Assembly of Sequence Data

Introduction

Improvements in DNA sequencing technology have led to new opportunities for studying organisms at the genomic and transcriptomic levels. Applications include studies of genomic variation within species and gene identification. In this module, we will concentrate on data generated by the Illumina and the Pacific Biosciences technologies, although the techniques you will learn are applicable to other technologies (e.g. 454 GS FLX, Capillary data, Ion torrent and or Oxford Nanopore). A single Illumina machine can produce over 3.5 terabases of sequence data in a week! This is the equivalent to many human genomes! The data from the Illumina machine comes as relatively short stretches (35-250 base pairs) of DNA. These individual sequences are called **sequencing reads**. The older **capillary sequencing** method produces longer reads of ~500bp, but is much slower and more expensive. The PacBio technologies produces reads over 30k, with a mean of 13kb, but there error level is not good - 15% of errors.

With the Illumina technologies one of the greatest challenges of of a genome project is determining how to arrange sequencing reads into chromosomes. This process of determining how the reads fit together by looking for overlaps between them is called **genome assembly**. Capillary sequencing reads (~500bp) are considered a good length for genome assembly, and assembling PacBio reads is now considered “easy”. The difficulty of sequence assembly is when the reads are shorter than repetitive sequences. Further, the programs have to deal with a lot of data.

When doing assembly with short reads (Illumina), the most time consuming part would be to find all the possible overlaps between all the reads. One efficient way is to look for k-mers (words of a specific length) in each read. If two reads contain the same k-mer they might also overlap. Each read contains several k-mers (n-k+1, assuming n is the read length). k-mers from the same read are connected in a graph. (A graph is an construct that helps to visualise abstract data structures, with ‘nodes’ that are connected by ‘edges’). Many short read assemblers exist and most are based on de Bruijn graphs, like Velvet (Zebrino et al., 2008) or ABYSS (Simpson et al., 2009). The k-mers are encoded in the de Bruijn assembly graph and then the software attempts to simplify the graph to generate connected sequences, where all k-mers are represented. For most purposes, you don’t really need to know how an assembler works in detail to use it and get good results, but a basic understanding is important.
Other good assembly pipelines are Masurca or Allpaths, which apart of doing just the assembly also perform other tasks like read correction or gap filling.

In this module we are going to assemble one chromosome of a malaria parasite, *Plasmodium falciparum* IT clone and compare it to the known reference. One aim is to compare the assembly of Illumina reads with PacBio reads. Important will be how to look at assemblies and understand possible errors. In the end we are going to annotate the final assembly with a new annotation pipeline.

A: Starting the PacBio *de novo* assembly

In this exercise we will look at a lab strain of *P. falciparum*, the IT clone. We have sequenced the genome with PacBio and Illumina. First we are going to start the PacBio assembly using the PBcR program. It first corrects the reads and then uses the Celera assembler to merge the long reads into contigs.

Double-click on the desktop icon "Module 7 Assembly", which will open a terminal in the correct directory for this module.

The filtered reads are called `PBReads.fastq`. Have a look at the configuration file pacbio.spec. Any idea what it means?
Here the command to start the assembly.

```
$ canu -p PB -d Pacbio -s file.specs -pacbio-raw
PBReads.fastq &> output.txt
```

Be sure that the process is running. For example use the `top` command to see if a program is running… it might be jellyfish. Use "q" for quit to exit top.

If something is not running, check the output file "output.txt" to see what the error is.

This assembly will run for a while. In the meantime we are going to assemble the same chromosome with Illumina. But first you can do a little bit a assembly by hand, if you want.

Doing a de Brujin graph by hand - OPTIONAL

Here we are going to do an example of the de Brujin graph by hand! Sit together with your neighbour and build the graph from the reads and find the contig(s).
Use a k-mer of five, to finish the graph. (In this example we ignore the reverse complement!)

Reads:

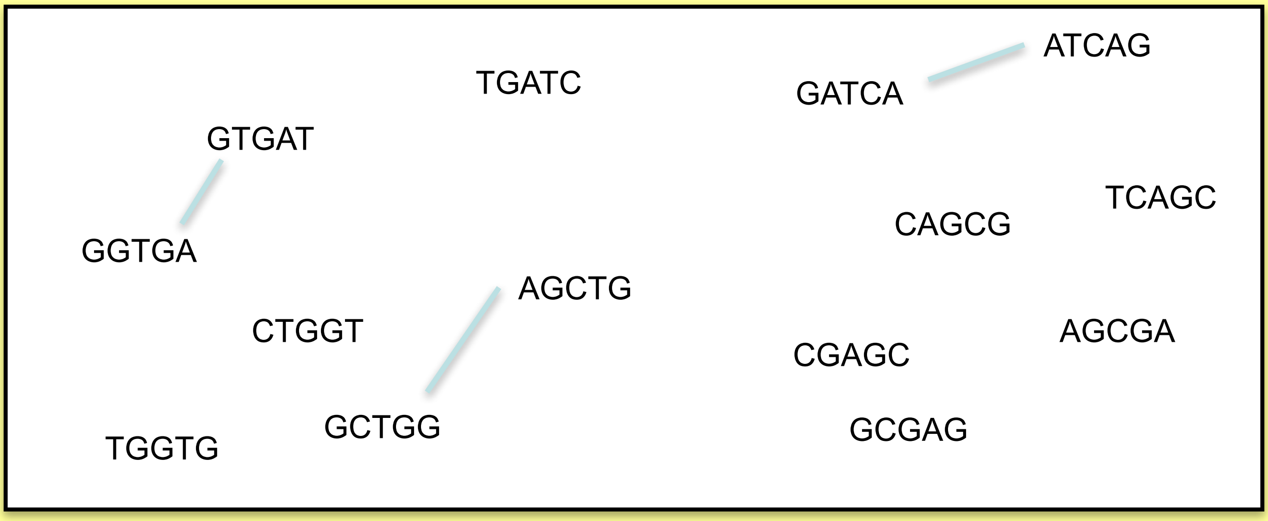
AGCTGG TGGTGA GATCAG

GAGCTG CTGGTG ATCAGC

GCTGGT GGTGAT TGATCA TCAGCG

AGCGAG CAGCGA GTGATC GCGAGC CGAGCT

To help you, we already generated all the k-mer of the length of five. Is it easy to find the contig? Some edges are already included.

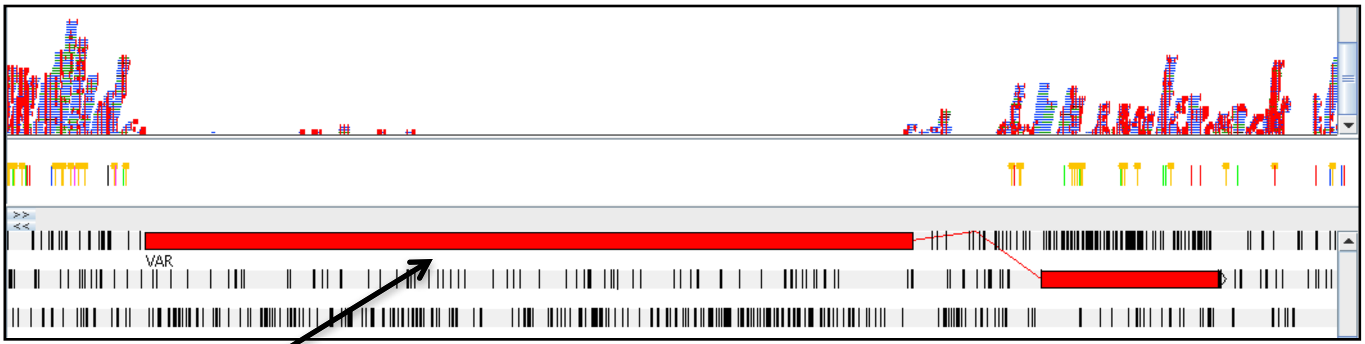


The contig has the sequence:

What is difficult here? Maybe translate it, to find the “origin”.

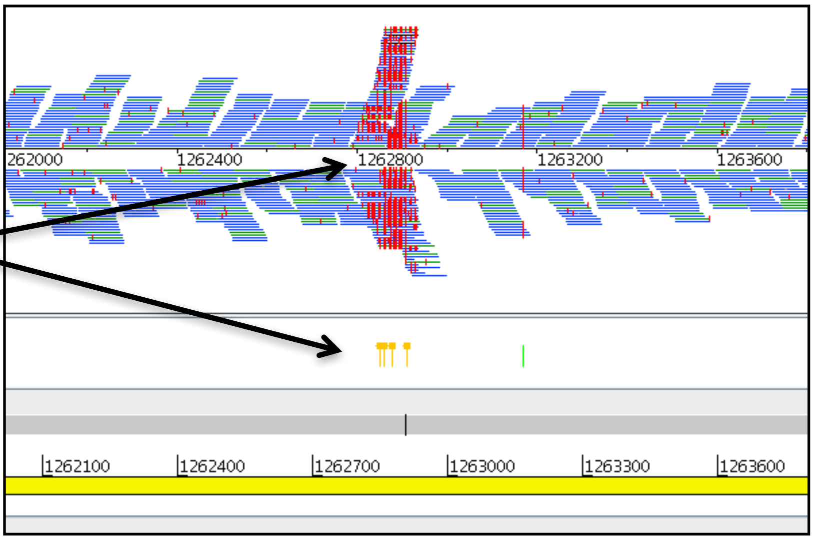
B: Generating *de novo* assemblies is useful even with a reference

Before we are doing the *de novo* assembly, please have a look at the following screenshots.
Why is the mapping approach not helpful in those regions?



1. Why is this gene not covered?

2. Should there be heterozygous SNPs in a haploid lab strain?



Here some examples of regions, with problematic mapping: 1) Gene not covered. 2) Heterozygous SNP. If you want to have a look by yourself:

```
$ art -Dbam=IT.Chr5.bam,var.IT.Chr5.bcf Pf3D7_05.embl
```

C: Doing the first Illumina assembly

One way to resolve problematic mapping is to do an assembly. Here, we go through step by step how to perform an assembly and analyse it. This will be just a draft assembly, which wouldn't be the final result, but it should give insight into some of the problems discussed above.

We are going to use the assembler velvet. As input we are using the same reads as the ones for the mapping. The only difference is that the files are zipped, so that they use less disc space. Be sure to be in the directory □`~/course_data/Module7_Assembly/`

```
$ velveth k.assembly.49 49 -shortPaired -fastq -separate
IT.Chr5_1.fastq IT.Chr5_2.fastq
```

49 is the k-mer size. "k.assembly.49" is the name of the directory where the results are going to be written. The other options specify the type of the input data. With the following command you can see all possible options, but don't be afraid, not all must be used.

```
$ velveth
```

Now the assembler has to build the graph and find the path, as we did before in the exercise:

```
$ velvetg k.assembly.49 -exp_cov auto  -ins_length 350
```

The first parameter specifies the working directory. The second is to let velvet find the median read coverage rather than specify it yourself. Last, the insert size of the library is given. There is a lot of output, but the most important is in the last line:
Final graph has 978 nodes and n50 of 10508, max 54529, total 1374552, using 1397134/1510408 reads. (Result might differ depending on the velvet version used).

The output can be different depending on the version of the software.

This line first gives you a quick idea of the result. 978 nodes are in the final graph. An n50 of 10508 means that 50% of the assembly is in contigs of at least 10508 bases, it is the median contig size. This number is most commonly used as an indicator of assembly quality. The higher, the better! "Max" is the length of the longest contig. "Total" is the size of the assembly, here 1347kb. The last two numbers tell us how many reads were used from the 7.5 million pairs.

That wasn't too bad! Now we have to try to improve the assembly a bit. The kmer size has the biggest impact. Also the -cov_cutoff parameter can play a role. This means that nodes with less than a specific k-mer count are deleted from the graph. More parameters can be changed, but we would run out of time. In the beginning the changes look a bit random, but with more experience, you will get a feeling for them.

First rerun velvet with a k-mer size of 49. As parts of the graph are already done, the program will run far quicker. velveth doesn't need to be rerun.

```
$ velvetg k.assembly.49 -exp_cov auto  -ins_length 350 \
-min_contig_lgth 200 -cov_cutoff 5
```

Maybe do assemblies for different k-mer sizes i.e. 55, 41, here the example is a k-mer length of 55

```
$ velveth k.assembly.55 55 -shortPaired -fastq –separate \
IT.Chr5_1.fastq IT.Chr5_2.fastq

$ velvetg k.assembly.55 -exp_cov auto  -ins_length 350
-min_contig_lgth 200 -cov_cutoff 5
```

Write down the results for each assembly made using different k-mer sizes. Which one looks the best?:

k-mer	Nodes	n50	largest contig
41			
49			
55			

If you want to play with other parameters, like the -min_pair_count, go for it. All the options can be seen by typing:

```
$ velvetg
```

All the results are written into the directory you specified, e.g. k.assembly.49. The final contigs are in contigs.fa. The stats.txt file holds some information about each contig, its length, the coverage, etc. The other files contain information for the assembler.

Another way to get more stats from all the runs is to use a little program called *stats*. It displays the number of contigs, the mean size and a lot of other numbers. It might help to pick “the best” assembly.

