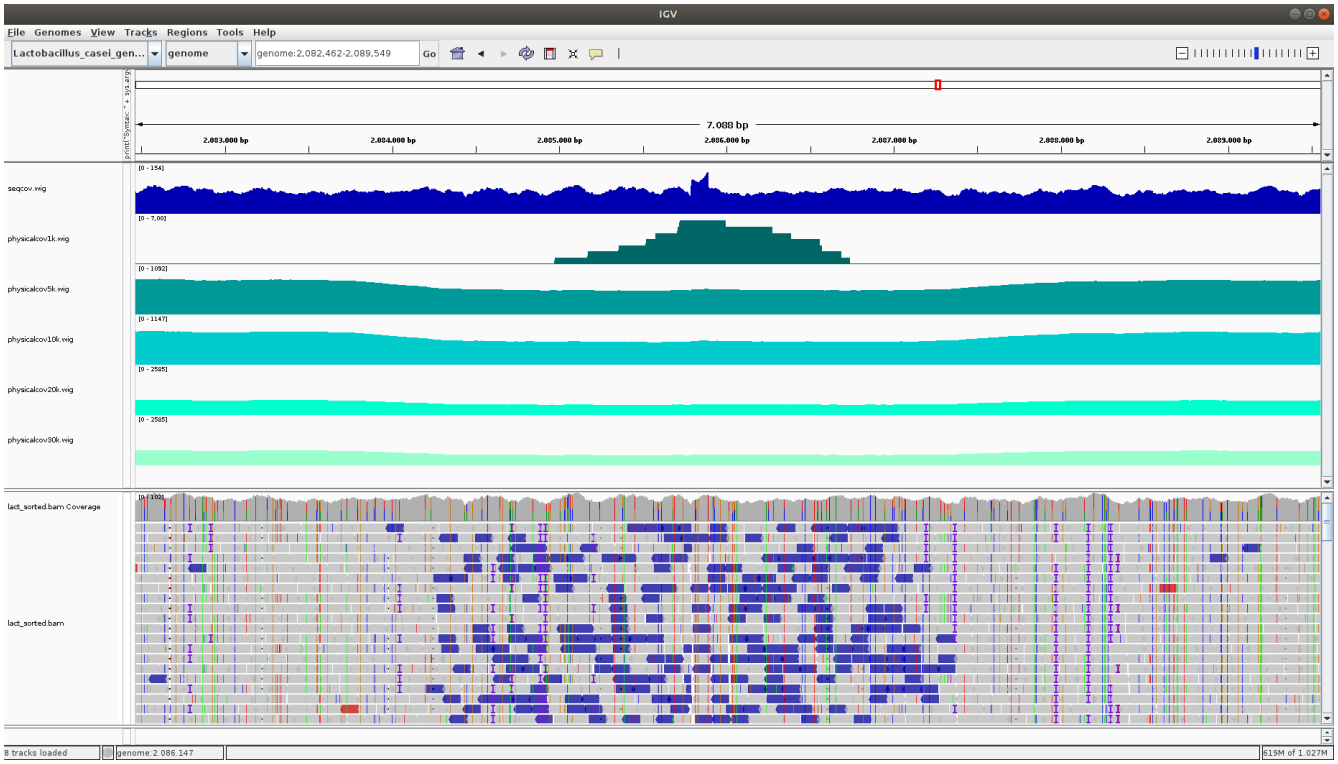
In the next figure it's possible to see that in correspondence to the gap identified by the 5k and 10k coverage, there is a long deletion (the same discovered with the sequence coverage). The gap is not noticed by the 20k and 30k tracks because probably, with a max length of 20k and 30k, the segment are too big and not allow to the program to find this gap.



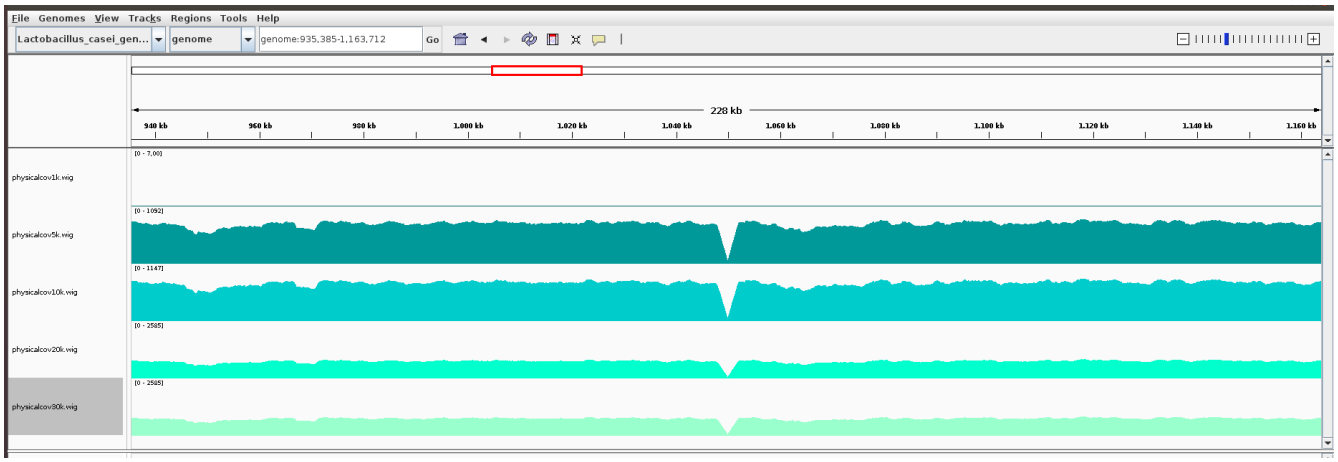
Instead, by looking in the zone identified by the peak present in 20k and 30k tracks, it's possible to see a region with a long inversion, identified by an increase and decay in the track, as shown in the next figure.



The big peak on the right which is identified by the 1k tracks is found in the correspondence of a long insertion, which is also indicated by the other physical coverages by a drop, as shown in the next picture.



The same drop seen for the long insertion, can be seen for a small one:



4.3 Multimappers coverage

Genome sequences have sometimes patterns that involve sequences that are repeated multiple times throughout the genome. Therefore they are a challenge in alignment. Multimappers are reads that will align to multiple locations across the genome. To try to solve the repeats problem, it's possible to create a multimapper coverage, exploiting the fact that BWA uses a special annotation for multimappers reads: the label *XA*.

4.3.1 Code

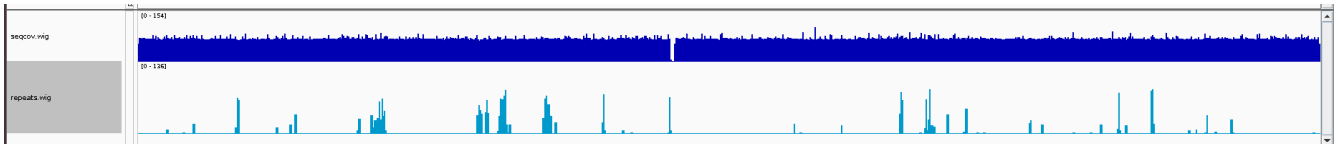
In this case a different input *.sam* file is given to the program that creates the sequence coverage. The new *.sam* file contains only the multimappers reads, so just the alignment that contains the label *XA*:

```
grep "XA:" lact.sam > repeats.sam
```

This file is then used as input to the same code for the sequence coverage.

