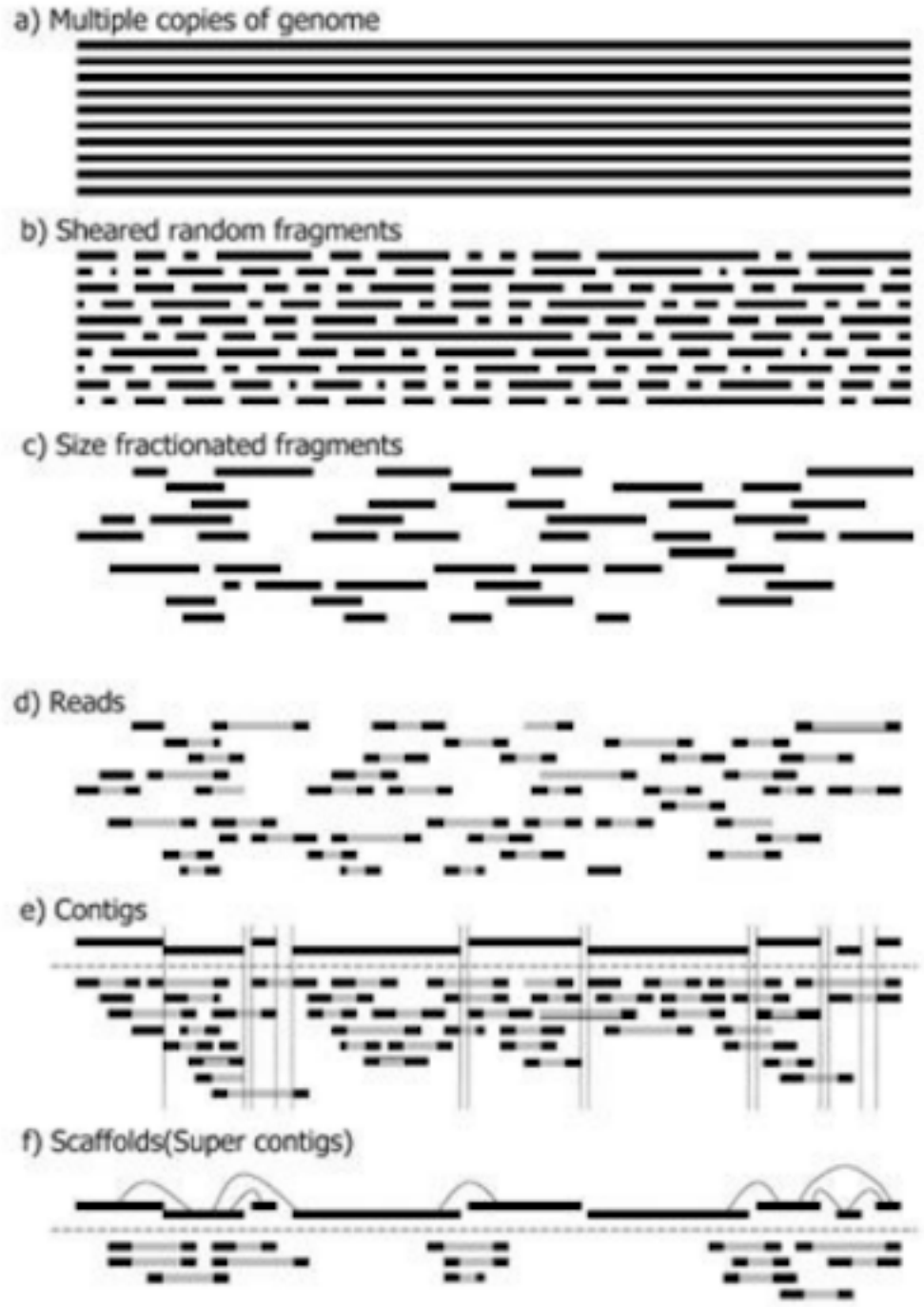
1 Genome Assembly

1.1 Introduction

Genome assembly is the process of taking a large number of fragments of DNA and putting them back together to create a representation of the original DNA sequence from which they originated.



