TRAINER'S MANUAL

Introduction to Next Generation Sequencing Hands-on Workshop

Bioplatforms Australia (BPA)
The Commonwealth Scientific and Industrial Research Organisation (CSIRO)

Brisbane and Adelaide, Australia Nov. 2012

TRAINER'S MANUAL

Licensing

Contents

Workshop Information

The Trainers

Dr. Nandan Deshpande

Postdoctoral Fellow
The University of New South Wales (UNSW), NSW
n.deshpande@unsw.edu.au

Dr. Konsta Duesing

Position
CSIRO
konsta.duesing@csiro.au

Dr. Xi (Sean) Li

Bioinformatics Analyst
Bioinformatics Core, CSIRO Mathematics, Informatics and Statistics, ACT
sean.li@csiro.au

Dr. Sean McWilliam

Position
CSIRO
sean.mcwilliam@csiro.au

Dr. Paula Moolhuijzen

Senior Bioinformatics Officer
Centre for Comparative Genomics, Murdoch University, WA
pmoolhuijzen@ccg.murdoch.edu.au

Dr. Sonika Tyagi

Senior Bioinformatics Officer
Australian Genome Research Facility Ltd, The Walter and Eliza Hall Institute, VIC
sonika.tyagi@agrf.org.au

Dr. Nathan S. Watson-Haigh

Senior Bioinformatician
The Australian Wine Research Institute (AWRI), SA
nathan.watson-haigh@awri.com.au

Do's and Don'ts of the Workshop

TODO

Document Structure

We have provided you with an electronic copy of the workshop's hands-on tutorial documents. We have done this for two reasons: 1) you will have something to take away with you at the end of the workshop, and 2) you can save time (mis)typing commands on the command line by using copy-and-paste.

While you could fly through the hands-on sessions doing copy-and-paste you will learn more if you take the time, saved from not having to type all those commands, to understand what each command is doing!

The commands to enter at a terminal look something like this:

```
1 tophat --solexa-quals -g 2 --library-type fr-unstranded -j \
    annotation/Danio_rerio.Zv9.66.spliceSites -o tophat/ZV9_2cells \
    genome/ZV9 data/2cells_1.fastq data/2cells_2.fastq
```

The following icons are used in the margin, throughout the documentation to help you locate:

 Important

 For reference

 Follow these steps

 Questions to answer

 Warning - STOP and read

 Bonus exercise for fast learners

 Advanced exercise for super-fast learners

Resources Used

We have provided you with an environment which contains all the tools and data you will need and use during this workshop. However, we have tried to include at the start of each module, details about the tools and data used.

Module: Data Quality

Primary Author(s):
Sonika Tyagi sonika.tyagi@agrf.org.au

Contributor(s):
Nathan S. Watson-Haigh nathan.watson-haigh@awri.com.au

Key Learning Outcomes

After completing this practical the trainee should be able to:

- Assess the overall quality of NGS sequence reads
- Visualise the quality, and other associated matrices, of reads to decide on filters and cutoffs for cleaning up data ready for downstream analysis
- Clean up and pre-process the sequences data for further analysis

Resources You'll be Using

Tools Used

FastQC
http://www.bioinformatics.babraham.ac.uk/projects/fastqc/

fastx-toolkit
http://hannonlab.cshl.edu/fastx_toolkit/

picard
http://picard.sourceforge.net/

fastq-mcf
http://code.google.com/p/ea-utils/wiki/FastqMcf

Introduction

Going on a blind date with your read set? For a better understanding of the consequences please check the data quality!

For the purpose of this tutorial we are focusing only on Illumina sequencing which uses 'sequence by synthesis' technology in a highly parallel fashion. Although Illumina high throughput sequencing provides highly accurate sequence data, several sequence artefacts, including base calling errors and small insertions/deletions, poor quality reads and primer/adapter contamination are quite common in the high throughput sequencing data. The primary errors are substitution errors. The error rates can vary from 0.5-2.0% with errors mainly rising in frequency at the 3' ends of reads.
One way to investigate sequence data quality is to visualize the quality scores and other metrics in a compact manner to get an idea about the quality of a read data set. Read data sets can be improved by post processing in different ways like trimming off low quality bases, cleaning up any sequencing adapters and removing PCR duplicates. We can also look at other statistics such as, sequence length distribution, base composition, sequence complexity, presence of ambiguous bases etc. to assess the overall quality of the data set. Highly redundant coverage (>15X) of the genome can be used to correct sequencing errors in the reads before assembly and errors. Various k-mer based error correction methods exist but are beyond the scope of this tutorial.

Prepare the Environment

To investigate sequence data quality we will demonstrate tools called FastQC and fastx-toolkit. FastQC will process and present the reports in a visual manner. Based on the results, the sequence data can be processed using the fastx-toolkit. We will use one data set in this practical, which can be found in the QC directory on your desktop.

Open the Terminal and go to the directory where the data are stored:

```
cd ~/QC/
pwd
```

At any time, help can be displayed for 'fastqc' using the following command:

```
fastqc -h
```

Quality Visualisation

We have a file for a good quality and bad quality statistics. FastQC generates results in the form of a zipped and unzipped directory for each input file.

Execute the following command on the two files:

```
fastqc -f fastq bad_example.fastq
fastqc -f fastq good_example.fastq
```

View the FastQC report file of the dab data using a web browser such as firefox.

```
firefox bad_example_fastqc/fastqc_report.html &
```

The report file will have a Basic Statistics table and various graphs and tables for different quality statistics. E.g.:

Table 1: FastQC Basic Statistics table

Filename	bad_example.fastq
File type	Conventional base calls
Encoding	Sanger / Illumina 1.9
Total Sequences	40000
Filtered Sequences	0
Sequence length	100
%GC	48

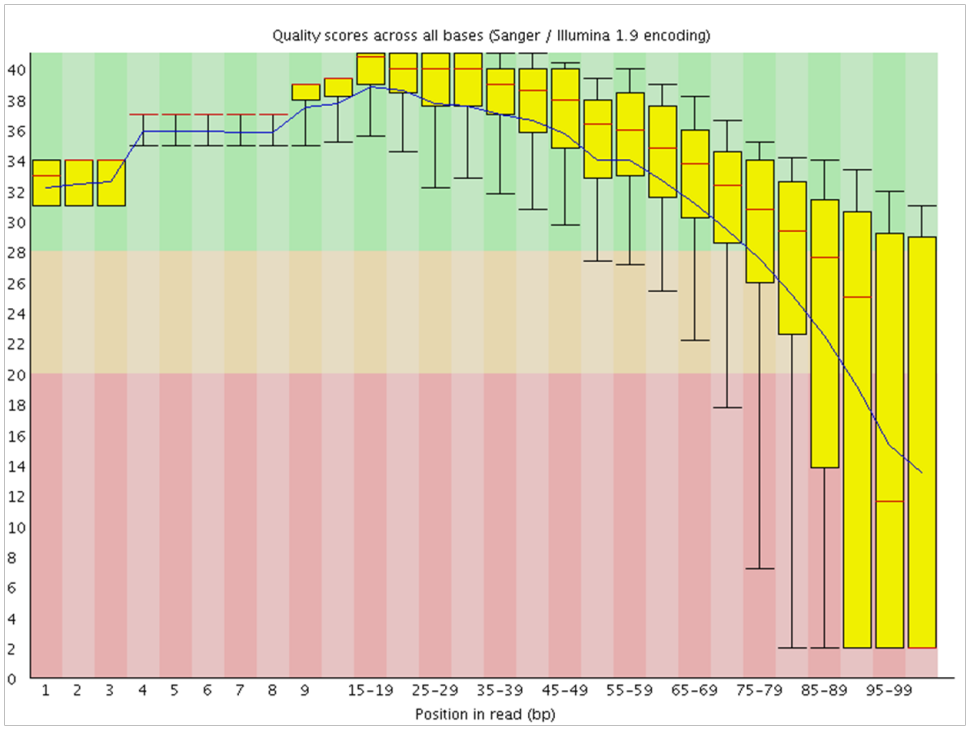


Figure 1: Per base sequence quality plot for `bad_example.fastq`. Base positions in the reads are shown on x-axis and quality score (Q Score) are shown on the Y-axis

A quality score (or Q-score) expresses an error probability. In particular, it serves as a convenient and compact way to communicate very small error probabilities. Given an assertion, A, the probability that A is not true, $P(\ A)$, is expressed by a quality score, $Q(A)$, according to the relationship:

$$Q(A) = -10log10(P(\sim A))$$

Where $P(\ A)$ is the estimated probability of an assertion A being wrong. The relationship between the quality score and error probability is demonstrated with the following table:

Table 2: Error probabilities associated with various quality (Q) values

Quality score, Q(A)	**Error probability, P(∼A)**
10	0.1
20	0.01
30	0.001
40	0.0001

How many sequences were there in your file? What is the read length? 40,000. read length=100bp
Does the quality score values vary throughout the read length? (hint: look at the 'per base sequence quality plot') Yes. Quality scores are dropping towards the end of the reads.
What is the quality score range you see? 2-40
At around which position do the scores start falling below Q20? Around 80 bp position
How can we trim the reads to filter out the low quality data? By trimming off the bases after a fixed position of the read. or by trimming off bases based on the quality score.

Good Quality Data

View the FastQC report files fastqc_report.html to see examples of a good quality data and compare the quality plot with that of the bad_example_fastqc.

```
1 firefox good_example_fastqc/fastqc_report.html &
```

Sequencing errors can complicate the downstream analysis, which normally requires that reads be aligned to each other (for genome assembly) or to a reference genome (for detection of mutations). Sequence reads containing errors may lead to ambiguous paths in the assembly or improper gaps. In variant analysis projects sequence reads are aligned against the reference genome. The errors in the reads may lead to more mismatches than expected from mutations alone. But if these errors can be removed or corrected, the read alignments and hence the variant detection will improve. The assemblies will also improve after pre-processing the reads with errors.

Read Trimming

Read trimming can be done in a variety of different ways. Choose a method which best suits your data. Here we are giving examples of fixed-length trimming and quality-based trimming.

Fixed Length Trimming

Low quality read ends can be trimmed using a fixed-length timmer. We will use the `fastx_trimmer` from the fastx-toolkit. Usage message to find out various options you can use with this tool. Type fastx_trimmer -h at anytime to display help.

We will no do fixed-length trimming of the `bad_example.fastq` file using the following command.

```
cd ~/QC
fastx_trimmer -h
fastx_trimmer -Q 33 -f 1 -l 80 -i bad_example.fastq -o \
    bad_example_trimmed01.fastq
```

We used the following options in the command above:

`-Q 33` Indicates the input quality scores are Phred+33 encoded

`-f` First base to be retianed in the output

`-l` Last base to be retained in the output

`-i` Input fastq file name

`-o` Output file name

Run fastqc on the trimmed file and visualise the quality scores of the trimmed file.

```
fastqc -f fastq bad_example_trimmed01.fastq
firefox bad_example_trimmed01_fastqc/fastqc_report.html &
```

The output should look like:

Table 3: FastQC Basic Statistics table

Filename	bad_example_trimmed01.fastq
File type	Conventional base calls
Encoding	Sanger / Illumina 1.9
Total Sequences	40000
Filtered Sequences	0
Sequence length	80
%GC	48

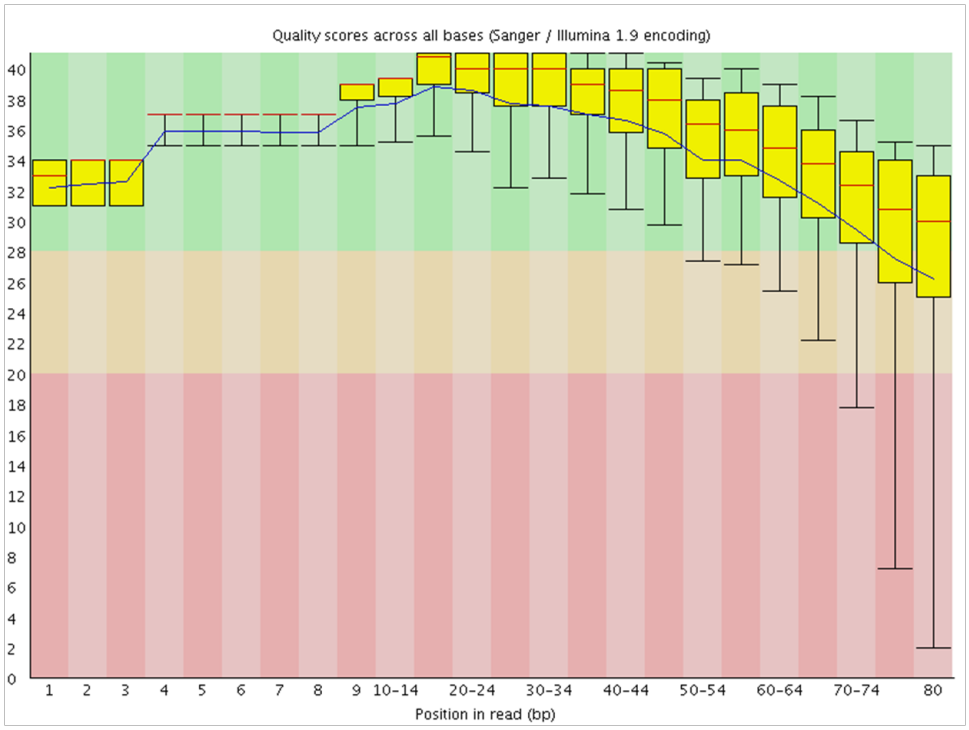


Figure 2: Per base sequence quality plot for the fixed-length trimmed `bad_example.fastq` reads. Base positions in the reads are shown on x-axis and quality score (Q Score) are shown on the Y-axis

What values would you use for `-f` if you wanted to trim off 10 bases at the 5' end of the reads? `-f 11`

Quality Based Trimming

Base call quality scores can also be used to dynamically determine the trim points for each read. A quality score threshold and minimum read length following trimming can be used to remove low quality data.