Just type:

```
$ assembly-stats k.*/*.fa
```

```
stats for k.assembly.41/contigs.fa
sum = 1435372, n = 199, ave = 7212.92, largest = 75293
N50 = 22282, n = 19
N60 = 16569, n = 27
N70 = 13251, n = 37
N80 = 9535, n = 49
N90 = 4730, n = 69
N100 = 202, n = 199
N_count = 51974
-------------------------------
stats for k.assembly.49/contigs.fa
sum = 1452034, n = 175, ave = 8297.34, largest = 85317
N50 = 28400, n = 17
N60 = 26582, n = 23
N70 = 16485, n = 29
N80 = 12065, n = 39
N90 = 6173, n = 55
N100 = 202, n = 175
N_count = 57000
-------------------------------
stats for k.assembly.55/contigs.fa
sum = 1461496, n = 181, ave = 8074.56, largest = 71214
N50 = 28059, n = 19
N60 = 22967, n = 25
N70 = 14871, n = 33
N80 = 11360, n = 44
N90 = 4885, n = 64
N100 = 205, n = 181
N_count = 69532
```

It looks that the best choice is a k-mer size of 49. The n50, average contig size and the largest contigs have the highest values, while contig number is the lowest. Before we look at the assembly itself, what could the N_count mean?

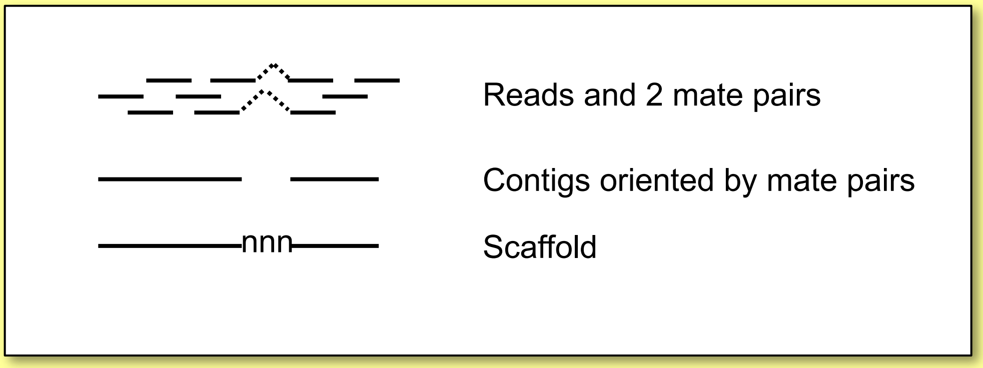
Scaffolding

As we discussed before, DNA templates can be sequenced from both ends, resulting in mate pairs. Their outer distance is the insert size. Imagine mapping the reads back onto the assembled contigs. In some cases the two mates don't map onto the same contig. We can use those mates to scaffold the two contigs e.g. orientate them to each other and put N's between them, so that the insert size is correct, if enough mate pairs suggest that join. Velvet does this automatically (although you can turn it off). The number of mates you need to join two contigs is defined by the parameter -min_pair_count.

Here is the description:

```
 -min_pair_count <integer>       : minimum number of paired
end connections to justify the scaffolding of two long
contigs (default: 5)
```

Here a schema:



It might be worth mentioning, that incorrect scaffolding is the most common source of error in assembly (so called miss-assemblies). If you lower the *min_pair_count* too much, the likelihood of generating errors increases.
Other errors are due to repeats. In a normal assembly one would expect that the repeats are all collapsed, if they are smaller than the read length. If the repeat unit is smaller than the insertsize, than it is possible to scaffold over it, leaving the space for the repeats with n's.

To get the statistic for the contigs, rather than supercontigs, you can use following command:

```
$ fastaq scaffolds_to_contigs k.assembly.49/contigs.fa
tmp.contigs.fasta
$ assembly-stats tmp.contigs.fasta
```

depending from which assembly you would like the statistics.

D: Reference based Assembly

Velvet has an option to use a reference to help to resolve repetitive regions. When velvet cannot resolve a repetitive region in the de Brujin graph, it can look where read with this k-mer are mapping in the reference. This way it is possible to untangle the graph to simplify to find the path in the graph. This module is called velvet columbus.

Although it is still work in progess, the results seem to be better than the normal assembly. The only difference would be to include: -reference -fasta Pf3D7_05.fasta in the velveth call. Important is to map the reads against the reference what we already did.

For the following exercises we are going to use this assembly, with a k-mer of 55.

Run it as follow:

```
$ velveth k.columbus.55 55 -reference -fasta Pf3D7_05.fasta
-shortPaired -fastq -separate IT.Chr5_1.fastq
IT.Chr5_2.fastq

$ velvetg k.columbus.55 -exp_cov auto  -ins_length 350 -
min_contig_lgth 200 -cov_cutoff 5
```

Now compare the results with the stats program. Are they better?

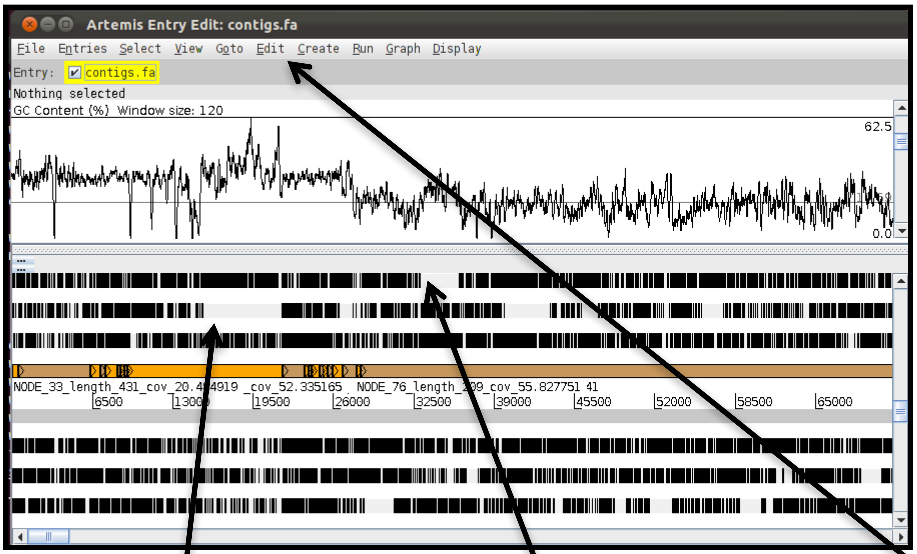
```
$ assembly-stats k.*/*.fa
```

E: Looking at the assembly

So far we have only looked at the stats for our assembly, and don't know anything about the content of each contig. One way would be just to open the contigs in Artemis.

Assuming you choose the k-mer 55 of the velvet columbus assembly, type:

```
$ art k.columbus.55/contigs.fa
```


Artemis Entry Edit: contigs.fa
File Entries Select View Goto Edit Create Run Graph Display
Entry: contigs.fa
Nothing selected
GC Content (%) Window size: 120
62.5
0.0
NODE_33_length_431_cov_20.444919 _cov_52.335165 NODE_76_length_109_cov_55.827751 41
6500
13000
19500
26000
32500
39000
45500
52000
58500
65000

Possible open reading frames (ORF), e.g. no stop codons and high GC content

Select create and "Mark Ambiguities". This will show you the gaps in the supercontigs.

Although we see some open reading frames, lot of work would need to be done to produce gene models e.g. find open readings frames, adjust the gene boundaries, do functional annotation, to then start to compare this assembly of the IT clone to the reference sequence.

There is a better way! Couldn't we use the reference somehow?

Optional: Which gene is in the first reading frame? Can you generate a gene model and blast it?
(Double click mouse wheel; Create -> Feature from base range; Accept the new model; Select the model and run a blast against Uniprot)

F: Contig ordering

At the Wellcome Trust Sanger Institute we developed a tool called ABACAS (Assefa et al., 2009) to order contigs against a reference. Spaces between the contigs (gaps) will be *N* characters. The result is called a pseudo-molecule. It can be loaded into ACT (a bit like a sandwich of two Artemis views) and then be analysed.

In order to start ABACAS you need a reference sequence (Pf3D7_05.fasta) and the contigs (we assume k.assembly.49/contigs.fa - but you can use another assembly). Next you decide if you want to do a comparison of nucleotides (nucmer) or amino acids (promer).

```
$ abacas.1.3.1.pl -r Pf3D7_05.fasta -q
k.columbus.55/contigs.fa -p promer -b -d -a -o IT.ordered
```

Abacas has many options. We use
-b to generate a bin of contigs that don’t map. This is very important
-a will append the bin onto the pseudo molecule
-d uses the standard comparison parameter, in this case faster
-o IT.ordered is the prefix for the output file.

The command

$ abacas.1.3.1.pl -h

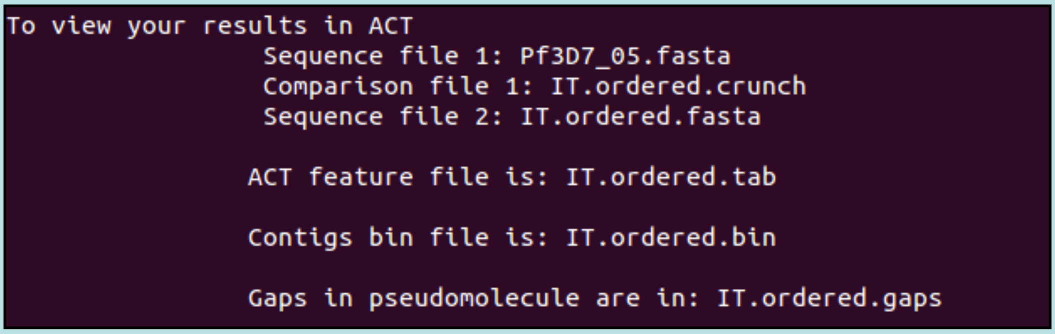
will give you a complete list of all options.

```
-s  int     minimum length of exact matching word (nucmer default = 12,
            promer default = 4)
```
Higher values decrease the runtime for the price of sensitivity.

```
-e          Escape contig ordering i.e. go to primer design
```
If you just would like to generate primers over gaps regions.

```
-c          Reference sequence is circular
```

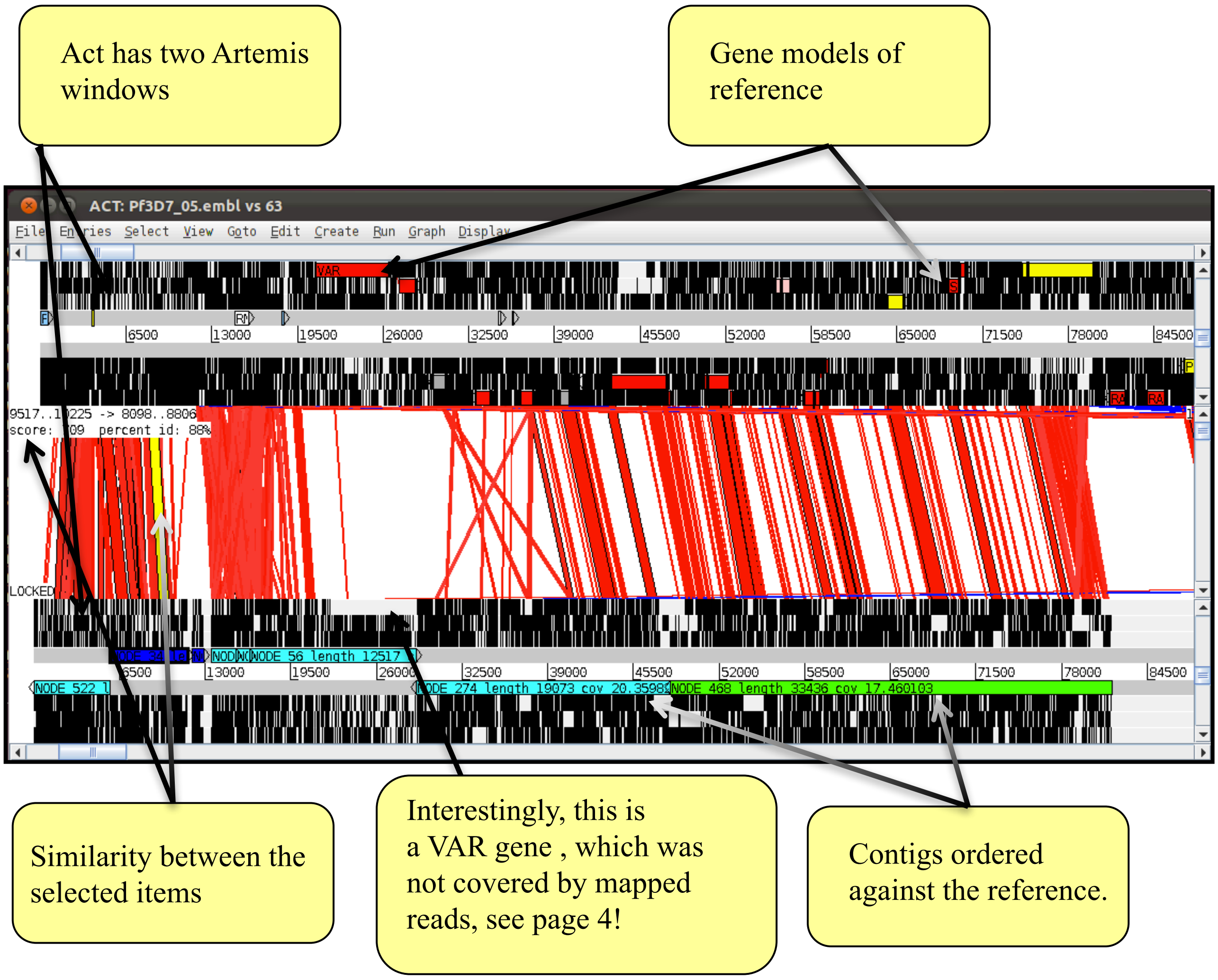
Once abacas is done, it indicates which data it generated and how to load them into Act:



```
To view your results in ACT
                 Sequence file 1: Pf3D7_05.fasta
                 Comparison file 1: IT.ordered.crunch
                 Sequence file 2: IT.ordered.fasta

                ACT feature file is: IT.ordered.tab

                Contigs bin file is: IT.ordered.bin

                Gaps in pseudomolecule are in: IT.ordered.gaps
```