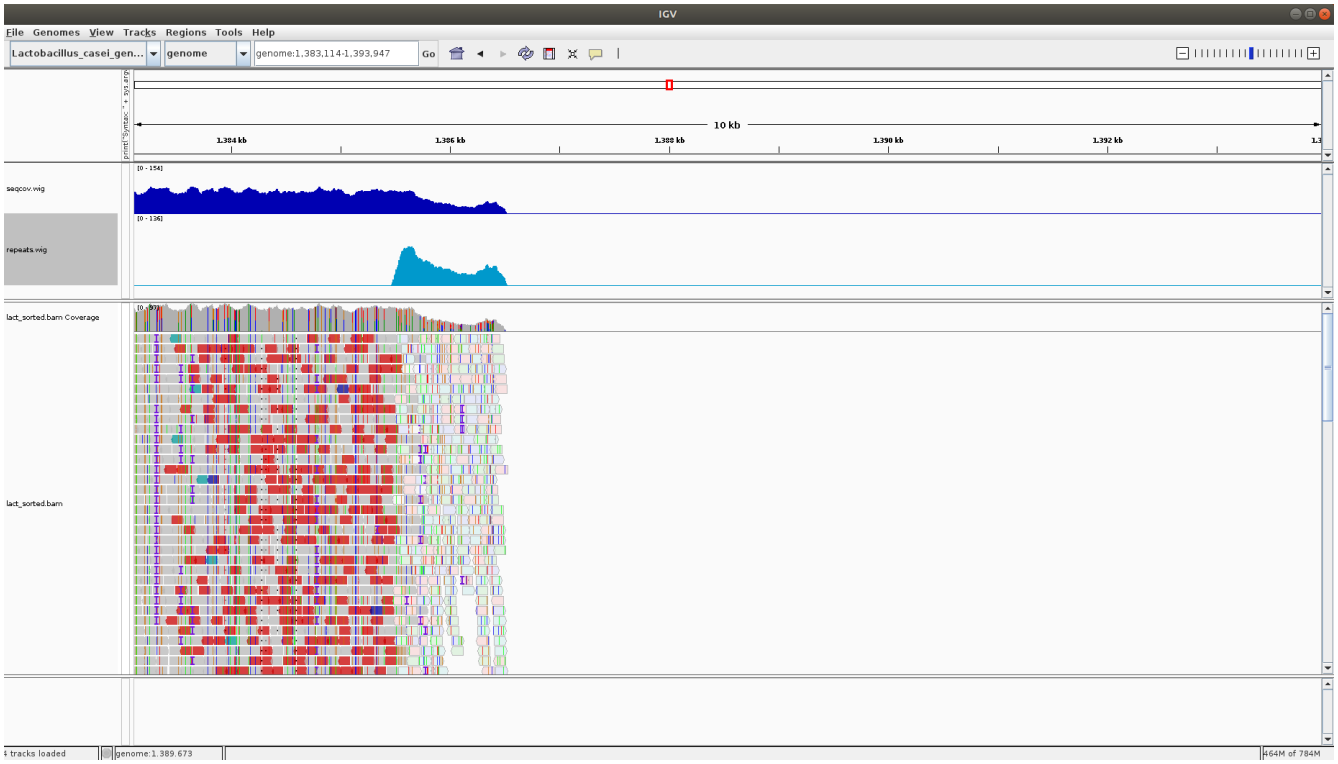
4.3.2 Results and observations

The repetitions track can be seen in the following picture. This track tends to be flatted to 0, except for the presence of multiple peaks.



As shown in next picture, one of the peaks is found in correspondence to a low quality region. Actually this phenomena is very frequent: a lot of the peaks are found often when there are low quality reads.

This phenomena happens also in the low quality region that precedes the long deletion, as shown in this figure.



4.4 Clipped reads coverage

The clipped reads coverage is a track with reads that map, but with the paired read that does not map. For this track is possible to use Hard and Soft clips, which can be easily identified because they are reported with H or S in the Cigar field in the SAM file.

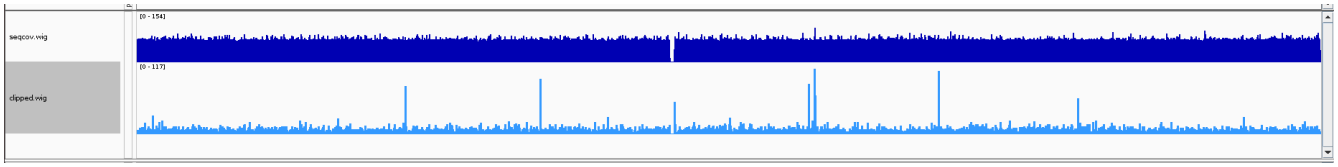
4.4.1 Code

The code for realizing this track is equal to the one used for the sequence coverage, except for the fact that it is necessary to verify, together with the check if the segment is mapped, if there are H or S in the CIGAR filed.

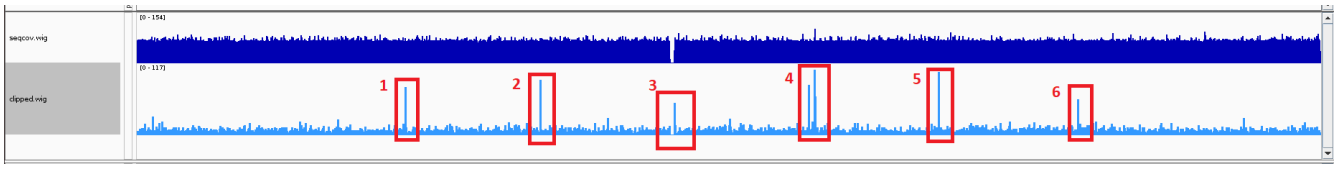
```
if((int(fields[1]) & 4)==0) and (("H" in fields[5]) or ("S" in fields[5])):
```

4.4.2 Results and observations

Very interesting results are given by the clipped coverage. In fact, it's possible to see form the image below, that this track identifies six peaks and almost each one of them is found in correspondence to a region with a structural variation.