Run the following command to quality trim your data:

```
cd ~/QC
fastq_quality_trimmer -h
fastq_quality_trimmer -Q 33 -t 20 -l 50 -i bad_example.fastq -o \
    bad_example_quality_trimmed.fastq
```

Run FastQC on the quality trimmed file and visualise the quality scores.

```
fastqc -f fastq bad_example_quality_trimmed.fastq
firefox bad_example_quality_trimmed_fastqc/fastqc_report.html &
```

The output should look like:

Table 4: FastQC Basic Statistics table

Filename	bad_example_quality_trimmed.fastq
File type	Conventional base calls
Encoding	Sanger / Illumina 1.9
Total Sequences	38976
Filtered Sequences	0
Sequence length	50-100
%GC	48

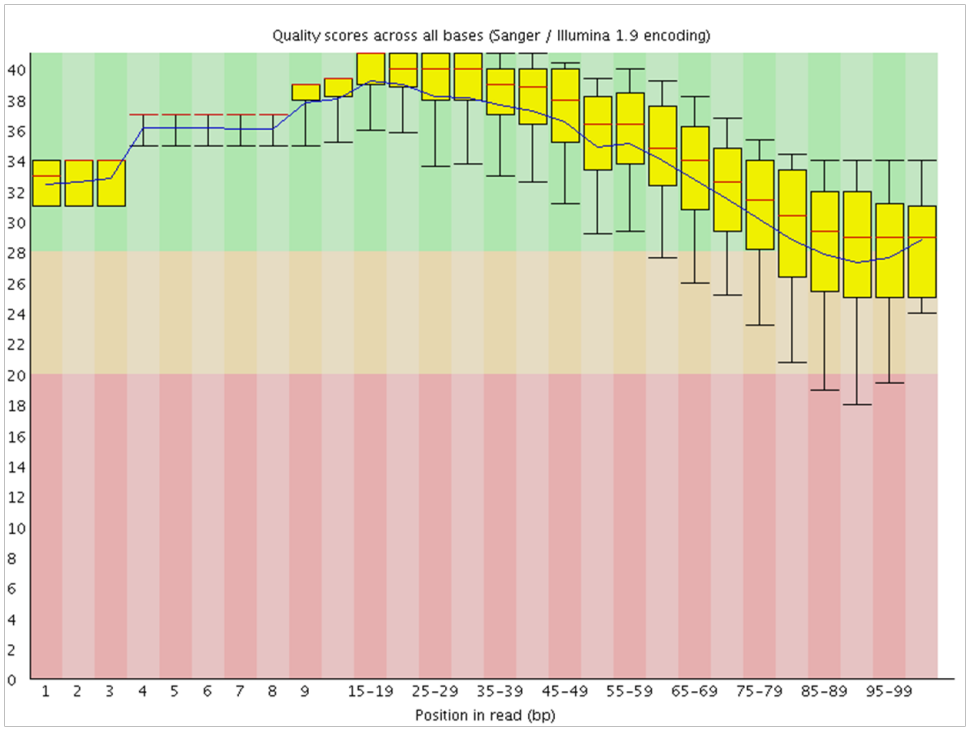


Figure 3: Per base sequence quality plot for the quality-trimmed `bad_example.fastq` reads. Base positions in the reads are shown on x-axis and quality score (Q Score) are shown on the Y-axis

How did the quality score range change with two types of trimming? Some poor quality bases (Q <20) are still present at the 3' end of the fixed-length trimmed reads. It also removes bases that are good quality.
Quality-based trimming retains the 3' ends of reads which have good quality scores.
Did the number of total reads change after two types of trimming? Quality trimming discarded >1000 reads. However, We retain a lot of maximal length reads which have good quality all the way to the ends.
What reads lengths were obtained after quality based trimming? 50-100
Reads <50 bp, following quality trimming, were discarded.
Did you observe adapter sequences in the data? No. (Hint: look at the overrepresented sequences.

Adapter Clipping

Sometime sequence reads may end up getting the leftover of adapters and primers used in the sequencing process. It's good practice to screen your data for these possible contaminations for more sensitive alignment and assembly based analysis. This is particularly important when read lengths can be longer than the molecules being sequenced. For example when sequencing miRNAs.

Various QC tools are available to screen and/or clip these adapter/primer sequences from your data. (e.g. FastQC, fastx, cutadapt).

Here we are demonstrating `fastx_clipper` to trim a given adapter sequence.

```
cd ~/QC
fastx_clipper -h
fastx_clipper -v -Q 33 -l 20 -M 15 -a \
    GATCGGAAGAGCGGTTCAGCAGGAATGCCGAG -i bad_example.fastq -o \
    bad_example_clipped.fastq
```

An alternative tool, not installed on this system, for adapter clipping is `fastq-mcf`. A list of adapters is provided in a text file. For more information, see Fastqmcf at the URL provided at the start of this module.

Removing Duplicates

Duplicate reads are the ones having the same start and end coordinates. This may be the result of technical duplication (too many PCR cycles), or over-sequencing (very high fold coverage). It is very important to put the duplication level in context of your experiment. For example, duplication level in targeted or re-sequencing projects may mean something different in RNA-seq experiments. In RNA-seq experiments oversequencing is usually necessary when detecting low abundance transcripts.

The duplication level computed by FastQC is based on sequence identity at the end of reads. Another tool, Picard, determines duplicates based on identical start and end positions in SAM/BAM alignment files.
We will not cover Picard but provide the following for your information.
Picard is a suite of tools for performing many common tasks with SAM/BAM format files. For more information see the Picard website and information about the various command-line tools available:
http://picard.sourceforge.net/command-line-overview.shtml

Picard 1.69 is installed on this system in `/usr/share/java/picard-1.69/`
One of the Picard tools (MarkDuplicates) can be used to analyse and remove duplicates from the raw sequence data. The input for Picard is a sorted alignment file in BAM format. Short read aligners such as, bowtie, BWA and tophat can be used to align FASTQ files against a reference genome to generate SAM/BAM alignment format.

Interested users can use the following general command to run the MarkDuplicates tool at their leisure and only need to provide a BAM file for INPUT:

```
cd ~/QC
java -jar /usr/share/java/picard/MarkDuplicates.jar \
    INPUT=<alignment_file.bam> VALIDATION_STRINGENCY=LENIENT \
    OUTPUT=alignment_file.dup METRICS_FILE=alignment_file.matric \
    ASSUME_SORTED=true REMOVE_DUPLICATES=true
```

Module: Read Alignment

Primary Author(s):
Myrto Kostadima kostadim@ebi.ac.uk

Contributor(s):
Xi Li sean.li@csiro.au

Key Learning Outcomes

After completing this practical the trainee should be able to:

- Perform the simple NGS data alignment task against one interested reference data
- Interpret and manipulate the mapping output using samtools
- Visualise the alignment via a standard genome browser, e.g. IGV browser

Resources You'll be Using

Tools Used

Bowtie
http://bowtie-bio.sourceforge.net/index.shtml

Bowtie 2
http://bowtie-bio.sourceforge.net/bowtie2/index.shtml

Samtools
http://picard.sourceforge.net/

BEDTools
http://code.google.com/p/bedtools/

UCSC tools
http://hgdownload.cse.ucsc.edu/admin/exe/

IGV genome browser
http://www.broadinstitute.org/igv/

Sources of Data

http://www.ebi.ac.uk/arrayexpress/experiments/E-GEOD-11431

Introduction

The goal of this hands-on session is to perform an unspliced alignment for a small subset of raw reads. We will align raw sequencing data to the mouse genome using *Bowtie* and then we will manipulate the SAM output in order to visualize the alignment on the *IGV browser*.

Prepare the Environment

We will use one data set in this practical, which can be found in the `ChIP-seq` directory on your desktop.

Open the Terminal.
First, go to the right folder, where the data are stored.

```
1 cd ~/ChIP-seq
```

The `.fastq` file that we will align is called `Oct4.fastq`. This file is based on Oct4 ChIP-seq data published by Chen et al. (2008). For the sake of time, we will align these reads to a single mouse chromosome.

Alignment

You already know that there are a number of competing tools for short read alignment, each with its own set of strengths, weaknesses, and caveats. Here we will try *Bowtie*, a widely used ultrafast, memory efficient short read aligner.

Bowtie has a number of parameters in order to perform the alignment. To view them all type

```
1 bowtie --help
```

Bowtie uses indexed genome for the alignment in order to keep its memory footprint small. Because of time constraints we will build the index only for one chromosome of the mouse genome. For this we need the chromosome sequence in FASTA format. This is stored in a file named `mm9`, under the subdirectory `bowtie_index`. The indexed chromosome is generated using the command:

```
1 bowtie-build bowtie_index/mm9.fa bowtie_index/mm9
```

This command will output 6 files that constitute the index. These files that have the prefix `mm9` are stored in the `bowtie_index` subdirectory. To view if they files have been successfully created type:

```
1 ls -l bowtie_index
```

Now that the genome is indexed we can move on to the actual alignment. The first argument for bowtie is the basename of the index for the genome to be searched; in our case is `mm9`. We also want to make sure that the output is in SAM format using the `-S` parameter. The last argument is the name of the FASTQ file.

Align the Oct4 reads using Bowtie:

```
1 bowtie bowtie_index/mm9 -S Oct4.fastq > Oct4.sam
```

The above command outputs the alignment in SAM format and stores them in the file `Oct4.sam`.

In general before you run *Bowtie*, you have to know which encoding your FASTQ files have. The available FASTQ encodings for bowtie are:

`--phred33-quals` Input quals are Phred+33 (default).

`--phred64-quals` Input quals are Phred+64 (same as `--solexa1.3-quals`).

`--solexa-quals` Input quals are from GA Pipeline ver. < 1.3.

`--solexa1.3-quals` Input quals are from GA Pipeline ver. ≥ 1.3.

`--integer-quals` Qualities are given as space-separated integers (not ASCII).

The FASTQ files we are working with are Sanger encoded (Phred+33), which is the default for *Bowtie*.
Bowtie will take 2-3 minutes to align the file. This is fast compared to other aligners which sacrifice some speed to obtain higher sensitivity.

Look at SAM format by typing:

```
1 head -n 10 Oct4.sam
```

Can you distinguish between the header of the SAM format and the actual alignments?
The header line starts with the letter '@', i.e.:
@HD VN:1.0 SO:unsorted
@SQ SN:chr1 LN:197195432
@PG ID:Bowtie VN:0.12.8 CL:"bowtie bowtie_index/mm9 -S Oct4.fastq"
While, the actual alignments start with read id, i.e.:
SRR002012.45 0 chr1 etc
SRR002012.48 16 chr1 etc
What kind of information does the header provide? 1. @HD: Header line; VN: Format version; SO: the sorting order of alignment. 2. @SQ: Reference sequence information; SN: reference sequence name; LN: reference sequence length. 3. @PG: Read group information; ID: Read group identifier; VN: Program version; CL: the command line that produces the alignment.
To which chromosome are the reads mapped? Chromosome 1.

Manipulate SAM output

SAM files are rather big and when dealing with a high volume of NGS data, storage space can become an issue. As we have already seen, we can convert SAM to BAM files (their binary equivalent that are not human readable) that occupy much less space.

Convert SAM to BAM using *samtools* and store the output in the file `Oct4.bam`. You have to instruct *samtools* that the input is in SAM format (`-S`), the output should be in BAM format (`-b`) and that you want the output to be stored in the file specified by the `-o` option:

```
1 samtools view -bSo Oct4.bam Oct4.sam
```

Compute simple stats of the alignment using *samtools*:

```
1 samtools flagstat Oct4.bam
```

Visualize alignments in IGV

IGV is a stand-alone genome browser. Please check their website (http://www.broadinstitute.org/igv/) for all the formats that *IGV* can display. For our visualization purposes we will use the BAM and bigWig formats.

When uploading a BAM file into the genome browser, the browser will look for the index of the BAM file in the same folder where the BAM files is. The index file should have the same name as the BAM file and the suffix `.bai`. Finally, to create the index of a BAM file you need to make sure that the file is sorted according to chromosomal coordinates.