Before opening the file in Act, we generate a BLAST comparison file:

```
$  blastn -subject Pf3D7_05.fasta -query IT.ordered.fasta \
-evalue 1e-20 -outfmt 6 -out comparison.blast
$ act Pf3D7_05.embl comparison.blast <(cat IT.ordered.tab
IT.ordered.fasta  ) &
```



Act has two Artemis windows

Gene models of reference

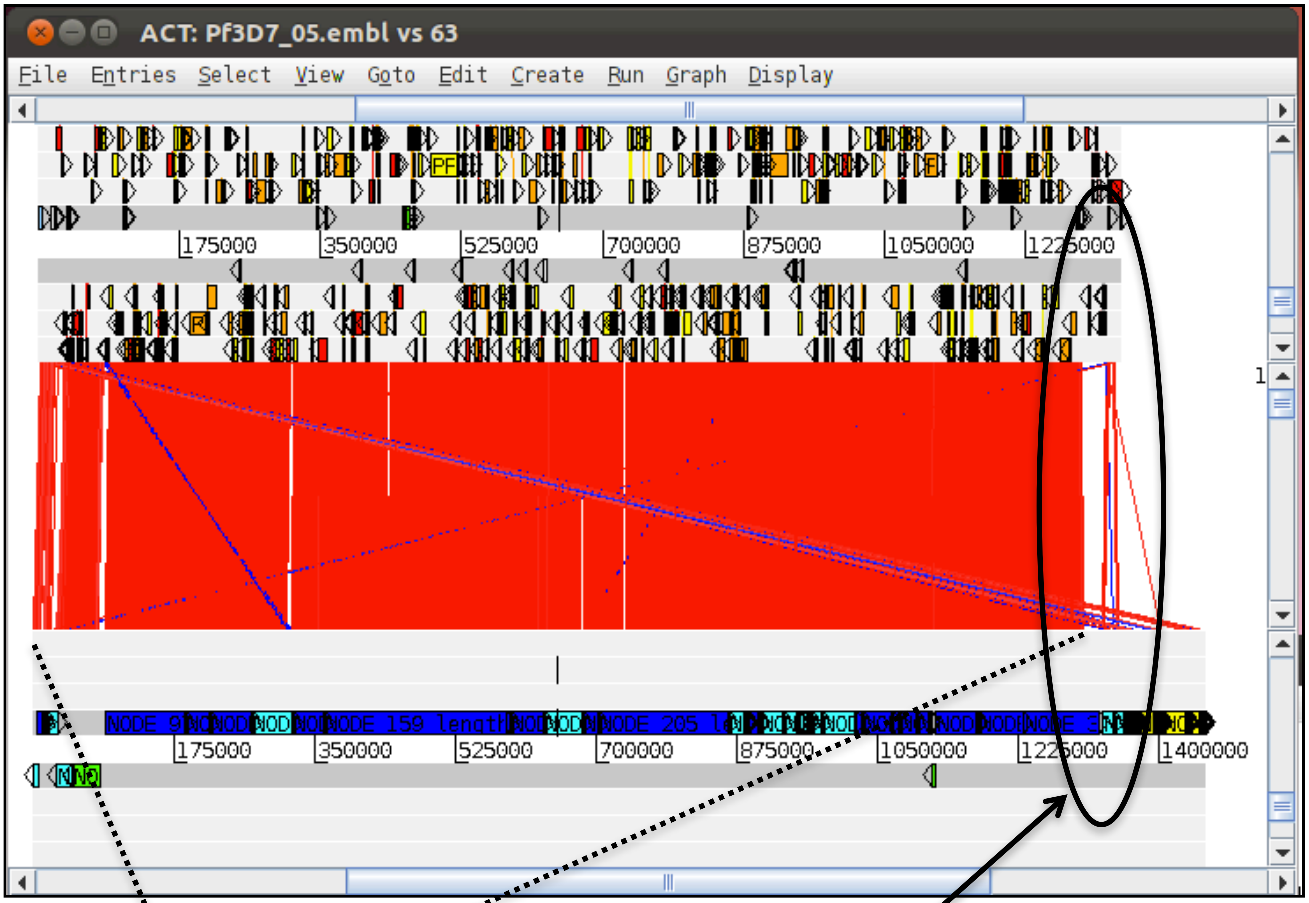
Similarity between the selected items

Interestingly, this is a VAR gene , which was not covered by mapped reads, see page 4!

Contigs ordered against the reference.

Scroll though the assembly. Maybe zoom in and out. How does it look? Are there any assembly errors?

What happened with the gene PF3D7_0532500?



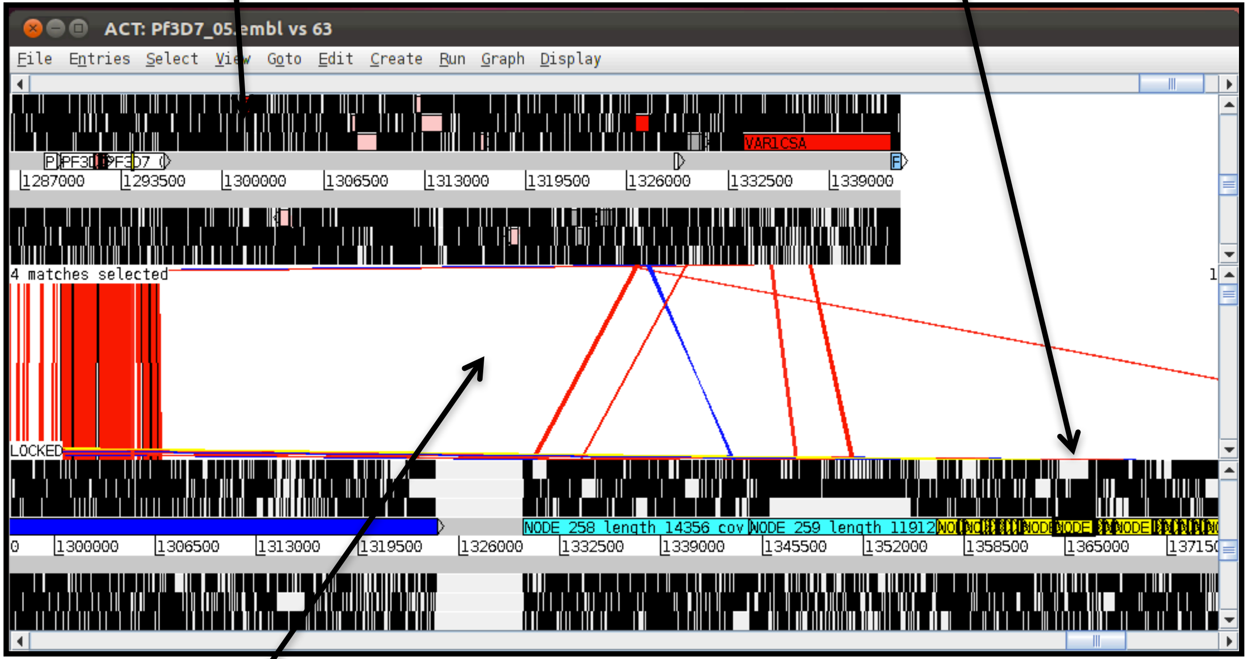
There are no big mis-assembly, as there are no synteny breaks.

Why is there no contig for the assembly here?

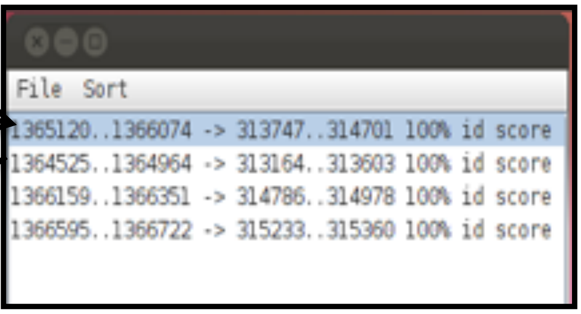
We ran abacas with the -a option. This means that contigs that didn't map against the reference are appended at the end. Scroll to the right hand site. Any idea what those contigs are? Could you order some into the core of the chromosome?

Has this gene not enough similarity or were the contigs not ordered well enough?

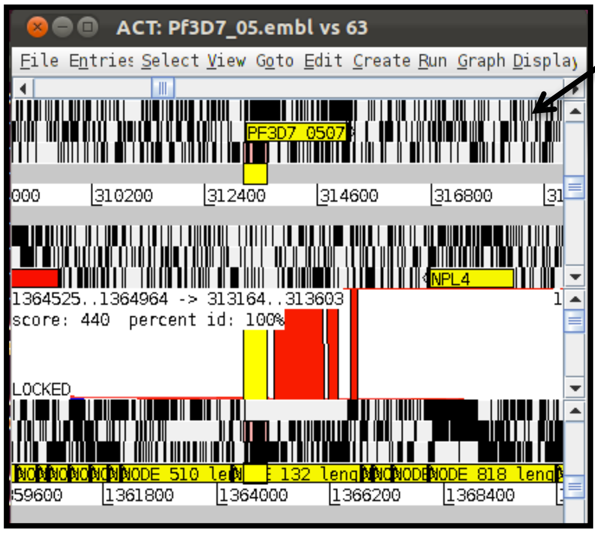
This looks like on open reading frame. Can you determine the function? It seems to have similarity with the reference.
Why is it not ordered against the reference?



1. Right click -> View selected matches. Double click on it.



```
1365120..1366074 -> 313747..314701 100% id score
1364525..1364964 -> 313164..313603 100% id score
1366159..1366351 -> 314786..314978 100% id score
1366595..1366722 -> 315233..315360 100% id score
```



```
1364525..1364964 -> 313164..313603
score: 440  percent id: 100%
```

This contig is indeed wrongly ordered. If you want you can do the following optional exercise to order the contig manually.

Don't close Act for the next exercise.

G. Looking at the PacBio assembly

So the Illumina assembly is not bad, but neither perfect. Let's have a look at the PacBio assembly, which should have finished by now.

First you have to find the final file. It is not obvious to find it, so here a bit of a help:

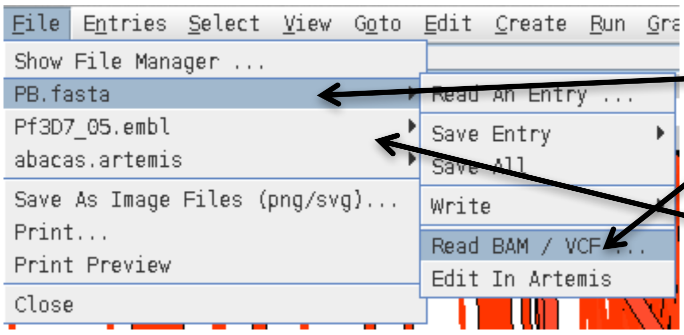
```
$ ln -s */PB.contigs.fasta PB.fasta
```

Now use the assembly-stats script to look at the stats of the assembly. What do you think?

Ok, isn't that great? If this is not, ask your neighbour or a teacher… next have a look in act to compare the results.

```
$ blastn -outfmt 6 -evalue 1e-20 -subject Pf3D7_05.fasta \
-query PB.fasta -out comp.PB.blast

$ act PB.fasta comp.PB.blast Pf3D7_05.embl Pre/comp.blast
Pre/abacas.artemis
```

If you have mapped the reads yourself, adapt the command. You can do the blast with:
formatdb -p F -i Pf3D7_05.fasta
blastall -p blastn -W 30 -m 8 -e 1e-20 -d Pf3D7_05.fasta -i <input> -o comp..blast
(adapt the command to your input file)



