An introduction to genome assemblies

Sebastian Schmeier

Institute of Natural and Mathematical Sciences
Massey University Auckland, New Zealand
http://sschmeier.com
s.schmeier@gmail.com

2016-03-16

Contents

An introduction to genome assemblies

1.0 Preface

In this genome assembly tutorial we will use our skill on the command-line interface to deal with the task of a genome assembly. You will learn how to judge the sequencing quality of a next generation sequencing (NGS) run and assemble the genome based on a collection of short sequences.

You will encounter some **To-do** sections at times. Write the solutions and answers into a text-file.

A PDF-version of this tutorial can be downloaded here or at http://sschmeier.com/bioinf-workshop/genome-assembly/doc/GenomeAssembly_sschmeier.pdf

There are two accompanying lectures for this tutorial:

- **Next-generation sequencing and quality control: An introduction** is available at figshare (https://dx.doi.org/10.6084/m9.figshare.2972320.v1).
- **Genome assembly: An Introduction** available at figshare (https://dx.doi.org/10.6084/m9.figshare.2972323.v1).

1.1 Learning outcomes

After studying this tutorial you should be able to:

1. Compute, investigate and evaluate the quality of sequence data from a sequencing experiment.
2. Describe the steps involved in pre-processing/cleaning sequencing data.
3. Distinguish between a good and a bad sequencing run.
4. Compute, investigate and evaluate the quality of sequence data from a sequencing experiment.
5. Compute, interpret and evaluate a whole genome assembly.
6. Judge the quality of a genome assembly.

1.2. The data

1.2.1 Get the data

You can download the data-file here (or at http://sschmeier.com/bioinf-workshop/genome-assembly/data/data_paired.fastq.gz). I will also bring the data on a USB drive, please copy it onto your system should the download not work.

The data is a down-sampled (randomly selected) small portion of the original sequencing data-set. This has been done because the amount of data produced was too high for the exercises today. Also, the original data was paired-end data (see *Figure 1*), thus we had two files, one for each end of the read. However, our assembler expects an read-interleaved file, which has been already created to make things easier.
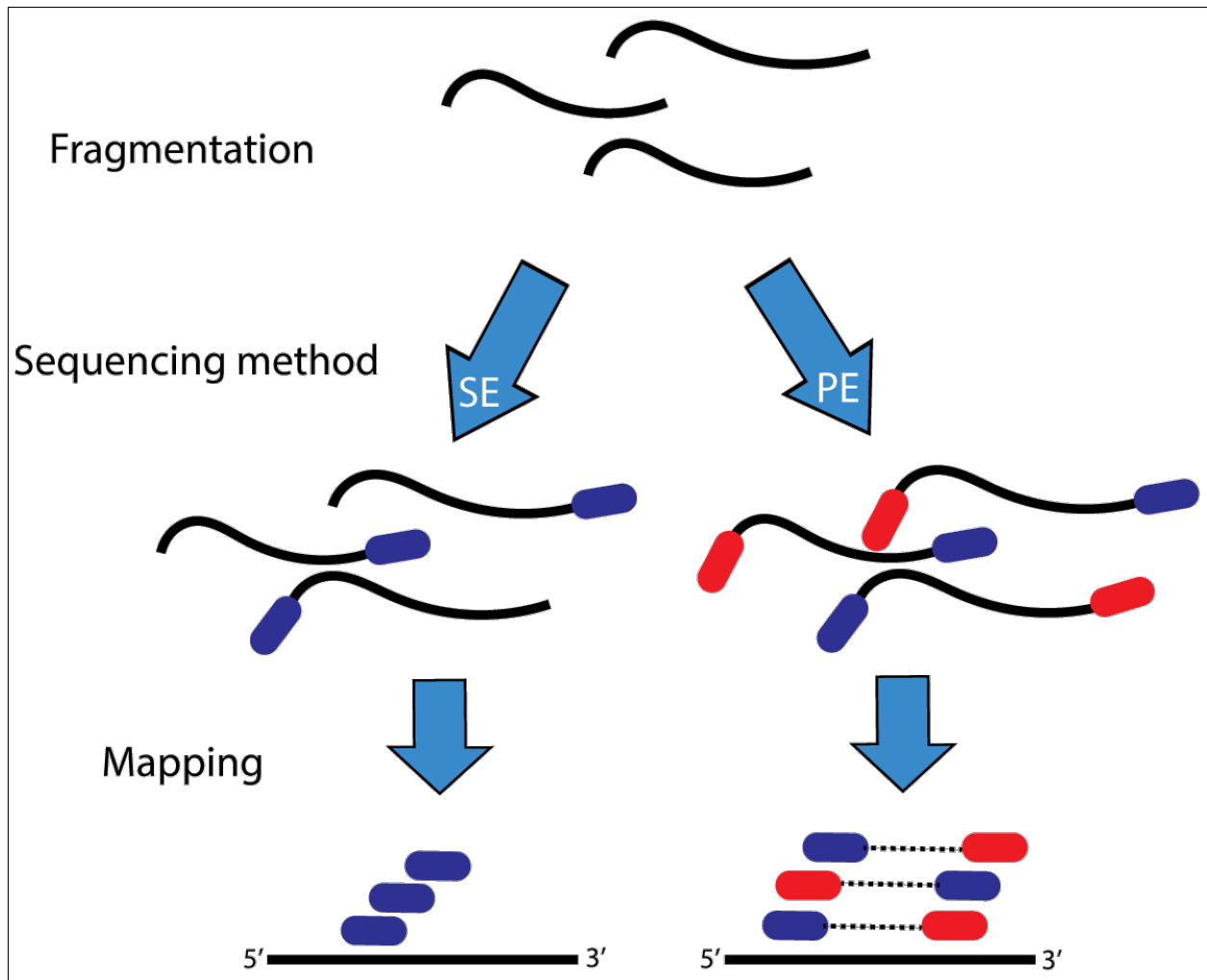
Figure 1: Illustration of single-end versus paired-end sequencing.