Many genomes contain large numbers of repeat sequences. Often these repeats are thousands of nucleotides long, and some occur in many different locations in the genome. This makes genome assembly a very difficult computational problem to solve. However, there are many genome assembly tools that exist that can produce long contiguous sequences (contigs) from sequencing reads. The assembly tool that you use will be determined by different factors, largely this will be the length of the sequencing reads and the sequencing technology used to produce the reads.

In this practical we will assemble one chromosome of a malaria parasite: Plasmodium falciparum, the IT clone. We have sequenced the genome with both PacBio and Illumina and pre-filtered the reads to select only those reads from a single chromosome.

1.2 Learning outcomes

On completion of the tutorial, you can expect to be able to:

- Describe the different approaches to genome assembly
- Generate a genome assembly from illumina data
- Generate a genome assembly from PacBio data
- Generate statistics to evaluate the quality of a genome assembly
- Estimate the size of a genome assembly

1.3 Tutorial sections

This tutorial comprises the following sections:
1. PacBio genome assembly
2. Assembly algorithms
3. Illumina genome assembly 4. Genome assembly estimation
5. PacBio genome assembly again

1.4 Authors

This tutorial was written by Jacqui Keane based on material from Shane McCarthy and Thomas Otto.

1.5 Running the commands from this tutorial

You can follow this tutorial by typing all the commands you see into a terminal window. This is similar to the "Command Prompt" window on MS Windows systems, which allows the user to type DOS commands to manage files.

To get started, open a new terminal window and type the command below:

[1]:
```
cd ~/course_data/assembly/data
```

1.6 Let's get started!

This tutorial requires that you have canu, jellyfish, velvet, assembly-stats and wtdbg installed on your computer. These are already installed on the virtual machine you are using. To check that these are installed, run the following commands:

```
[ ]: canu
```

```
[ ]: jellyfish
```

```
[ ]: velvetg
```

```
[ ]: velveth
```

```
[ ]: assembly-stats
```

```
[ ]: wtdbg2
```

This should return the help message for software canu, jellyfish, velvet, assembly-stats and wtdbg2 respectively.

If after this course you would like to download and install this software the instructions can be found at the links below, alternatively we recommend bioconda for the installation and management of your bioinformatics software.

- The canu website
- The jellyfish github page
- The velvet wesite
- The wtdbg2 github page

To get started with the tutorial, go to the first section: Pacbio genome assembly

2 PacBio Genome Assembly

First, check you are in the correct directory.

[ ]:
```
pwd
```

It should display something like:

`/home/manager/course_data/assembly/data`

Now we are going to start the PacBio assembly using the `canu` program. It first corrects the reads and then uses the `Celera` assembler to merge the long reads into contigs. To see the contents of the directory, type:

[ ]:
```
ls
```

The pre-filtered PacBio reads are called `PBReads.fastq.gz` - take a look at the contents of this file.

[ ]:
```
zless -S PBReads.fastq.gz
```

What do you notice compared to the Illumina fastq files you have seen previously?

Now we will start the assembly with `canu` (https://canu.readthedocs.io/).

NOTE: This will take some time, so we will start it running now in the background and hopefully it will complete while we work on the other exercises.

[ ]:
```
canu -p PB -d canu-assembly -s file.specs -pacbio-raw PBReads.fastq.gz &>␣
↪canu_log.txt &
```

The `-p` option sets the prefix of output files to PB, while the `-d` option sets the output directory to `canu-assembly/`. The `&` at the end will set this command running in the background while you work through the next sections of this practical.

Before we move on, let's just make sure the program is running. Use the `top` or better `htop` command to show you all processes running on your machine (type `q` to exit `top`). You should hopefully see processes associated with `canu` running (or maybe something called `meryl`). We can also check the `canu_log.txt` file where the `canu` logs will be written. If we see error messages in the file, then something has gone wrong.

