NORWEGIAN SEQUENCING CENTRE

DAY 2 - Thursday October 27th

09:00-09:15 Practical Demonstration 5: hybrid *de novo* assembly (Lex)

- Multiple input data for Newbler

09:15-10:45 Practical 6: *de novo* assembly of Illumina reads - Part 1

- optimizing k-mer lengths
- optimizing other velvet parameters
- adding paired end and mate-pair information

10:45-11:00 Coffee/tea break

11:00-12:00 Practical 6: *de novo* assembly of Illumina reads - Part 2

12:00-13:00 Lunch break (cantina of the new informatics building across the street)

13:00-14:00 Practical 5 and 6: evaluation of the results

14:00-15:00 Practical 7: What can our assemblies tell us about *E. coli* O104? (Nick)

- MAUVE to visually check for mis-assembly
- align to reference genome, find difference(s)

15:00-15:45 Discussion: what have we learnt

- QC and filtering
- 454 only assembly
- Illumina only assembly
- hybrid assembly
- how to determine which assembly is ‘best’

15:45-16:00 Coffee/tea break

16:00-16:30 Lecture 3: what’s next (Lex)

- other popular assembly programs
- resources for annotation
- challenges for *de novo* assembly of large (eukaryotic) genomes

16:30-17:00 Discussion: rounding up

- short evaluation of the course
- Q&A

Practical 4: *de novo* assembly of Illumina reads - Part 1

Learning points

- understand k-mers and how to estimate genome size from short-read data
- understand how to estimate read error, sequence diversity, repeats from k-mers
- understand how to run Velvet
- understand the concept of k-mer coverage
- understand the effect of k-mer settings on assemblies
- understand how to use paired-end information in Velvet
- understand the significance of setting expected coverage values

Assembling short-reads with Velvet

We will now use Velvet to assemble Illumina reads on their own. Ensure you have set up your PATH like yesterday.

To initialise your environment:

```
module load velvet
module load python
```

We will assemble the *E. coli* O104:H4 strain 280 which was sequenced on an Illumina MiSeq. The sequencing centre read 150 bases from each direction. We know from the sequencing centre that the fragment size they chose was approximately 450bp.

You should have two files from yesterday's filtering practical, named `Sample280_1_filtered.fastq` and `Sample280_2_filtered.fastq`. We will use a simple Python script to interleave the two files, ensuring that the proper pairing is preserved.

Although we can identify pairs of reads by looking at their FASTQ headers (reads end in either `/1` or `/2`), Velvet does not understand this system. It requires paired-end data to be *interleaved*, e.g. read 2 should always *follow* read 1.

"Orphan" reads which have no pair will be sent to a separate file which we will not use in this practical.

```
pair_up_reads.py Sample280_1_filtered.fastq \
     Sample280_2_filtered.fastq \
     Sample280_1_good.fastq \
     Sample280_2_good.fastq \
     Sample280_orphans.fastq
```

Examine this file (Compare with `/data/illumina/Sample280_1.fastq` and `/data/illumina/Sample280_2.fastq`) and check you understand the format.

Are there the same number of sequences in `Sample280_1_filtered.fastq` as `Sample280_2_filtered.fastq`? Why? (Hint: `wc -l filename` will tell you the number of lines in a file)	
Are there the same number of sequences in `Sample280_1_good.fastq` and `Sample280_2_good.fastq`? Why?	
How many orphan sequences were produced? (hint, `wc -l` **`filename`** will tell you the number of lines in a file)	

Now interleave the files:

```
shuffleSequences_fastq.pl \
    Sample280_1_good.fastq Sample280_2_good.fastq \
    Sample280_interleaved.fastq
```

If you have problems with these steps, we have provided a Velvet-compatible file for you to use here:

```
/data/illumina/Sample280_interleaved.fastq
```

Building the Velvet Index File

Velvet requires an index file to be built before the assembly takes place. Just like in the previous practical we must choose a *k*-mer value. Longer *k*-mers result in a more stringent assembly, at the expense of coverage. There is no definitive value of *k* for any given project. However, there are several absolute rules - *k* must be less than the read length and it should be an odd number.