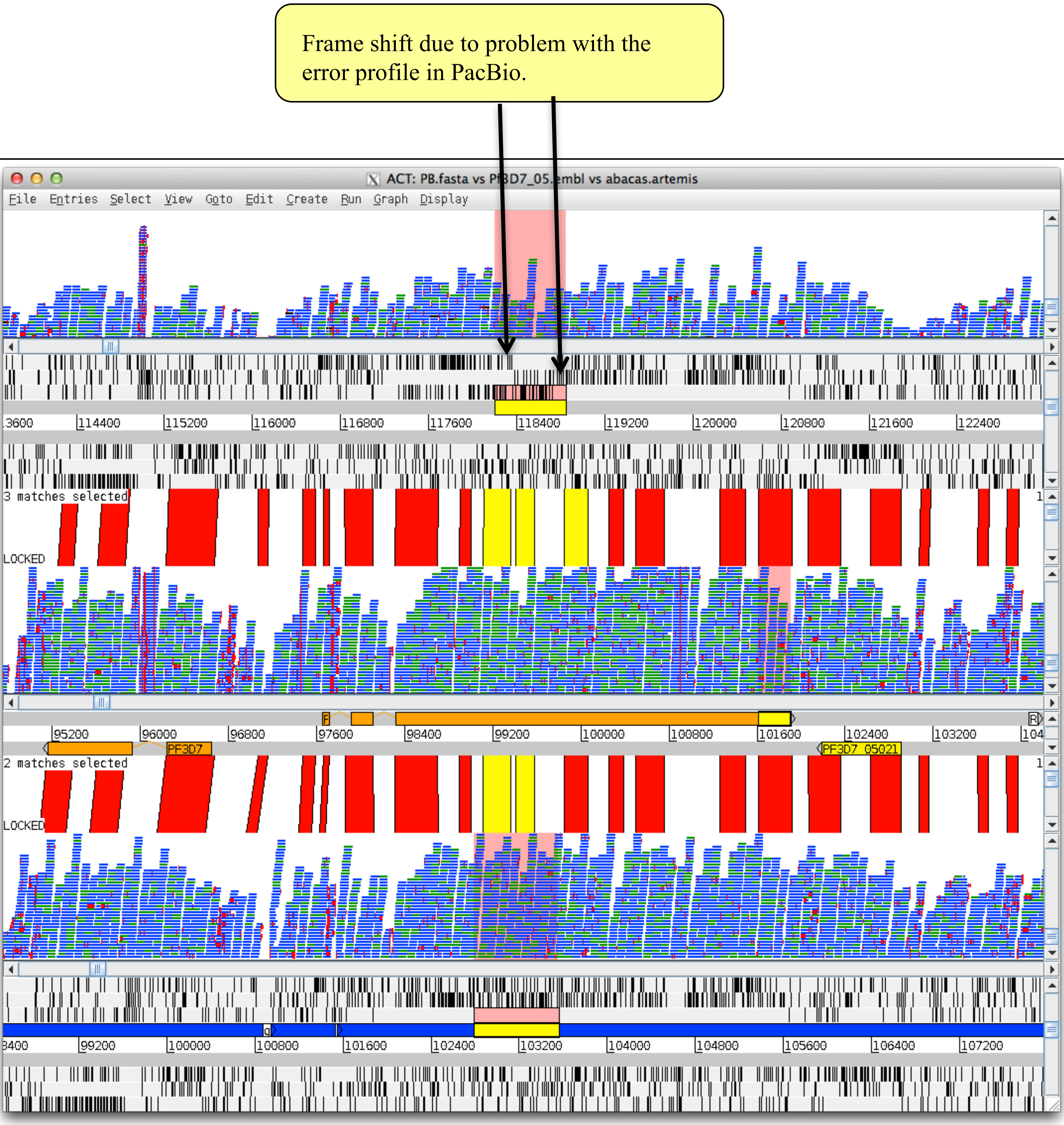
First load the bam **IT_onPacBio.bam** file onto the reference

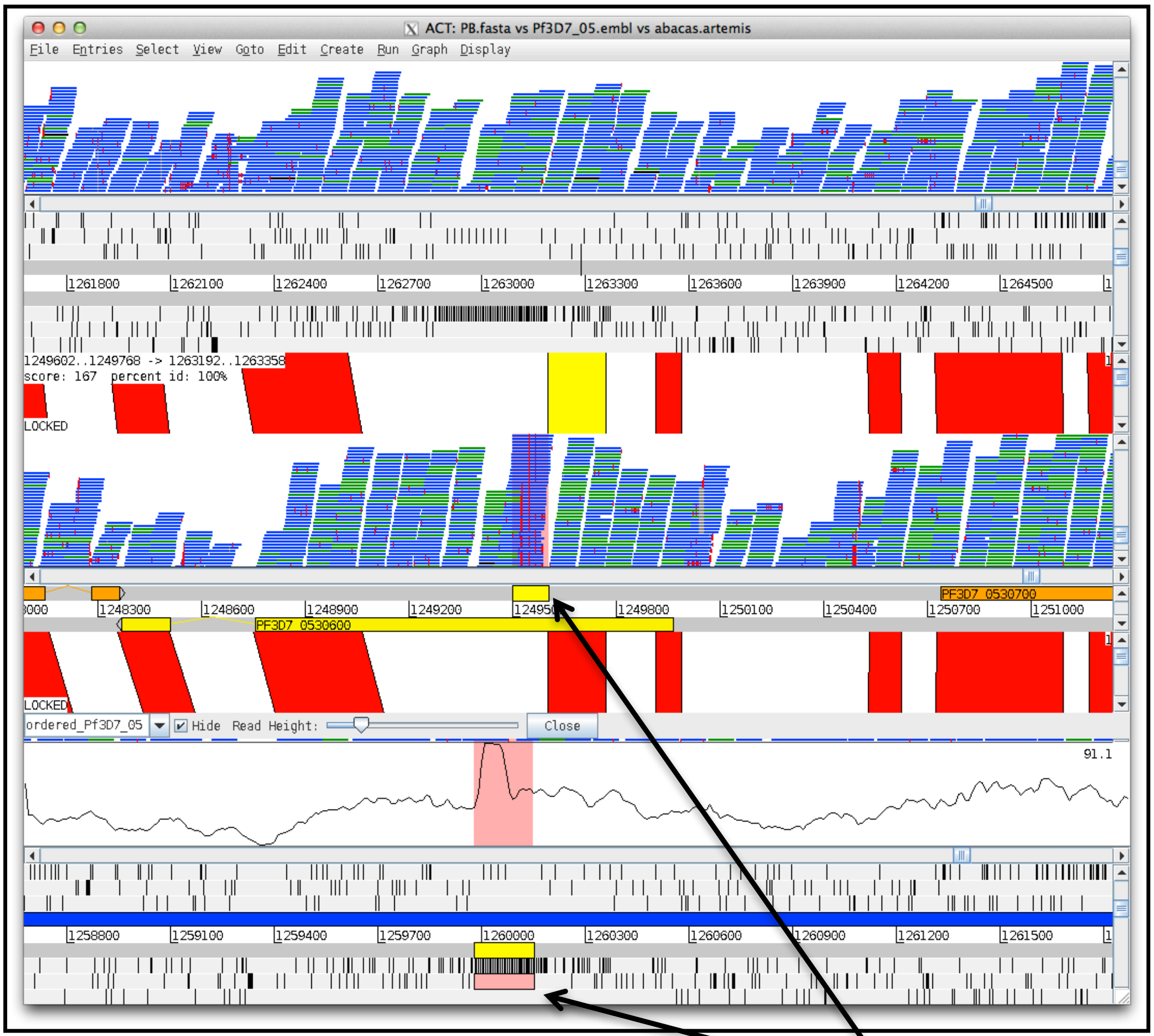
Next load the bam file **IT.Chr5.bam** on Pf3D7_05 and **Pre/IT_onDenovo.bam** on the first genome.(this file is different if you remapped the reads).

So the next task would be to compare the mapped reads against the reference with the mapped reads of the assembly. Notice, these are the same reads! Skim through the assembly, and look for:
1. Regions that weren't covered before
2. Regions where the mapping was weird (i.e. heterozygous SNP, distant mate pairs)
3. Partially mapped reads

Important: highlight SNPs (BAMVIEW, right click, show SNP MARKS). Also show the forward and reverse frame lines (ask for help if needed!).

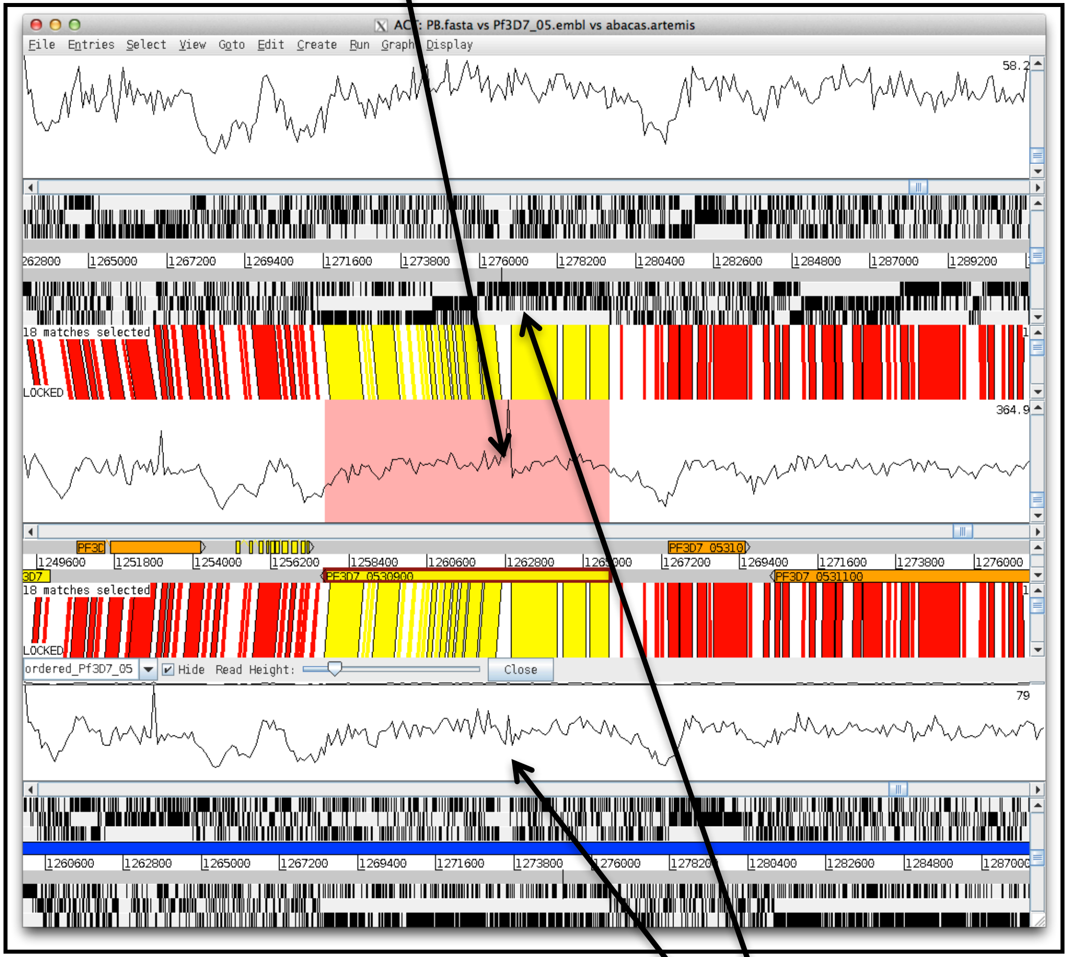
So far we have already established that the VAR genes were too divergent to have mapped reads, but the *de novo* assembly of them looks good.


Frame shift due to problem with the error profile in PacBio.
ACT: PB.fasta vs Pf3D7_05.embl vs abacas.artemis
File Entries Select View Goto Edit Create Run Graph Display
3 matches selected
LOCKED
2 matches selected
LOCKED
PF3D7
PF3D7_05021


ACT: PB.fasta vs Pf3D7_05.embl vs abacas.artemis
File Entries Select View Goto Edit Create Run Graph Display
1249602..1249768 -> 1263192..1263358
score: 167 percent id: 100%
LOCKED
PF3D7_0530600
PF3D7_0530700
ordered_Pf3D7_05
Hide Read Height:
Close
91.1

Those regions look better! No gaps, but compressed repeats. Does it look ok in the PacBio?

PF3D7_0530900: IT has an insertion at this position.


ACT: PB.fasta vs Pf3D7_05.embl vs abacas.artemis
File Entries Select View Goto Edit Create Run Graph Display
18 matches selected
LOCKED
PF3D7_0530900
PF3D7_05310
PF3D7_0531100
ordered_Pf3D7_05
Hide Read Height:
Close

This regions has definitely improved in the Illumina. If you zoom further in, you'll see it is a repetitive region. The PacBio has again frameshifts.

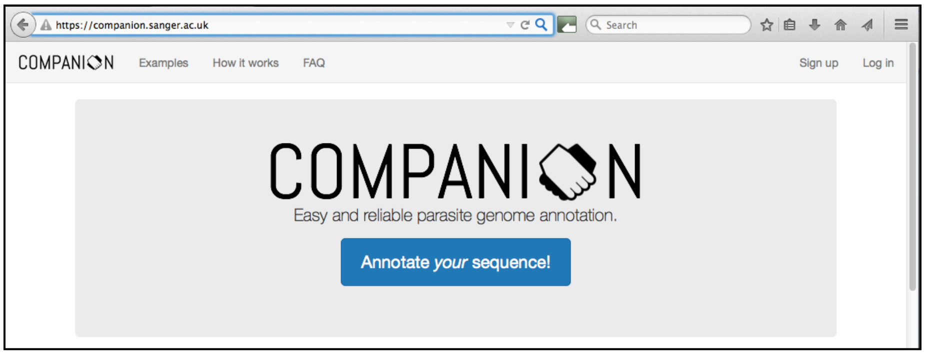
In summary, we did different assemblies and could improve some regions. The PacBio assembly does look very good, but still has some frame shifts. The Illumina has some gaps, but most of the difficult regions are now in a better shape. The best way to prove this is to see which reads map with their full length, without any differences to the reference (red point).

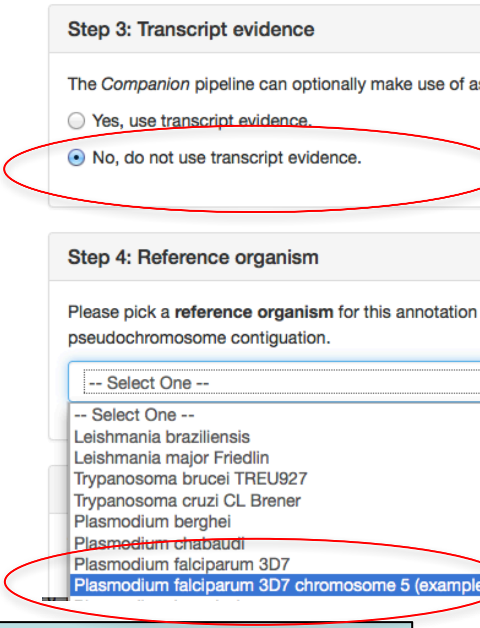
You may have noticed that the new assembly doesn't have any annotation. Before we work on this, some more comments on assembly…

H: Annotation

With the PacBio sequence with have a pretty good assembly. But for the analysis we do need an annotation! Normally we would need to do *ab initio* gene finding. Or we could use a tool that uses the annotation of the reference, and adapt it to the new assembly. We developed a tool called RATT (Otto *et al*., 2011 "Rapid annotation transfer tools"), that can transfer the annotation from a reference to a new assembly.

