# 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 concentrate on data generated using the Illumina Genome Analyzer II, although the techniques you will learn are applicable to other technologies (e.g. 454 GS FLX and ABI SOLiD). A single machine can produce around 60 Gigabases of sequence data in a week! This is the equivalent of around 20 human genomes! The data from the Illumina machine comes as relatively short stretches (35-105 base pairs) of DNA - around 300 million of them. These individual sequences are called **sequencing reads**. The older **capillary sequencing** method produces longer reads of ~500bp, but is much slower and more expensive.

One of the greatest challenges of sequencing a genome 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. Genome assembly using sequence reads of <100bp is more complicated due the high frequency of repeats longer than the read length and the massive amount of data.

Nevertheless assembly is possible, although the results won't be "perfect"; there will be many separate contigs for each replicon. When doing assembly with short reads, the most complicated part is 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. It has nodes that are connected by edges.) For our purpose we encoded the k-mer in a de Bruijn assembly graph (see overleaf). The next step is to simplify the graph and try to generate connected sequences, where k-mers in the graph are connected.

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). Although you don't really need to know how an assembler works, merely get good results, here is a little example of a graph.

Here we are going to do an assembly by hand.

You have been given a set of reads. Generate all the different k-mers of length 5 and include them as nodes in the de Brujin Graph (second white box). We already generated most of the nodes, just four are missing. Next connect all the k-mers that are shared in a read. This should connect most of the nodes in the graph. Last find the path through the graph and generate contigs. (In this example we ignore the reverse complement!)

Reads	AGCTGG	TGGTGA	GATCAG	3
GAGCTG		CTGGTG	ATCAGC	
GCTGGT	GGTGAT		TGATCA	TCAGCG
AGCGAG	CAGCGA	GTGATC	GCGAGC	CGAGCT

de Brujin Graph	ATCAG		
GTGAT GGTGA CTGGT	CAGCG AGCGA		
TGGTG GCTGG			

Enter here the contig sequence:

What is the difficulty here?

Tip: In bacteria this kind of problem always occurs. The aim would be that the contig starts at the origin of replication. Can you do this here? To help you, maybe translate the contig into amino acids....

### A: Generating de novo assemblies is useful even with a reference

In this exercise we will look at a the genome of a lab strain of *P. falciparum*, the IT clone. The reads are already mapped against the reference. We will look at some features in the mapping, as we did with *Chlamydia*. Maybe we will find regions that we cannot properly analyse without doing assembly.

Open up a terminal and navigate to the course data directory, then type:

\$ cd /home/wt/Module\_5\_Assembly/

Now open Artemis with chromosome 5 of *P. falciparum* 3D7 including the mapped reads of the IT clone.

\$ art -Dbam=IT.Chr5.bam,var.IT.Chr5.bcf Pf3D7 05.embl

If you like, you can also map the reads yourself, the fastq files are called: IT.Chr5\_1.fastq and IT.Chr5\_2.fastq and the reference is Pf3D7\_05.fasta.

Maybe have a look through the genome. Do you find something unexpected?

As a hint:

- Should all the region be covered by reads?

- Should the coverage be even?

- What kind of variance (i.e. SNP's) would you expect to see? Can they be heterozygous in a haploid genome?

- Are there read pairs that outside of their expected insert size?

Here are some examples of regions, with problematic mapping: 1) Gene not covered. 2) Heterozygous SNP. 3) Read pairs too far apart.



### B: Doing the first 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 an ordered sam file. You could do this from the original bam file, but this would take around 10minutes, so we will skip to the pre-generated sam file. The command would be: samtools view IT.Chr5.bam | sort > IT2.Chr05.sam Samtools view prints all the read information out, which will be ordered by the sort command. The ">" redirects the sort output to the file IT.Chr05.sam. From this file we

are going to do the k-mer hashing.

```
$ velveth k.assembly.49 49 -shortPaired -sam IT.Chr5.sam
```

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. If you want to see all input options, use:

\$ velveth
(Don't worry - not all options have to be used!)