Firstly we are going to run Velvet in single-end mode, ignoring the pairing information. Later on we will incorporate this information.

Pick a value of *k* between 21 and 99 to start with (mark on blackboard) and run `velveth` to build the hash index.

```
velveth my_assembly_directory value_of_k -short -fastq \
     Sample280_interleaved.fastq
```

After it has finished, look in **`my_assembly_directory`**. You should see the following files:

```
Log
Roadmaps
Sequences
```

`Log` is a useful file, this is a useful reminder of what commands you typed to get this assembly result, useful for reproducing results later on. Sequences contains the sequences we put in, and `Roadmaps` contains the index you just created.

Now we will run the assembly with default parameters:

```
velvetg my_assembly_directory
```

Look again at my_assembly_directory, you should see the following extra files;

```
contigs.fa
Graph
LastGraph
PreGraph
stats.txt
```

The important files are:

`contigs.fa` - the assembly itself
`Graph` - a textual representation of the contig graph
`stats.txt` - a file containing statistics on each contig

What *k*-mer did you use?	
What is the N50 of your assembly?	
What is the size of the largest contig?	
What is the total assembly size?	

Now try running Velvet again using the best *k*-mer value as determined by the group. You can try a few more values now you know where the sweet spot is.

Record the N50 and the total assembly size for each assembly in the box:

Assembly ID	K	N50	Total assembly size

Advanced tip: You can also use VelvetOptimiser to automate this process of selecting appropriate *k*-mer values.

Group task: Record on the board the best value of *k* you found with the N50.

Practical 5: hybrid *de novo* assembly

Learning points

- How to add miseq reads to a 454 read dataset using newbler
- How to judge the effect

Starting the Assembly

Since this assembly will take around three hours, we will start this before we continue with the velvet assemblies.

We will redo the very first assembly we did for this course, assembly using newbler and the 454 GS FLX+ shotgun reads, together with 8kb Paired End (better called Mate Pair) reads. However, this time we will add the filtered MiSeq reads as well. Therefore, these files will be used for this assembly:

```
/data/454/454_gsflx+_all_reads.sff
/data/454/454_titanium_pe_reads.sff
/data/illumina/Sample280_1_filtered.fastq
/data/illumina/Sample280_2_filtered.fastq
```

Luckily for us, from version 2.6 newbler automatically recognizes fastq files, the quality scoring version and the read pairing. So, we can basically just add the miseq files to the commandline we used before. With one cpu, however, this assembly is going to take around 3 hours. So, let's get started!

Run this assembly as (again, all on one line):

```
runAssembly -o projectname /data/454/454_gsflx+_all_reads.sff
/data/454/454_titanium_pe_reads.sff
/data/illumina/Sample280_1_filtered.fastq
/data/illumina/Sample280_2_filtered.fastq
```

TIP: If you have a long command, remember you can split it over multiple lines by adding a space followed by a backslash '\', as such:

```
runAssembly -o projectname \
/data/454/454_gsflx+_all_reads.sff  \
/data/454/454_titanium_pe_reads.sff  \
/data/illumina/Sample280_1_filtered.fastq \
/data/illumina/Sample280_2_filtered.fastq
```

Do not put a backslash at the end of the last part.

Practical 6: *de novo* assembly of Illumina reads Part 2

Estimating and setting exp_cov

Much better assemblies are produced if Velvet understands the expected coverage for unique regions of your genome. This allows it to try and resolve repeats.

This is done by setting exp_cov.

The command velvet-estimate-exp-cov.pl is supplied with Velvet and will plot a histogram of k-mer frequency for each node in the graph.

```
velvet-estimate-exp_cov.pl my_assembly_directory/stats.txt
```

The peak value in this histogram can be used as a guide to the best value for *exp_cov*.