But this would not covered the new genes… therefore we generated Companion which not only merges *ab initio* gene finding with reference based transfer, but also does RNA detection, functional annotation, pseudo gene detect and in the end generates a ready to submit format. This tool is designed for parasites. But there are alternatives, for Bacteria the RAST server or the Maker server for human. Just search a bit in the web.


https://companion.sanger.ac.uk
COMPANION
Examples
How it works
FAQ
Sign up
Log in
COMPANION
Easy and reliable parasite genome annotation.
Annotate *your* sequence!


Step 3: Transcript evidence

The *Companion* pipeline can optionally make use of a

Yes, use transcript evidence. [✓ No, do not use transcript evidence.]

Step 4: Reference organism

Please pick a **reference organism** for this annotation

pseudochromosome contiguation.

-- Select One --

- -- Select One --
- Leishmania braziliensis
- Leishmania major Friedlin
- Trypanosoma brucei TREU927
- Trypanosoma cruzi CL Brener
- Plasmodium berghei
- Plasmodium chabaudi
- Plasmodium falciparum 3D7
- [✓ Plasmodium falciparum 3D7 chromosome 5 (example]

Open the webpage https://companion.sanger.ac.uk/. Click on "annotate your sequence" and follow the instructions.

The steps should be obvious. Important is:
Step one: Give some names.
Step two: Upload the PacBio assembly.
Step three: Leave it the way it is. (No protein evidence as they triples the run time)
Step four: Select Plasmodium falciparum *P. falciparum* 3D7 Chromosome 5 (example)
Step five: Select "No, do not modify my input sequences."
And confirm through "create job".

This might take 12 minutes for the first to submit… and the results will come through accordingly the submission order of the job. In case you don't want to wait, bookmark the run you started and then look for the preprocessed results, see:

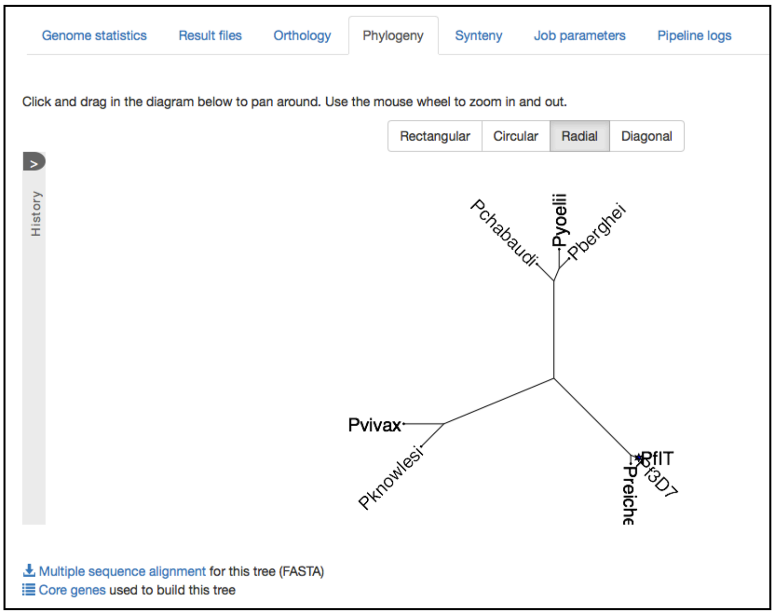
https://tinyurl.com/WTAC-PB2017

Examples of the output of companion. The phylogenetic tree indicates where your newly sequences sits.

The orthmcl is helpful to understand which genes are new in the new genomes.

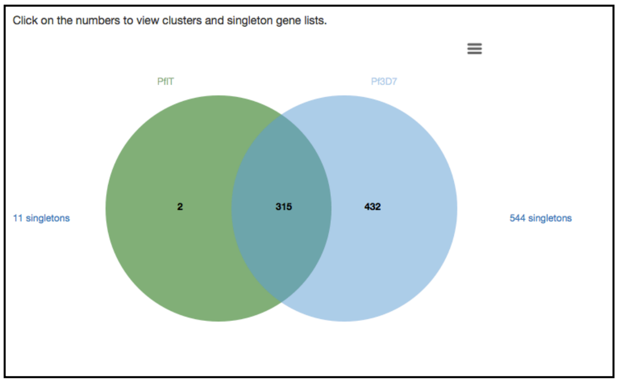
The circos view allows to see the overall synteny between the reference and the new genome.

Obviously, you want to be able to download the result files.

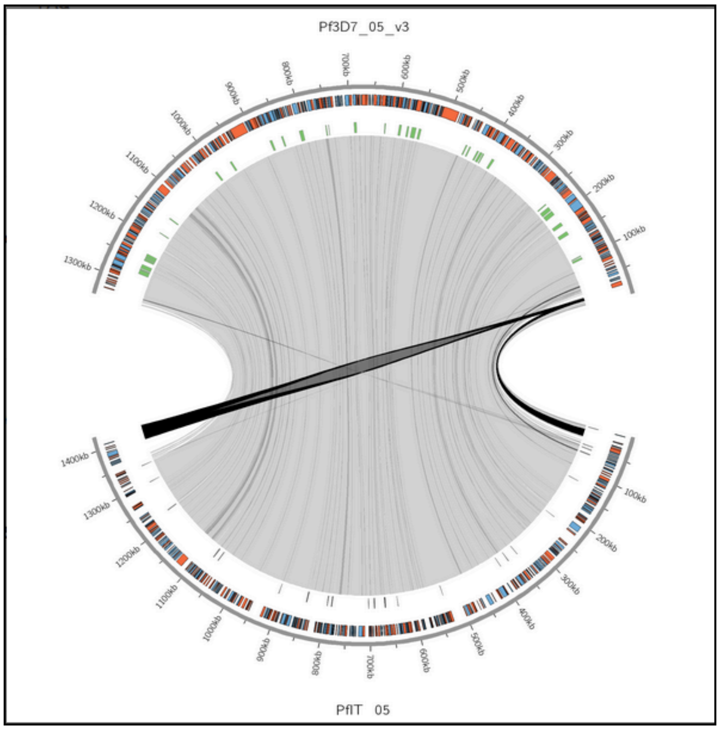


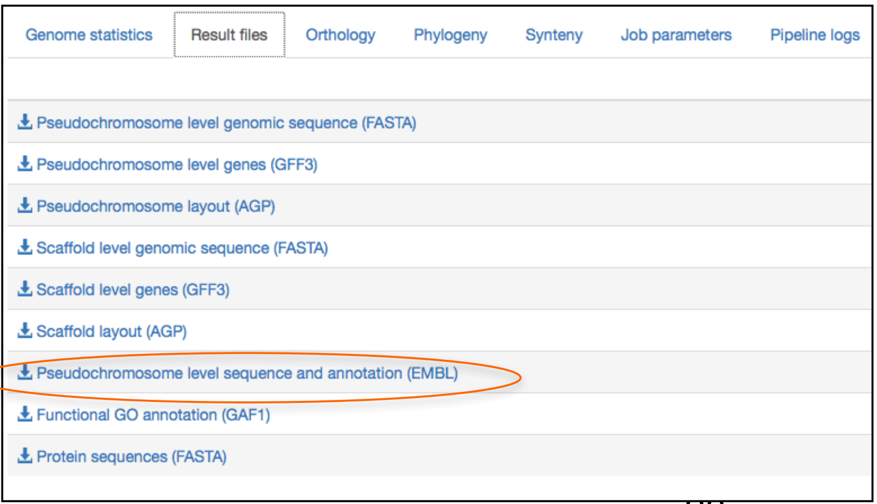
Click and drag in the diagram below to pan around. Use the mouse wheel to zoom in and out.

Multiple sequence alignment for this tree (FASTA)
Core genes used to build this tree



Click on the numbers to view clusters and singleton gene lists.





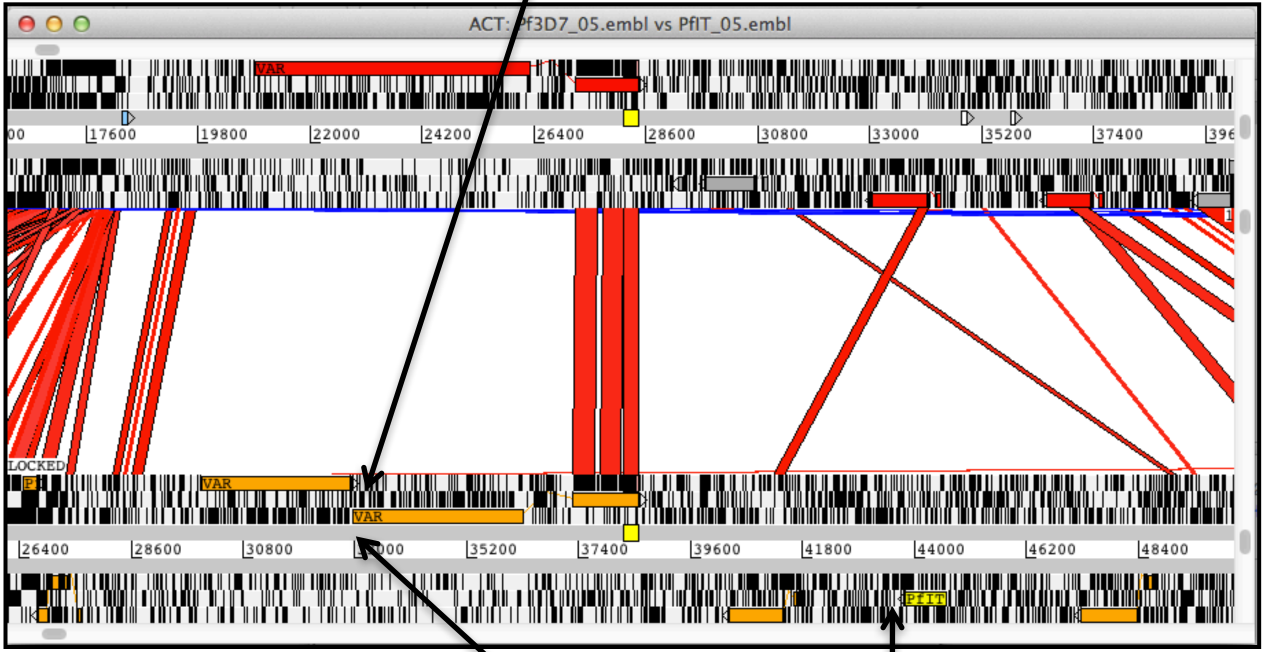
Pseudochromosome level genomic sequence (FASTA)
Pseudochromosome level genes (GFF3)
Pseudochromosome layout (AGP)
Scaffold level genomic sequence (FASTA)
Scaffold level genes (GFF3)
Scaffold layout (AGP)
Pseudochromosome level sequence and annotation (EMBL)
Functional GO annotation (GAF1)
Protein sequences (FASTA)

Skim a little bit through the visualization, orthomcl, tree and the statistics. In the end, download the annotation results (result file tab) as embl file, untar them (tar xzvf embl.tar.gz) and load them into act.

A good way to analyse the transferred gene models is to load the data into act. Start it by:

```
$ act Pf3D7_05.embl comp.PB.blast <name of downloaded embl
file>
```

Gene model is adapted to the frameshift.



Regions without synteny. Diverse or new genes.

In summary, we did different assemblies and could look into some regions. Our new PacBio chromosome 5 of the IT clone has no gaps. The genes are ok annotated. But never the less we can see some over prediction and could improve the models manually if required.

I: Fixing frameshifts

The PacBio sequence is not perfect. First, the telomere repeats are missing - but this has to do how we obtained the subset to do the assembly of just chromosome 5. But on the other site, we still have those frame shift, see pages 16 and 18**.**

One way of correcting those is to run Quiver or icorn2. Let's try icorn2.

With this command you can run RATT. As input we use the reference annotation and the output of abacas.

```
$ icorn2.serial.sh IT.Chr5 500 PB.fasta 1 3 > out.icorn.txt
&
```

Iteratively (3 times) the reads are being mapped, variation is called with gatk and the differences are corrected. Last, the corrections are checked, by looking if the amount of perfectly mapped reads does not decrease. If you don't want to wait, you can have a look at the corrected and annotated sequence.

```
$ act PB.fasta comp.PB.blast Pf3D7_05.embl \
Pre/comp.Pbcor.blast Pre/PfIT.pb.chr5.embl
```

Upload the annotation if you want and then check how many of the frame shifts are corrected…

Here the correction stats of the three iterations (`icorn2.collectResults.pl`):

	Increase in mapped reads		Performed changes			Rejected changes		
#iter	perfect1%	perfect2%	SNP	INS	DEL	Rej.SNP	Rej.INS	Rej.DEL
1	37.39	42.02	153	2493	34	0	71	0
2	41.89	42.26	24	261	3	1	13	1
3	42.23	42.3	11	39	0	0	3	1

Important aspects of the assembly procedure

The secret is the read quality and the insert size

For a good assembly you need good libraries. For Illumina, in our case we used a PCR-free library (Kozarewa et al., 2009), with a good insert size of 350bp. A standard library would have generated far more contigs, and probably an N50 of 1-3kb. With a large insert size library (approx 3kb) our assembly might have returned in less than 10 pieces. For good libraries, you need enough DNA, good hands (experience) and a well tuned sequencing machine.