If you need to refresh how Illumina paired-end sequencing works have a look at the Illumina webpage (http://www.illumina.com/technology/next-generation-sequencing/paired-end-sequencing_assay.html) and this video at https://youtu.be/HMyCqWhwB8E.

1.2.2 Investigate the data

Make use of your newly developed skills on the command-line to investigate the files in two folders.

To-do

1. Unzip the data using `gzip`.
2. Use the command-line to get some ideas about the file.
3. What kind of files are we dealing with?
4. How many sequence reads are in the file?

1.3 Quality assessment of sequencing reads (SolexaQA++)

To assess the sequence read quality of the Illumina run we make use of a program called SolexaQA (http://solexaqa.sourceforge.net/) [Cox2010]. This was originally developed to work with Solexa data (since bought by

Illumina), but long since working with Illumina data. It produces nice graphics that intuitively show the quality of the sequences. it is also able to dynamically trim the bad quality ends off the reads.

From the webpage:

> "SolexaQA calculates sequence quality statistics and creates visual representations of data quality for second-generation sequencing data. Originally developed for the Illumina system (historically known as"Solexa"), SolexaQA now also supports Ion Torrent and 454 data."

1.3.1 Download/install SolexaQA++

Download SolexaQA from the developer webpage here.

Note! Emergency link here or at http://sschmeier.com/bioinf-workshop/genome-assembly/apps/SolexaQA++_v3.1.3.zip.

```
# unzip the archive
$ unzip SolexaQA++_v3.1.3.zip
# make the file executable
$ chmod a+x SolexaQA++_v3.1.3/Linux_x64/SolexaQA++
# copy the executable to my working directory
$ cp ./SolexaQA++_v3.1.3/Linux_x64/SolexaQA++ .
# run the program
$ ./SolexaQA++
# should now run the program
```

1.3.2 Understand SolexaQA++

SolexaQA++ has three modes that can be run. Type:

```
$ ./SolexaQA++
```

```
SolexaQA++ v3.1.3
Released under GNU General Public License version 3
C++ version developed by Mauro Truglio (M.Truglio@massey.ac.nz)

Usage: SolexaQA++ <command> [options]

Command: analysis      quality analysis and graphs generation
         dynamictrim    trim reads using a chosen threshold
         lengthsort  sort reads by a chosen length
```

The three modes are: `analysis`, `dynamictrim`, and `lengthsort`

`analysis` - the primary quality analysis and visualization tool. Designed to run on unmodified FASTQ files obtained directly from Illumina, Ion Torrent or 454 sequencers.

`dynamictrim` - a read trimmer that individually crops each read to its longest contiguous segment for which quality scores are greater than a user-supplied quality cutoff.

