Quality control and assembly

For the exercises of today and tomorrow, there is a sample data set stored on the server. This dataset consists of 12 fastq files that have been truncated to 200 000 reads (50 base-pair, single-end reads from an Ilumina HiSeq machine). They are from a red abalone, a marine snail that suffered a severe mass mortality event in 2011. Six of the samples are from before the event, and 6 are from after. We will see if these samples can help us learn anything about the effects of the mortality event during this class.

As a reminder from this morning – base quality in fastq files are denote by the phred scale Quality score [Q_{phred} =-10 log10(p)], which ranges in practise from 0 to 40. Quality scores are coded in the fastq files as ASCII characters, where ASCII character 33 (which is an !) is 0. The quality of any given base call is given by the sequencer and it is a measure of how good the color-space signal was at that site. Usually the quality decreases toward the end of the reads due to the sequencing chemistry.
Before we can use our data to answer any biological questions, we must remove poorly called bases as well as any adapter sequences from our reads (the sample prep procedure includes ligating a special set of sequencing adapters). The Illumina adapters all have this sequence:
GATCGGAAGAGCACACGTCTGAACTCCAGTCACNNNNNNATCTCGTATGCCGTCTTCTGCTTG,
where the 6 Ns represent a sample-specific barcode that allows for sequencing several samples in the same sequencing reaction. The barcode for each sample is given in the file name.

To evaluate the data set, it is useful to know what the distribution of quality scores and nucleotides looks like. As the FASTQ files are too large to overview manually, we have to summarize the data and graph it using command-line based software.

It is also useful to know the fraction of duplicate reads (identical reads present more than once in the dataset) and singletons (reads only present once in the dataset), for which we use command-line tools. There is still debate over whether duplicate reads represent very common transcripts or if they are due to primer or PCR bias, but a large fraction of duplicate reads may be indicative of a poor cDNA library. It seems quite typical to have 30-50 % duplicate reads.

In this exercise, we will use bash scripts to process multiple files at once; before executing them we will need to open them and make sure that input and output filenames as well as adapter sequences are correct. The objectives are to 1) remove all bases with a Phred quality score of less than 20, 2) remove any adapter sequences present in the data, 3) graph the distributions of quality scores and nucleotides, and 4) calculate the fractions of duplicate and singleton reads in the data.

1) Make sure that the class files are located in your personal folder, under "data".

```
cd ~/course/data
ls
```

1) Quality trimming and adapter clipping, using the bash script `TrimClip.sh`

   a. Open the bash script in the text editor "nano" with

   ```
   nano ../scripts/TrimClip.sh
   ```

   The bash script first invokes the quality trimmer, which scans through all reads, and when it encounters a base with a quality score of less than 20, trims off the rest of the read and then subsequently removes reads shorter than 20 bases. A temporary file is created, which is then used as an input file for the adapter clipper. The clipper removes any read ends that match the defined adapter sequences, and then removes reads that after clipping are shorter than 20 bases.

   To close the text editor, just hit Control-X

   b. For each step in the bash script, make sure that all the input and output file names, as well as the adapter sequences, match the data.

   c. Execute the bash script by typing:

   ```
   sh ../scripts/TrimClip.sh
   ```

   while in the folder containing your data. Make note of how many reads are being trimmed and clipped through the screen output.

3) Calculate the fraction of duplicate and singleton reads, using the bash script `CollapseDuplicateCount.sh`.

   a. Open the bash script in nano with

   ```
   nano ../scripts/CollapseDuplicateCount.sh
   ```

   The bash script first uses `fastx_collapser` to combine and count all identical reads. A temporary FASTA-formatted file called YOURFILE_collapsed.txt is created, which is then used as an input file for a python script (`fastqduplicatecounter.py`) that calculates the fractions of duplicate reads and singletons. This file is removed at the end of the program since it was just an intermediate step.

   To close the text editor, just hit Control-X

   b. For each step in the bash script, make sure that all the input and output file names in the script match your data

   c. Execute the bash script by typing:

   ```
   sh ../scripts/CollapseDuplicateCount.sh
   ```

   while in the folder containing your data.

d. Open the files named YOURFILE.duplicatecount.txt and note what percentage of your reads is duplicates, how many different sequences you have (unique reads) and how many singletons there are. Depending on the quality of your initial tissue and sample preparation procedure, the fraction of duplicate reads can be as low as 5-10% or higher than 50 %.

```
To open the file, you can use head (see the
first ten lines), tail (see the last ten
lines) or less (to scroll through).  If you
use less, press "q" to get out of the
scrollable format

Example:

less FR82_GGCTAC_after.duplicateCount.txt
```

4) Summarize quality score and nucleotide distribution data, and plot.

a. Open the bash script in a text editor with

```
nano ../scripts/QualityStats.sh
```

```
To close
the text
editor,
just hit
Control-X
```

The bash script uses `fastq_quality_stats` to summarize the data in the FASTQ file by read position (1-50 or 101) into a file named YOURFILE_qualstats.txt

b. Make sure that all the input and output file names in the script match your data.