[ ]:
```
htop
```

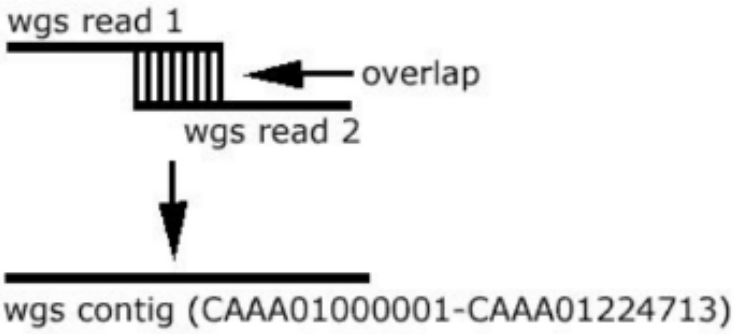
[ ]:
```
less canu_log.txt
```

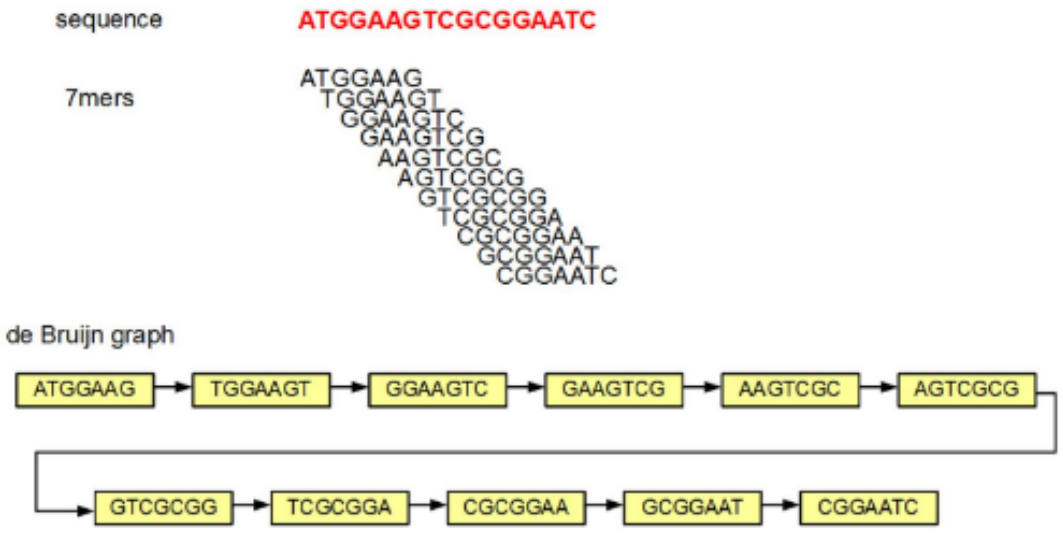
While the `canu` assembly is running, move on to the next section: Assembly algorithms

3 Assembly algorithms

There are several approaches (algorithms) that can be used to generate a set of contiguous sequences (contigs) from a set of DNA fragments (reads). Two of the main approaches are:

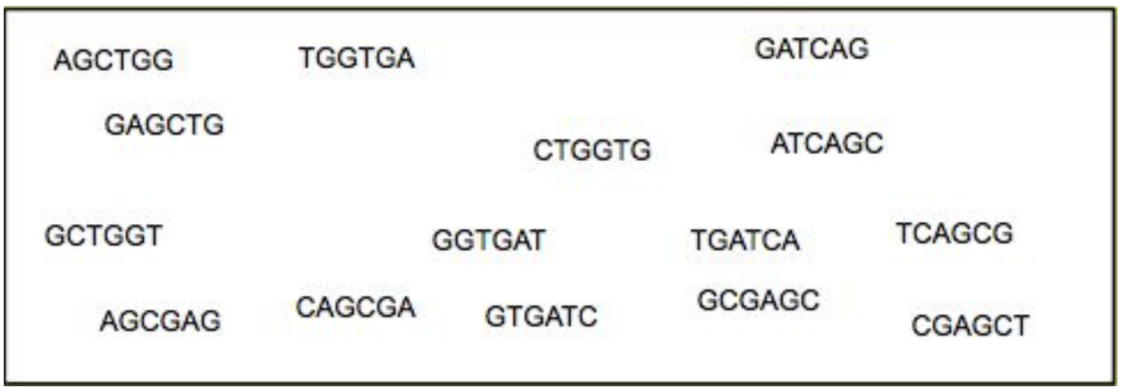
- **Overlap Layout Consensus (OLC):** This approach looks at all pairs x,y of all reads and determines if there is a sufficient overlap of the two reads. If there is, then it bundles the stretches of overlap graphs into contigs. Examples of assembly software that use this approach are Falcon (PacBio), Canu(Pacbio, ONT), mnimap/miniasm.
- **de Brujin graph (DBG):** This approach builds a graph of all subsequences of lenght k (k-mers). Examples of assembly software that use this approach are velvet, ABySS, SPAdes, wtdbg2.



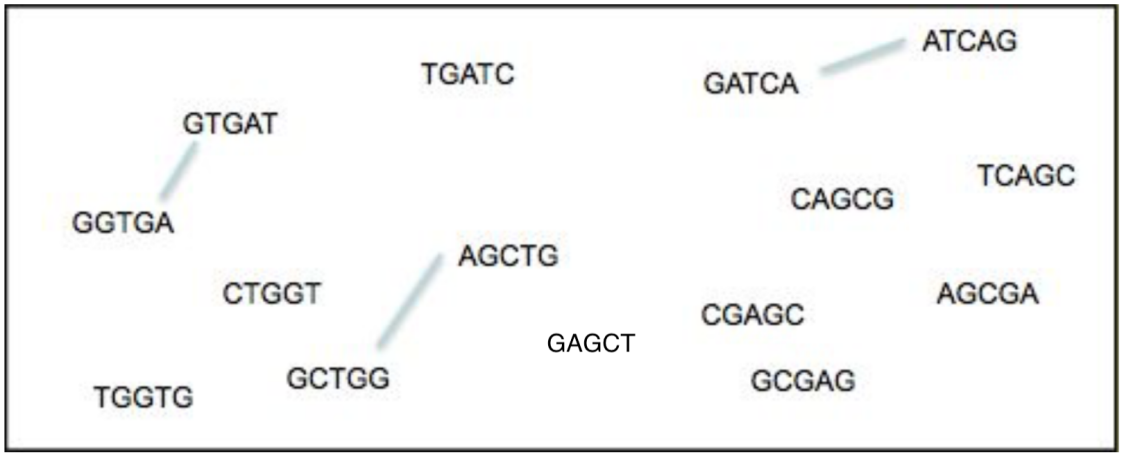


3.1 Exercise

Here we are going to build a de Brujin graph by hand from a set of short reads! Using the 6bp reads listed below, manually create the de Brujin graph and find the contig(s).



Using a k-mer value of k=5 produces the following k-mers of length 5 from the reads above. To finish the graph, join k-mers that overlap by 4 bases.



Questions

1. What is the contig sequence?
2. What was difficult here?

Now move on to the next section: Illumina genome assembly

4 Illumina Genome Assembly

We are now going to use the assembler velvet https://www.ebi.ac.uk/~zerbino/velvet/ to assemble the Illumina reads. Our Illumina reads are from the same sample we used to generate the PacBio data.