For Pacific Bioscience you need enough high molecular weight DNA (> 5ug). Important is it size select your library, for small **and** large fragment. We consider a good run, if more then 600mb per cell are product with a mean ROI length > 9kb.

Powerful computers

In this example we assembled a single chromosome. We used reads that were already mapped onto the known IT chromosome. The methodology for the complete assembly would be the same. The k-mer size might need to be bigger because a 49 k-mer might be unique in a chromosome, but not for the complete genome. Also the computer would need to have more memory (up to 30 gb) and more time. For even bigger genomes (>100gb), memory requirements would increase significantly.

Bacterial genomes are easier

The good news is that for bacterial genomes the Illumina methodology should generate sufficiently good results to do the analysis, so between 10-100 contigs. For PacBio one cell should generate the complete genome of a bacteria!

Bin assembly

In some cases, you may just want to do an assembly of the reads that didn't map. This should return you contigs unique to your sample, with less computer power.

Tips

- It is always a good idea to try different programs for any particular problem in computational biology. If they all produce the same answer you can be more certain it is correct.
- The field of assembly develops fast! There are always new tools to improve assembly. For example the reads can be corrected before the assembly, or the assembly can be improved by image and iCORN, as mentioned.
- **Always evaluate your assembly. You can use a reference and map reads back and look for badly mapped mates/reads!**
- **There will be always problems with repeats!**

What to do without a reference?

In some cases no closely related reference is available, or the quality of the reference is rather poor (many contigs, bad annotation). To obtain then a good assembly is far more work intensive (and expensive).

Mix of insert size libraries

To obtain large supercontigs you will need a range of insert sizes. A small one, 300-500bp, a 3kb, and if working with mid size (> 40mb) eukaryotes, 8k and 20kb.

Better your assembly is, easier will be the analysis - less genes split, more context around genes.

Improve assembly

You can improve the assembly as described on page 23 (Further improvement to assemblies)

Expression data

To find the genome models in eukaryotes you will need to use expression data, to manual curate the genome and train gene finder. You would need at least 200 models which can take up to two weeks. In bacteria you would just run a gene finder.

This topic will be covered in the next module.

Different assembler

There are many assembler and to date even for Illumina new assembler are produced. There are several comparison papers, like the Assemblathon and the CAGE paper. Important is to always try different assemblers.

For PacBio also several assemblers are available, e.g. HGAP, SPRAI or Falcon. The first two like PBcR are build around the Celera assembler. Falcon suppose to be able to split haplotypes.

With the advent of long reads, manual improvement of assemblies might be obsolete. Anyhow, in case you are ultra keen or you have Illumina assemblies to correct, below examples how to do so.

IX: Bin assembly (OPTIONAL)

So far we did a completely new assembly of one chromosome of *Plasmodium*. The main motivation was to resolve issues of mid size indels and too divergent or novel sequence. If you are just interested in novel sequence (or very divergent one, so that reads don' t map), you won' t need to do a complete assembly, but just an assembly of non mapping reads. We call this "Bin" assembly.
First we will get all the reads that don' t map against the reference from the bam and then assemble them.

First we get the reads that don't map.

```
$ samtools view -f 0X04 IT.Chr5.bam > bin.IT.Chr5.sam
```

This command returns all the reads that don't map (they have the 0x04 flag in the bamfile). Next we do a normal assembly as before. Note that the way we used the command, we don't have the correct format for read pair data, so we use the reads as single end reads. (Any idea how to get both mates?).

```
$ velveth k.bin_assembly.49 49 -short -sam bin.IT.Chr5.sam

$ velvetg k.bin_assembly.49 -exp_cov auto
```

Have look at the n50. Which genes would you expect to be in this assembly?

The next steps could be to abacas it, or to blast open reading frames… but we might run out of time doing this.

Important to notice is that the bin assembly runs faster, as less reads have to be assembled. The assembly of a certain region might also be better, as less reads are included into the assembly, so certain k-mer might not be repetitive as in the complete assembly.

A perl script would be one option to get also the mate pair of the reads that doesn't map.

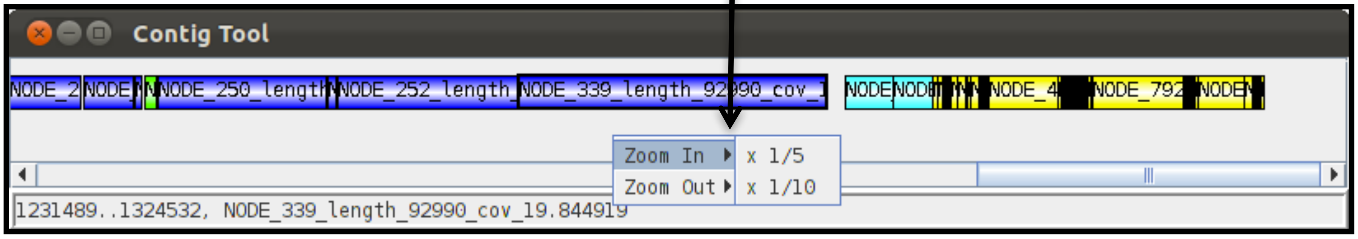
X: Manual contig ordering (Optional)

Here we describe how you can reposition (or order) a contig of your choice manually by drag and drop. At first it might feel fidely, but just give it a go.

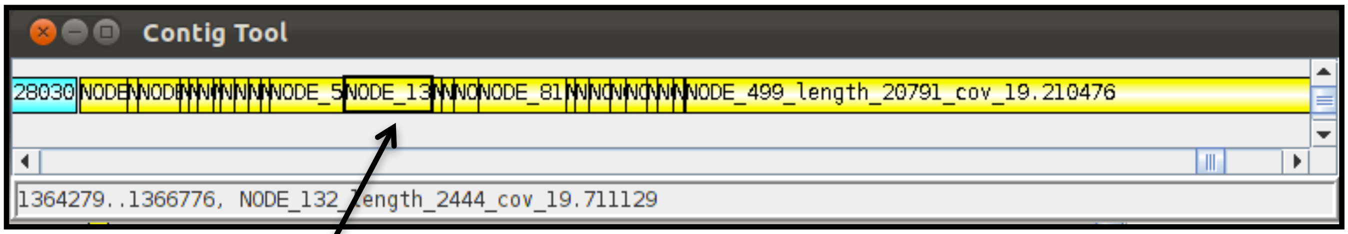
First note the name of the contig you want to reposition (NODE_132). Next note down between which contig you want to place it (NODE_131 and NODE_138).

1. Right click on the contig to order -> Edit -> Contig Ordering.

2. Right click and zoom in x1/10. Be a bit patient when you zoom, for the window to update.

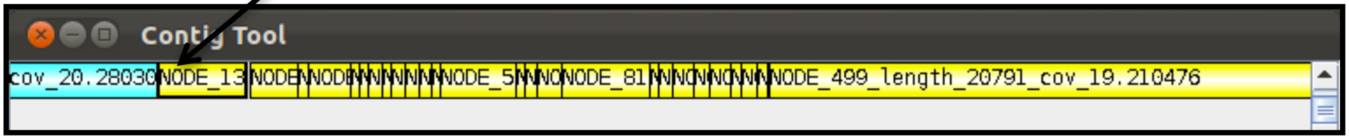


1231489..1324532, NODE_339_length_92990_cov_19.844919

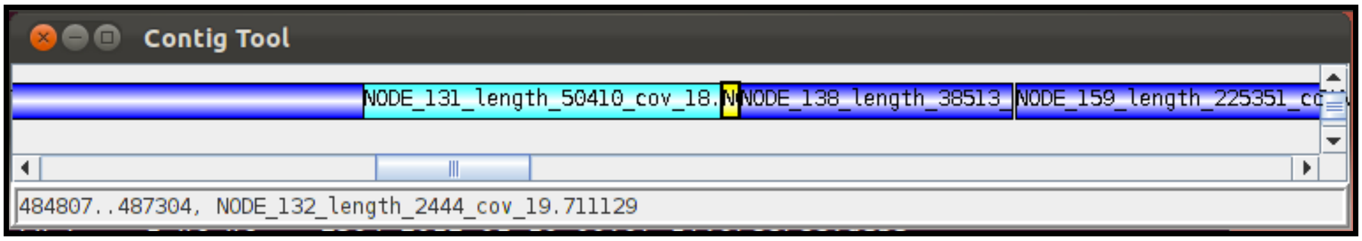


1364279..1366776, NODE_132_length_2444_cov_19.711129

3. Select the contig, and drag it to the left



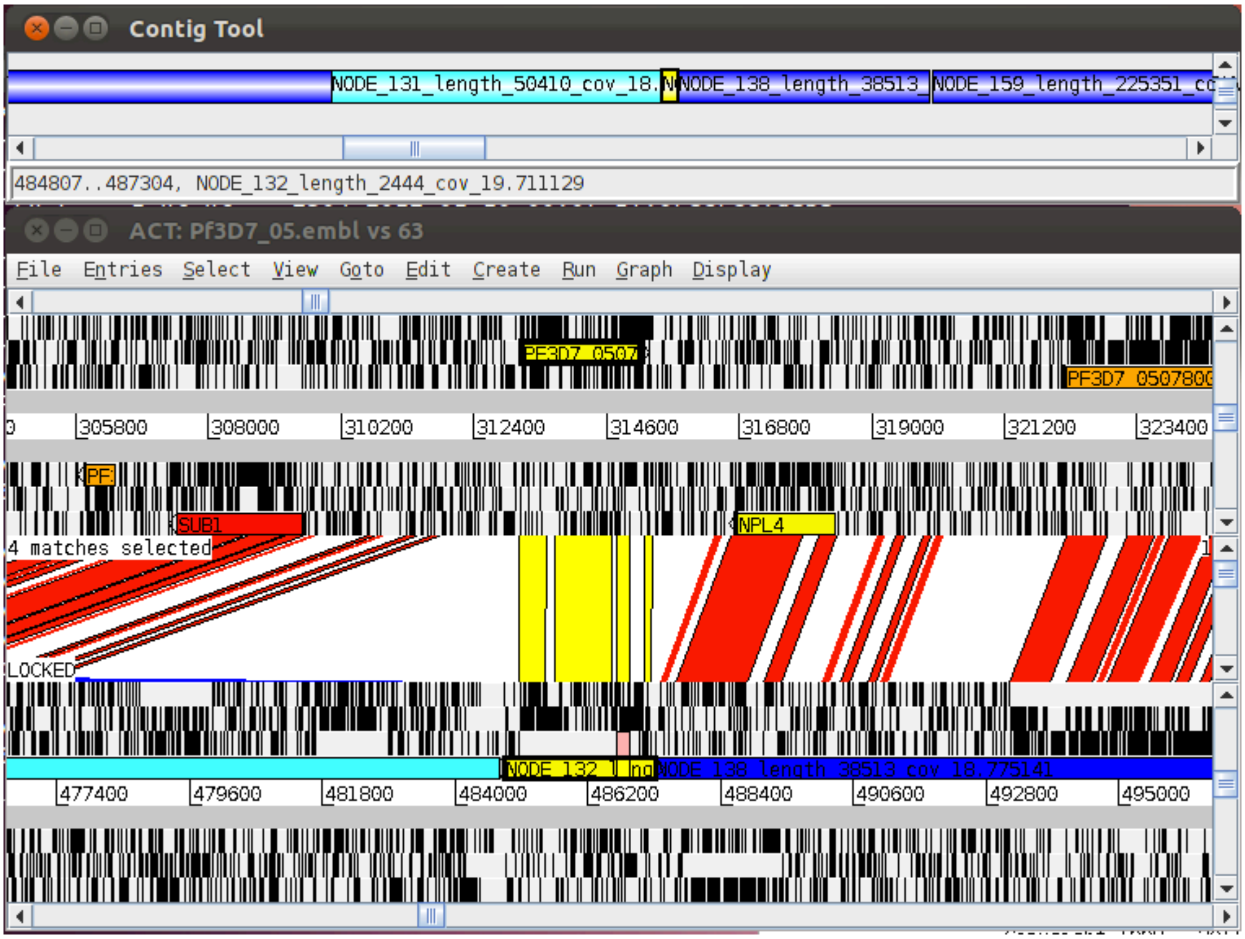
4. Do this until you place the contig at the correct position. Maybe use the zoom out zoom it. Just play around.



484807..487304, NODE_132_length_2444_cov_19.711129

Observe, when you move the contig around, the act window get automatically updated.
At first, the blast hits look weird, just score a bit to a site, and it should look as below.

The gene order is now restored. Any idea why the contig might have been placed wrongly?