What do you think is the approximate k-mer coverage for your assembly?	
Convert this value from k-mer coverage into genome coverage. The formula is: $\frac{Ck * L}{(L - k + 1)} = C$ Where Ck = k-mer coverage, L = read length, k = k-mer size for your assembly	

Now run velvet again, supplying the value for exp_cov corresponding to your answer:

```
velvetg -exp_cov mean_k_mer_coverage
```

What improvements do you see in the assembly by setting a value for exp_cov?	

Setting cov_cutoff

You can also clean up the graph by removing low-frequency nodes from the *de Bruijn* graph using the cov_cutoff parameter. This will often result in better assemblies, but setting the cut-off too high will also result in losing bases. Using the histogram from previously, estimate a good value for cov_cutoff.

```
velvetg my_assembly_directory -exp_cov your_value \
     -cov_cutoff your_value
```

Try some different values for cov_cutoff, keeping exp_cov the same and record your assembly results:

Exp_cov	Cov_cutoff	N50	Total assembly size

Now ask Velvet to predict the values for you:

```
velvetg -exp_cov auto -cov_cutoff auto
```

What values of exp_cov and cov_cutoff did Velvet chose?	
Is this assembly better than your one?	

Incorporating paired-end information

Paired end information contributes additional information to the assembly, allowing contigs to be scaffolded. Velvet needs to be told to use paired-end information as follows:

Now, re-index your reads telling Velvet to use paired-end information and re-run Velvet using the best value of *k, exp_cov and cov_cutoff* from the previous step.

```
velveth my_assembly_directory value_of_k -shortPaired -fastq \
     /data/illumina/Sample280_interleaved_filtered.fastq

velvetg my_assembly_directory -exp_cov value_of_exp_cov \
     -cov_cutoff value_of_cov_cutoff
```

How does doing this affect the assembly?

The contigs are actually scaffolds. Use the assemblathon scripts to generate metrics for this assembly.

Looking for repeats

Have a look for contigs which are long and have a much higher coverage than the average for your genome:

E.g.

```
awk '($2>=1000 && $6>=60)' stats.txt
```

Will find contigs larger than 1kb with k-mer coverage greater than 60.

Find the contig with the highest coverage. BLAST it against NT using NCBI. What is it?

Generate assembly output

When you have produced the best assembly you can, you can re-run Velvet and produce output files for viewing:

```
velvetg my_assembly_directory options_you_used \
     -read_trkg yes -amos_file yes
```

This will produce an AMOS-compatible .afg file which can be read in Tablet.

Download the file and view in Tablet. How does it compare to 454 assemblies?

The effect of mate-pair libraries

Long-range “mate-pair” libraries can also dramatically improve an assembly by scaffolding contigs. Typical sizes for Illumina are 2kb and 6kb, although any size is theoretically possible. You can supply a second library to Velvet. However, it is important that files are reverse-complemented first as Velvet expects a specific orientation.

We have supplied a 6kb mate-pair library in the correct orientation. However, these are only 50bp reads and therefore you must choose a *k*-mer value lower than this (the alternative is to use a dedicated scaffolder such as SSPACE2).

```
velveth my_assembly_directory value_of_k \
      -shortPaired
      -fastq /data/illumina/Sample280_interleaved.fastq \
      -shortPaired2
      -fastq /data/illumina/TY2482_6kb_RC_interleaved.fastq

velvetg my_assembly_directory -cov_cutoff auto \
      -exp_cov auto
```

We use auto values here because the addition of new reads will change the genome coverage.

What is the N50 of this assembly?	
How many scaffolds?	
How many bases are in gaps?	

Make a copy of the contigs file and call it '`mate_pair_contigs.fa`'

Mate-pair data can contain significant paired-end contamination which generates misassemblies. To account for this, let Velvet know about the problem with the following flag:

```
velvetg my_assembly_directory -cov_cutoff auto \
      -exp_cov auto -shortMatePaired2 yes
```

What is the N50 of this assembly?	
How many scaffolds?	
How many bases are in gaps?	