4.1 Generating an Illumina assembly with velvet

Creating a genome assembly using velvet is a two stage process:

- First, the command `velveth` is used to generate the k-mers from the input data
- Second, the command `velvetg` is used to build the de Bruijn graph and find the optimal path through the graph

To assemble the data, start with the command:

[ ]:
```
velveth k.assembly.49 49 -shortPaired -fastq IT.Chr5_1.fastq IT.Chr5_2.fastq
```

The input option `k.assembly.49` is the name of the directory where the results are to be written. The vallue 49 is the k-mer size. The other options specify the type of the input data (`-shortPaired`) and `-fastq` is used to specify the fastq files that contain the sequencing reads.

To see all possible options for velveth use:

[ ]:
```
velveth
```

Now use velvetg to build the graph and find the path through the graph (similar to what we did manually in the previous section):

[ ]:
```
velvetg k.assembly.49 -exp_cov auto -ins_length 350
```

The first parameter `k.assembly.49` specifies the working directory as created with the `velveth` command. The second `-exp_cov auto` instructs velvet to find the median read coverage automatically rather than specifying it yourself. Finally, `-ins_length` specifies the insert size of the sequencing library used. There is a lot of output printed to the screen, but the most important information is the last line:

```
Final graph has 1455 nodes and n50 of 7527, max 38045, total 1364551, using
700301/770774 reads. (Your exact result might differ depending on the velvet
version used - don't worry).
```

This gives you a quick idea of the result.

- 1455 nodes are in the final graph.
- An n50 of 7527 means that 50% of the assembly is in contigs of at least 7527 bases, it is the median contig size. This number is most commonly used as an indicator of assembly quality. The higher, the better! (but not always!)
- Max is the length of the longest contig.
- Total is the size of the assembly, here 1346kb.
- The last two numbers tell us how many reads were used from the 7.7 million pairs input data.

To see all possible options for velvetg use:

[ ]:
```
velvetg
```

Now let's try to improve the quality of the assembly by varying some of the input parameters to velvet. Two parameters that can play a role in improving the assembly are `-cov_cutoff` and `-min_contig_lgth`.

Using the `-cov_cutoff` parameter means that nodes with less than a specific k-mer count are removed from the graph.

Using the `-min_contig_lgth` parameter means that contigs with less than a specific size are removed from the assembly.

Try re-running the assembly with a kmer of 49 and using a `-cov_cutoff` of 5 and `-min_contig_lgth` of 200.

[ ]:
```
velvetg k.assembly.49 -exp_cov auto -ins_length 350 -min_contig_lgth 200 -
cov_cutoff 5
```

Note as we are not changing the k-mer size, we do not need run the `velveth` command again.

Generally, the k-mer size has the biggest impact on assembly results. Let us make a few other assemblies for different k-mer sizes i.e. 55, 41. Here is the example for k-mer length of 55.

[ ]:
```
velveth k.assembly.55 55 -shortPaired -fastq 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
```

Note: If you find that you are having trouble running the velvet assemblies or if it is running for longer than 10-15 mins then quit the command (Ctrc-C). A pre-generated set of Illumina assemblies can be found at:

[ ]:
```
ls ~/course_data/assembly/data/backup/illumina_assemblies
```

4.2 Assembly metrics

All the assembly results are written into the directory you specified with the `velvet` commands, e.g. `k.assembly.41`,`k.assembly.49`,`k.assembly.55`. The final contigs are written to a file called `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 assembly statistics is to use a program called `assembly-stats`. It displays the number of contigs, the mean size and a lot of other useful statistics about the assembly. These numbers can be used to assess the quality of your assemblies and help you pick the "best" one.

Type:

[ ]:
```
assembly-stats k.assembly*/*.fa
```

EXAMPLE ONLY - YOUR NUMBERS WILL DIFFER!

```
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
```

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

k-mer	nodes	n50	average contig	largest contig
41				
49				
55				

Question: What is the best choice for k?

We want to choose the set of parameters that produce the assembly where the n50, average contig size and the largest contigs have the highest values, while contig number is the lowest.

You will notice another statistic produced by assembly-stats is N_count, what does the N_count mean?