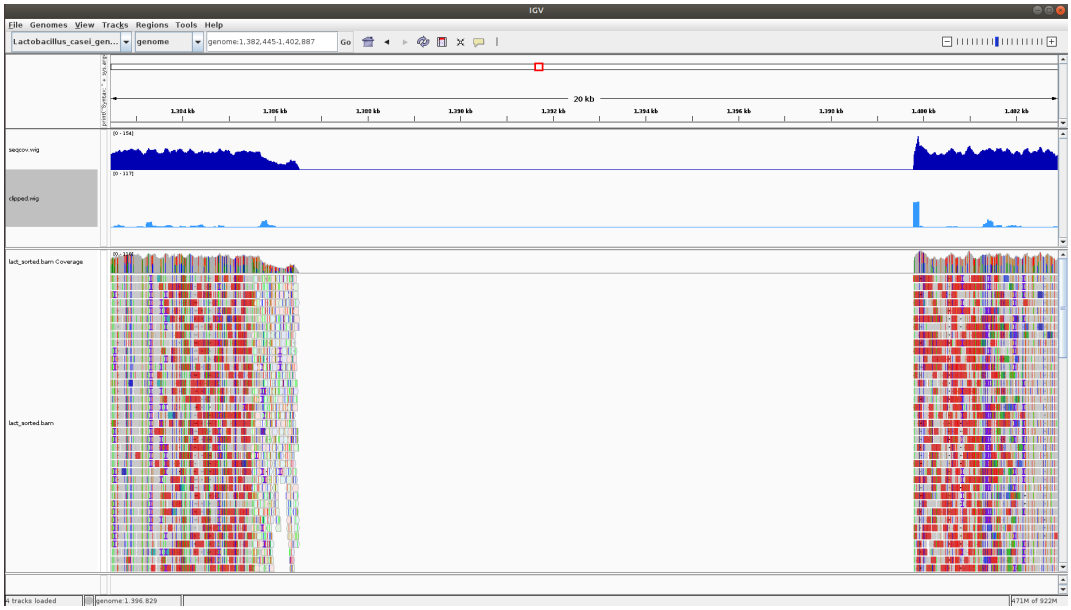


Here the peaks are highlighted and then are going to be studied in details later:



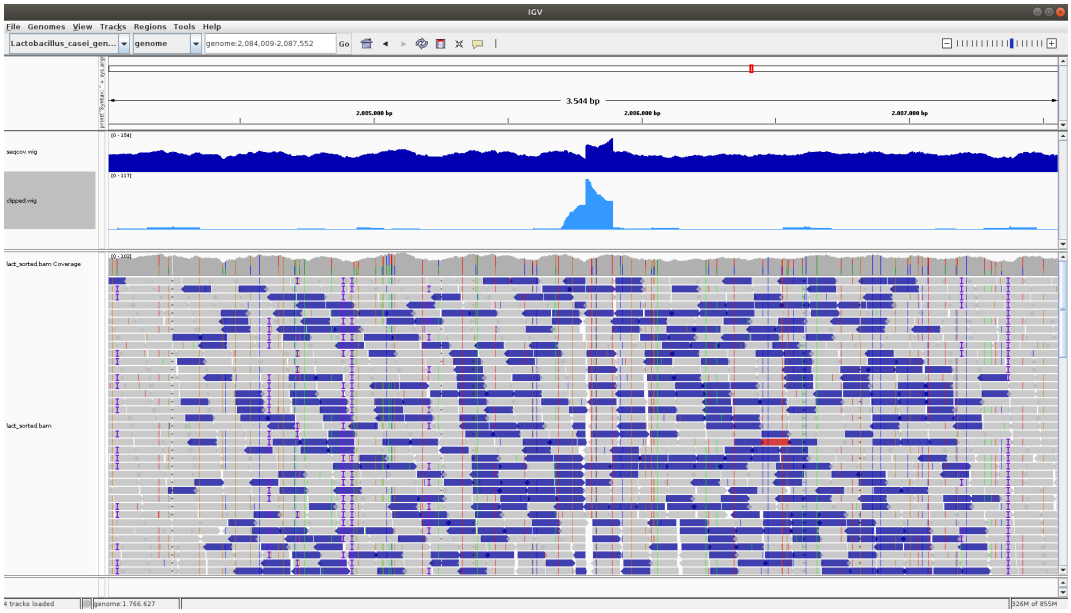
Peak 1: Looking closely to the first peak, it's possible to see that there are actually two peaks, which identified a region with a small inversion. This is displayed in the following picture.

Peak 3: The third peak, instead, is found in the same position where is present the long deletion, more specifically where the long deletion ends.

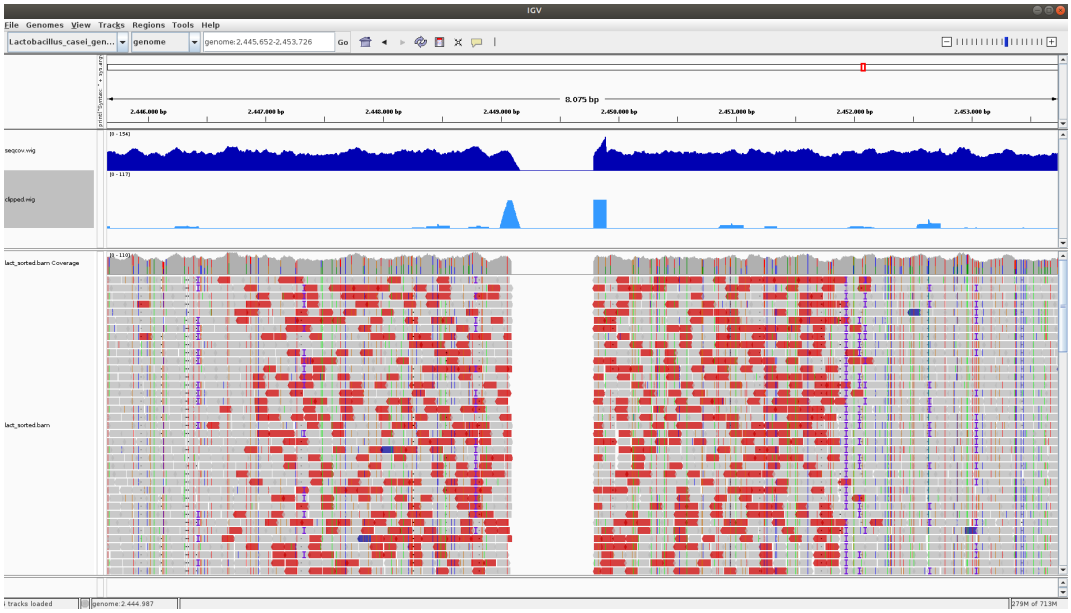


Peak 4: In peak number four, by zooming in, it's possible to see, that, as for peak 1, there are actually two peak, which identified a long inversion region, as shown here.

Peak 5: Focusing, instead, on peak number 5, a region with a big insertion can be detected.



Peak 6: Finally, peak number 6 is found in correspondence with a small deletion, as shown in the following picture.



4.5 Average length coverage

The average length coverage track can be used to find insertions and deletions. This means creating a track with the average length of the physical inserts covering each position of the genome. If there is a deletion the two resulting reads will map at a greater distance on the reference genome, while the opposite happens with an insertion. To create this track two values must be computed for each genomic position: the physical coverage and the sum of the length of the inserts at every position. Then for each genomic position, the sum of the length of fragments divided by the number of fragments must be saved in the wig file.