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 (k.assembly.49) specifies the working directory. The second (-exp\_cov auto) is to let velvet find the median read coverage rather than specify it yourself. Last, the insert size of the library is given (-ins\_length 350).

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).

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, ie it is the median contig size. This n50 parameter 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 as before, with a k-mer size of 49, but using more parameters. 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 try 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 -sam IT.Chr5.sam

```
$ 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:

#### \$ stats k.vel\*/\*.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 mis-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 all the repeats to be collapsed, if they are smaller than the read length. If the repeat unit is smaller than the insert size, then it is possible to scaffold over it, leaving space for the repeats with "N"s.

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

\$ countContigVel.sh k.assembly.49/contigs.fa

Changing the files name depending on the assembly for which you would like the statistics.

### **C: Reference based Assembly**

Velvet has an option to use a reference sequence to help to resolve repetitive kmers. When velvet cannot resolve a repetitive region in the de Brujin graph, it can look up where reads map against in the reference. Remember, the k-mer is always shorter than the reads. So the k-mer can be repetitive (it occurs at least twice in the genome), but not necessarily the read. Knowing from which regions of the genome the repetitive k-mer originates from can help to untangle the graph into a simpler graph by splitting nodes. This simplifies the search of the path through the graph. This module is called velvet columbus.

Although it is still a 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. It is important to map the reads against an accurate reference which we have made previously.

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

Run it as follow:

```
$ velveth k.columbus.55 55 -reference -fasta Pf3D7_05.fasta
-shortPaired -sam IT.Chr5.sam
```