c. Execute the bash script by typing:

```
     sh ../scripts/QualityStats.sh

in order to summarize your data files. Then execute the
plotting software by typing:

     sh ../scripts/Boxplots.sh

This will create an output by running two programs:

     fastq_quality_boxplot_graph.sh [don't copy this]

     and

     fastx_nucleotide_nucleotide_distribution_graph.sh
     [don't copy this]
```

The script also combines all the plots and graphs into a file called "Boxplots.pdf"

The easiest way to view the plots is by copying this file to your local drive and opening it there. It is also possible to view them on the server using an x-windows application if your computer is setup to do so. The plots should look something like Figs. 1a-b. If the mean quality scores are low throughout or if the nucleotides are non-randomly distributed, something could have gone wrong during sample preparation or sequencing.

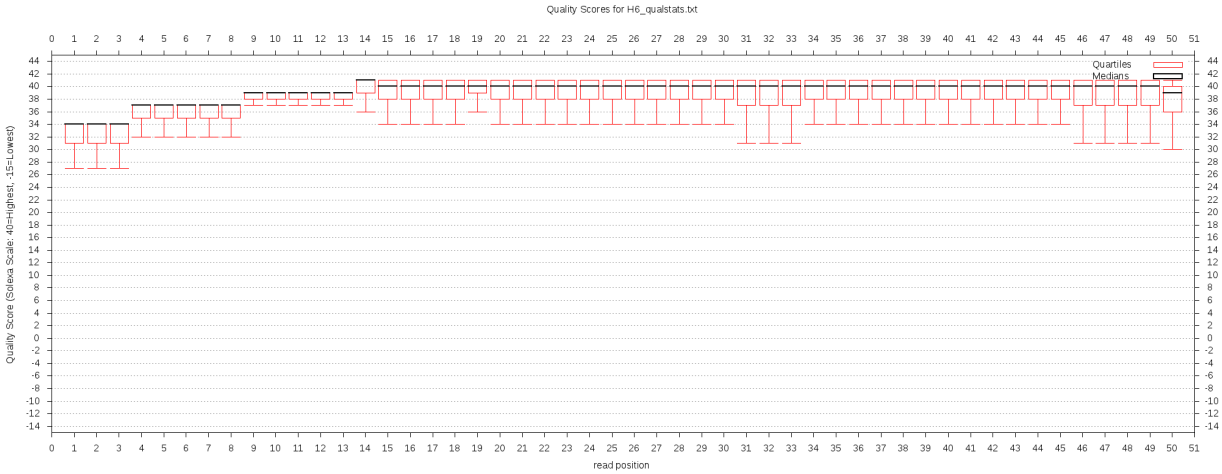

Figure 1a. Quality score boxplot of 50-bp Illumina reads (after quality trimming, Q>20), summarized by read position. Lower scores in the beginning of the reads is an artifact of the software used to calculate base quality scores.

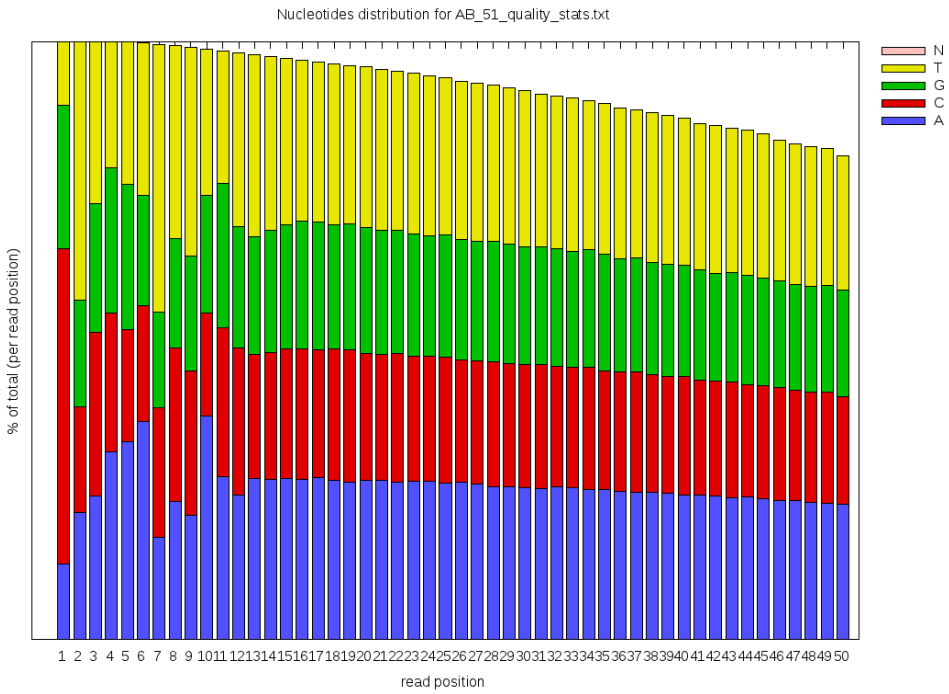

Figure 1b. Nucleotide distribution chart of 50-bp Illumina reads, summarized by read position. A non-random distribution in the first 12 bases is common, and is thought to be an artifact of the random hexamer priming during sample preparation.