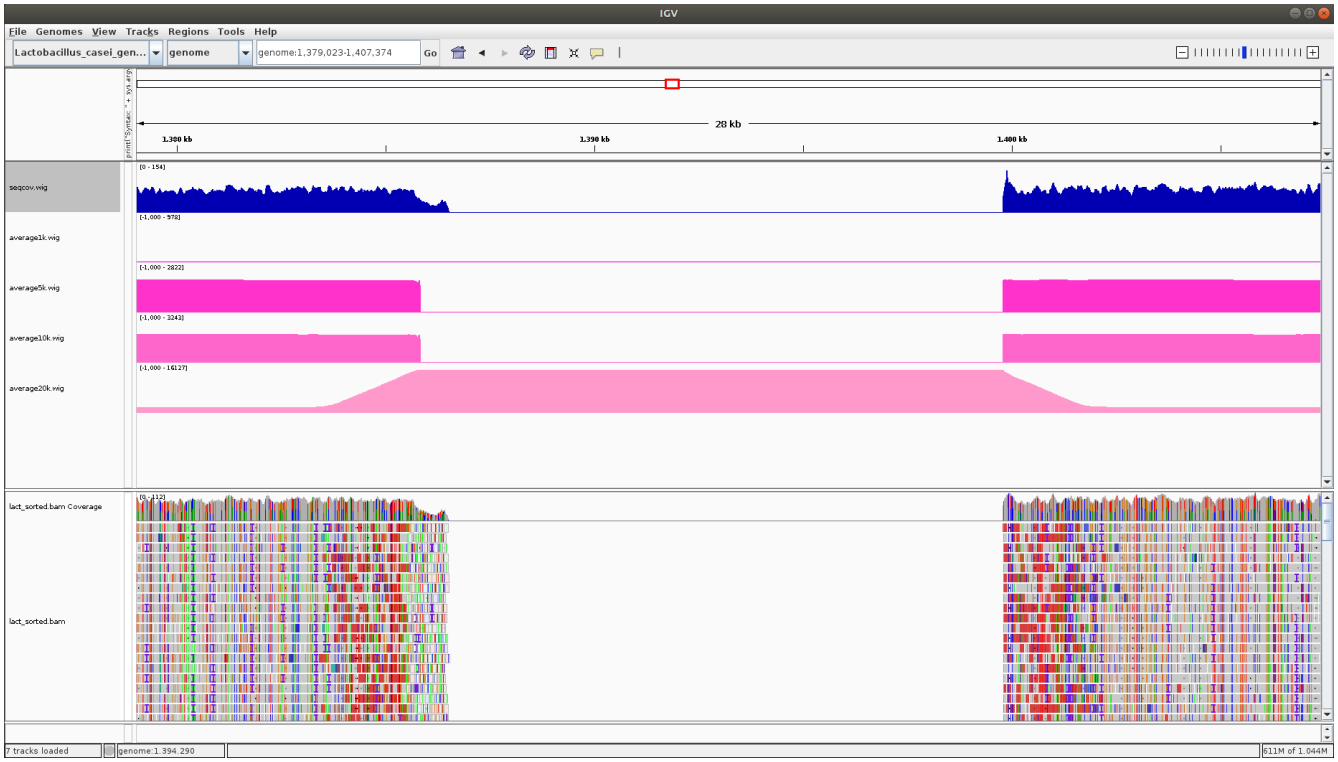
4.5.1 Code

In this case, the code is very similar to the one of the physical coverage: even here is required a maximum length as input, furthermore, the same conditions are verified. Here another array of zeros has been initialized with a dimension equal to the length genome. In this array, the value stored in each starting position is going to be incremented by TLEN value, and the value stored in each ending position is going to be decremented by TLEN value. After that, to create the wig file, the usual coverage (the number of reads that map) is computed together with the cumulative sum. If the value of coverage for a specific position is positive, the ratio between the sum of the length of fragments and the number of fragments is going to be printed in the wig file. Different maximum lengths have been tried: 1000, 5000, 10000 and 20000.

4.5.2 Results and observations

Here it is possible to see the four average length coverage tracks: the first one is the sequence coverage, then the coverage with 1k max length, right after the one with 5k max length, then coverage with 10k max length, and finally the one with 20k max length. As for the physical coverage, the track with 1k max length is flatten to 0 almost everywhere, except for just a couple of peaks. Instead, the ones with 5k and 10k, again, seems very similar between each other.

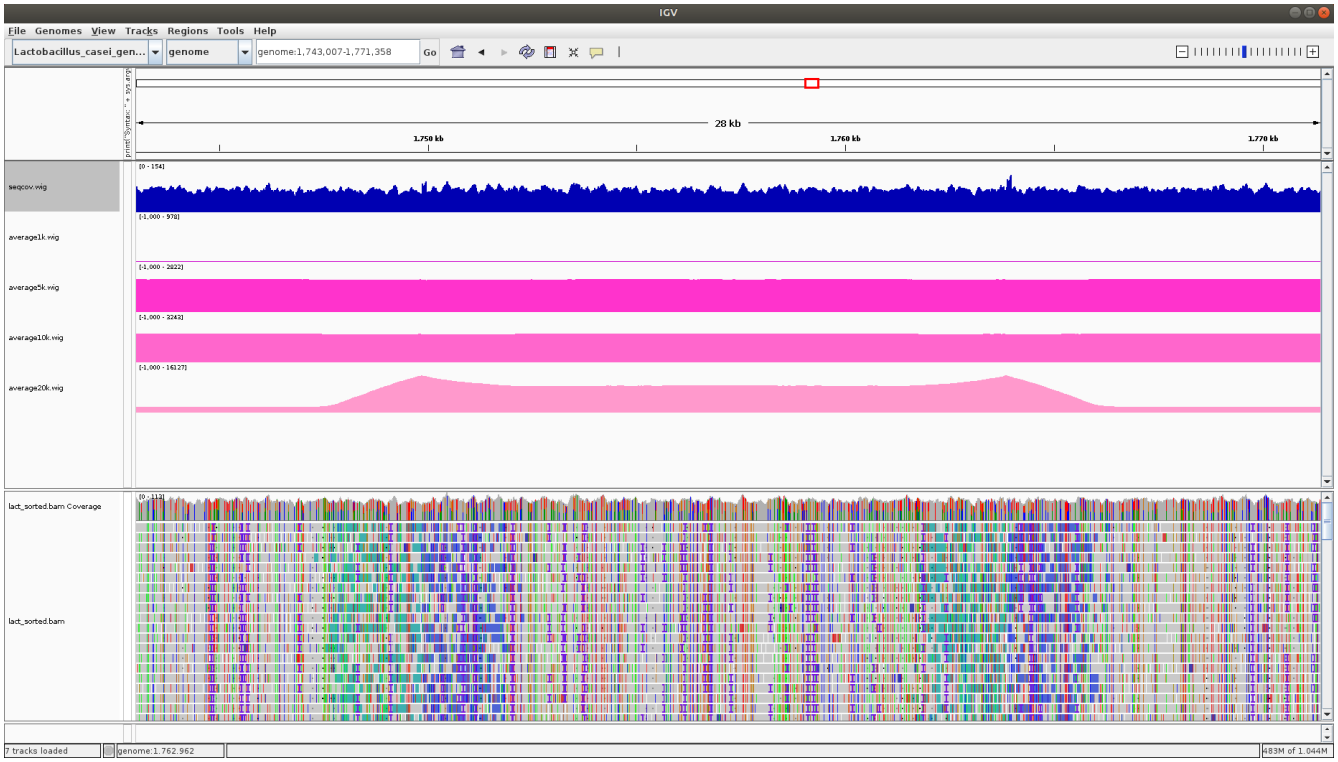
As for the physical coverage, a long deletion can be detected by looking at the empty gap identified by the 5k and 10k tracks, while when the max length is too big it can be found through a peak.