```
$ velvetg k.columbus.55 -exp_cov auto -ins_length 350
-min contig lgth 200 -cov cutoff 5
```

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

\$ stats k.\*/\*.fa

### D: 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 to look at this would be just to open the contigs in Artemis.

Assuming you want to look at the k-mer 55 of the velvet columbus assembly, type:

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

Entry: <mark>I contigs.fa</mark> Nothing selected	t <u>C</u> reate <u>Run G</u> raph <u>D</u> isplay		
GC Content (%) Window size: 120			62.5
Anna Maria Ing ang katalan kata Mana	La construction and the	1	
	pagangangan 1 hatak Masser and and a shake when	at whe at what when the other to	h du alla Maria
	A second a second second	nikku, "Mu Mu Au Maku, niku niku	M Malata A A
· · · · ·			0.0
▲			
D D D D D D D D D D D D D D D D D D D	<b>D ID ID ID ID ID ID ID </b>		leson
D D 10 148 NODE_33_length_431_cov_20.494919_cov_ 1295	D         IB-III D           52.335165         JODE_76           126600         [32500	7751 41 [45500 [52000 [58500	65000
D D 10 089 NODE_33_length_431_cov_20.484914 [6500 1_3000 1_295	D         UB/UID D           52.335165         JODE 76 length 209 cov 55.82           60         [29:00	7751 41 [45500 [52000 [58500	65000
D D1D D49 NODE_33_length_431_cov_20.484910 [6500 ]13000 [1390	D 1001010 D 10 52.335155 JODE 76 length 209 cov 55.82 29:00 [39:00 ] 32:500 [39:00 ]	7751 41 [45500 [52000 [58500	
D D1D HB NOOE_33_length_431_cov_20.484916 _cov [esco 1_3000 _1295]	District         District           23:35165         500E         76         Length         209         cov         55.62         60         290:00         200:00 <td< td=""><td>7751 41 [45500 [52000 [58500</td><td></td></td<>	7751 41 [45500 [52000 [58500	
D         D         ID         ID <td>Directory         Directory         <thdirectory< th="">         Directory         <thdirectory< th="">         Directory         <thdirectory< th=""> <thdirectory< th=""> <thdir< td=""><td>775] 41 [45500 [52000 [58500</td><td></td></thdir<></thdirectory<></thdirectory<></thdirectory<></thdirectory<></td>	Directory         Directory <thdirectory< th="">         Directory         <thdirectory< th="">         Directory         <thdirectory< th=""> <thdirectory< th=""> <thdir< td=""><td>775] 41 [45500 [52000 [58500</td><td></td></thdir<></thdirectory<></thdirectory<></thdirectory<></thdirectory<>	775] 41 [45500 [52000 [58500	
D         D <thd< th=""> <thd< th=""> <thd< th=""> <thd< th=""></thd<></thd<></thd<></thd<>	Directory         Directory <thdirectory< th="">         Directory         <thdirectory< th="">         Directory         <thdirectory< th=""> <thdirectory< th=""> <thdir< td=""><td>775] 41 [45500 [52000 [58500</td><td></td></thdir<></thdirectory<></thdirectory<></thdirectory<></thdirectory<>	775] 41 [45500 [52000 [58500	

Possible CDS have 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, a 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 open reading frame? Can you generate a gene model and blast it?

(Double click mouse wheel or middle button over the open reading frame (ORF); "Create" - > "Feature from base range"; Accept the new model; Select the model and "Run" a "BLAST against Uniprot")

## E: Contig ordering

At the Wellcome Trust Sanger Institute we have developed a tool called ABACAS (Assefa et al., 2009) to order contigs against a reference. Any spaces between the contigs (gaps) can be filled in with "N" characters. The result is called a pseudo-molecule. This 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 to append the bin onto the pseudo-molecule

-d to use the standard comparison parameter, which is in this case faster

-o to give the prefix for the output file name (IT.ordered)

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 but gives lower sensitivity.

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

-c Reference sequence is circular





-12-

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?





-14-

# F: Manual contig ordering (Optional)

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

First note the name of the contig you want to reposition (NODE\_132). Next note down between which contigs you want to place it (NODE\_131 and NODE\_138).





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



Observe, when you move the contig around, the ACT window gets automatically updated. (In some cases, the ACT comparison might look strange; just scroll to update the comparison file.)

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

😸 🖨 💷 Contig Tool	
NODE_131_length_50410_cov_18. <mark>N</mark> NODE_138_length_38513_NODE_159_leng	th_225351_cc
	► ►
484807487304, NODE_132_length_2444_cov_19.711129	
🗙 🖃 🗉 ACT: Pf3D7_05.embl vs 63	
File Entries Select View Goto Edit Create Run Graph Display	•
anii 1111 mmaanii 1111111 11111111111111111111111111	323400
4 matches selected	
LOCKED	// //-
ի հայ է որ վել այն հայտապատան հայտությունը հայտարածած հայտարանին ու ու հարավոր հայտարանը հայտարանը հայտարանը հա	
UNDE 132 1 holy00E 138 length 38513 Cov 18.775141	

As this exercise is optional, and we are using a pre-computered bam file for the next exercise, you can either:

- 1. Save the assembly with the ordered contig, and remap the reads, or
- 2. Close ACT without saving and reopen it.

### G. Evaluating the assembly

The motivation for the assembly was to untangle regions where the mapping looked weird. For example, reads don't map against VAR genes as the amino acid identity between them is less than 30%.

Obviously, we want to see if the two gene examples from page 4 are in better shape.