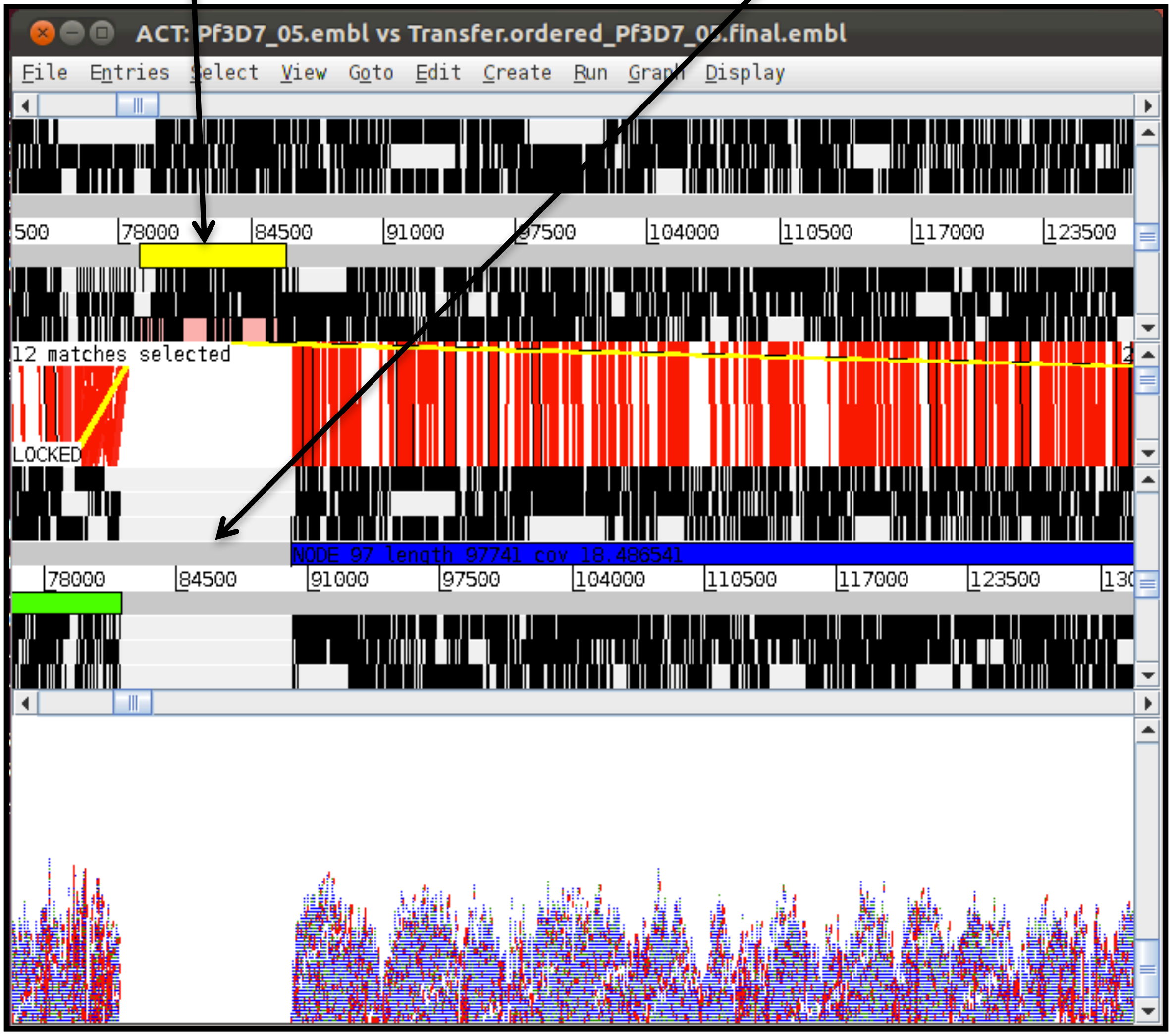
As this exercise is optional, and we are using for the next exercise a pre-computered bam file, you can
1.Save the assembly with the ordered contig, and remap the reads.
2.or, close Act without saving and reopen it.

XI: Manual correction of assembly (Optional)

Were you able to find a miss-assembly? Look for synteny breaks, were a region maps to another part of the genome.

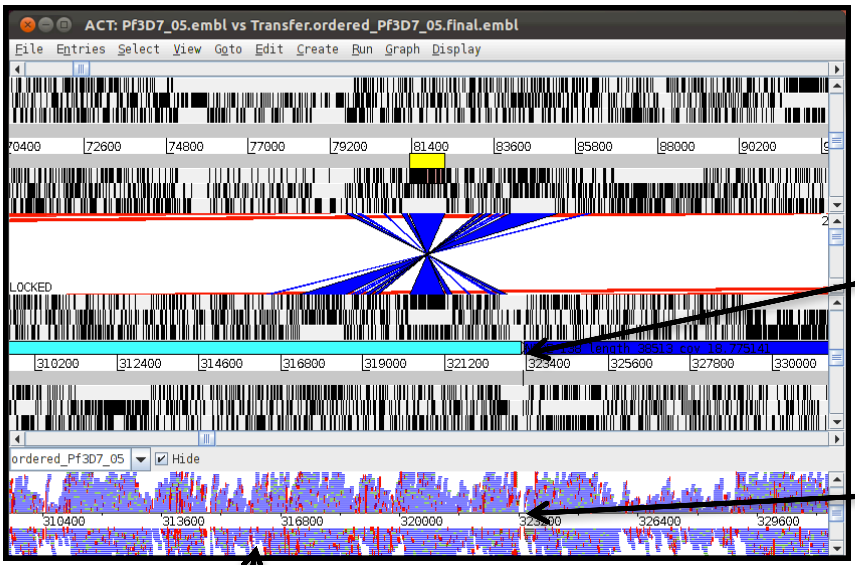
This regions of the reference has similarity to another region of assembly.

In the new assembly is a gap.


ACT: Pf3D7_05.embl vs Transfer.ordered_Pf3D7_05.final.embl
File Entries Select View Goto Edit Create Run Graph Display
500
78000
84500
91000
97500
104000
110500
117000
123500
12 matches selected
LOCKED
NODE_97_length_97741_cov_18.486541
78000
84500
91000
97500
104000
110500
117000
123500

Find the region with the similarity in the new assembly. Does it look like a miss-assembly?

What kind of information could you use to track down a possible miss-assembly?

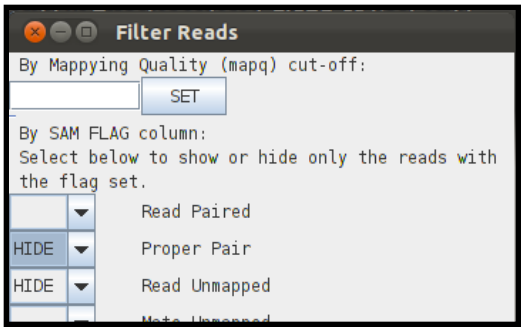

ACT: Pf3D7_05.embl vs Transfer.ordered_Pf3D7_05.final.embl
File Entries Select View Goto Edit Create Run Graph Display
LOCKED
ordered_Pf3D7_05
Hide

This is the region with the similarity.

There is again a gap.

Change the view -> Strand Stack. Can you see read pairs that bridge this gap?

This looks like a miss-assembly. But yet we don' t have the full prove! Zoom in that position and have a look at the mate pairs.
Filter the Proper Pairs (right click, Filter)


Filter Reads

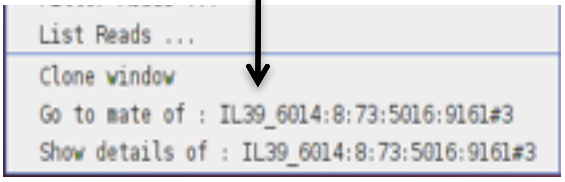
By Mapping Quality (mapq) cut-off:

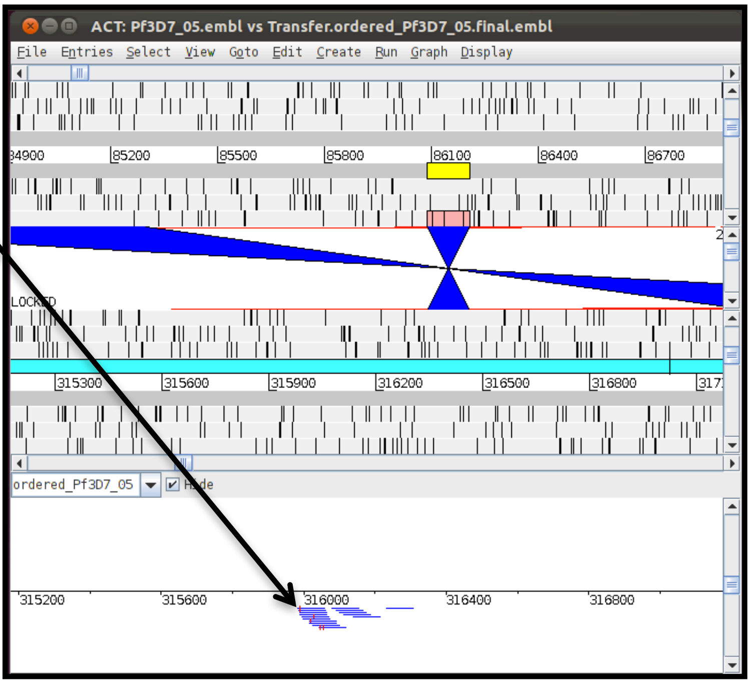
SET

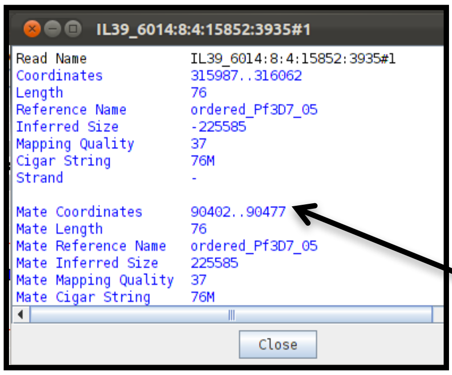
By SAM FLAG column:
Select below to show or hide only the reads with the flag set.

	Read Paired
HIDE	Proper Pair
HIDE	Read Unmapped

Now you can see reads that map on the reverse strand. Right click on one read and select first the last , then the second last option.


List Reads ...
Clone window
Go to mate of : IL39_6014:8:73:5016:9161#3
Show details of : IL39_6014:8:73:5016:9161#3


ACT: Pf3D7_05.embl vs Transfer.ordered_Pf3D7_05.final.embl
File Entries Select View Goto Edit Create Run Graph Display
LOCKED
ordered_Pf3D7_05
Hide


IL39_6014:8:4:15852:3935#1

Read Name	IL39_6014:8:4:15852:3935#1
Coordinates	315987..316062
Length	76
Reference Name	ordered_Pf3D7_05
Inferred Size	-225585
Mapping Quality	37
Cigar String	76M
Strand	-
Mate Coordinates	90402..90477
Mate Length	76
Mate Reference Name	ordered_Pf3D7_05
Mate Inferred Size	225585
Mate Mapping Quality	37
Mate Cigar String	76M

Close

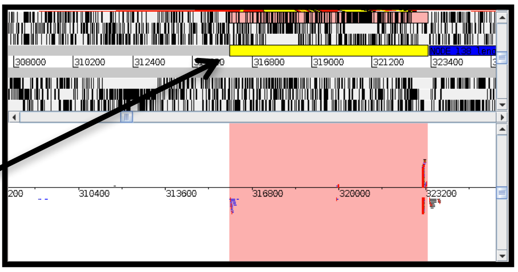
The last option show you were the mate is mapping.
The second last option will just to the position. Where is it?

Over ten mate pairs indicate that they are mapping very far apart, and suggest that the contig should be broken here and place in the gap, of two site before. With the fact that synteny breaks in the core regions of plasmodium are rare and the sequencing gaps at the left hand site, we can assume that this is a miss-assembly.

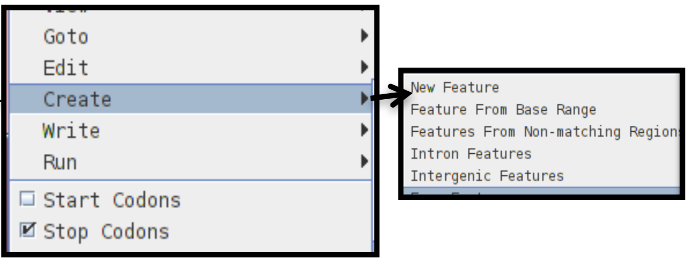
If you want to fix the miss-assembly, this is the way to do it:

1. Delete the light blue contig.

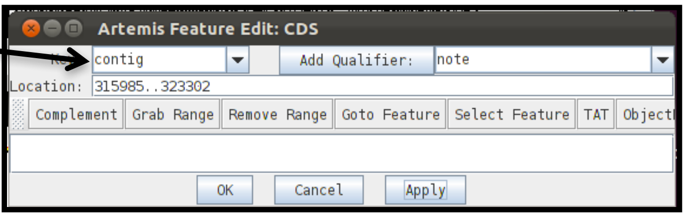
2. Select the region that is miss-place. The reads can help you!



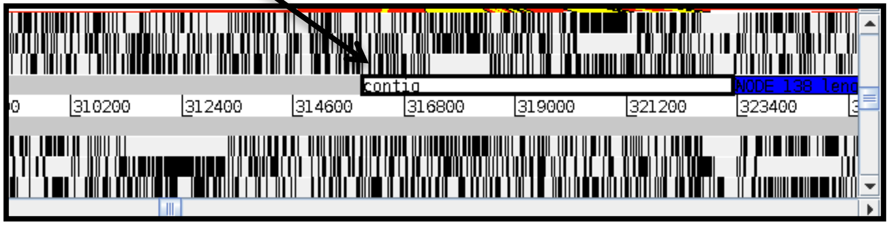
3. Right click on the selected region -> Create -> Feature from Base Range



4. Change the key to contig, and press apply.



5. Now you can order this contig manually as before to the correct postion



Please be aware that this is pretty advanced edit. At the Wellcome Trust we are currently working on software to break this regions automatically (Maybe google REAPER).
After this step one order the new contig set again with abacas.

Perl could here help to find the reads that map wrongly in a more automated easier way.