Instead, on the right side of the tracks, when there is a small peak in the 5k and 10k tracks, a small deletion is found, so it is identified by an increase in the tracks,

Instead, where there is a peak in the 1k track, again on the right side, in correspondence with a lower value in the 5k and 10k coverage, there is a long insertion.

Finally, where a high peak (the one after the peak related to the long deletion) is found in the 20k track, a region with a long inversion is present.



4.6 Orientation coverage

Another way to detect structural variations is with the orientation coverage track, which is a track with the reciprocal orientation of the mates. The orientation can be found on bit 16 and bit 32 of the FLAG fields of the SAM file. Since there are 2 bits there will be 4 possible results: 0, 16, 32, and 48, each indicating one of the possible orientations (forward/forward, forward/-reverse, reverse/forward, reverse/reverse). Therefore 4 wig tracks, one for each orientation, are computed.

4.6.1 Code

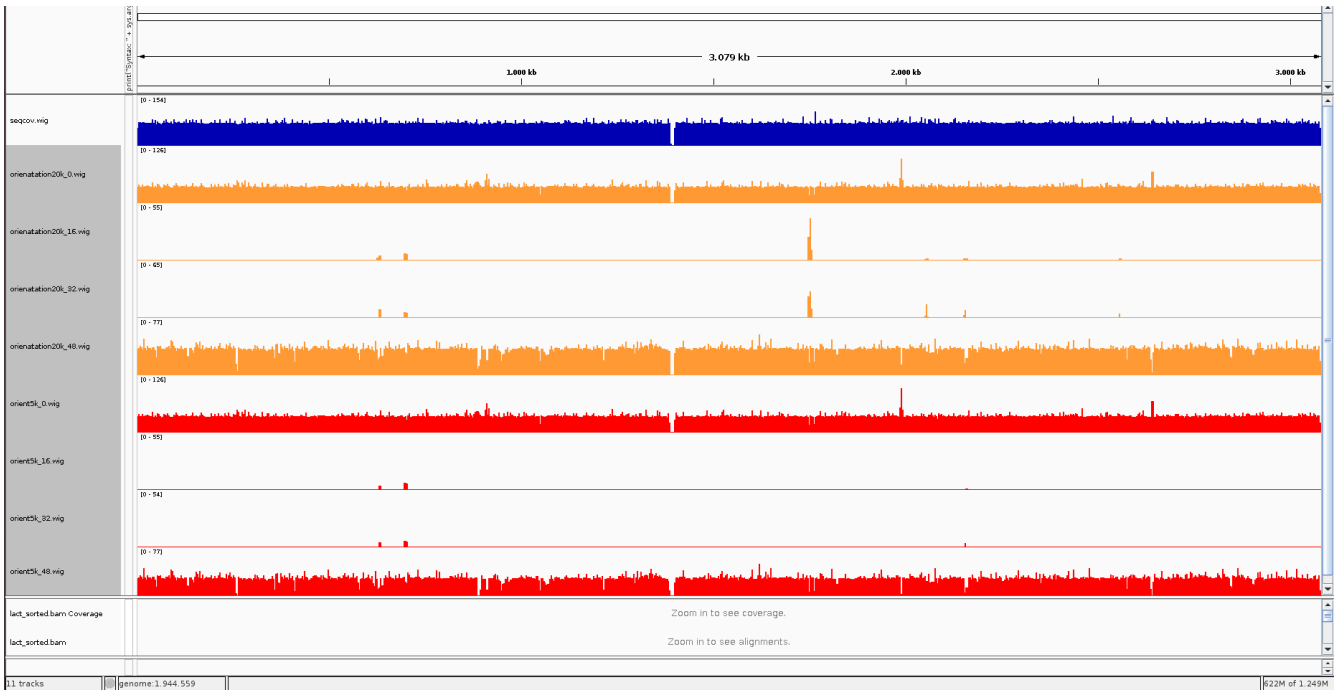
The code is the same as the one used for the physical coverage. The only thing that is added in this case is the check for the orientation. The type of orientation is given as input to the program. After checking if the read and its mate are mapped and if the TLEN value is positive but smaller than the maximum length (input feature), it is verified if the orientation is the same as the one given as input:

```
if ((int(fields[1])) & 48 == orientation):
```

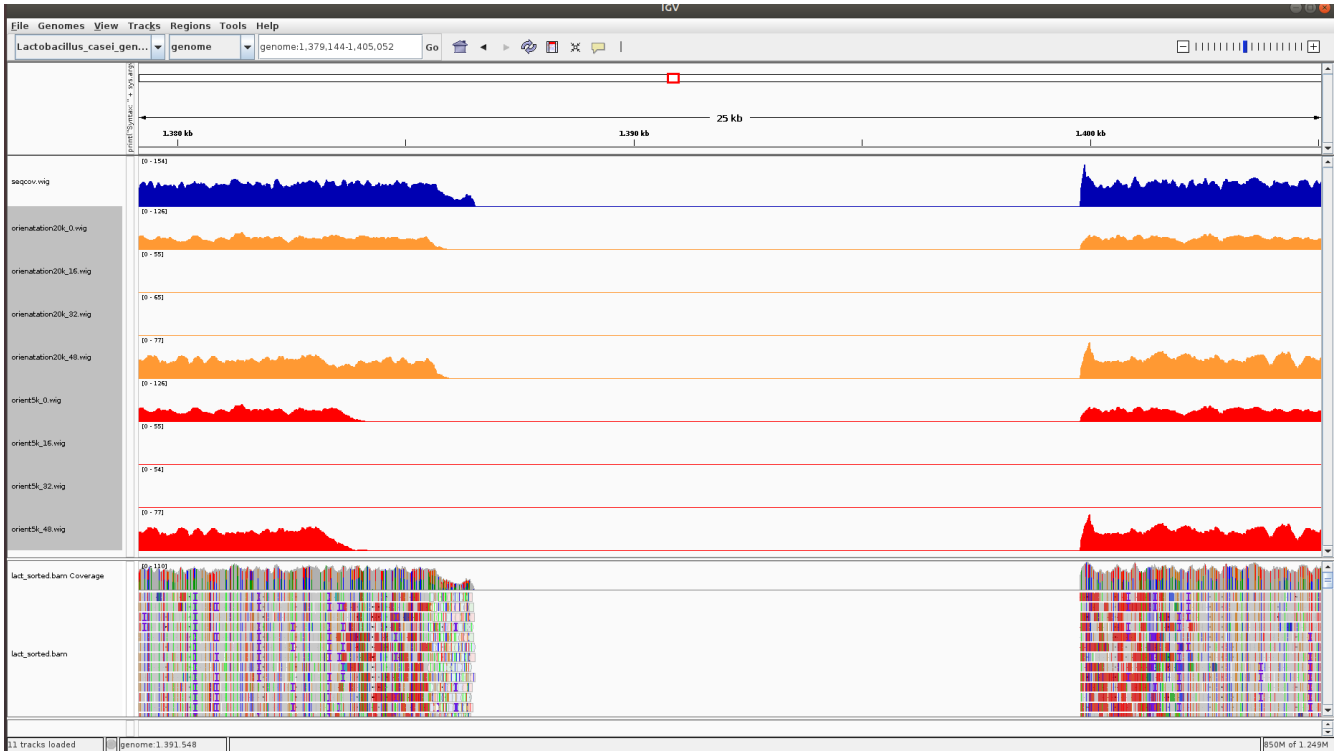
If all these conditions are true, then the program behaves like before and computes the coverage. Different maximum lengths have been tried: 5000 and 20000.