Sort alignments according to chromosome position and store the result in the file with the prefix `Oct4.sorted`:

```
1 samtools sort Oct4.bam Oct4.sorted
```

Index the sorted file.

```
1 samtools index Oct4.sorted.bam
```

The indexing will create a file called `Oct4.sorted.bam.bai`. Note that you don't have to specify the name of the index file when running samtools.

Another way to visualize the alignments is to convert the BAM file into a bigWig file. The bigWig format is for display of dense, continuous data and the data will be displayed as a graph. The resulting bigWig files are in an indexed binary format.

The BAM to bigWig conversion takes place in two steps. Firstly, we convert the BAM file into a bedgraph, called `Oct4.bedgraph`, using the tool `genomeCoverageBed` from *BEDTools*:

```
genomeCoverageBed -bg -ibam Oct4.sorted.bam -g \
    bowtie_index/mouse.mm9.genome > Oct4.bedgraph
```

Then we convert the bedgraph into a binary graph, called `Oct4.bw`, using the tool `bedGraphToBigWig` from the *UCSC* tools:

```
bedGraphToBigWig Oct4.bedgraph bowtie_index/mouse.mm9.genome Oct4.bw
```

Both of the commands above take as input a file called `mouse.mm9.genome` that is stored under the subdirectory `bowtie_index`. These genome files are tab-delimited and describe the size of the chromosomes for the organism of interest. When using the UCSC Genome Browser, Ensembl, or Galaxy, you typically indicate which species/genome build you are working. The way you do this for *BEDTools* is to create a "genome" file, which simply lists the names of the chromosomes (or scaffolds, etc.) and their size (in basepairs). *BEDTools* includes pre-defined genome files for human and mouse in the `genomes` subdirectory included in the *BEDTools* distribution.

Now we will load the data into the IGV browser for visualization. In order to launch IGV double click on the IGV 2.1 icon on your Desktop. Ignore any warnings and when it opens you have to load the genome of interest.
On the top left of your screen choose from the drop down menu `Mus musculus (mm9)`. Then in order to load the desire files go to:

```
File > Load from File
```

On the pop up window navigate to Desktop > ChIP-seq folder and select the file `Oct4.sorted.bam`. Repeat these steps in order to load `Oct4.bw` as well. Select chr1 from the drop down menu on the top left. Right click on the name of `Oct4.bw` and choose Maximum under the Windowing Function. Right click again and select Autoscale. In order to see the aligned reads of the BAM file, you need to zoom in to a specific region. For example, look for gene `Lemd1` in the search box.

What is the main difference between the visualization of BAM and bigWig files? The actual alignment of reads that stack to a particular region can be displayed using the information stored in a BAM format. The bigWig format is for display of dense, continuous data that will be displayed in the Genome Browser as a graph.

Using the + button on the top right zoom in more to see the details of the alignment.

What do you think the different colors mean? The different color represents four nucleotides, e.g. blue is Cytidine (C), red is Thymidine (T).

Practice

In the ChIP-seq folder you will find another `.fastq` file called `gfp.fastq`. Follow the above described analysis for this dataset as well.

Module: ChIP-Seq

Primary Author(s):
Remco Loos, EMBL-EBI remco@ebi.ac.uk
Myrto Kostadima kostadim@ebi.ac.uk

Contributor(s):
Xi Li sean.li@csiro.au

Key Learning Outcomes

After completing this practical the trainee should be able to:

- Perform simple ChIP-Seq analysis, e.g. the detection of immuno-enriched areas using the chosen peak caller program MACS
- Visualize the peak regions through a genome browser, e.g. Ensembl, and identify the real peak regions
- Perform functional annotation and detect potential binding sites (motif) in the predicted binding regions using motif discovery tool, e.g. MEME.

Resources You'll be Using

Tools Used

MACS
http://liulab.dfci.harvard.edu/MACS/index.html

Ensembl
http://www.ensembl.org

PeakAnalyzer
http://www.ebi.ac.uk/bertone/software

MEME
http://meme.sdsc.edu/meme/cgi-bin/meme.cgi

TOMTOM
http://meme.sdsc.edu/meme/cgi-bin/tomtom.cgi

DAVID
http://david.abcc.ncifcrf.gov

GOstat
http://gostat.wehi.edu.au

Sources of Data

http://www.ebi.ac.uk/arrayexpress/experiments/E-GEOD-11431

Introduction

The goal of this hands-on session is to perform some basic tasks in the analysis of ChIP-seq data. In fact, you already performed the first step, alignment of the reads to the genome, in the previous session. We start from the aligned reads and we will find immuno-enriched areas using the peak caller MACS. We will visualize the identified regions in a genome browser and perform functional annotation and motif analysis on the predicted binding regions.

Prepare the Environment

The material for this practical can be found in the ChIP-seq directory on your desktop. This directory also contains an electronic version of this document, which can be useful to copy and paste commands. Please make sure that this directory also contains the SAM/BAM files you produced during the alignment practical.

If you didn't have time to align the control file called `gfp.fastq` during the alignment practical, please do it now.
Follow the same steps as for the `Oct4.fastq` file.

Of course, the same bowtie index can be used, so the index building step can be skipped.

In ChIP-seq analysis (unlike in other applications such as RNA-seq) we often would like to exclude all reads that map to more than one location in the genome, to avoid false positives. When using Bowtie, this can be done using the `-m 1` option, which tells it to report only unique matches (See `bowtie --help` for more details).

Open the Terminal and go to the `ChIP-seq` directory:

```
1 cd ~/ChIP-seq
```

Finding enriched areas using MACS

MACS stands for Model based analysis of ChIP-seq. It was designed for identifying transcription factor binding sites. MACS captures the influence of genome complexity to evaluate the significance of enriched ChIP regions, and improves the spatial resolution of binding sites through combining the information of both sequencing tag position and orientation. MACS can be easily used for ChIP-Seq data alone, or with a control sample to increase specificity.

Consult the MACS help file to see the options and parameters:

```
1 macs --help
```

The input for MACS can be in ELAND, BED, SAM, BAM or BOWTIE formats (you just have to set the `--format` option).
Options that you will have to use include:

`-t` To indicate the input ChIP file.

`-c` To indicate the name of the control file.

`--format` To change the file format. The default format is bed..

`--name` To set the name of the output files.

`--gsize` This is the mappable genome size. With the read length we have, 70% of the genome is a fair estimation. Since in this analysis we include only reads from chromosome 1, we will use as gsize 70% of the length of chromosome 1 (197 Mb).

`--tsize` To set the read length (look at the fastq files to check the length).

`--wig` To generate signal wig files for viewing in a genome browser. Since this process is time consuming, it is recommended to run MACS first with this flag off, and once you decide on the values of the parameters, run MACS again with this flag on.

`--diag` To generate a saturation table, which gives an indication whether the sequenced reads give a reliable representation of the possible peaks.

Now run macs using the following command:

```
1 macs -t [Oct4 aligned bam file] -c [gfp aligned bam file] --format=BAM \
    --name=Oct4 --gsize=138000000 --tsize=26 --diag --wig
```

Look at the output saturation table (`Oct4 diag.xls`). To open Excel files, right-click on them, choose Open with and select OpenOffice. Do you think that more sequencing is necessary? Open the Excel peak file and view the peak details. Note that the number of tags (column 6) refers to the number of reads in the whole peak region and not the summit height.

Viewing results with the Ensembl genome browser

It is often instructive to look at your data in a genome browser. Before, we used IGV, a stand-alone browser, which has the advantage of being installed locally and providing fast access. Web-based genome browsers, like Ensembl or the UCSC browser, are slower, but provide more functionality. They do not only allow for more polished and flexible visualisation, but also provide easy access to a wealth of annotations and external data sources. This makes it straightforward to relate your data with information about repeat regions, known genes, epigenetic features or areas of cross-species conservation, to name just a few. As such, they are useful tools for exploratory analysis.
They will allow you to get a 'feel' for the data, as well as detecting abnormalities and problems. Also, exploring the data in such a way may give you ideas for further analyses.

Launch a web bowser and go to the Ensembl website at `http://www.ensembl.org/index.html`
Choose the genome of interest (that is, mouse) on the left side of the page.
Click on the **Manage your data** link on the left, then choose **Attach remote file**.

One option is to browse to the wig file MACS generated (usually under MACS wiggle/treat) and to upload it to Ensembl (option **Upload data**). However, since wig files describing sequencing signal are very big, they would take a long time to upload, and the browsing process will be very slow.
As a better alternative, wig files can be converted to an indexed binary format and put into a web accessible server (http, https, or ftp) instead on the Ensembl server. This makes all the browsing process much faster. Detailed instructions for generating a bigWig from a wig type file can be found at:
`http://genome.ucsc.edu/goldenPath/help/bigWig.html`.

We have generated bigWig files in advance for you to upload to the Ensembl browser. They are at the following URL: `http://www.ebi.ac.uk/~remco/ChIP-Seq_course/Oct4.bw`.
To visualise the data:

- Paste the location above in the field File URL.
- Choose data format Bigwig.
- Choose some informative name and in the next window choose the colour of your preference.
- Click **Save** and close the window to return to the genome browser.

Repeat the process for the control sample, located at `http://www.ebi.ac.uk/~remco/ChIP-Seq_course/gfp.bw`.
After uploading, choose **Configure this page**, and under **Your data** tick both boxes. Closing the window will save these changes.
Go to a region on chromosome 1 (e.g. `1:34823162-35323161`), and zoom in and out to view the signal and peak regions. Be aware that the data scale automatically, so bigger-looking peaks need not actually be bigger. Always look at the values on the left hand side axis.

What can you say about the profile of Oct4 peaks? There is no significant peaks of Oct4 over the selected region.
Compare it with H3K4me3 histone modification wig file we have generated at http://www.ebi.ac.uk/~remco/ChIP-Seq_course/H3K4me3.bw. H3K4me3 has a region that contains relatively high peaks than Oct4.
Jump to `1:36066594-36079728` for a sample peak. Do you think H3K4me3 peaks regions contain one or more modification sites? What about Oct4? Yes. There are roughly three peaks, which indicate the possibility of having more than one modification sites in this region.
For Oct4, no peak can be observed.

MACS generates its peak files in a file format called bed file. This is a simple text format containing genomic locations, specified by chromosome, begin and end positions, and some more optional information.
See http://genome.ucsc.edu/FAQ/FAQformat.html#format1 for details.
Bed files can also be uploaded to the Ensembl browser.

Try uploading the peak file generated by MACS to Ensembl. Find the first peak in the file (use the `head` command to view the beginning of the bed file), and see if the peak looks convincing to you.

Annotation: From peaks to biological interpretation

In order to biologically interpret the results of ChIP-seq experiments, it is usually recommended to look at the genes and other annotated elements that are located in proximity to the identified enriched regions. This can be easily done using PeakAnalyzer.

Go to the PeakAnalyzer directory and launch the program by typing:

```
1 java -jar PeakAnalyzer.jar &
```

The first window allows you to choose between the split application (which we will try next) and peak annotation. Choose the peak annotation option and click **Next**. We would like to find the closest downstream genes to each peak, and the genes that overlap with the peak region. For that purpose you should choose the **NDG** option and click **Next**. Fill in the location of the peak file `Oct4 peaks.bed`, and choose the mouse GTF as the annotation file. You don't have to define a symbol file since gene symbols are included in the GTF file. Choose the output directory and run the program.

When the program has finished running, you will have the option of generating plots, by pressing **Generate plots** (You can only do this if R is installed on your computer, as is the case here. Otherwise, if you don't want to install R, you can generate similar plots

with Excel using the output files that were generated by PeakAnalyzer.). A pdf file with the plots will be generated in the output folder.

This list of closest downstream genes (contained in the file `Oct4_peaks.ndg.bed`) can be the basis of further analysis. For instance, you could look at the Gene Ontology terms associated with these genes to get an idea of the biological processes that may be affected. Web-based tools like DAVID (`http://david.abcc.ncifcrf.gov`) or GOstat (`http://gostat.wehi.edu.au`) take a list of genes and return the enriched GO categories.

Motif analysis

It is often interesting to find out whether we can associate identified the binding sites with a sequence pattern or motif. We will use MEME for motif analysis. The input for MEME should be a file in fasta format containing the sequences of interest. In our case, these are the sequences of the identified peaks that probably contain Oct4 binding sites. Since many peak-finding tools merge overlapping areas of enrichment, the resulting peaks tend to be much wider than the actual binding sites. Sub-dividing the enriched areas by accurately partitioning enriched loci into a finer-resolution set of individual binding sites, and fetching sequences from the summit region where binding motifs are most likely to appear enhances the quality of the motif analysis. Sub-peak summit sequences can be retrieved directly from the Ensembl database using PeakAnalyzer.

If you have closed the PeakAnalyzer running window, open it again. If it is still open, just go back to the first window.
Choose the split peaks utility and click **Next**. The input consists of files generated by most peak-finding tools: a file containing the chromosome, start and end locations of the enriched regions, and a `.wig` signal file describing the size and shape of each peak. Fill in the location of both files `Oct4 peaks.bed` and the wig file generated by MACS, which is under `Oct4_MACS_wiggle/treat`, check the option to **Fetch subpeak sequences** and click **Next**.
In the next window you have to set some parameters for splitting the peaks.
Separation float - This value determines when a peak will be separated into sub-peaks. This is the ratio between a valley and its neighbouring summit (the lower summit of the two). For example, if you set this height to be 0.5, two sub-peaks will be separated only if the height of the lower summit is twice the height of the valley. Keep the default value.
Minimum height - only sub-peaks with at least this number of tags in their summit region will be separated. Set this to be 5. Change the organism name from the default human to mouse and run the program.