If you are using your own assembly (not the k.columbus.55/contigs ordered with abacas), map the reads (IT.Chr5\_1.fastq / IT.Chr5\_2.fastq) with bwa against your abacas results. If you want to use a preprocessed version (we use a BLAST comparison rather the crunch file, as this will make the differences more obvious), just type :

\$ act 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 <your abacas ouptut> -o comp.blast (adapt the command to your input file)

😁 🔿 🔿 📉 🗚 Eile Entries Select View Goto E	<b>CT: Pf3D7_05.embl</b> vs e	First load the bam file IT.Chr5.bam onto the reference
Show File Manager		
Pf3D7_05.emb1	Read An Entry	
ordered_Pf3D7_05.embl	Save Entry 🔹 🕨	Next load the bam file
Save As Image Files (png/jpeg)	Save ATT	Pre/IT onDenovo.bam (this file
Print	Write 🕨	is different if you remapped the
Print Preview	Read BAM	reads onto the ordered contigs).
Close	Edit In Artemis	

So the next task would be to compare the reads mapped against the reference to the reads mapped against the assembly. Notice, these are the same reads! Skim through the assembly, and look for:

1.Regions that aren't covered in the reference but are in the ordered contigs 2.Regions where the mapping against the reference is weird (i.e. heterozygous SNP, distant mate pairs)

3.Partially mapped reads

Important: highlight SNPs (BAMVIEW window, right click, show SNP MARKS).

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.

PF3D7\_0504700: Though the assembly has better mapped reads, there are still gaps in the new gene model.





In summary, we did different assemblies and could improve some regions. Our new chromosome 5 of the IT clone 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 marks).

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

# H: Manual correction of assembly (Optional)

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



Find the region with the similarity in the new assembly. Does it look like a mis-assembly? What kind of information could you use to track down a possible mis-assembly?



Over ten mate pairs indicate that they are mapping very far apart, and suggest that the contig should be broken here and placed in the gap that we saw on page 20. With the fact that synteny breaks in the core regions of *Plasmodium* are rare and the assembly has a contig break at the left hand side, we can assume that this is a mis-assembly.

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



Please be aware that this is a pretty advanced editing. At the Wellcome Trust we are currently working on software to break these regions automatically. (Perhaps google in for REAPR) After this step you can order the new contig set again with abacas.

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

### **Further improvement to assemblies**

Though until now we could determine that the assembly represents 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 that of most bacteria (due to the high AT content and repeats). It is nevertheless a useful example, highlighting the problems you will encounter when assembling genomes.

A good assembly of a bacterial genome will return 20-100 supercontigs. Here we describe which methods you could use to improve the assembly. All the tools are installed on the USB stick.

**SSPACE:** This tool can scaffold contigs. Although velvet also scaffolds contigs, SSPACE generally performs better.

**Abacas** 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.

**Image** (Tasi *et al*, 2010) is a tool that can close gaps in the assembly automatically. First the reads are mapped against the assembly. When one mate maps close to a gap and its mate 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. Again, 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 mate map in the expected insert size without any difference to the iteratively derived reference.

If you are interested in running these programs it is worth downloading the PAGIT suite (post assembly genome improvement toolkit), that contains tools including IMAGE and iCORN to improve genome assembly. (http://www.sanger.ac.uk/resources/software/pagit/)

### H: 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 sequence which is too divergent or novel.

If you are just interested in novel sequence (or very divergent regions such 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.

It is important to notice that the bin assembly runs faster, as fewer reads have to be assembled. The assembly of a certain region might also be better, as fewer reads are included in the assembly, so certain k-mers might not be as repetitive as they were in the complete assembly.

A PERL script would be one option to get the mate pairs of the reads that don't map.

### I: Annotation transfer

Now we have the contigs ordered against the reference. This is an achievement, but yet we know nothing about the position and function of possible genes. Normally we would need to do *ab initio* gene finding (this will be covered later). But can't we just use 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. In the first step the similarity between the two sequences is determined and a synteny map is constructed. This map is used to map the annotation of the reference onto the new sequence. In a second step, it tries to correct gene models. One advantages of RATT is that the complete annotation is transferred, including descriptions. Thus careful manual annotation from the reference becomes available in the newly sequenced genome. Obviously, where no synteny exists, no transfer can be done. Let's see if this will work for our assembly.

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