4.6.2 Results and observations

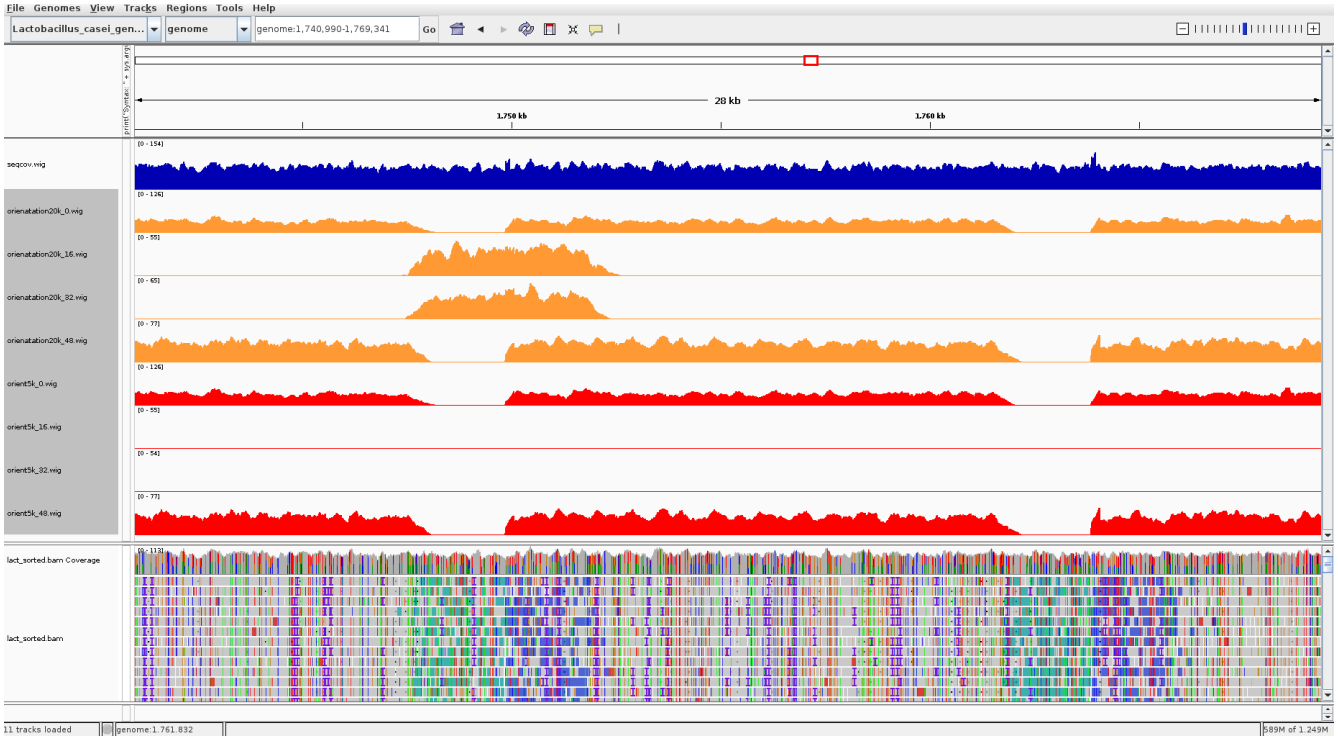
Here it is possible to see the two average length coverage tracks: the first one is the sequence coverage, then the coverage with 5k max length, right after the one with 20k max length. It's possible to see that the tracks related to orientation 16 and 32 (forward/reverse, reverse/forward) are very similar to each other, even between the 5k and 20k tracks: both coverages are flattened to 0, except for very few peaks, which means that there are some errors with the orientation. Instead, the coverage related to 0 and 48 (forward/forward, reverse/reverse) are quite different from one another, but they both reach 0 almost never.

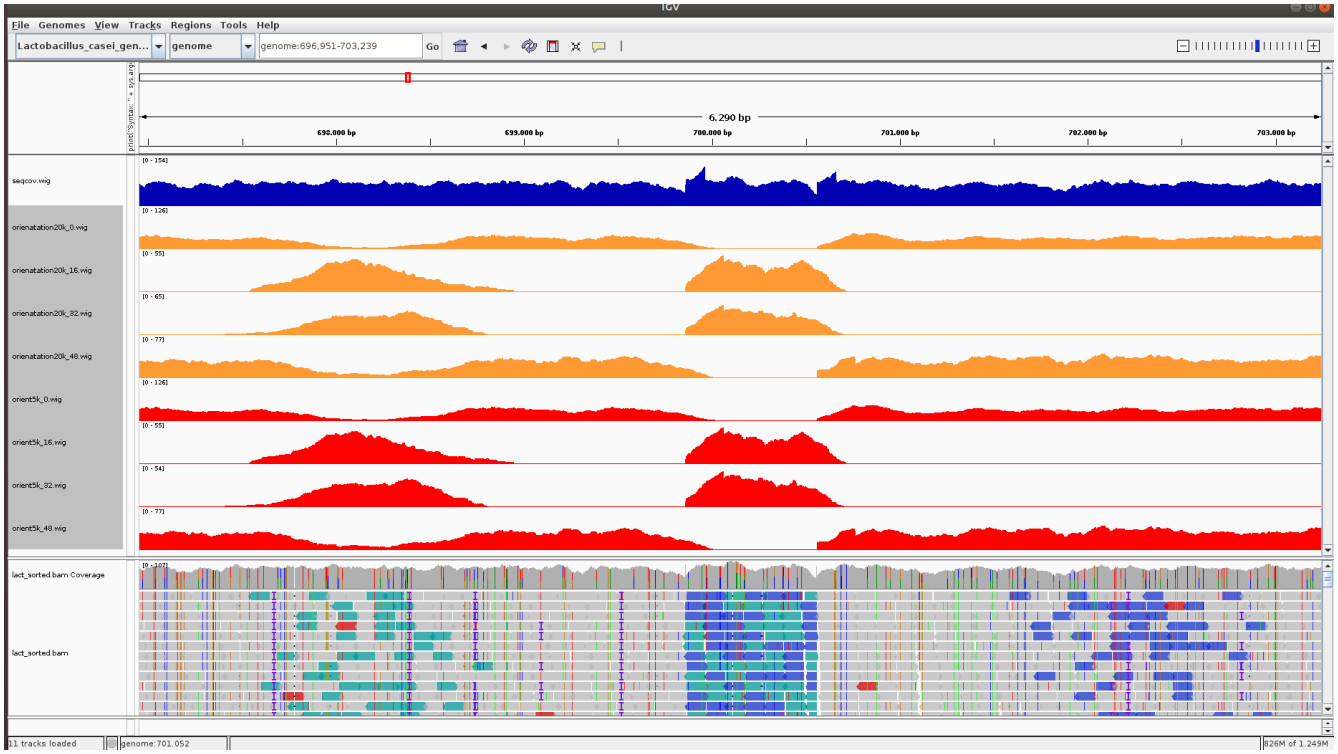


By looking closely to the region where the tracks for 0 and 48 orientation is equal to 0, it's possible to see that the long deletion is found.