Since the program has to read large wig files, it will take a few minutes to run. Once the run is finished, two output files will be produced. The first describes the location of the sub-peaks, and the second is a fasta file containing 300 sequences of length 61 bases, taken from the summit regions of the highest sub-peaks.

Open a web bowser and go to the MEME website at http://meme.ebi.edu.au/meme/cgi-bin/meme.cgi, and fill in the necessary details, such as:

- your e-mail address
- the sub-peaks fasta file `Oct4 peaks.bestSubPeaks.fa` (will need uploading), or just paste in the sequences.
- the number of motifs we expect to find (1 per sequence)
- the width of the desired motif (between 6 to 20)
- the maximum number of motifs to find (3 by default). For Oct4 one classical motif is known.

You will receive the results by e-mail. This usually doesn't take more than a few minutes.

Open the e-mail and click on the link that leads to the html results page.
Scroll down until you see the first motif logo. We would like to know if this motif is similar to any other known motif. We will use TOMTOM for this. Scroll down until you see the option **Submit this motif to**. Click the TOMTOM button to compare to known motifs in motif databases, and on the new page choose to compare your motif to those in the JASPAR and UniPROBE database.

Which motif was found to be the most similar to your motif? Sox2

Reference

Chen, X et al.: Integration of external signaling pathways with the core tran-scriptional network in embryonic stem cells. Cell 133:6, 1106-17 (2008).

Module: RNA-Seq

Primary Author(s):
Myrto Kostadima, EMBL-EBI kostadmi@ebi.ac.uk
Remco Loos, EMBL-EBI remco@ebi.ac.uk

Contributor(s):
Nathan S. Watson-Haigh nathan.watson-haigh@awri.com.au

Key Learning Outcomes

After completing this practical the trainee should be able to:

- TODO 1
- TODO 2

Resources You'll be Using

Tools Used

Tophat
http://tophat.cbcb.umd.edu/

Cufflinks
http://cufflinks.cbcb.umd.edu/

Samtools
http://samtools.sourceforge.net/

BEDTools
http://code.google.com/p/bedtools/

UCSC tools
http://hgdownload.cse.ucsc.edu/admin/exe/

IGV
http://www.broadinstitute.org/igv/

DAVID Functional Analysis
http://david.abcc.ncifcrf.gov/

Sources of Data

http://www.ebi.ac.uk/ena/data/view/ERR022484
http://www.ebi.ac.uk/ena/data/view/ERR022485

Introduction

The goal of this hands-on session is to perform some basic tasks in the downstream analysis of RNA-seq data. We will start from RNA-seq data aligned to the zebrafish genome using Tophat.
We will perform transcriptome reconstruction using Cufflinks and we will compare the gene expression between two different conditions in order to identify differentially expressed genes.

Prepare the Environment

We will use a dataset derived from sequencing of mRNA from Danio rerio embryos in two different developmental stages. Sequencing was performed on the Illumina platform and generated 76bp paired-end sequence data using polyA selected RNA. Due to the time constraints of the practical we will only use a subset of the reads.
The data files are contained in the subdirectory called `data` and are the following:

`2cells_1.fastq` and `2cells_2.fastq`
: These files are based on RNA-seq data of a 2-cell zebrafish embryo

`6h_1.fastq` and `6h_2.fastq`
: These files are based on RNA-seq data of zebrafish embryos 6h post fertilization

Open the Terminal and go to the directory where the data is stored:

```
1 cd ~/RNA-seq/
```

All commands entered into the terminal for this tutorial should be from within the `RNA-seq` directory above.

Check that the `data` directory contains the above-mentioned files by typing:

```
1 ls -l data
```

Alignment

There are numerous tools performing short read alignment and the choice of aligner should be carefully made according to the analysis goals/requirements. Here we will use Tophat, a widely used ultrafast aligner that performs spliced alignments.
Tophat is based on Bowtie to perform alignments and uses an indexed genome for the alignment to keep its memory footprint small. We have already seen how to index the genome (see Alignment hands-on session), therefore for time purposes we have already generated the index for the zebrafish genome and placed it under the `genome` subdirectory.

Tophat has a number of parameters in order to perform the alignment. To view them all type:

```
1 tophat --help
```

The general format of the tophat command is:

```
1 tophat [options]* <index_base> <reads_1> <reads_2>
```

Where the last two arguments are the `.fastq` files of the paired end reads, and the argument before is the basename of the indexed genome.

Like with Bowtie before you run Tophat, you have to know which quality encoding the fastq formatted reads are in.

Can you tell which quality encoding our fastq formatted reads are in?
Look at the first few reads in the file `data/2cells_1.fastq` and then compare the quality strings with the table found at: http://en.wikipedia.org/wiki/FASTQ_format#Encoding.

```
1 head -n 20 data/2cells_1.fastq
```

Some other parameters that we are going to use to run Tophat are listed below:

`-g` Maximum number of multihits allowed. Short reads are likely to map to more than one locations in the genome even though these reads can have originated from only one of these regions. In RNA-seq we allow for a restricted number of multihits, and in this case we ask Tophat to report only reads that map at most onto 2 different loci.

`--library-type` Before performing any type of RNA-seq analysis you need to know a few things about the library preparation. Was it done using a strand-specific protocol or not? If yes, which strand? In our data the protocol was NOT strand specific.

`-J` Improve spliced alignment by providing Tophat with annotated splice junctions. Pre-existing genome annotation is an advantage when analysing RNA-seq data. This file contains the coordinates of annotated splice junctions from Ensembl. These are stored under the sub-directory `annotation` in a file called `ZV9.spliceSites`.

`-o` This specifies in which subdirectory Tophat should save the output files. Given that for every run the name of the output files is the same, we specify different directories for each run.

Since it takes a long time to perform spliced alignments, even for this subset of reads, we have pre-aligned the `2cells` data for you using the following command:

DO NOT run this command yourself - you may overwrite the output files of the pre-aligned data.

```
tophat --solexa-quals -g 2 --library-type fr-unstranded -j \
    annotation/Danio_rerio.Zv9.66.spliceSites -o tophat/ZV9_2cells \
    genome/ZV9 data/2cells_1.fastq data/2cells_2.fastq
```

Using the above command as a guide, can you devise and run the command for aligning the `6h` data?

The alignment will take approximately 17 minutes. In the meantime please move on with the practical and we will get back to the terminal once the alignment is done.

We will firstly look at some of the files produced by Tophat. For this please open the `RNA-seq` directory which can be found on your Desktop. Click on the `tophat` subdirectory and then on the directory called `ZV9_2cells`.
Tophat reports the alignments in a BAM file called `accepted_hits.bam`. Among others it also creates a junctions.bed files that stores the coordinates of the splice junctions present in your dataset, as these have been extracted from the spliced alignments.
Now we will load the BAM file and the splice junctions onto IGV to visualize the alignments reported by Tophat.
In order to launch IGV type double click on the IGV 2.1 icon on your Desktop. Ignore any warnings and when it opens you have to load the genome of interest.
On the top left of your screen choose from the drop down menu Zebrafish (Zv9). Then in order to load the desire files go to:

```
File >> Load from File
```

In the pop-up window, navigate to the `Desktop/RNA-seq/tophat/ZV9_2cells` directory and select the file `accepted_hits.sorted.bam`. Once the file is loaded right-click on the name of the track on the left and choose Rename Track. Give the track a meaningful name. Follow the same steps in order to load the `junctions.bed` file from the same directory. Using the same proceedure, load the Ensembl annotation `Danio_rerio.Zv9.66.gtf` file stored under the `RNA-seq/annotation` directory. On the top middle box you can specify the region you want your browser to zoom. Type: `chr12:20,270,921-20,300,943`

Can you identify the splice junctions from the BAM file?
Are the junctions annotated for `CBY1` consistent with the annotation?
Are all annotated genes, from both RefSeq and Ensembl, expressed?

We already know that in order to load a BAM file onto IGV we need to have this file sorted by genomic location and indexed. HereŠs a reminder of the command syntax to perform these tasks

```
1 samtools sort [bam_file_to_be_sorted] [prefix_of_sorted_bam_output_file]
2 samtools index [sorted_bam_file]
```

Once Tophat finishes running, sort the output BAM file and then index the sorted BAM file using the information above to guide you.
Then load them onto IGV following the steps above.

Isoform Expression and Transcriptome Assembly

There are a number of tools that perform reconstruction of the transcriptome and for this workshop we are going to use Cufflinks. Cufflinks can do transcriptome assembly either ab initio or using a reference annotation. It also quantifies the isoform expression in FPKMs. A reminder from the presentation this morning that FPKM stands for 'Fragments Per Kilobase of exon per Million fragments mapped'.

Cufflinks has a number of parameters in order to perform transcriptome assembly and quantification. To view them all type:

```
1 cufflinks --help
```

We aim to reconstruct the transcriptome for both samples by using the Ensembl annotation both strictly and as a guide. In the first case Cufflinks will only report isoforms that are included in the annotation, while in the latter case it will report novel isoforms as well. The annotation from Ensembl of Danio rerio is stored under the directory `annotation` in a file called `Danio_rerio.Zv9.66.gtf`.

The general format of the cufflinks command is:

```
1 cufflinks [options]* <aligned_reads.(sam|bam)>
```

Where the input is the aligned reads (either in SAM or BAM format).

Some of available parameters of Cufflinks that we are going to use to run Cufflinks are listed below:

`-o` Output directory

`-G` Tells Cufflinks to use the supplied annotation strictly in order to estimate isoform annotation

`-b` Instructs Cufflinks to run a bias detection and correction algorithm which can significantly improve accuracy of transcript abundance estimates. To do this Cufflinks requires a multi-fasta file with the genomic sequences against which we have aligned the reads

`-u` Tells Cufflinks to do an initial estimation procedure to more accurately weight reads mapping to multiple locations in the genome (multi-hits).

`--library-type` See Tophat parameters

In the terminal type:

```
1 cufflinks -o cufflinks/ZV9_2cells_gff -G \
    annotation/Danio_rerio.Zv9.66.gtf -b \
    genome/Danio_rerio.Zv9.66.dna.fa -u --library-type fr-unstranded \
    tophat/ZV9_2cells/accepted_hits.bam
```

Don't forget to change the output directory for the question section below. Otherwise your command will overwrite the results of the previous run.

Using the above command, which we used to pre-align the `2cells` data, can you devise and run a command for the `6h` data?
Take a look at the output directories that have been created. The results from Cufflinks are stored in 4 different files, see below.

Here's a short description of these files:

`genes.fpkm_tracking` Contains the estimated gene-level expression values.

`isoforms.fpkm_tracking` Contains the estimated isoform-level expression values.

`skipped.gtf` Contains loci skipped as a result of exceeding the maximum number of fragments.

`transcripts.gtf` This GTF file contains Cufflinks' assembled isoforms.

The complete documentation can be found at: http://cufflinks.cbcb.umd.edu/manual.html#cufflinks_output

Now in order to perform guided transcriptome assembly (transcriptome assembly that reports novel transcripts as well) we need to change the `-G` option of the previous command to `-g` instead. This tells Cufflinks to assemble the transcriptome using the supplied annotation as a guide, thus allowing for novel transcripts.

Performing the guided transcriptome analysis for the `2cells` and `6h` data sets would take 15-20min each. Therefore, we have pre-computed these for you and have the results available under subdirectories: `cufflinks/ZV9_2cells` and `cufflinks/ZV9_6h`.

Rather than running the commands for guided transcriptome assembly, which would take too much time, please have a think about the cufflinks command you would need to use to run in order to perform a guided transcriptome assembly for the `2cells` dataset. Write your answer below:

Go back to the IGV browser and load the file transcripts.gtf which is located in the subdirectory `cufflinks/ZV9_2cells/`. Rename the track into something meaningful.

This file contains the transcripts that Cufflinks assembled based on the alignment of our reads onto the genome.

In the search box type `ENSDART00000082297` in order for the browser to zoom in to the gene of interest. Compare between the already annotated transcripts and the ones assembled by Cufflinks. Do you observe any difference?

Differential Expression

One of the stand-alone tools that perform differential expression analysis is Cuffdiff. We use this tool to compare between two conditions; for example different conditions could be control and disease, or wild-type and mutant, or various developmental stages. In our case we want to identify genes that are differentially expressed between two developmental stages; a `2cell` embryo and `6h` post fertilization.