Make a copy of the contigs file and call it '`mate_pair_no_contamination.fa`'

Advanced exercise

Calculate genome size using k-mer counting

We can count the number of unique k-mers in a sample to give us an estimate of genome size. For this we will use the program Jellyfish:

Program	**Options**	**Explanation**
jellyfish count		Count unique and total k-mers from a FASTA or FASTQ file
	-m 22	Use a *k*-mer size of 22
t	-o	The output file prefix
	-s 10000000	The hash table size (influences memory usage)
	-c	Length of counting field in bits (don't need to change)
jellyfish stats		Provide statistics on total and distinct k-mers
	-L	Lower frequency bound to count from
	-U	Upper frequency bound to count from
jellyfish histo		Produce a histogram from k-mer counting data

Perform the following:

```
jellyfish count -m 22 -o your_prefix -c 3 \
     -s 10000000 \
     /data/illumina/Sample280_interleaved.fastq
```

This will produce 2 or more files starting with your_prefix and then an integer. Make sure you supply all the generated files to the next step:

```
jellyfish merge your_prefix_0 your_prefix_1
     your_prefix_2 your_prefix_3 \
     -o your_database.jf
```

This will merge together your files.

Now we can generate some k-mer statistics from the database.

```
jellyfish stats your_database.jf
```

How many unique (singleton) k-mers are there? How many distinct (different) k-mers are there? What is the total number of k-mers in the database?

To determine genome size accurately, we must determine which k-mers are real and which represent sequencing error or variation. This is best done by plotting the k-mer frequency. We can do this easily through jellyfish:

```
jellyfish histo your_database.jf > histogram.txt
```

There are two ways of using k-mers to estimate genome size.

Ignoring the singletons, where is the modal coverage? Where do you think the boundary of erroneous k-mers is? Write down the numbers and consider plotting the histogram (e.g. in Excel).

What is the modal coverage (excluding singletons?)	
Where is the boundary between erroneous k-mers?	
Where is the upper bound of non-unique parts of the genome?	

Estimating genome size

Now we can estimate the genome size. Re-run the count, setting a lower bound as follows:

```
jellyfish stats -L lower_bound -U upper_bound your_database.jf
```

Two simple methods for genome size estimation

The first is simple; you simply divide the number of distinct (different) k-mers in your dataset by 2.

The second way is to divide the total number of good k-mers by 2 and then by the mean genome coverage. As an approximation you can use the modal coverage for the non-unique region of the genome.

Practical 5 and 6: evaluation of the results

Hybrid assembly evaluation

Run the assemblathon_metrics script on this assembly. Compare the hybrid assembly with the individual newbler and velvet assemblies.

Question	**Answer**
Did newbler filter away much reads or bases from the MiSeq data?	
What was the coverage of the hybrid assembly (trimmed bases)	
What did newbler determine to be the insert size ('average pair distance') of the MiSeq reads?	
In the hybrid assembly, how many pieces make up the main chromosome? Interpretation?	
Did the hybrid assembly have more or fewer contigs?	
Which assembly produced the best contig N50? Contig NG50?	
Which assembly had the most gaps bases in scaffolds?	
Which of the two newbler assemblies had the highest percent Q40plus bases?	

Generate assembly statistics for all assemblies in Practical 3

Use the assemblathon scripts to generate statistics for all the assemblies you generated so far:

```
assemblathon_stats.pl \
      my_assembly/454Scaffolds.fna
```

Tabulate the results in the spreadsheet located here:

http://bit.ly/uGYEOk

Which assembly gives the best N50? How might the total assembly size affect the N50 value?

Advanced exercise

Calculating A50 plots

A50 plots are visual guides to assembly quality and can be more informative than the N50 value. We have provided you a script which calculates the A50 values using R.

```
python /data/bin/a50.py my_assembly_1.fasta my_assembly_2.fasta \
      > lengths.txt
/data/bin/A50.R lengths.txt out.png "my title"
```

NORWEGIAN SEQUENCING CENTRE

Practical 7: What can our assemblies tell us about *E. coli* O104?