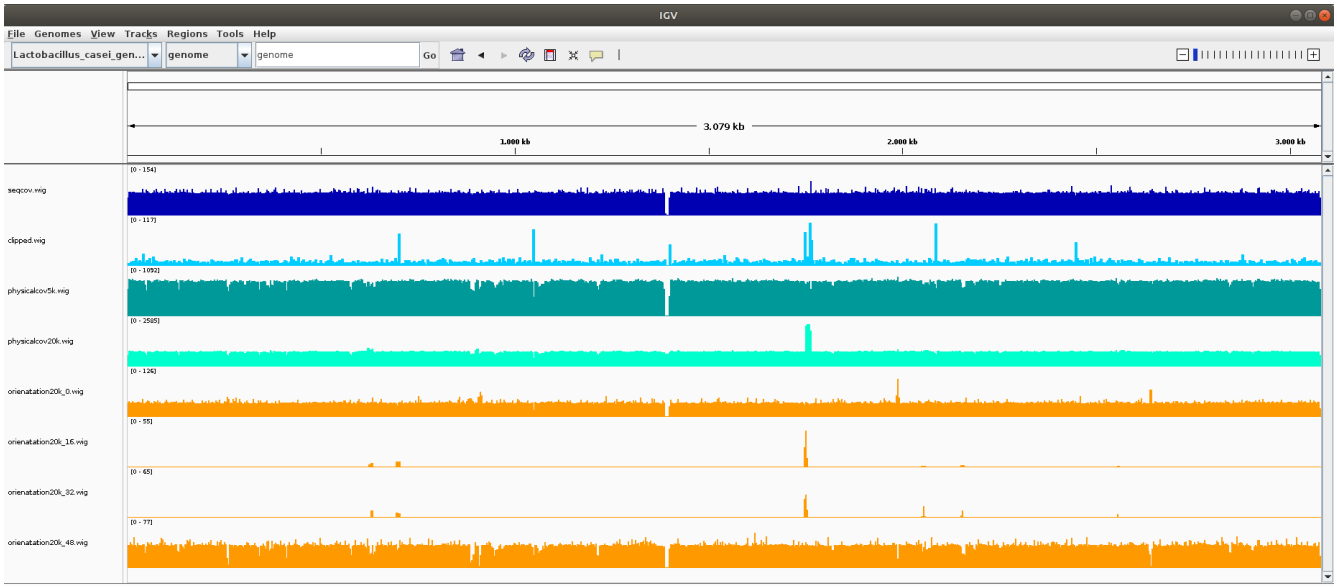
When tracks for 0 and 48 orientation is very low or when tracks for 16 and 32 orientation have high peaks, there is usually a region with inversion, as shown in the two following picture: the first one is a big inversion, while the other one is a small one. This behavior poit out problems with orientations.


File Genomes View Tracks Regions Tools Help
Lactobacillus_casei_gen...
genome
genome:1,740,990-1,769,341
Go
28 kb
1,759 kb
1,760 kb
seqcov.wig
orienatation20k_0.wig
orienatation20k_16.wig
orienatation20k_32.wig
orienatation20k_48.wig
orient5k_0.wig
orient5k_16.wig
orient5k_32.wig
orient5k_48.wig
lact_sorted.bam Coverage
lact_sorted.bam
11 tracks loaded
genome:1,761,832
589M of 1,249M


File Genomes View Tracks Regions Tools Help
Lactobacillus_casei_gen...
genome
genome:696,951-703,239
Go
6,290 bp
698,000 bp
699,000 bp
700,000 bp
701,000 bp
702,000 bp
703,000 bp
seqcov.wig
orienatation20k_0.wig
orienatation20k_16.wig
orienatation20k_32.wig
orienatation20k_48.wig
orient5k_0.wig
orient5k_16.wig
orient5k_32.wig
orient5k_48.wig
lact_sorted.bam Coverage
lact_sorted.bam
11 tracks loaded
genome:701,052
826M of 1,249M

4.7 Comparison between tracks

Another interesting thing to be investigated is the comparison between the different coverage, in order to see if there are any types of connections. This can also help to find easily the structural variation. Here, the following tracks are compared: sequence coverage, clipped coverage, physical coverage (5k and 20k), and the 4 orientation tracks (20k).

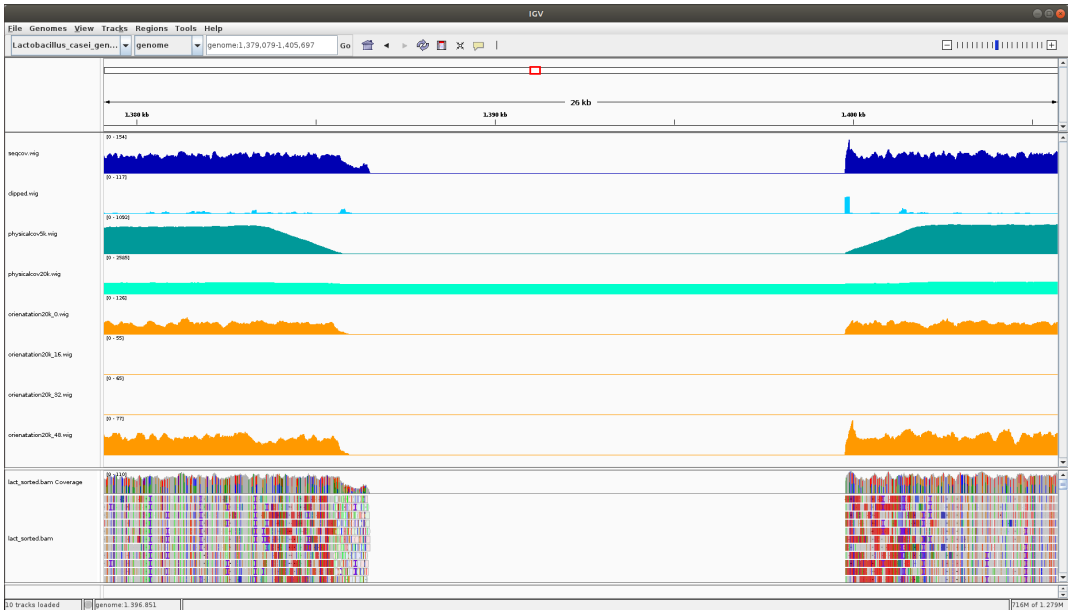


By looking again at the correspondence of the peaks present in the clipped tracks, it is possible to notice that the structural variations pointed out by them are also confirmed by the behavior the others track have in those positions