The general format of the cuffdiff command is:

```
cuffdiff [options]* <transcripts.gtf> \
    <sample1_replicate1.sam[,...,sample1_replicateM]> \
    <sample2_replicate1.sam[,...,sample2_replicateM.sam]>
```

Where the input includes a `transcripts.gtf` file, which is an annotation file of the genome of interest, and the aligned reads (either in SAM or BAM format) for the conditions. Some of the Cufflinks options that we will use to run the program are:

`-o` output directory

`-L` labels for the different conditions

`-T` Tells Cuffdiff that the reads are from a time series experiment

`-b` Same as for Cufflinks

`-u` Same as for Cufflinks

`--library-type` Same as for Cufflinks

To run cufdiff type on the terminal:

```
cuffdiff -o cuffdiff/ -L ZV9_2cells,ZV9_6h -T -b \
    genome/Danio_rerio.Zv9.66.dna.fa -u --library-type fr-unstranded \
    annotation/Danio_rerio.Zv9.66.gtf \
    tophat/ZV9_2cells/accepted_hits.bam tophat/ZV9_6h/accepted_hits.bam
```

In the command above we have assumed that the directory where you stored the results of Tophat for dataset `6h` was named `ZV9_6h`. If this is not the case please change the previous command accordingly otherwise you will get an error.

We are interested in the differential expression at the gene level. The results are reported by Cuffdiff in the file `cuffdiff/gene_exp.diff`. Look at the first few lines of the file using the following command:

```
1 head -n 20 cuffdiff/gene_exp.diff
```

We would like to see which are the most significantly differentially expressed genes. Therefore we will sort the above file according to the q value (corrected p value for multiple testing). The result will be stored in a different file called `gene_exp_qval.sorted.diff`.

```
1 sort -t$'\t' -g -k 13 cuffdiff/gene_exp.diff > \
    cuffdiff/gene_exp_qval.sorted.diff
```

Look again at the first few lines of the sorted file by typing:

```
1 head -n 20 cuffdiff/gene_exp_qval.sorted.diff
```

Copy the Ensembl identifier of one of these genes. Now go back to the IGV browser and paste it in the search box. Look at the raw aligned data for the two datasets.

Do you see any difference in the gene coverage between the two conditions that would justify that this gene has been called as differentially expressed?

Note that the coverage on the Ensembl browser is based on raw reads and no normalisation has taken place contrary to the FPKM values.

Functional Annotation of Differentially Expressed Genes

After you have performed the differential expression analysis you are interested in identifying if there is any functionality enrichment for your differentially expressed genes. On your Desktop click:

```
1 Applications >> Internet >> Firefox Web Browser
```

And go to the following URL: http://david.abcc.ncifcrf.gov/ On the left side click on Functional Annotation. Then click on the Upload tab. Under the section Choose from File, click Choose File and navigate to the `cuffdiff` directory. Select the file called `globalDiffExprs_Genes_qval.01_top100.tab`. Under Step 2 select ENSEMBL_GENE_ID from the drop-down menu. Finally select Gene list and then press Submit List. Click on Gene Ontology and then click on the CHART button of the GOTERM_BP_ALL item.

Do these categories make sense given the samples weŠre studying?
Browse around DAVID website and check what other information are available.

References

1. Trapnell, C., Pachter, L. & Salzberg, S. L. TopHat: discovering splice junctions with RNA-Seq. Bioinformatics 25, 1105-1111 (2009).

2. Trapnell, C. et al. Transcript assembly and quantification by RNA-Seq reveals unannotated transcripts and isoform switching during cell differentiation. Nat. Biotechnol. 28, 511-515 (2010).

3. Langmead, B., Trapnell, C., Pop, M. & Salzberg, S. L. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. Genome Biol. 10, R25 (2009).

4. Roberts, A., Pimentel, H., Trapnell, C. & Pachter, L. Identification of novel transcripts in annotated genomes using RNA-Seq. Bioinformatics 27, 2325-2329 (2011).

5. Roberts, A., Trapnell, C., Donaghey, J., Rinn, J. L. & Pachter, L. Improving RNA-Seq expression estimates by correcting for fragment bias. Genome Biol. 12, R22 (2011).

Module: *de novo* Assembly

Primary Author(s):
Matthias Haimel mhaimel@ebi.ac.uk
Nathan S. Watson-Haigh nathan.watson-haigh@awri.com.au

Contributor(s):

Key Learning Outcomes

After completing this practical the trainee should be able to:

- TODO 1
- TODO 2

Resources You'll be Using

Although we have provided you with an environment which contains all the tools and data you will be using in this module, you may like to know where we have sourced those tools and data from.

Tools Used

Velvet
http://www.ebi.ac.uk/~zerbino/velvet/

AMOS Hawkeye
http://apps.sourceforge.net/mediawiki/amos/index.php?title=Hawkeye

gnx-tools
https://github.com/mh11/gnx-tools

FastQC
http://www.bioinformatics.bbsrc.ac.uk/projects/fastqc/

R
http://www.r-project.org/

Sources of Data

- http://www.ebi.ac.uk/ena/data/view/SRS004748
- ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR022/SRR022825/SRR022825.fastq.gz
- ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR022/SRR022823/SRR022823.fastq.gz
- http://www.ebi.ac.uk/ena/data/view/SRS004748
- ftp://ftp.ensemblgenomes.org/pub/release-8/bacteria/fasta/Staphylococcus/s_aureus_mrsa252/dna/s_aureus_mrsa252.EB1_s_aureus_mrsa252.dna.chromosome.Chromosome.fa.gz
- ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR022/SRR022852/SRR022852_1.fastq.gz

- ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR022/SRR022852/SRR022852_2.fastq.gz
- ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR023/SRR023408/SRR023408_1.fastq.gz
- ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR023/SRR023408/SRR023408_2.fastq.gz
- ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR000/SRR000892/SRR000892.fastq.gz
- ftp://ftp.sra.ebi.ac.uk/vol1/fastq/SRR000/SRR000893/SRR000893.fastq.gz
- http://www.ebi.ac.uk/ena/data/view/SRX007709
- http://www.ebi.ac.uk/ena/data/view/SRX000181

Introduction

Some general introduction

Prepare the Environment

The first exercise should get you comfortable with the computer environment. I am going to assume that either you have some minimal experience with command line UNIX, or that the very short introduction I should have given you by now was sufficient. Please ask if neither of the above is true. It is not as tricky as it looks . . . honest!

First make sure that you are in your home directory by typing:

```
1 cd
```

and making absolutely sure you're there by typing:

```
1 pwd
```

Now create sub-directories for this and the two other velvet practicals. All these directories will be made as sub-directories of a directory for the whole course called NGS. For this you can use the following commands:

```
1 mkdir -p NGS/velvet/{part1,part2,part3}
```

The `-p` tells `mkdir` (make directory) to make any parent directories if they don't exist. You could create the directories one-at-a-time by doing:

```
1 mkdir NGS
2 mkdir NGS/velvet
3 mkdir NGS/velvet/part1
4 mkdir NGS/velvet/part2
5 mkdir NGS/velvet/part3
```

After creating the directories, examine the structure and move into the directory ready for the first velvet exercise by typing:

```
1 ls -R NGS
2 cd NGS/velvet/part1
3 pwd
```

Downloading and Compiling Velvet

For the duration of this workshop, all the software you require has been set up for you already. This might not be the case when you return to Şreal lifeŤ. Many of the programs you will need, including velvet, are quite easy to set up, it might be instructive to try a couple.

Although you will be using the preinstalled version of velvet, it is useful to know how to compile velvet as some of the parameters you might like to control can only be set at compile time. You can find the latest version of velvet at:

http://www.ebi.ac.uk/~zerbino/velvet/

You could go to this URL and download the latest velvet version, or equivalently, you could type the following, which will download, unpack, inspect, compile and execute your locally compiled version of velvet:

```
cd ~/NGS/velvet/part1
pwd
cp ~/NGS/Data/velvet_1.2.07.tgz ./
tar xzf ~/NGS/Data/velvet_1.2.07.tgz
ls -R
cd velvet_1.2.07
make velveth velvetg
./velveth
```

Take a look at the executables you have created. They will be displayed as green by the command:

```
ls --color=always
```

The switch `--color`, offensively misspelled in American!, instructs that files be coloured according to their type. This is often the default. Here we are just being cautious. The `=always` is included as colouring is usually suppressed in scripts. If you run this exercise using the script provided, just `--color` would not be enough. `--color=always` insists on file colouring even from a script.

Have a look of the output the command produces and you will see that `MAXKMERLENGTH=31` and `CATEGORIES=2` parameters were passed into the compiler.
This indicates that the default compilation was set for De Bruijn graph k-mers of maximum size 31 and to allow a maximum of just 2 read categories. You can override these, and other, default configuration choices using command line parameters. Assume, you want to run velvet with a k-mer length of 41 using 3 categories, velvet needs to be recompiled to enable this functionality by typing:

```
make clean
make velveth velvetg MAXKMERLENGTH=41 CATEGORIES=3;
./velveth
```

Discuss with the persons next to you the following questions:
What are the consequences of the parameters you have given make for velvet?
MAXKMERLENGTH: increase the max kmer length from 31 to 41
CATEGORIES: paired-end data require to be put into separate categories. By increasing this parameter from 2 to 3 allows you to process 3 paired / mate-pair libraries and unpaired data.
Why does Velvet use k-mer 31 and 2 categories as default? Possibly a number of reason:
- odd number to avoid palindromes
- The first reads were very short (20-40 bp) and there were hardly any paired-end data
around so there was no need to allow for longer kmer lengths / more categories.
- For programmers: 31 bp get stored in 64 bits (using 2bit encoding)
Should you get better results by using a longer k-mer length? If you can achieve a good kmer coverage - yes.

velvet can also be used to process SOLID colour space data. To do this you need a further make parameter. With the following command clean away your last compilation and try the following parameters:

```
make clean
make MAXKMERLENGTH=41 CATEGORIES=3 color
./velveth_de
```

What effect would the following compil-time parameters have on velvet:
`OPENMP=Y` Turn on multithreading
`LONGSEQUENCES=Y` Assembling reads / contigs longer than 32kb long
`BIGASSEMBLY=Y` Using more than 2.2 billion reads
`VBIGASSEMBLY=Y` Not documented yet
`SINGLE_COV_CAT=Y` Merge all coverage statistics into a single variable - save memory

For a further description of velvet compile and runtime parameters please see the velvet Manual: https://github.com/dzerbino/velvet/wiki/Manual

Assembling Single-end Reads

The data you will examine is from Staphylococcus aureus USA300 which has a genome of around 3MB . The reads are Illumina and are unpaired, also known as single-end library. Even though you have carefully installed velvet in your own workspace, we will use a pre-installed version.
The data needed for this section can be obtained from the Sequence Read Archive (SRA). For the following example use the run data SRR022825 and SRR022823 from the SRA Sample SRS004748. The SRA experiment could be viewed by setting your browser to the URL:

http://www.ebi.ac.uk/ena/data/view/SRS004748

We have already downloaded the data files for you.

The following exercise focuses on velvet using single-end reads, how the available parameters effect an assembly and how to measure and compare the changes.

To begin with, first move back to the directory you prepared for this exercise, create a new folder with a suitable name for this part and move into it. There is no need to download1 the read files, as they are already stored locally. Instead we will create soft links to the files. Continue by copying (or typing):

```
cd ~/NGS/velvet/part1
mkdir SRS004748
cd SRS004748
pwd
ln -s ~/NGS/Data/SRR022825.fastq.gz ./
ln -s ~/NGS/Data/SRR022823.fastq.gz ./
ls -l
```

You are ready to process your data with velvet, which is a compressed fastq file. Velvet has two main components:

velveth Used to construct, from raw read data, a dataset organised in the fashion expected by the second component, velvetg.

velvetg The core of velvet where the de Bruijn graph assembly is built and manipulated.

You can always get further information about the usage of both velvet programs by typing `velvetg` or `velveth` in your terminal.