```
$ start.ratt.sh embl Pre/abacas.fasta Transfer Strain >
out.ratt.txt
```

Transfer is a prefix for the result and Strain is the transfer type. RATT generates a lot of output, such as the synteny block, which genes were corrected and most importantly how many genes were transferred. It's all in the file out.ratt.txt. With the following step you get the transfer statistics:

\$ grep -iA 1 "gene models" out.ratt.txt

285 Gei	ne models	transferred	correctly
---------	-----------	-------------	-----------

5 Gene models partially transferred.

```
5 Exons not transferred from partial CDS matches.
```

```
31 Gene models not transferred.
```

So around 10% of the genes couldn't be transferred at all. Those are probably in the subtelomeric regions. It looks like that in 5 genes, one exon couldn't be transferred.

Next we are going to have a look at the annotation in Artemis. It can be loaded by

```
$ art Transfer.ordered_Pf3D7_05.final.embl &
```

Skim through the genome. Can you see regions that have clear open reading frames (ORFs) that are not annotated?

In Malaria around 50% of the genes are spliced. In the first round *Plasmodium falciparum* annotation it was difficult to find all exons correctly. How well do you think RATT performs?

😣 🔿 🗊 Artemis Entry Edit: Transfer	ordered_Pf3D7_05.final.er	nbl		
<u>E</u> ile E <u>n</u> tries <u>S</u> elect <u>V</u> iew G <u>o</u> to <u>E</u> dit <u>C</u> reate <u>B</u> un <u>G</u> raph <u>D</u> isplay				
Entry: 🔽 Transfer.ordered_Pf3D7_05.final.embl				
72 selected bases on reverse strand: 13	341201334191 = complemer	it (123097123168)		
			PF3D7_050280	0
<u>120000 120800 121600</u>	<u>1</u> 22400 <u>1</u> 23200	124000 124800	<u>1</u> 25600 <u>1</u> 26400	127200 128
				PF3D7_0502
		PE3D7_0502700		
				K <b>oʻt oʻ i i</b> ii ii i i i i i
<b>•</b>	PF3D7 050	2600		▼ 
 IISICYICCSIL#	T * N H R R R H	FPVCSCLP	LOECVHIY	ΙΥΙΥΙΥΙΑ
# F Q Y V T F Y V F C K	HETIEEDIL	FLFVRVYP	FKNVYIYI	YIYIYI
ATAATTTCAATATGTTACATTTG TGTAGTATTTTGTAA	ACATGAAACCATAGAACAAGACATTCT	ТТТССТОПТТОПСОТОТСТАССС	CTTCAAGAATGTGTACATATATATA	
		40  123160		
	/ H F K L L L C E	KGTQEHRG	R * S H T C I Y I	YIYIYI
	C S V M S S S M R M F G Y F F V N K	KRNTRT+G CEOKNTDVG	K L F T Y M Y I E L I H V Y I Y	Y I Y I Y I Y I Y I Y I Y I Y
				•
			1	
			1	
Those models seem to b	<mark>be</mark>	Also the s	plice site is	
correctly transferred.		correct (99	9% GT -AG)	

Some models are wrong. In Artemis it is easy to find certain types of incorrect gene model:

View -> Feature filter -> Suspicious Start / Stop codon or Stop in Translation To find incorrect splice sites, select -> feature select. Enable "Contains introns without GT..AG"

Those should find you most of the problematic gene models.

Look for the following gene PF3D7\_0515600. Can you correct this gene model?



Due to the gap, RATT placed the last three exons incorrectly. Tomorrow we are going to revisit annotation issues.



In summary, we did different assemblies and looked into some regions. Our new chromosome 5 of the IT clone has some gaps, but most of the core gene models could be trensferred.

### Important aspects of the assembly procedure

#### The secret is the read quality and the insert size

For a good assembly you need good libraries. 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 fewer than 10 contigs. For good libraries, you need enough DNA, good hands (experience) and a well tuned sequencing machine.

#### **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 chromsome, 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 this methodology should generate sufficiently good results to do the analysis.

#### **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 a good assembly in this case 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.

The better your assembly is, the easier the analysis will be – fewer genes will be split, and there will be more context around the genes.

#### Improve assembly

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

#### Annotation

To find the genome models in eukaryotes you will need to manually generate gene models and train gene finders. You would need at least 200 manually curated gene models which can take up to two weeks to generate and curate. It is very useful to have expression data (ESTs or RNA-Seq). In bacteria you would just run a gene finder like glimmer.

Actually, this topic will be covered in the next module.