`lengthsort` - a program to separate high quality reads from low quality reads. LengthSort assigns trimmed reads to paired-end, singleton and discard files based on a user-defined length cutoff.

1.3.3 Run SolexaQA++ analysis on untrimmed data

To-do

1. Create a directory for the result-files –> **qa_untrimmed/**.
2. Run SolexaQA++ with the untrimmed data, and submit result-directory **qa_untrimmed/**.
3. Investigate the result-files in **qa_untrimmed/**.
4. Compare your results to these examples of a particularly bad MiSeq run (*Figure 2-5*, taken from http://solexaqa.sourceforge.net/). Write down your observations.
5. What elements in these example figures (*Figure 2-5*) show that we are dealing with a bad run? Write down your explanations.

Hint! Should you not get 1 and/or 2 right, try these commands here or at http://sschmeier.com/bioinf-workshop/genome-assembly/code/solexaqa1.txt.
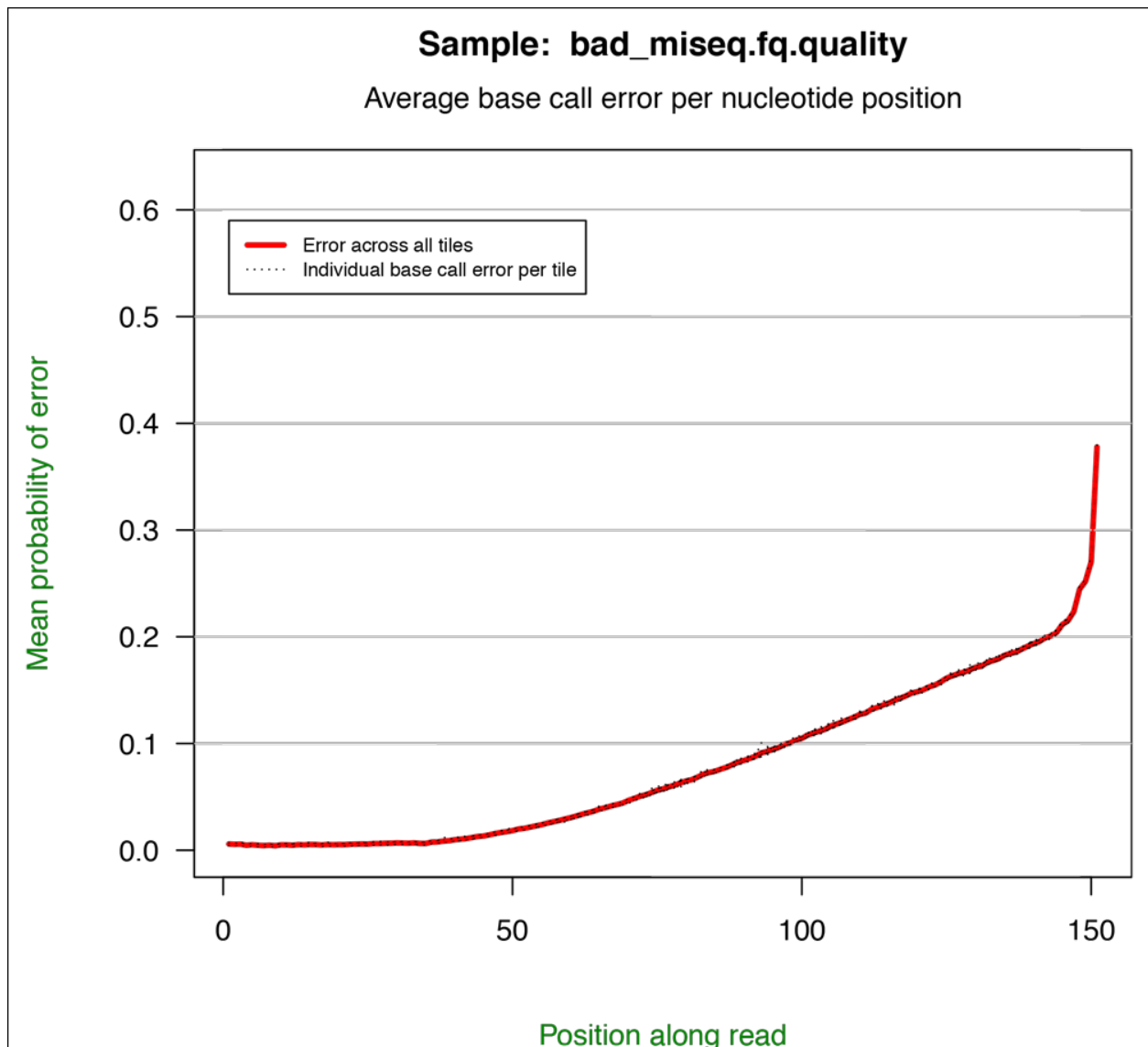

Figure 2: SolexaQA++ example quality plot along reads of a bad MiSeq run.