Now run velveth for the reads in `SRR022825.fastq.gz` and `SRR022823.fastq.gz` using the following options:

- A de Bruijn graph k-mer of 25
- An output directory called run_25

```
velveth run_25 25 -fastq.gz -short SRR022825.fastq.gz SRR022823.fastq.gz
```

`velveth` talks to itself for a while and ends with some files in the output directory. Move into the output directory `run_25` and take a look around at what `velveth` had done so far. The UNIX command `less` allows you to look at output files (press `q` for quit). Just in case you still need a hint:

```
cd run_25
ls -l
head Sequences
cat Log
```

What did you find in the folder `run_25`? Sequences, Roadmaps, Log
Describe the content of the two `velveth` output files? Sequences: fasta file version of provided reads
Roadmaps: Internal file of velvet - basic information about reads, k-mer size
What does the `Log` file store for you? Time stamp, Executed commands; velvet version + compiler parameters, results

Now move one directory level up and run velvetg on your output directory, with the commands:

```
cd ../
time velvetg run_25
```

Move back into your results directory to examine the effects of velvetg:

```
cd run_22
ls -l
```

What extra files do you see in the folder `run_25`? PreGraph, Graph, stats.txt, contigs.fa, LastGraph
What do you suppose they might represent? PreGraph, Graph, LastGraph: Velvet internal graph representation at different stages (see manual for more details about the file format)
stats.txt: tab-delimited description of the nodes of the graph incl. coverage information
contigs.fa: assembly output file
In the Log file in `run_25`, what is the N50? 4409 bp

Hopefully, we will have discussed what the N50 statistic is by this point. Broadly, it is the median (not average) of a sorted data set using the length of a set of sequences. Usually it is the length of the contig whose length, when added to the length of all longer contigs,

makes a total greater that half the sum of the lengths of all contigs. Easy, but messy Ű a more formal definition can be found here:

http://www.broadinstitute.org/crd/wiki/index.php/N50

Backup the `contigs.fa` file and calculate the N50 (and the N25,N75) value with the command:

```
cp contigs.fa contigs.fa.0
gnx -min 100 -nx 25,50,75 contigs.fa
```

Does the value of N50 agree with the value stored in the Log file? No
If not, why do you think this might be? K-mer N50 vs bp N50; contig length cut-off value, estimated genome length

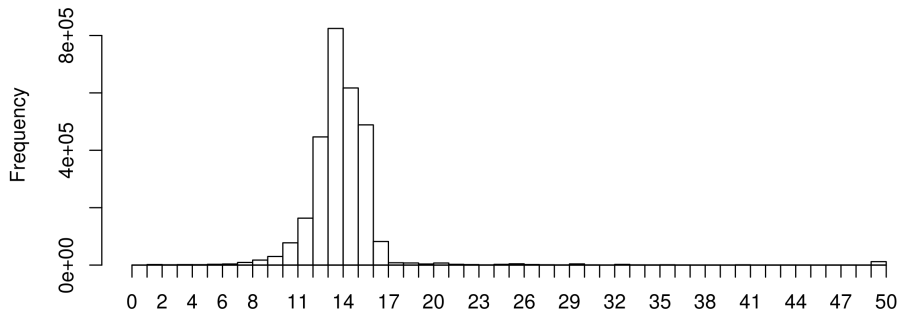
In order to improve our results, take a closer look at the standard options of `velvetg` by typing `velvetg` without parameters. For the moment focus on the two options `-cov_cutoff` and `-exp_cov`. Clearly `-cov_cutoff` will allow you to exclude contigs for which the k-mer coverage is low, implying unacceptably poor quality. The `-exp_cov` switch is used to give `velvetg` an idea of the coverage to expect.
If the expected coverage of any contig is substantially in excess of the suggested expected value, maybe this would indicate a repeat. For further details of how to choose the parameters, go to 'Choice of a coverage cutoff':

http://wiki.github.com/dzerbino/velvet/

Briefly, the k-mer coverage (and much more information) for each contig is stored in the file `stats.txt` and can be used with R to visualize the coverage distribution. Take a look at the `stats.txt` file, start R, load and visualize the data using the following commands:

```
R --no-save --no-restore
library(plotrix)
data <- read.table("stats.txt", header=TRUE)
weighted.hist(data$short1_cov, data$lgth, breaks=0:50)
```

A weighted histogram is a better way of visualizing the coverage information, because of noise (lots of very short contigs). You can see an example output below:



After choosing the expected coverage and the coverage cut-off, you can exit R by typing:

```
q()
```

The weighted histogram suggests to me that the expected coverage is around 14 and that everything below 6 is likely to be noise. Some coverage is also represented at around 20, 30 and greater 50, which might be contamination or repeats (depending on the dataset), but at the moment this should not worry you. To see the improvements, rerun velvetg first with `-cov_cutoff` 6 and after checking the N50 use only / add `-exp_cov` 14 to the command line option. Also keep a copy of the contigs file for comparison:

```
cd ~/NGS/velvet/part1/SRS004748
time velvetg run_25 -cov_cutoff 6

# Make a copy of the run
cp run_25/contigs.fa run_25/contigs.fa.1

time velvetg run_25 -exp_cov 14
cp run_25/contigs.fa run_25/contigs.fa.2

time velvetg run_25 -cov_cutoff 6 -exp_cov 14
cp run_25/contigs.fa run_25/contigs.fa.3
```

What is the N50 with no parameter: 4,447 bp
What is the N50 with `-cov_cutoff` 6: 5,168 bp
What is the N50 with `-exp_cov` 14: 4,903 bp
What is the N50 with `-cov_cutoff` 6 `-exp_cov` 14: 5,417 bp
Did you notice a variation in the time velvetg took to run? If so, can you explain why that might be? Velvet reuses already calculated results (from PreGraph,Graph)

You were running `velvetg` with the `-exp_cov` and `-cov_cutoff` parameters. Now try to experiment using different cut-offs, expected parameters and also explore other settings (e.g. `-max_coverage`, `-max_branch_length`, `-unused_reads`, `-amos_file`, `-read_trkg` or see `velvetg` help menu).

Make some notes about the parameters you've played with and the results you obtained. Please also comment on the `-max_coverage` and `-max_branch_length` parameters. -max_coverage: cut-off value for the upper range (like cov_cutoff for the lower range)
-max_branch_length: length of branch to look for bubble
-unused_reads: write unused reads into file
-amos_file: write amos file for further usage
-read_trkg: tracking read (more memory usage) - automatically on for certain operations

In particular, look at the `-amos_file` parameter which instructs `velvetg` to create a version of the assembly that can be processed and viewed with a program called AMOS Hawkeye. Another program, called tablet, can also understand and display the AMOS file. For now, we will take a look at Hawkeye.

Run velvetg with just `-cov_cutoff` 6 but requesting an amos file:

```
velvetg run_25 -cov_cutoff 6 -amos_file yes
```

Now convert the AMOS message file `velvet_asm.afg` into an AMOS bank and view the assembly using AMOS Hawkeye.

```
bank-transact -c -b run_25/velvet_asm.bnk -m run_25/velvet_asm.afg
hawkeye run_25/velvet_asm.bnk
```

Once the file has loaded in Hawkeye, select and display one of the longer contigs. Select the Contigs tab on the top of Hawkeye and take a closer look at the Contigs information. Note the lack of Read information and the one dimensional nature of the Contigs display. Close down Hawkeye when you have seen enough and delete the AMOS bank file.

```
rm -r run_25/velvet_asm.bnk
```

Now rerun velvetg adding the additional parameter `-exp_cov` 14 with no need to save any files as the AMOS file needs to be changed.

```
velvetg run_25 -cov_cutoff 6 -exp_cov 14 -amos_file yes
```

Again, convert the AMOS message file into an AMOS bank and view the assembly using Hawkeye or tablet:

```
bank-transact -c -b run_25/velvet_asm.bnk -m run_25/velvet_asm.afg
hawkeye run_25/velvet_asm.bnk
```

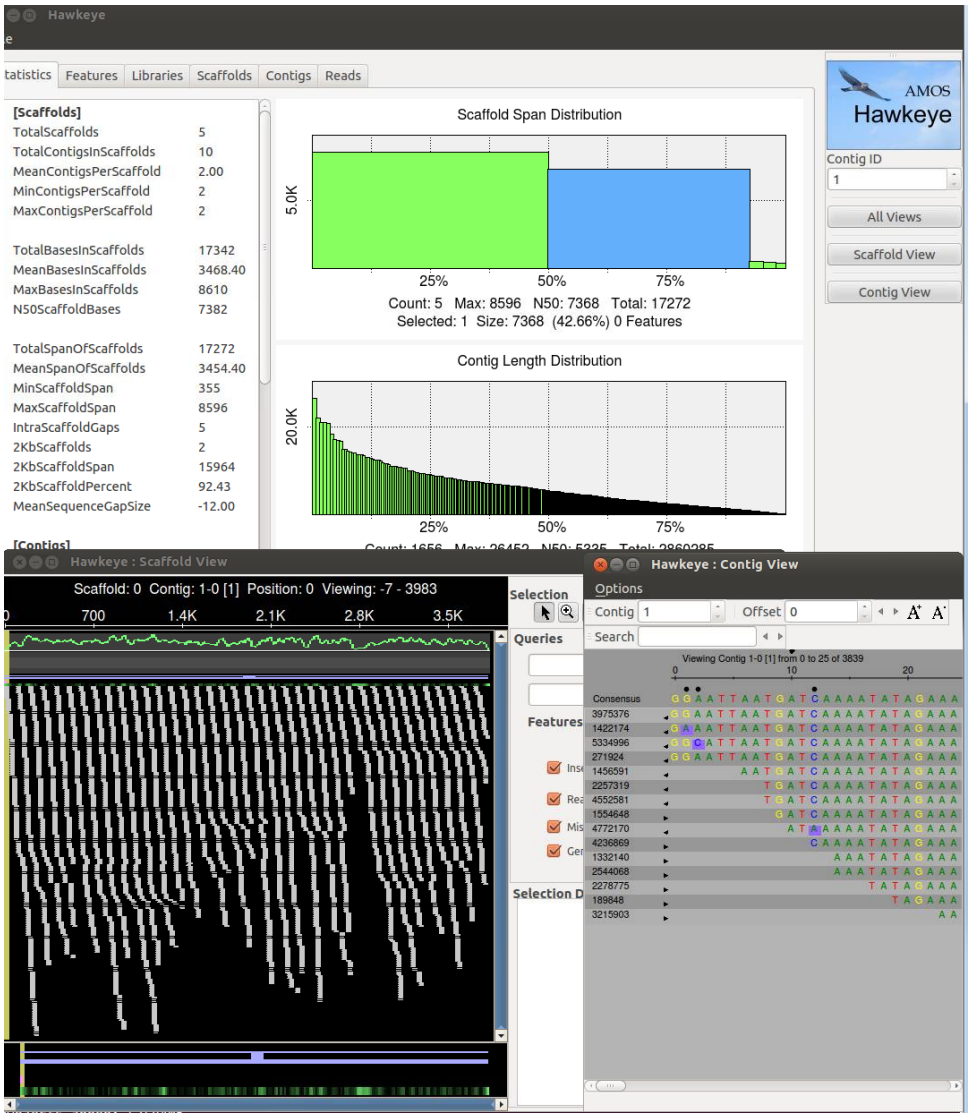


Figure 4: You should get a similar looking result with Hawkeye. If you are happy that you explored enough functionality, close down Hawkeye and continue with the exercise.

Why do you think there was read information only for the second use of Hawkeye and tablet? Read tracking was needed to provide the position information
You may have noticed `velvetg` took a little longer with `-exp_cov 14` on. Does this make sense? Reads are tracked automatically when using expected coverage parameter

If you want to explore the behaviour of velvet even further, you could experiment with the following.
Reduce the sequence coverage by only choosing one input file for the assembly e.g.

`SRR022825.fastq.gz`
Increase the sequence coverage by downloading further files from the same ENA sample SRS004748 http://www.ebi.ac.uk/ena/data/view/SRS004748.

How did the N50 change by a) reducing the coverage and b) increasing the coverage: Reducing the coverage: N50 reduced
Increasing the coverage: N50 increased

Simple Assembly Simulation

The data for this section is from Staphylococcus aureus MRSA252, a genome closely related to the genome that provided the short read data in the earlier sections of this exercise. The sequence data this time is the fully assembled genome. The genome size is therefore known exactly and is 2,902,619 bp.

In this exercise you will process the single whole genome sequence with velveth and velvetg, look at the output only and go no further. The main intent of processing this whole genome is to compute its N50 value. This must clearly be very close to the ideal N50 value for the short reads assembly and so aid evaluation of that assembly.

To begin, move back to the main directory for this exercise, make a sub-directory for the processing of this data and move into it. All in one go, this would be:

```
cd ~/NGS/velvet/part1/
mkdir MRSA252
cd MRSA252
```

Next, you need to download the genome sequence from
http://www.ensemblgenomes.org/ which holds five new sites, for bacteria, protists, fungi, plants and invertebrate metazoa. You could browse for the data you require or use

the file which we have downloaded for you. For the easier of these options, make and check a soft link to the local file and with the commands:

```
ln -s ~\
   /NGS/Data/s_aureus_mrsa252.EB1_s_aureus_mrsa252.dna.chromosome.Chromosome.fa.gz \
   ./
ls -l
```

Usually Velvet expects relatively short sequence entries and for this reason has a read limit of 32,767 bp per sequence entry. As the genome size is 2,902,619 bp - longer as the allowed limit and does not fit with the standard settings into velvet. But like the maximum k-mer size option, you can tell Velvet during compile time, using `LONGSEQUENCES=Y`, to expect longer input sequences than usual. I already prepared the executable which you can use by typing `velveth_long` and `velvetg_long`.

Now, run `velveth_long`, using the file you either just downloaded or created a soft link to as the input:

```
velveth_long run_25 25 -fasta.gz -long \
   s_aureus_mrsa252.EB1_s_aureus_mrsa252.dna.chromosome.Chromosome.fa.gz
velvetg_long run_25
```

What is the N50? 24,142 bp
How does the N50 compare to the previous single end run (SRS004748)? Big difference

Does the total length differ from the input sequence length? 2,817,181 (stats) vs 2,902,619 (input)
What happens when you rerun velvet with a different k-mer length? K-mer 31: N50: 30,669 bp, total 2,822,878

Assembling Paired-end Reads

The data you will examine in this exercise is again from Staphylococcus aureus which has a genome of around 3MB. The reads are Illumina paired end with an insert size of 350 bp. The required data can be downloaded from the SRA. Specifically, the run data (SRR022852) from the SRA Sample SRS004748.

http://www.ebi.ac.uk/ena/data/view/SRS004748

The following exercise focuses on preparing the paired-end FASTQ files ready for velvet, using velvet in paired-end mode and comparing results with velvet's 'auto' option.