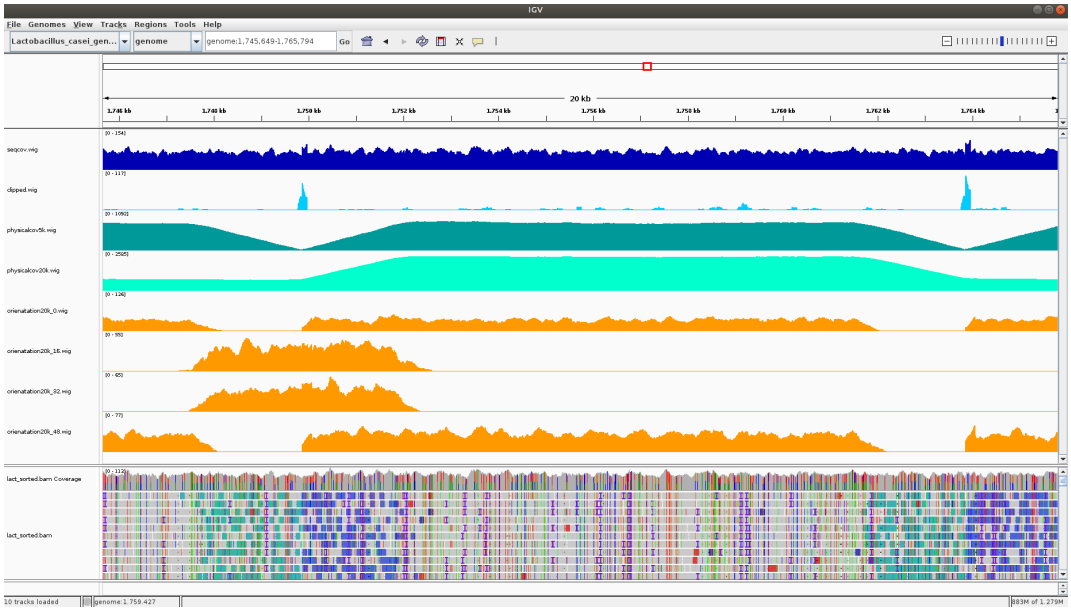
Peak 1 In the first peak, a small inversion is identified, and this is supported also by the orientation tracks since in this region the orientation tracks of 0 and 48 are flatted to zero, while the others 2 (16, 32) are higher, indicating that there are some problems with the orientations.

Peak 2 The second peak recognizes a small insertion, fact that is confirmed by the drop in the physical coverage.

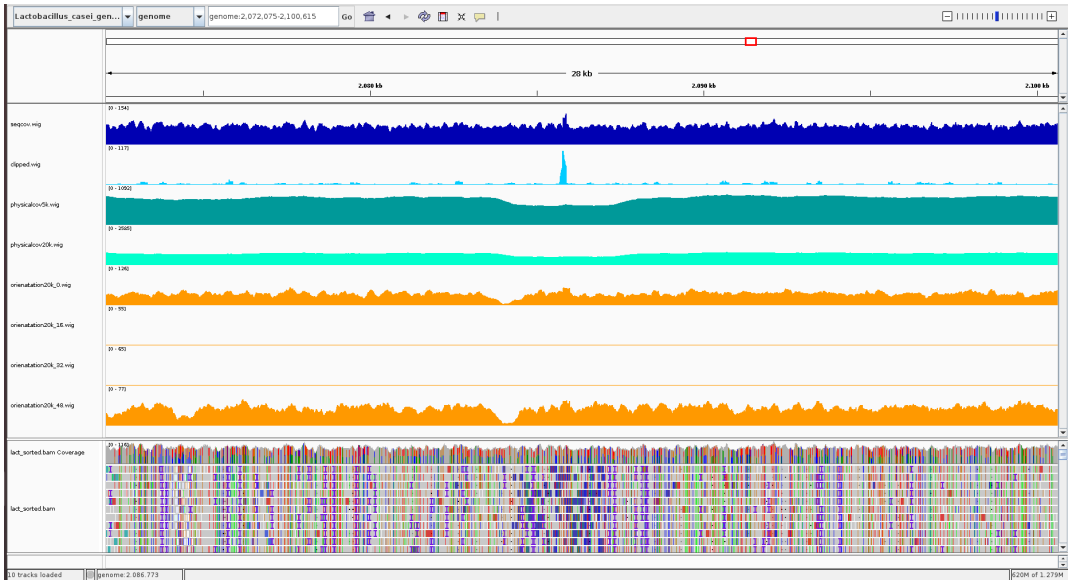
Peak 3 The third peak found a long deletion, which is supported by all the tracks since all of them present a long empty spot. This doesn't happen for the physical coverage with a max length of 20k, meaning that the length considered in this track is too large to let it be possible to spot the long deletion.



Peak 4 For what regards the fourth peak, a long inversion is spotted. This is also supported by the other tracks since a similar situation to the one present for the small insertion happens: the physical coverage has an increase and a subsequent decay, while the tracks of the orientation indicate a problem with the orientation since the 0 and 48 tracks drop to zero while the 16 and 48 increase.



Peak 5 In correspondence with the fifth peak there is the long insertion, accompanied by a drop in the physical coverage track.



Peak 6 Finally, in peak number six, a small deletion is found. Since the deletion is small, the two physical coverage are not able to spot it due to the max lengths, which are too big to let the track finds the deletion, meaning that the deletion is shorter that 5k.

5 Conclusions

5.1 Conclusion for coverage

5.1.1 Sequence coverage

- Sequence coverage has similar behavior to the coverage track;
- negative peaks are indicators for long deletion.

5.1.2 Physical coverage

- Negative peaks are indicators for long deletion, but these peaks are not visible if the fragment is larger than the area related to the deletion (20k and 30k max length case);
- inversion can be identified by an increase and a subsequent decay of the track;
- insertions are in correspondence with a decrease in the track.

5.1.3 Multimappers coverage

- Repetitions seem to have some relationship with low-quality regions.

5.1.4 Clipped reads coverage

- they are very useful in detecting all types of structural variations since each peak has been found in correspondence to one of them.

5.1.5 Average length coverage

- Short and long deletion seems to have a different relationship with the average length track. In a region with long deletion it is probable to have reads before and mates after this region, instead with small deletion the track is likely to have a positive curve;
- long insertion seems to produce a decay in the track;
- long inversion, instead, produces an increase in the track.

5.1.6 Orientation coverage

- The 16 and 32 bits are useful to identify inversion, both short and long, since in this situation there are peaks in these tracks, while the 0 and 48 go to zero, pointing out problems with orientation;
- the 0 and 48 bits are useful to spot deletion, both long and short, since they present an empty spot.

5.2 Positions of structural variations

5.2.1 Deletions

- **Long**: the long deletion is found in the middle of the genome, more specifically between position around 1.386.526 and 1.399.789;
- **Short**: the short deletion is found in the right part of the genome, more specifically between position around 2.449.091 and 2.449.790.

5.2.2 Insertions

- **Long**: the long insertion is found in the right part of the genome, more specifically around position 2.085.801;
- **Short**: the short insertion is found in the left part of the genome, more specifically around position 1.049.899 (more or less).

5.2.3 Inversions

- **Long**: the long inversion is found in the right part of the genome (after the long deletion), more specifically around position 1.749.445;
- **Short**: the short inversion is found in the left part of the genome (at the beginning), more specifically around position 700.272 (more or less).