Figure 3: *SolexaQA++ example histogram plot of a bad MiSeq run.*

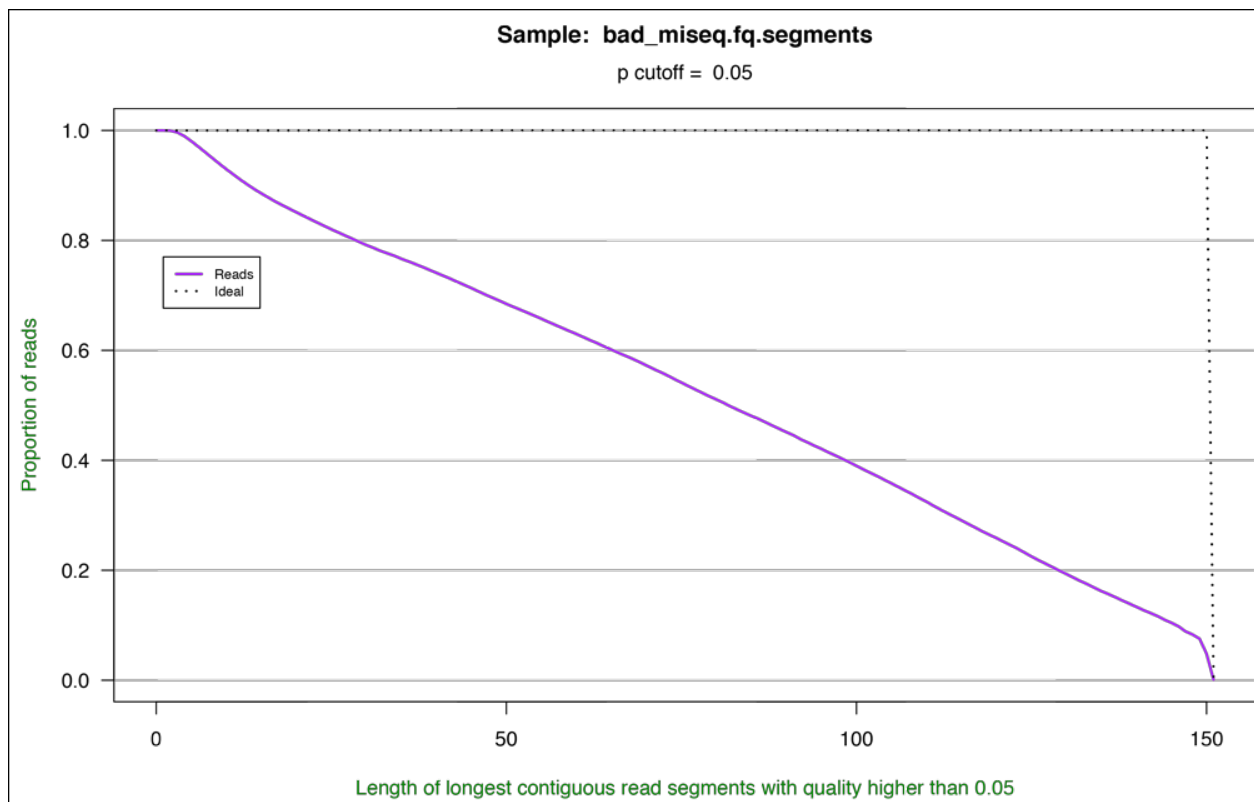
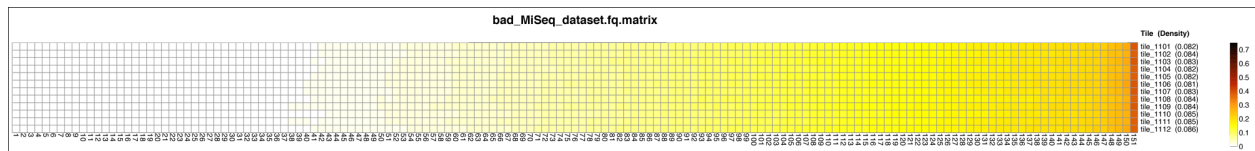Figure 4: *SolexaQA++ example cumulative plot of a bad MiSeq run.*

Figure 5: SolexaQA++ example quality heatmap of a bad MiSeq run.