Further improvement to assemblies

Though until now we could determine that the assembly represent the sequence, it is not perfect, as it still has compressions and gaps. One reason for this is that the assembly of *Plasmodium* is more difficult than for most bacteria (due to the high AT content and repeats). It is nevertheless a useful example, highlighting the problems you will encounter with assembling genomes.
A good assembly of bacteria will return you 20-100 supercontigs. Here we describe which methods you could use to improve the assembly. All the tools are installed on the USB stick, except SSPACE.

SSPACE (Boetzer *et al*, 2011) is a tool that scaffolds contigs. Although Velvet also does this, SSPACE generally performs better.

Abacas (Assefa *et al*, 2009) has the option to design primers which can be used to generate a PCR product to span a possible gap. This new sequence can then be included in the assembly. This process is called finishing.

Circlator (Hunt *et al*, 2015) is a tool that uses long sequencing reads (PacBio or Oxford Nanopore) to tidy up and circularize bacterial chromosomes, plasmids, or eukaryotic mitochondria.

IMAGE (Tsai *et al*, 2010) is a tool that can close gaps in the assembly automatically. First the reads are mapped against the assembly. When a read maps close to a gap and its pair would be “in the gap”, all those reads and their mates are gathered together and a local assembly is done. This is repeated iteratively. This procedure can close up to 80% of the sequencing gaps.

iCORN (Otto *et al*, 2010) can correct base errors in the sequence. Reads are mapped against the reference and differences are called. Those differences that pass a certain threshold are corrected. A correction is accepted if the amount of perfect mapping reads doesn’t decrease. This algorithm also runs iteratively. N.B. perfect mapping = the read and its pair map in the expected insert size without any difference to the iteratively derived reference.

REAPR (Hunt *et al*, 2013) can find errors in the assembly and automatically break at those errors. It does this by checking how read pairs map to an assembly, and so does not need a reference sequence to compare the assembly against.

Several of these tools come as part of the **PAGIT** suite (post assembly genome improvement toolkit) to improve genome assembly (Swain *et al* 2012, http://www.sanger.ac.uk/resources/software/pagit/).

References

Abbot, J. C. et al. (2005) *Bioinformatics* 21(18) 3665-3666.
WebACT – an online companion for the Artemis Comparison Tool

Carver T.J. et al. (2012) *Bioinformatics* 28 (4): 464-9.
Artemis: an integrated platform for visualization and analysis of high-throughput sequence-based experimental data.

Carver T.J. et al. (2012) *Brief Bioinform* (doi: 10.1093/bib/bbr073).
BamView: visualizing and interpretation of next-generation sequencing read alignments.

Carver T. J. et al. (2005) *Bioinformatics* 21: 3422-3.
ACT: the Artemis Comparison Tool.

Kozarewa, I., Z. Ning, et al. (2009). "Amplification-free Illumina sequencing-library preparation facilitates improved mapping and assembly of (G+C)-biased genomes." Nature methods **6**(4): 291-295.

Rutherford et al. (2000) *Bioinformatics* 16 (10) 944-945.
Artemis: sequence visualization and annotation.

Hunt M, Kikuchi T, Sanders M, Newbold C, Berriman M, Otto TD. **REAPR**: a universal tool for genome assembly evaluation. Genome Biol. 2013 May 27;14(5):R47.

Swain MT, Tsai IJ, Assefa SA, Newbold C, Berriman M, Otto TD. A post-assembly genome-improvement toolkit (**PAGIT**) to obtain annotated genomes from contigs. Nat Protoc. 2012

Otto TD, Sanders M, Berriman M, Newbold C. Iterative Correction of Reference Nucleotides (**iCORN**) using second generation sequencing technology. Bioinformatics. 2010

Tsai IJ, Otto TD, Berriman M. Improving draft assemblies by iterative mapping and assembly of short reads to eliminate gaps. Genome Biol. 2010 (**IMAGE**)

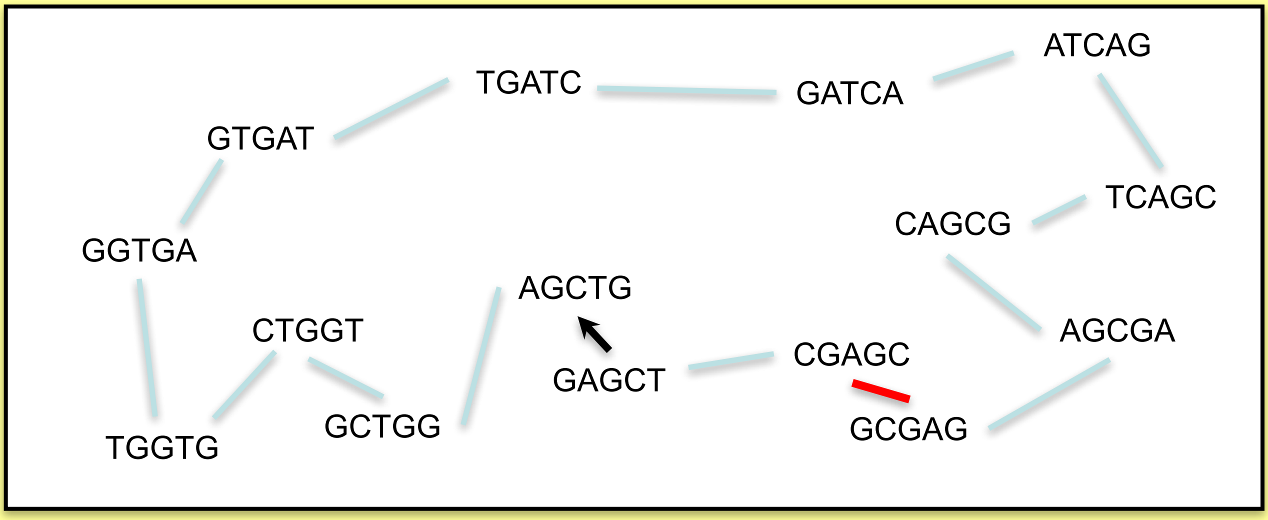
Assefa S, Keane TM, Otto TD, Newbold C, Berriman M. **ABACAS**: algorithm-based automatic contiguation of assembled sequences. Bioinformatics. 2009 Aug 1;25(15):1968-9. doi: 10.1093/bioinformatics/btp347. Epub 2009 Jun 3. PubMed PMID: 19497936; PubMed Central PMCID: PMC2712343.

Step by step of *de novo* assembly:
Otto TD. From sequence mapping to genome assemblies. Methods Mol Biol. 2015;1201:19-50. doi: 10.1007/978-1-4939-1438-8_2

Appendix

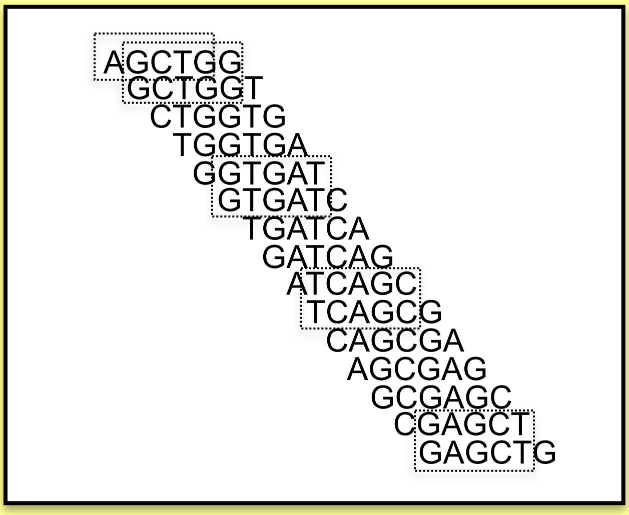
Here we present the solution for the de Bruijn graph exercise, as well as the code for the PERL scripts we mentioned in the assembly module.

The exercise is on page 2 of the assembly module. The first step of the solution would be to generate all the k-mers from the reads. For example the k-mers of length 5 for the read GCGAGC are GCGAG and CGAGC. Those two k-mers will be nodes in the de Bruijn graph, and moreover, will be connected (bold red edge). Doing this for all the reads, generates the following graph:



It is not always easy and might be confusing which node to connect. Remember, the concept of k-mers and the de Bruijn graph are needed to be able to process the large amount of short reads generated by the sequencing machines.

The graph can also be represented as a multiple alignment, as shown on the right hand site. All reads are aligned against each other. The dotted boxes are examples of k-mers.



Now just follow the path thought the graph. Starting at the arrow, the first k-mer is GAGCT, so this would be the start of our contig. The graph indicates the next k-mer AGCTG. So we add a G to the contig. The next k-mer is GCTGG. The new letter is another G. Doing this for the whole graph, we get**: GAGCTGGTGATCAGC.** As you see, the graph is circular. So depending where you start, you get a different contig! If you do a six frame translation, you might see which is a good starting point for the contig.

PERL: Find read pairs that map too far apart

For some applications it would be useful to know whether read pairs map too far apart or whether they don't map pointing to each other. This could be an indication of mis-assemblies, but also duplications or rearrangements, which are are looking for when comparing sequences of different strains.

To find read pairs (RPs) that map too far apart we just need columns 2 and 9 from the BAM file (mapping flag and insert size), and a PERL one-liner. We successively make the query more and more complex, until we find the mis-assembly. Please keep in mind that this is advanced programming! It should give you an idea how useful programming could be.

Assuming your BAM file is called IT_onDenovo.bam and you want to list RPs that map more than 2000bp apart:

```
$ samtools view IT_onDenovo.bam | perl -nle 'my
($read,$flag,$ref,$pos,$mappingQual,$cigar,$mateRef,$matePos,$insertSize,$seq,$
seqQual,$other)=split(/\t/); if($insertSize>2000){print}' | head
```

Here is also a shorter version, using an array (not as readable):

```
$ samtools view IT_onDenovo.bam | perl -nle 'my @ar=split(/\t/);
if($ar[8]>2000){print}' | head
```

There is a lot of output. Many read pairs map all over the place. We would like to bin those into chunks of 1kb, and then list of the most abundant:

```
$ samtools view IT_onDenovo.bam | perl -nle 'my
($read,$flag,$ref,$pos,$mappingQual,$cigar,$mateRef,$matePos,$insertSize,$seq,$
seqQual,$other)=split(/\t/); if($insertSize>2000){print
int($pos/1000)."\t".int($matePos/1000)}' | sort | uniq -c | sort -rn | head
```

This does look more complex! In the output the first column is the number of RPs that connect the first bin (2nd column) with the second bin (3rd column). For example `418 1360 1373` means that 481 RPs connect the region 1360000-1361000 of genome with the regions 1373000-13731000 of the genome. The list shows us that in the subtelomeric regions many RP map far apart!
The following command ignores the subtelomeric ends, by excluding 75kb at each end.

```
$ samtools view IT_onDenovo.bam | perl -nle 'my
($read,$flag,$ref,$pos,$mappingQual,$cigar,$mateRef,$matePos,$insertSize,$seq,$
seqQual,$other)=split(/\t/); if($insertSize>10000 && $pos>75000 && $matePos <
1300000){print int($pos/1000)."\t".int($matePos/1000)}' | sort | uniq -c | sort
-nr
```

The third line   21 81   323 shows us our mis-assembly. What are the other entries?

We are fully aware that is this a quite complex piece of code, and just used as a one liner. It uses the LINUX commands sort and uniq. But keep in mind that this command canfind you all mate pairs mapping too far apart for any bam file (if you adjust the insertSize parameter for your data)!

Getting non mapping reads and their mates

Here is an example of how to get the mates of non mapping reads. It is a good example of PERL one-liners.

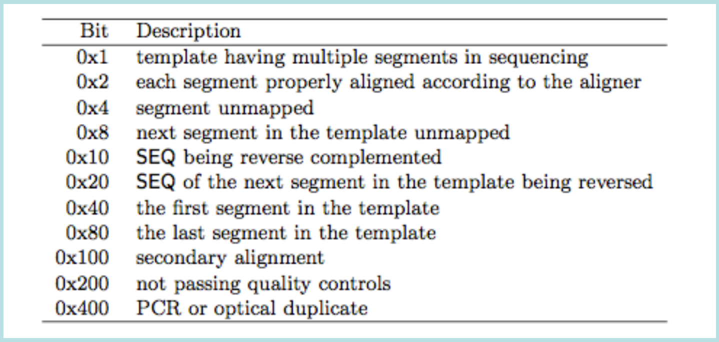
First we are going to get reads that don’t map with PERL. The original command is:

```
$ samtools view -f 0X4 IT.Chr5.bam | head
```

In PERL this would be:

```
$ samtools view IT.Chr5.bam | perl -nle 'my
($read,$flag)=split(/\t/); if ($flag & 0x4) {print }’ | head
```

Now we need to get the reads where the mate is not mapped. Looking at the samtools manual:



Bit	Description
0x1	template having multiple segments in sequencing
0x2	each segment properly aligned according to the aligner
0x4	segment unmapped
0x8	next segment in the template unmapped
0x10	SEQ being reverse complemented
0x20	SEQ of the next segment in the template being reversed
0x40	the first segment in the template
0x80	the last segment in the template
0x100	secondary alignment
0x200	not passing quality controls
0x400	PCR or optical duplicate

The 0x8 tells if the mate pair is not mapped. So if the read is not mapping (0x4) or the mate is not mapping (0x8) then print the sam line into a file:

```
$ samtools view IT.Chr5.bam | perl -nle 'my
($read,$flag)=split(/\t/); if ($flag & 0x4 or $flag & 0x8)
{print }' | sort > NonmappingReadsPlusmate.sam
```

This file can now be used in VELVET for *de novo* assembly as explained in the Assembly module.

We hope that this illustrates the power of PERL one-liners!