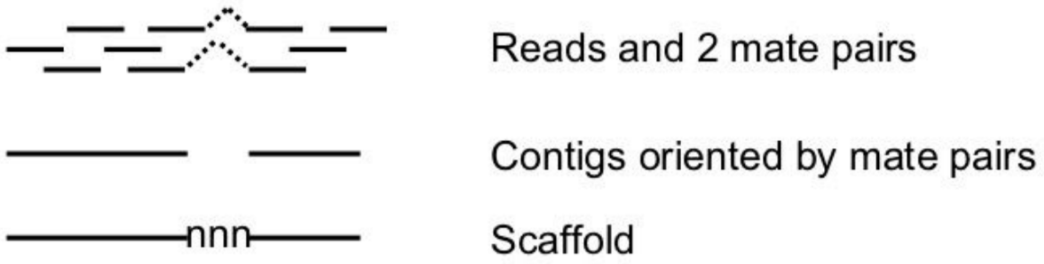
As we know, DNA templates can be sequenced from both ends, resulting in mate pairs. Their outer

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 is 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 that the repeats are all collapsed, if they are smaller than the read length. If the repeat unit is smaller than the insert size, 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 scaffolds (supercontigs), you can use `seqtk` to break the scaffold at any stretch of N's with the following commands:

[ ]:
```
seqtk cutN -n1 k.assembly.41/contigs.fa > assembly.41.contigs.fasta
```

[ ]:
```
assembly-stats assembly.41.contigs.fasta
```

[ ]:
```
seqtk cutN -n1 k.assembly.49/contigs.fa > assembly.49.contigs.fasta
```

[ ]:
```
assembly-stats assembly.49.contigs.fasta
```

[ ]:
```
seqtk cutN -n1 k.assembly.55/contigs.fa > assembly.55.contigs.fasta
```

[ ]:
```
assembly-stats assembly.55.contigs.fasta
```

Question: How does the contig N50 compare to the scaffold N50 for each of your assemblies?

k-mer	nodes	contig n50	scaffold n50
41			
49			
55			

Congratulations you have sucessfully created a genome assembly using Illumina sequence data. Now move on to the next section: Assembly estimation

5 Assembly estimation

Fortunately with this dataset, we have a known reference genome and therefore some expectations about the size and composition of the Plasmodium falciparum genome.

But what if we are working with a new genome, one which has not been sequenced before? One approach is to look at k-mer distributions from the reads. This can be done using two pieces of software, `jellyfish` (https://github.com/gmarcais/Jellyfish) and `Genomesope` (http://qb.cshl.edu/genomescope/). `Jellyfish` is used to determine the distribution of k-mers in the dataset and then `Genomescope` is used to model the single copy k-mers as heterozygotes, while double copy k-mers will be the homozygous portions of the genome. It will also estimate the haploid genome size.

Let's check with our Plasmodium falciparum Illumina data that the k-mer distribution gives us what we expect. To get a distribution of 21-mers:

```
[ ]: jellyfish count -C -m 21 -s 1G -t 2 -o IT.jf <(cat IT.Chr5_1.fastq IT.Chr5_2.
     ↪fastq)
```

This command will count canonical (`-C`) 21-mers (`-m 21`), using a hash with 1G elements (`-s 1G`) and 2 threads (`-t 2`). The output is written to `IT.jf`.

To compute the histogram of the k-mer occurences, use

```
[ ]: jellyfish histo IT.jf > IT.histo
```

Look at the output:

```
[ ]: less IT.histo
```

Now analyse the output with `genomescope`:

```
[ ]: Rscript genomescope.R IT.histo 21 76 IT.jf21
```

Where 21 is the k-mer size, 76 is the read length of the input Illumina data and IT.jf21 is the directory to write the output to. To look at the output use:

```
[ ]: less IT.jf21/summary.txt
```

```
[ ]: firefox IT.jf21/plot.png &
```

You should see an image similar to the ones shown below. Notice the bump to right of the main peak. These are the repeated sequences.