1.3.4 Use SolexaQA++ to dynamic trim the data

Despite what you may have found out about the untrimmed data, it is a good idea to trim the data before further analyses.

To-do

1. Create a directory for the result-files –> **qa_toTrimmed/**
2. Use SolexaQA++ to trim the reads based on quality and a probability cutoff of 0.01.

Hint! Should you not get 1 and/or 2 it right, try these commands here or at http://sschmeier.com/bioinf-workshop/genome-assembly/code/solexaqa2.txt.

Attention! Should you be unable to trim with SolexaQA++ go to 1.3.6

1.3.5 Run SolexaQA++ analysis on trimmed data

To-do

1. Create a directory for the result-files –> **qa_trimmed/**.
2. Do the quality assessment again with the trimmed data-set.
3. Compare the results in **qa_trimmed/** to the untrimmed results in **qa_untrimmed/**. Can you say something about the differences in the untrimmed and trimmed data sets? Write down your observations.

Hint! Should you not get 1 and/or 2 it right, try these commands here or at http://sschmeier.com/bioinf-workshop/genome-assembly/code/solexaqa3.txt.

1.3.6 Sickle for dynamic trimming

To-do
Get the sickle program either here or here. Unzip and un on the untrimmed data:

```
$ tar xvzf v1.33.tar.gz
$ cd sickle
$ make
$ mv sickle ../
$ cd ..
```

Now we are going to run the program on our interleaved paired-end data:

```
$ mkdir qa_toTrimmed/
$ sickle pe -c data_paired.fastq -t sanger -m qa_toTrimmed/data_paired.fastq.trimmed -s qa_toTrimmed/s
```

Attention! Should you be unable to run sickle or SolexaQA++ at all to trim the data. You can download the trimmed data **here** (or at http://sschmeier.com/bioinf-workshop/genome-assembly/data/data_paired.fastq.trimmed.gz). Move the file into the qa_toTrimmed/ directory and unzip it with `gzip -d ata_paired.fastq.trimmed.gz`

1.4 Quality assessment of sequencing reads (FastQC)

1.4.1 Download/install FastQC (might already be installed)

Download FastQC (http://www.bioinformatics.babraham.ac.uk/projects/fastqc/) from the developer webpage here.

Note! Emergency link here or at http://sschmeier.com/bioinf-workshop/genome-assembly/apps/fastqc_v0.11.3.zip.

```
# Download file on the command-line
$ wget http://www.bioinformatics.babraham.ac.uk/projects/fastqc/fastqc_v0.11.3.zip
# Unzip the archive
$ unzip fastqc_v0.11.3.zip
# copy the program to the working directory
$ cp ./FastQC/fastqc .
# run the program
$ ./fastqc --help
# should now run the program
```

```
          FastQC - A high throughput sequence QC analysis tool

SYNOPSIS

        fastqc seqfile1 seqfile2 .. seqfileN

    fastqc [-o output dir] [--(no)extract] [-f fastq|bam|sam]
           [-c contaminant file] seqfile1 .. seqfileN

DESCRIPTION

    FastQC reads a set of sequence files and produces from each one a quality
    control report consisting of a number of different modules, each one of
    which will help to identify a different potential type of problem in your
    data.

    If no files to process are specified on the command line then the program
    will start as an interactive graphical application.  If files are provided
    on the command line then the program will run with no user interaction
    required.  In this mode it is suitable for inclusion into a standardised
    analysis pipeline.
```

1.4.2 Understand FastQC

FastQC is a very simple program to run that provides similar and additional information to SolexaQA++.

From the webpage:

> "FastQC aims to provide a simple way to do some quality control checks on raw sequence data coming from high throughput sequencing pipelines. It provides a modular set of analyses which you can use to give a quick impression of whether your data has any problems of which you should be aware before doing any further analysis."

The basic command looks like:

```
$ fastqc -o result-dir/ input-file.[txt/fa/fq] ...
```

- `-o result-dir/` is the directory where the result files will be written
- `input-file.[txt/fa/fq]` is the sequence file to analyze, can be more than one file.