Learning points:

- Using Mauve to re-order and orient contigs according to a reference genome
- Using Mauve to visually inspect assemblies for issues
- The effect of different assemblies on our biological understanding of this strain

Re-order velvet contigs against a reference

We have provided a reference, annotated assembly in GenBank format in /data/refs/280.gbk. Download this to your hard-drive.

You have hopefully installed Mauve already but otherwise it can be downloaded from http://gel.ahabs.wisc.edu/mauve.

Choose “Align with ProgressiveMauve” from the menu and add the 280.gbk file as the first sequence and then mate_pair_contigs.fa and mate_pair_no_contamination.fa as second and third files. Enter a descriptive name for the output file, e.g. 280reference_vs_illumina

Wait for the alignment to complete and then examine the viewer.

What is the largest contiguous block of alignment?	
Which file contains more misassemblies? Why?	
Are there regions missing from the genome? What gene features are in those places?	

Using the Mauve Contig Mover

You can re-order the contigs for ease of viewing by using the Mauve Contig Mover. Choose Tools > Contig Mover. For the first genome chose again ref.gbk and for the second chose the Illumina assembly (this process has to be done two at a time).

How could this view be improved further?	

How well do assemblies reconstruct the biology of the strain?

We have prepared an alignment bundle for you which we will look at together during the practical, pointing out interesting biological features. Download the file from `/data/hybrid/alignments.zip`. Unzip the file. Then load the file “all_alignments” into Mauve.

NORWEGIAN SEQUENCING CENTRE

Sources

Website for the course
with all presentations and handouts, etc

https://github.com/lexnederbragt/denovo-assembly-tutorial

To download the entire repository in one go, go here:

https://github.com/lexnederbragt/denovo-assembly-tutorial/zipball/master

Sequence data

Much of the sequence data used in this practical is available to download.

We are grateful to the Health Protection Agency for the 454 & Illumina MiSeq data - http://www.hpa-bioinformatics.org.uk/lgp/genomes

We are grateful to the BGI for the Illumina 6kb mate-pair data.

We will try to release all of the data for this course, please refer to https://github.com/lexnederbragt/denovo-assembly-tutorial for the links.

Software

- fastqc: http://www.bioinformatics.bbsrc.ac.uk/projects/fastqc/
- fastx toolkit: http://hannonlab.cshl.edu/fastx_toolkit/
- newbler: you can ask a copy through a form at the 454 website (http://my454.com/contact-us/software-request.asp)
- velvet: http://www.ebi.ac.uk/~zerbino/velvet/
- jellyfish: http://www.cbcb.umd.edu/software/jellyfish/
- Tablet assembly viewer http://bioinf.scri.ac.uk/tablet/
- MAUVE multiple genome alignment http://asap.ahabs.wisc.edu/software/mauve/

Scripts

Scripts especially written or modified for this course are available on the github repository.

- shuffleSequences_fastq.pl is part of velvet
- interleave_pairs.py by Peter Cock
- pair_up_reads.py by Peter Cock
- A50.py/A50.R by Nick Loman
- assemblathon_stats.pl modified after http://korflab.ucdavis.edu/Datasets/Assemblathon/Assemblathon1/assemblathon_stats.pl. NOTE 1: it requires a perl module called FALite.pm, available here http://homepage.mac.com/iankorf/FAlite.pm. NOTE 2: an updated version of this script is available from assemblathon.org
- fastq_trim_trailing_Bs.pl written by one of us

Further Reading

Li, Waterman Estimating the Repeat Structure and Length of DNA Sequences Using ℓ-Tuples. DOI: 10.1101/gr.1251803 Genome Res. 2003. 13:1916-1922

A guide to using Jellyfish for estimating genome size: https://banana-slug.soe.ucsc.edu/bioinformatic_tools:jellyfish

Velvet manual http://bioweb2.pasteur.fr/docs/velvet/Manual.pdf