First move to the directory you made for this exercise and make a suitable named directory for the exercise:

```
cd ~/NGS/velvet/part2
mkdir SRS004748
cd SRS004748
```

There is no need to download1 the read files, as they are already stored locally. You will simply create a soft link to this pre-downloaded data using the following commands:

```
ln -s ~/NGS/Data/SRR022852_?.fastq.gz ./
```

It would be interesting to monitor the way that the programs you will run utilize your computer's resources, particularly memory. A simple way to do this is to open a second terminal and in it type:

```
top
```

`top` is a program that continually monitors all the processes running on your computer, showing the resources used by each. Alternatively you could use a system monitor from your operating system as well. Leave this running and refer to it at intervals, especially when programs appear to be taking a long time, of just whenever your curiosity gets the better of you. You should find that as this practical progresses, memory usage will increase significantly, but hopefully not beyond the capacity of your workstation.
Now, back to the first terminal, you are ready to run `velveth` and `velvetg`. The reads are `-shortPaired` this time and for the first run you should not use any parameters for `velvetg`.

From this point on, where it will be informative, time your runs. This is very easy to do, just prefix the command to run the program with the command `time`. This will cause UNIX to report how long the program took to complete its task.

Set the two stages of velvet running, whilst you watch the memory usage as reported by `top`. time the `velvetg` stage. The commands to enter are:

```
velveth run_25 25 -fmtAuto -create_binary -shortPaired -separate \
    SRR022852_1.fastq.gz SRR022852_2.fastq.gz
time velvetg run_25
```

What does `-fmtAuto` and `-create_binary` do? (see help menu) TODO
Comment on the use of memory and CPU for `velveth` and `velvetg`? `velveth` uses only one CPU while `velvetg` uses all possible CPUs for some parts of the calculation.

How long did `velvetg` take? My own measurements on an EBI machine are:
`real 1m8.877s; user 4m15.324s; sys 0m4.716s`

Next, after saving your `contigs.fa` file from being overwritten, set the cut-off parameters that you investigated in the previous exercise and rerun `velvetg`. time and monitor the use of resources as previously. Start with `-cov_cutoff 16` thus:

```
mv run_25/contigs.fa run_25/contigs.fa.0
time velvetg run_25 -cov_cutoff 16
```

Up until now, `velvetg` has ignored the paired-end information. Now try running `velvetg` with both `-cov_cutoff 16` and `-exp_cov 26`, but first save your `contigs.fa` file. By using `-cov_cutoff` and `-exp_cov`, `velvetg` tries to estimate the insert length, which you will see in the `velvetg` output. The command is, of course:

```
mv run_25/contigs.fa run_25/contigs.fa.1
time velvetg run_25 -cov_cutoff 16 -exp_cov 26
```

Comment on the time required, use of memory and CPU for `velvetg`? Runtime is lower when velvet can reuse previously calculated data. By using `-exp_cov`, the memory usage increases.
Which insert length does Velvet estimate? Paired-end library 1 has length: 228, sample standard deviation: 26

Next try running `velvetg` in 'paired-end mode'. This entails running `velvetg` specifying the insert length with the parameter `-ins_length` set to 350. Even though velvet estimates the insert length it is always advisable to check / provide the insert length manually as velvet can get the statistics wrong due to noise. Just in case, save your last version of `contigs.fa`. The commands are:

```
mv run_25/contigs.fa run_25/contigs.fa.2
time velvetg run_25 -cov_cutoff 16 -exp_cov 26 -ins_length 350
mv run_25/contigs.fa run_25/contigs.fa.3
```

How fast was this run? Down to my own measurements on an EBI machine:
`real 0m29.792s; user 1m4.372s; sys 0m3.880s`

Take a look into the Log file.

What is the N50 value for the `velvetg` runs using the switches:
Base run: 19,510 bp `-cov_cutoff 16` 24,739 bp
`-cov_cutoff 16 -exp_cov 26` 61,793 bp
`-cov_cutoff 16 -exp_cov 26 -ins_length 350` n50 of 62,740 bp; max 194,649 bp; total 2,871,093 bp

Try giving the `-cov_cutoff` and/or `-exp_cov` parameters the value `auto` - the `velvetg` help output could show you how. The information Velvet prints during running includes information about the values used (coverage cut-off or insert length) when using the `auto` option.

Which coverage values does Velvet choose (hint: look at the output which velvet produces while running)? Median coverage depth = 26.021837
Removing contigs with coverage ¡ 13.010918 . . .
How does the N50 value change? n50 of 68,843 bp; max 194,645 bp; total 2,872,678 bp

Run `gnx` on all the `contig.fa` files you have generated in the course of this exercise. The command will be:

```
1 gnx -min 100 -nx 25,50,75 run_25/contigs.fa*
```

For which runs are there Ns in the `contigs.fa` file and why? contigs.fa.2, contigs.fa.3, contigs.fa
Velvet tries to use the provided (or infers) the insert length and fills ambiguous regions with Ns.
Comment on the number of contigs and total length generated for each run.

Filename Total length	# contigs # Ns
Contigs.fa.0 2,830,659	631 0
Contigs.fa.1 2,832,670	580 0
Contigs.fa.2 2,849,919	166 4,847
Contigs.fa.3 2,856,795	166 11,713
Contigs.fa 2,857,439	163 11,526

Similar to the single-end assembly you created previously, we can output a paired-end assembly as an AMOS message format file. This file can then be converted into an AMOS bank and viewed using AMOS Hawkeye.

```
1 time velvetg run_25 -cov_cutoff 16 -exp_cov 26 -ins_length 350 \
     -amos_file yes -read_trkg yes
2 time bank-transact -c -b run_25/velvet_asm.bnk -m \
     run_25/velvet_asm.afg
3 hawkeye run_25/velvet_asm.bnk
```

Looking at the scaffold view of a contig, comment on the proportion of happy mates to compressed mates. TODO
What is the mean and standard deviation of the insert size report in the Libraries tab of Hawkeye? TODO
Look at the actual distribution of insert sizes for this library. Can you explain where there is a difference between the mean and SD reported in those two places? TODO

You can get AMOS to re-estimate the mean and SD of insert sizes. First, close Hawkeye and then run the following commands before reopening the AMOS bank to see what has changed.

```
asmQC -b run_25/velvet_asm.bnk -scaff -recompute -update -numsd 2
hawkeye run_25/velvet_asm.bnk
```

Looking at the scaffold view of a contig, comment on the proportion of happy mates to compressed mates. TODO
What is the mean and standard deviation of the insert size report in the Libraries tab of Hawkeye? TODO
Look at the actual distribution of insert sizes for this library. Does the mean and SD reported in each of those two places now match? TODO

Data Quality

As discussed previously, fastq format files include quality assessment of each called base. As also mentioned earlier, velvet does not use this information directly. However, by taking into account the quality of the read data, it should be possible to both improve the efficiency (in terms of memory usage and runtime) and the quality of the output.

To investigate the effect of data quality, we will use the run data (SRR023408) from the SRA experiment SRX008042. The reads are Illumina paired end with an insert size of 92 bp.

As ever, move up a directory to the main directory for the whole of this exercise and create and enter a new directory dedicated to this phase of the exercise. The commands are:

```
cd ~/NGS/velvet/part2
mkdir SRX008042
cd SRX008042
```

Create soft links to the read data files we downloaded for you from the SRA with the command:

```
ln -s ~/NGS/Data/SRR023408_?.fastq.gz ./
```

To look at the quality scores, you could use a tool called FastQC to process and present the scores in a compact manner of basic statistics. FastQC can be downloaded from:
http://www.bioinformatics.bbsrc.ac.uk/projects/fastqc/
Where there are links to an excellent manual and an even more excellent youtube video. I

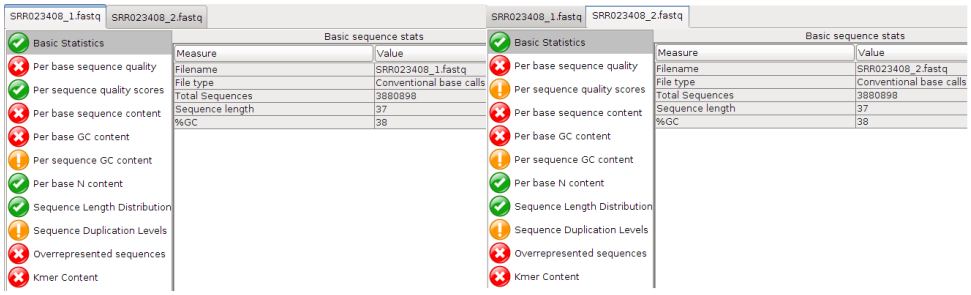
hope to show you the video and recommend you keep the manual pages open whilst you are using FastQC in this exercise.

Start up FastQC Graphical User Interface (GUI) to examine your compressed fastq files by typing:

```
1 fastqc &
```

The `&` sets FastQC running as a background job, thus freeing your terminal for further commands if required. FastQC emerges as a big blank window.

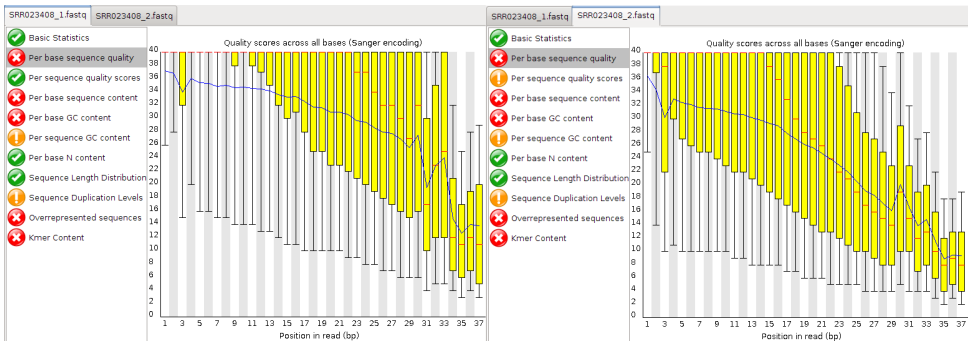
Open both your compressed fastq files as was done in the video (Choose Open from the File pull down menu, browse to your files, select them both and click OK). Look at tabs for both files:



Are the quality scores the same for both files? Overall yes
Which value varies? Per sequence quality scores
Take a look at the Per base sequence quality for both files. Did you note that it is not good for either file? The quality score of both files drop very fast, the REV strand faster than the FWD strand
At which positions would you cut the reads off? Looking at the ŞPer base qualityŤ and ŞPer base sequence content, both around position 27
Why does the quality deteriorate towards the end of the read? Errors more likely for later cycles
Does it make sense to trim the start? Looking at the ŞPer base sequence contentŤ, yes - there is a clear signal at the beginning.

Have a look at the other options that fastqc offers. Make good use of the Help to remind you of what you learned from the video. I suggest you open the Help window and keep it on your screen, open at the relevant page, as you browse through the fastqc possibilities.