De novo assembly

RNA-Seq reads represent short pieces of all the mRNA present in the tissue at the time of sampling. In order to be useful, the reads need to be combined – assembled- into larger fragments, each representing an mRNA transcript. These combined sequences are called "contigs", which is short for "contiguous sequences". A *de novo* assembly joins reads that overlap into contigs without any template (i.e. no reference genome/transcriptome).

When comparing the lengths and numbers of contigs acquired from *de novo* assemblies to the predicted number of transcripts from genome projects, the *de novo* contigs typically are shorter and more numerous. This is because the assembler cannot join contigs together unless there is enough overlap and coverage in the reads, so that several different contigs will match one mRNA transcript. Biologically, alternative splicing of transcripts also inflates the number of contigs when compared to predictive data from genome projects. This is important to keep in mind, especially when analyzing gene expression data based on mapping to a *de novo* assembly. To minimize this issue, we want to use as many reads as possible in the assembly to maximize the coverage level. We'll therefore pool the reads from all our samples, which means that no information about the individual samples can be extracted from the assembly. In order to get sample-specific information, we need to map our reads from each sample individually to the assembly once it has been created (which we'll do later today).

Building a *de novo* assembly is a very memory-intensive process. There are many programs for this, some of which are listed below. We will use Trinity in this exercise, an assembler that is thought to work very well for transcriptomes, as opposed to others that are optimized for genome assembly. Trinity uses *De Bruijn graphs* to join reads together. De Bruijn graphs summarize sequence variation in a very cost-effective way, speeding up the assembly process. Nevertheless, it is a very memory-intensive step, and having access to a computer cluster might be necessary if the number of reads is high. However, as we only have 1.2 million reads in our dataset, we should be able to produce an assembly with only 1GB RAM memory in about 30 minutes.

Figure 2a. An example De Bruijn graph with k-mer size 16 and 5 nodes.

Figure 2b. A bubble caused by 2 SNPs or sequencing errors.

There are several parameters one can vary when assembling a transcriptome or genome. The perhaps most important one is the k-mer (word) length of the De Bruijn graphs. Longer k-mers can help resolve repeat regions in genome assemblies and can be useful to resolve homeolog genes in polyploid species, whereas shorter one can increase performance in polymorphic sequences. As Trinity focuses on transcriptome assembly, the k-mer length is preset to 25. In other assemblers, it can vary considerably.

We will now create our own *de novo* assembly from our short reads. The objectives are to 1) build a *de novo* assembly, using default parameters, and 2) examine the properties of the newly-created assembly.

Exercise

1) Concatenate the sample files into one.
   a. Move to the folder where the trimmed and clipped fastq files are located, if you are not there already ("data").
   b. Concatenate the files using the *cat* command:

```
cat *.trimmed.clipped.fastq > assembly_ready.fastq
```

2) Run Trinity to create a *de novo* assembly:
   a. Start the assembly software by typing:

```
Trinity.pl --seqType fq --max_memory 1G --CPU 4 --single
assembly_ready.fastq --output TrinityTest
```

This specifies to Trinity that we are dealing with single-end reads in a fastq file, to use 1 GB of RAM, 4 processors and to store output files in a folder called "TrinityTest" (The assembly file will by default be named "Trinity.fasta"). The command will produce a lot of output that you can monitor to follow the progress of the assembler. It will take a little while for it to finish, so let's go have a fika while the computers are working!

There is a possibility that this will take too long to finish despite the truncated data files, as transcriptome assembly is a quite memory- and CPU demanding process. If this is the case, there is a pre-made assembly located in a directory called "assembly" in your data directory. The assembly is called "Trinity.fasta" - you can use this file for downstream exercises.

3) Examine the statistics of the assembly, using the count_fasta.pl script, by typing:

```
../scripts/count_fasta.pl ./TrinityTest/Trinity.fasta
> ./TrinityTest/trinityStats.txt
```

This will output some statistics of the assembly to a file called trinityStats.txt which can be found inside the "TrinityTest" folder

```
nano ./TrinityTest/trinityStats.txt
```

Open this file using nano and look at the length distribution on the contigs. Then copy the data into an Excel spread sheet and plot the distribution to get an overview.

A few other assemblers

CLC genomics workbench: http://www.clcbio.com/index.php?id=1240
ABySS: http://www.bcgsc.ca/platform/bioinfo/software/abyss
Velvet: http://www.ebi.ac.uk/~zerbino/velvet/
Oases: http://www.ebi.ac.uk/~zerbino/oases/

Here's also an excellent review describing and contrasting the different software packages in use:

Zhang W, Chen J, Yang Y, Tang Y, Shang J, et al. 2011. A practical comparison of *de novo* genome assembly software tools for next-generation sequencing technologies. *PLoS ONE* 6: e17915. doi:10.1371/journal.pone.0017915