5.1 Exercises

1. What is the predicted heterozygosity?
2. What is the predicted genome size?
3. Does this seem reasonable?

We have used `jellyfish` to pre-generate a set of k-mer histograms for a handful of other species. These histo files can be found in the data directory.

[ ]:
```
ls *.histo
```

Try running genomscope on these. The read length for all of the datasets is 150bp.

fAnaTes1.jf21.histo: What is the bulge to the left of the main peak here?

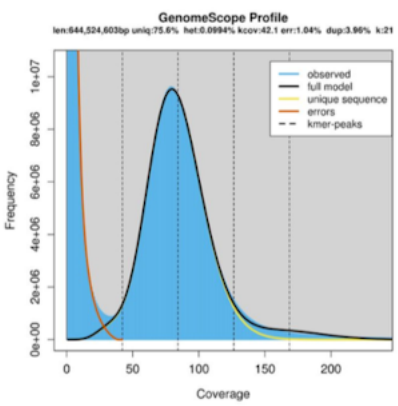
[ ]:
```
Rscript genomescope.R fAnaTes1.jf21.histo 21 150 fAnaTes1.jf21
```

fDreSAT1.jf21.histo: What is the striking feature of this genome?

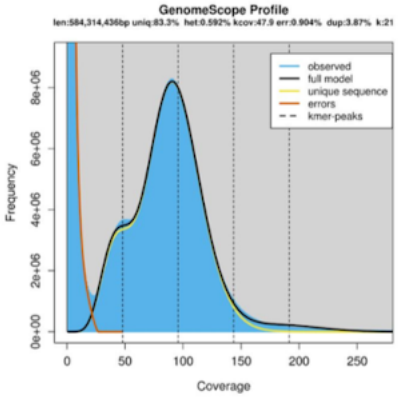
fMasArm1.jf21.histo: You should see a nice tight diploid peak for this sample. It has very low heterozygosity - similar to human data.

fSalTru1.jf21.histo: This genome was actually haploid. How do we interpret the features in the genomescope profile?

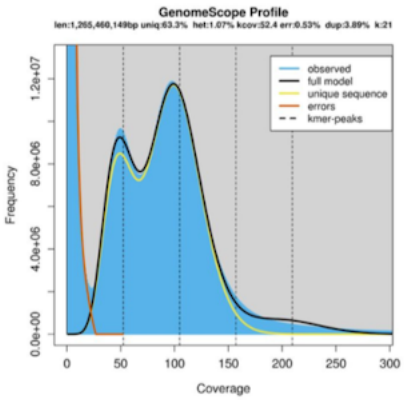
fMasArm1



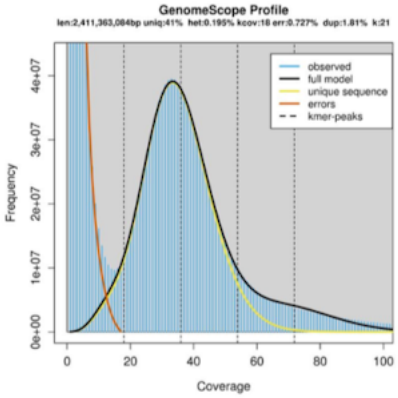
fAnaTes1



fDreSAT1



fSalTru1



Now let us go back to our PacBio assembly: PacBio assembly

6 PacBio Genome Assembly contd.

We have generated an assembly for the Plasmodium falciparum chromosome from our Illumina data. Now, let's have a look at the PacBio assembly.

6.1 Generating pacbio assemblies

The `canu` pacbio assembly should have hopefully finished by now (check the log less canu_log.txt).

If it has not finished, you can find a pre-generated canu assembly at:

```
[ ]: ls ~/course_data/assembly/data/backup/pacbio_assemblies
```

Now use the `assembly-stats` script to look at the assembly statistics.

```
[ ]: assembly-stats canu-assembly/PB.contigs.fasta
```

How does it compare to the Illumina assembly?