Hint! The result will be a HTML page per input file that can be opened in a web-browser.

1.4.3 Run FastQC on the untrimmed and trimmed data

To-do

1. Create a directory for the results –> **fastqc_results/**
2. Run FastQC with both files
3. Compare the two result-files in a browser
4. Visit the FastQC website (http://www.bioinformatics.babraham.ac.uk/projects/fastqc/) and read about sequencing QC reports for good and bad Illumina sequencing runs.
5. Compare your results to these examples (*Figure 6-8*) of a particularly bad run (taken from http://www.bioinformatics.babraham.ac.uk/projects/fastqc/). Write down your observations.
6. What elements in these example figures (*Figure 6-8*) show that we are dealing with a bad run? Write down your observations.

Hint! Should you not get it right, try these commands here or at http://sschmeier.com/bioinf-workshop/genome-assembly/code/fastqc.txt.
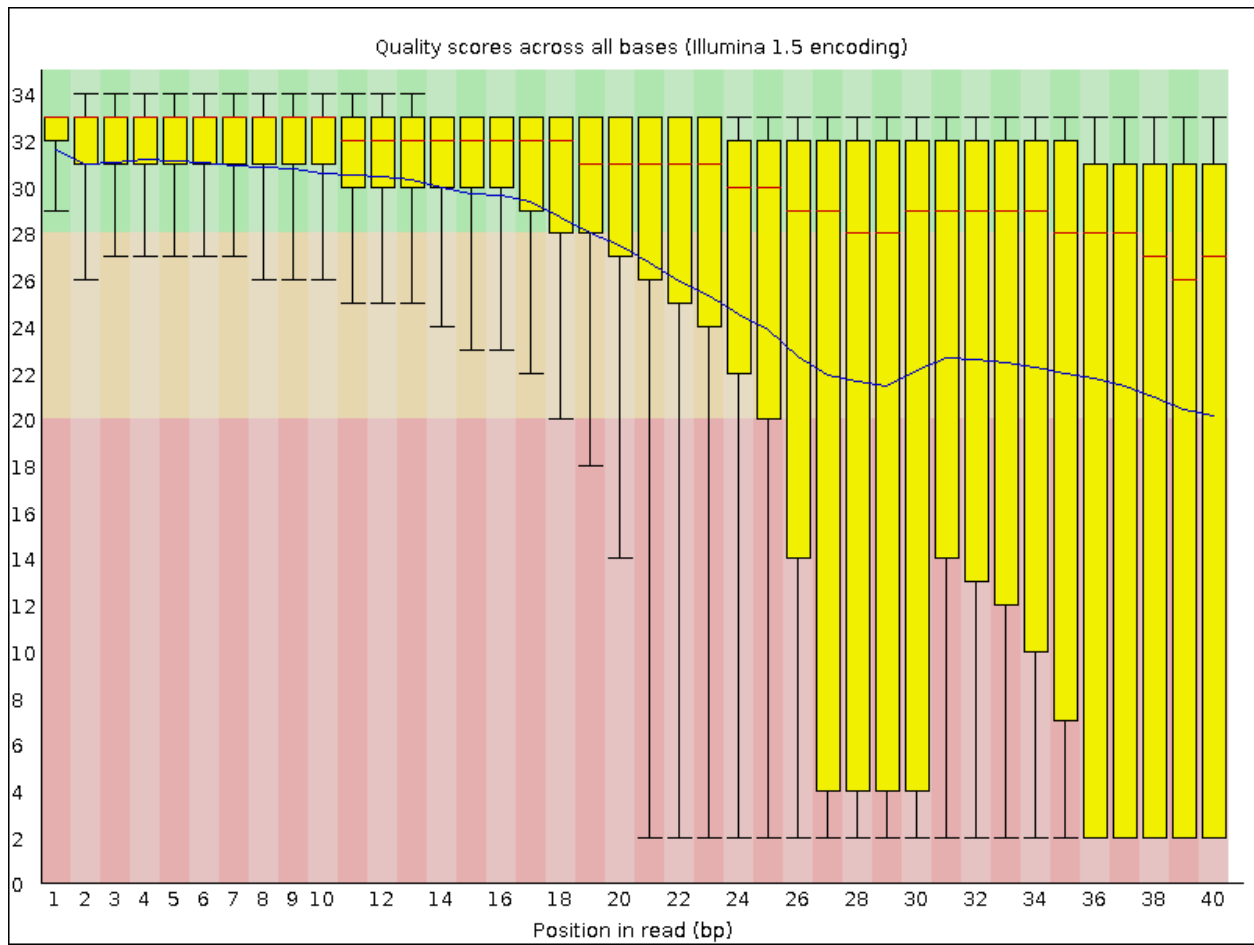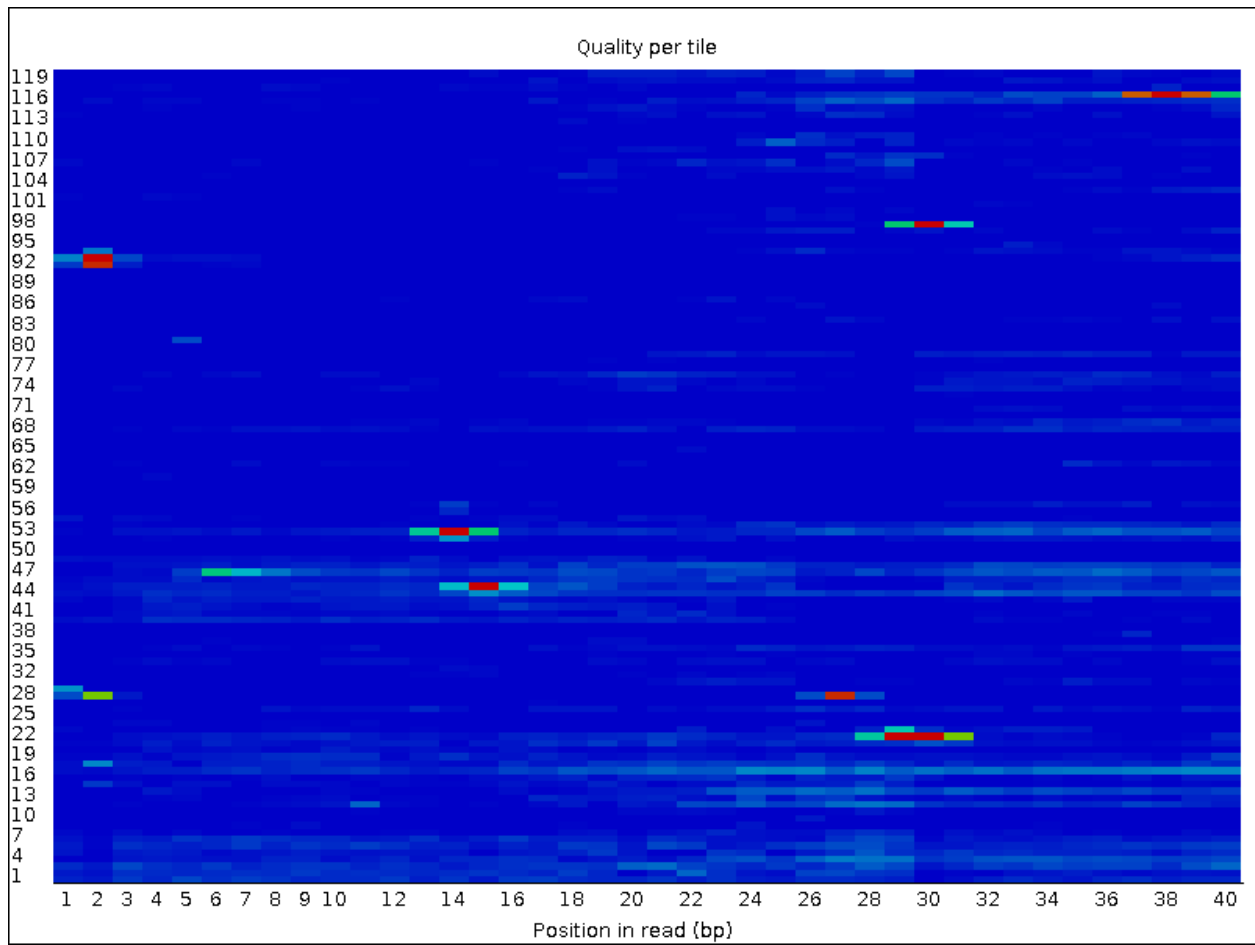
Figure 6: Quality score across bases.