Which other statistics could you use to support your trimming strategy? ŞPer base sequence contentŤ, ŞPer base GC contentŤ, ŞKmer contentŤ, ŞPer base sequence qualityŤ



Once you have decided how it might be sensible to trim your reads, close down fastqc by selecting Exit from the File pull down menu. Now apply the knife to your reads with `fastx_trimmer` from the FASTX-Toolkit. However, you will first need to decompress the FASTQ files. The usage of the tool can be found here:
http://hannonlab.cshl.edu/fastx_toolkit/commandline.html#fastx_trimmer_usage
The suggestion (hopefully not far from your own thoughts?) is that you trim your reads as follows:

```
gunzip < SRR023408_1.fastq.gz > SRR023408_1.fastq
gunzip < SRR023408_2.fastq.gz > SRR023408_2.fastq
fastx_trimmer -Q 33 -f 4 -l 32 -i SRR023408_1.fastq -o \
    SRR023408_trim1.fastq
fastx_trimmer -Q 33 -f 3 -l 29 -i SRR023408_2.fastq -o \
    SRR023408_trim2.fastq
```

Now run velveth with a k-mer value of 21 for both the untrimmed and trimmed read files in `-shortPaired` mode. Separate the output of the two executions of `velveth` into suitably named directories, followed by `velvetg`. The commands would be:

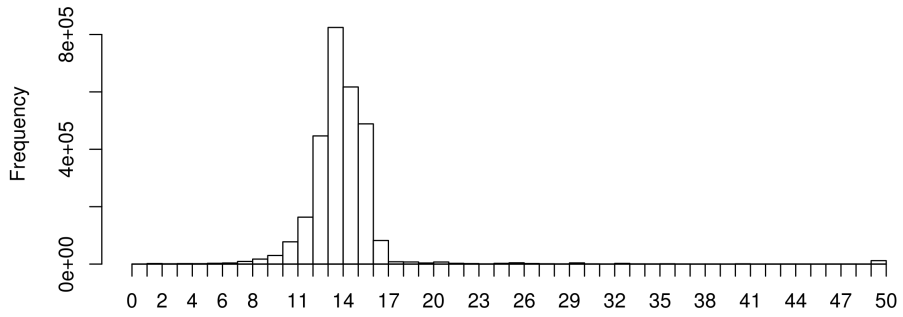
```
velveth run_21 21 -fmtAuto -create_binary -shortPaired -separate \
    SRR023408_1.fastq SRR023408_2.fastq
time velvetg run_21

velveth run_21trim 21 -fmtAuto -create_binary -shortPaired -separate \
    SRR023408_trim1.fastq SRR023408_trim2.fastq
time velvetg run_21trim
```

What are the times for the two `velvetg` runs? run_25: `real 3m16.132s; user 8m18.261s; sys 0m7.317s`
run_25trim: `real 1m18.611s; user 3m53.140s; sys 0m4.962s`
What N50 scores did you achieve? 11 (untrimmed), 115 (trimmed)
What were the overall effects of trimming? Time saving, increased N50, reduced coverage

The evidence is that trimming improved the assembly. The thing to do surely, is to run velvetg with the `-cov_cutoff` and `-exp_cov`. In order to use `-cov_cutoff` and `-exp_cov` sensibly, you need to investigate with R, as you did in the previous exercise, what parameter values to use. To start up R and produce the weighted histograms, type:

```
R --no-save
library(plotrix)
data <- read.table("run_21/stats.txt", header=TRUE)
data2 <- read.table("run_21trim/stats.txt", header=TRUE)
x11()
par(mfrow=c(1,2))
weighted.hist(data$short1_cov, data$lgth, breaks=0:50)
weighted.hist(data2$short1_cov, data2$lgth, breaks=0:50)
```



One histogram suggests to me an expected coverage of around 13 with a coverage cut-off of around 7. The trimmed histogram instead suggests an expected coverage of around 9 with a coverage cut-off of around 5. If you disagree, feel free to try different settings, but first leave R before running velvetg:

```
q()

time velvetg run_21 -cov_cutoff 7 -exp_cov 13 -ins_length 92
time velvetg run_21trim -cov_cutoff 5 -exp_cov 9 -ins_length 92
```

How good does it look now?
Still not great Comment on:
Runtime Reduced runtime
Memory Lower memory usage
k-mer choice (Can you use kmer 31 for a read of length 30 bp?) K-mer has to be lower than the read length and the Kmer coverage should be sufficient to produce results.
Does less data mean 'worse' results? Not necessarily. If you have lots of data you can savely remove poor data without affecting overall coverage.
How would a smaller/larger Kmer size behave? TODO

Compare the results, produced during the last exercises, with each other:

Metric	SRR022852	SRR023408	SRR023408.trimmed
Overall Quality (1-5)			
bp Coverage			
k-mer Coverage			
N50 (k-mer used)			

Metric	SRR022852	SRR023408	SRR023408.trimmed
Overall Quality (1-5)	2	5	4
bp Coverage	136 x (36 bp;11,374,488)	95x (37bp; 7761796)	82x (32bp; 7761796)
k-mer Coverage	45x	43x (21); 33x (25)	30x (21); 20.5x (25)
N50 (k-mer used)	68,843 (25)	2,803 (21)	2,914 (21)

What would you consider as the 'best' assembly? SRR022852
If you found a candidate, why do you consider it as 'best' assembly? Overall data quality and coverage

Assembling Long (454) Read

The data you will examine in this exercise is again from Staphylococcus aureus which has a genome of around 3MB. The reads are 454 single end.
The required data can be downloaded from the SRA. Specifically, the run data (SRR000892,SRR000893) from the SRA Experiment SRX000181.

http://www.ebi.ac.uk/ena/data/view/SRX000181

The following exercise focuses on processing 454 long reads with velvet and how this differs compared to short reads.

First move to the directory you made for this exercise and make a suitable named directory for the exercise before downloading the read files:

```
cd ~/NGS/velvet/part3
mkdir SRX000181
cd SRX000181
```

The downloaded files can be used directly in velvet. To let velvet know that these fastq files are long reads, you pass in the parameter `-long` by using the commands:

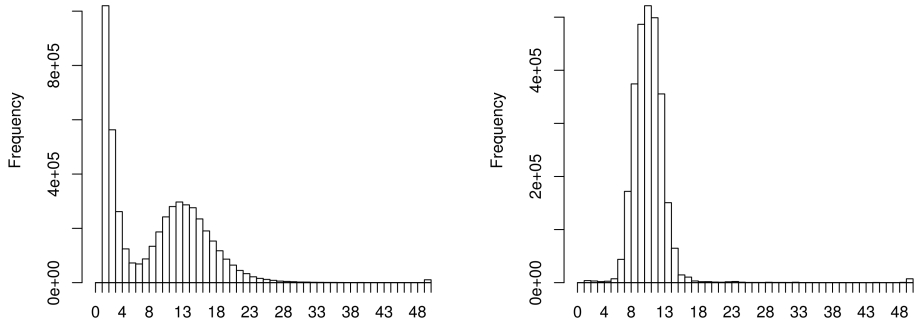
```
ln -s ~/NGS/Data/SRR000892.fastq.gz
ln -s ~/NGS/Data/SRR000893.fastq.gz
velveth run_25 25 -create_binary -fastq.gz -long *.fastq.gz
time velvetg run_25
```

Take a look at the stats.txt file. Which columns are used compared to short reads and why? long_cov instead of e.g. short1_cov
Which N50 do you get? 28 (k-mer N50)
How long did the velvetg run take? `real 0m36.333s; user 2m33.238s; sys 0m3.893s`

The right thing to do is to run velvetg setting the cut-offs. But for long reads there is an option called `-long_cov_cutoff` to filter them independently because of the difference in usage in velvet. To investigate with R, as you did in the previous exercises, start up R and produce the weighted histogram using the column `long_cov` by typing:

```
R --no-save
```

```
library(plotrix)
data <- read.table("run_25/stats.txt", header=TRUE)
x11()
weighted.hist(data$long_cov, data$lgth, breaks=0:50)
```



For me the histogram suggests to me to choose a coverage cut-off of around 9 with an expected coverage of about 19. If you disagree, feel free to try different settings, but first leave R before running `velvetg` with the coverage parameters typing:

```
q()

cp run_25/contigs.fa run_25/contigs.fa.0

time velvetg run_25 -long_cov_cutoff 9
cp run_25/contigs.fa run_25/contigs.fa.1

time velvetg run_25 -long_cov_cutoff 9 -exp_cov 19
cp run_25/contigs.fa run_25/contigs.fa.2

gnx -min 100 -nx 25,50,75 run_25/contigs.fa.*
```

What is the N50 and runtime using:
`-long_cov_cutoff 9` `5,747; real 0m11.457s; user 0m10.855s; sys 0m0.329s`
`-long_cov_cutoff 9 -exp_cov 19` `16,781; real 0m29.830s; user 2m32.530s; sys 0m3.523s`
other runs?
Which other parameters could improve the assembly quality for long reads? `-conserveLong`

What do you think about assembling 454 reads with Velvet? It's working!

Assembling Mixed Insert Length Libraries

Like the previous examples, the data you will examine in this exercise is again from Staphylococcus aureus which still has a genome of around 3MB. The reads are Illumina paired end with an insert size of 170 bp and 350 bp.
You already downloaded the required reads from the SRA in previous exercises. Specifically, the run data (SRR022863, SRR022852) from the SRA Study SRP001086.

http://www.ebi.ac.uk/ena/data/view/SRP001086

The following exercise focuses on handing two insert length libraries with velvet and the changes you have to look out for.

First move to the directory you made for this exercise, make a suitable named directory for the exercise and check if all the files are in place:

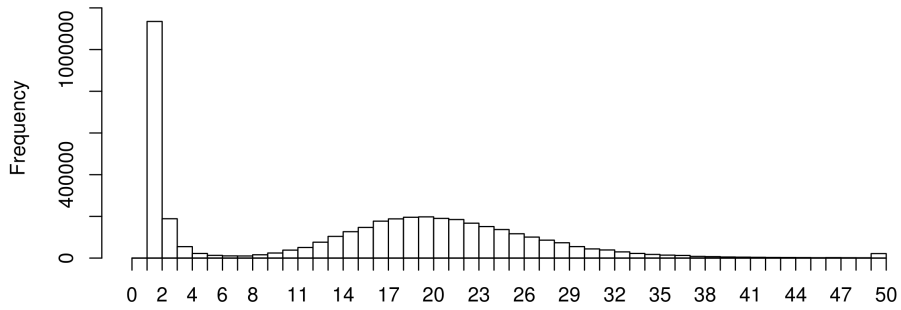
```
cd ~/NGS/velvet/part3
mkdir SRP001086
cd SRP001086
ln -s ~/NGS/Data/SRR022863_?.fastq.gz ./
ln -s ~/NGS/Data/SRR022852_?.fastq.gz ./
```

Now run `velveth` and `velvetg` using the appropriate command line options by typing:

```
time velveth run_25 25 -fmtAuto -create_binary -shortPaired -separate \
    SRR022863_1.fastq.gz SRR022863_1.fastq.gz -shortPaired2 -separate \
    SRR022852_1.fastq.gz SRR022852_2.fastq.gz
time velvetg run_25
```

The right thing to do is to run velvetg setting the cut-offs. To investigate with R, as you did in the previous exercises, start up R and produce the weighted histogram using the columns `short1_cov` and `short2_cov` by typing:

```
R --no-save
library(plotrix)
data <- read.table("run_25/stats.txt", header=TRUE)
weighted.hist(data$short1_cov+data$short2_cov, data$lgth, breaks=0:70)
```



For me the histogram suggests to me to choose a coverage cut-off of around 20 with an expected coverage of about 36. If you disagree, feel free to try different settings, but first leave R before running velvetg with the coverage parameters typing:

```
q()

cp run_25/contigs.fa run_25/contigs.fa.0

time velvetg run_25 -cov_cutoff 20
cp run_25/contigs.fa run_25/contigs.fa.1

time velvetg run_25 -cov_cutoff 20 -exp_cov 36
cp run_25/contigs.fa run_25/contigs.fa.2

time velvetg run_25 -cov_cutoff 20 -exp_cov 36 -ins_length 170 \
   -ins_length2 350
cp run_25/contigs.fa run_25/contigs.fa.3

gnx -min 100 -nx 25,50,75 run_25/contigs.fa.*
```

What N50 did you get? 61,644 bp
How did the runtime/results compare to one paired-end library runs? More memory, longer runtime, (in this case) similar results
How many different libraries would you be able to run with this velvet version? 2 paired-end (or mate-pair) + one single end
Would you be able to add a single-end library as well with this velvet version? Yes

Find and download a different insert length library [1] from the study SRP001086 and recompile velvet to allow the use of three insert length libraries. Maybe you could use the library you trimmed during previous exercises. You should be able to find these files here: `/NGS/velvet/part2/SRX008042/SRR023408_trim?.fastq`. If you don't still have these files, you can find a copy of them here: `/NGS/Data/SRR023408_trim?.fastq` Use the fresh compiled Velvet version with the three (two provided and one downloaded library) to assemble the genome.

Does the extra library make any difference? TODO
How does the overall coverage change? TODO
Any other comments? TODO

[a]Paired insert lengths can be found on the NCBI SRA page in the library section (Nominal length) e.g. http://www.ncbi.nlm.nih.gov/sra?term=SRR022866

Hybrid Assembly

Like the previous examples, the data you will examine in this exercise is again from Staphylococcus aureus which has a genome of around 3MB. The reads are 454 single end and Illumina paired end with an insert size of 170 bp. You already downloaded the required reads from the SRA in previous exercises. Specifically, the run data (SRR022863, SRR000892, SRR000893) from the SRA experiments SRX007709 and SRX000181.
The following exercise focuses on handing 454 long reads and paired-end reads with velvet and the differences in setting parameters.

First move to the directory you made for this exercise, make a suitable named directory for the exercise and check if all the three files are in place:

```
1 cd ~/NGS/velvet/part3
2 mkdir SRR000892-SRR022863
3 cd SRR000892-SRR022863
4 ln -s ~/NGS/Data/SRR00089[2-3].fastq.gz ./
5 ln -s ~/NGS/Data/SRR022863_?.fastq.gz ./
```

The following command will run for a LONG time. This indicated the amount of calculations being preformed by Velvet to reach a conclusion. To wait for velvet to finish would exceed the time available in this workshop, but it is up to you to either let it run over night or kill the process by using the key combination `CTRL+c`.

```
velveth run_25 25 -fmtAuto -create_binary -long \
    SRR00089?.fastq.gz -shortPaired -separate \
    SRR022863_1.fastq.gz SRR022863_2.fastq.gz
time velvetg run_25
```

If you have decided to continue, we already inspected the weighted histograms for the short and long read library separately, you can reuse this for the cut-off values:

```
time velvetg run_25 -cov_cutoff 7 -long_cov_cutoff 9
```

What are your conclusions using velvet in an hybrid assembly? 17 min: time velvetg run_25