Another long read assembler based on de Bruijn graphs is `wtdbg2` https://github.com/ruanjue/wtdbg2. Let's try to build a second assembly using this assembler and compare it to the assembly produced with `canu`.

```
[ ]: wtdbg2 -t2 -i PBReads.fastq.gz -o wtdbg
```

```
[ ]: wtpoa-cns -t2 -i wtdbg.ctg.lay.gz -fo wtdbg.ctg.lay.fasta
```

```
[ ]: assembly-stats wtdbg.ctg.lay.fasta
```

6.2 Comparing pacbio assemblies

How does the wtdbg2 assembly compare to the canu assembly? Both assemblies may be similar in contig number and N50, but are they really similar? Let's map the Illumina reads to each assembly, call variants and compare.

```
[ ]: bwa index canu-assembly/PB.contigs.fasta
```

```
[ ]: samtools faidx canu-assembly/PB.contigs.fasta
```

```
[ ]: bwa mem -t2 canu-assembly/PB.contigs.fasta IT.Chr5_1.fastq IT.Chr5_2.fastq |␣
↪samtools sort -@2 - | samtools mpileup -f canu-assembly/PB.contigs.fasta -ug␣
↪- | bcftools call -mv > canu.vcf
```

Do the same for `wtdbg.ctg.lay.fasta` and then compare some basic statistics.

```
[ ]: bwa index wtdbg.ctg.lay.fasta
```

```
[ ]: samtools faidx wtdbg.ctg.lay.fasta
```

[ ]:
```
bwa mem -t2 wtdbg.ctg.lay.fasta IT.Chr5_1.fastq IT.Chr5_2.fastq | samtools sort␣
↪-@2 - | samtools mpileup -f wtdbg.ctg.lay.fasta -ug - | bcftools call -mv >␣
↪wtdbg.vcf
```

[ ]:
```
bcftools stats canu.vcf | grep ^SN
```

[ ]:
```
bcftools stats wtdbg.vcf | grep ^SN
```

Question: What do you notice in terms of the number of SNP and indel calls?

The `wtdbg2` assembly has more variants due to having more errors. This is mainly due to a lack of polishing or error correction - something that the `canu` assembler performs, but the `wtdbg2` assembler does not.

6.3 Polishing pacbio assembly

Correcting errors is an important step in making an assembly, especially from noisy long ready data. Not polishing an assembly can lead to genes not being identified due to insertion and deletion errors in the assembly sequence. To polish a genome assembly with Illumina data we use `bcftools consensus` to change homozygous differences between the assembly and the Illumina data to match the Illumina data

Run the following steps to polish the `canu` assembly

[ ]:
```
bgzip -c canu.vcf > canu.vcf.gz
```

[ ]:
```
tabix canu.vcf.gz
```

[ ]:
```
bcftools consensus -i'QUAL>1 && (GT="AA" || GT="Aa")' -Hla -f canu-assembly/PB.
↪contigs.fasta canu.vcf.gz > canu-assembly/PB.contigs.polished.fasta
```

[ ]:
```
Run the following steps to polish the `wtdbg2` assembly
```

[ ]:
```
bgzip -c wtdbg.vcf > wtdbg.vcf.gz
```

[ ]:
```
tabix wtdbg.vcf.gz
```

[ ]:
```
bcftools consensus -i'QUAL>1 && (GT="AA" || GT="Aa")' -Hla -f wtdbg.ctg.lay.
↪fasta wtdbg.vcf.gz > wtdbg.contigs.polished.fasta
```

Finally, align and call variants like before (bwa index/bwa mem/samtools-sort/mpileup/bcftools call) using the polished assemblies as the reference this time.

When running this analysis on these polished genomes, do we still get variants? More or less than with the raw `canu` and `wtdbg2` assemblies? Why?

Congratulations, you have reached the end of the Genome Assembly tutorial.