Figure 7: Quality per tile.
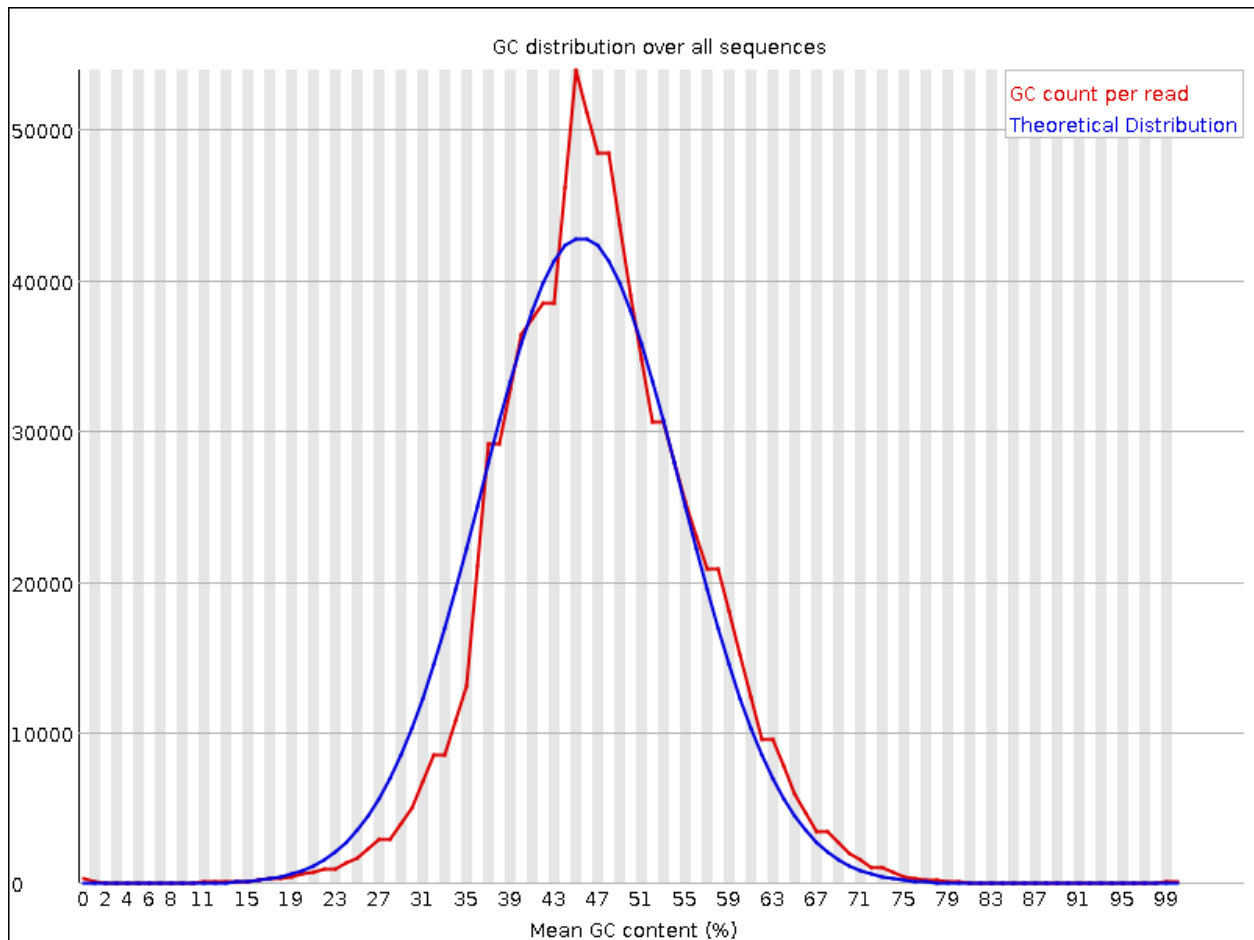
Figure 8: GC distribution over all sequences.

1.5 Whole genome assembly (velvet)

Velvet (https://www.ebi.ac.uk/~zerbino/velvet/) is a *de Bruijn* graph assembly program [Zerbino2008]. It is developed with the aim of assembling very short reads like in our case.

From the webpage:

> "Velvet is a de novo genomic assembler specially designed for short read sequencing technologies, such as Solexa or 454. Velvet currently takes in short read sequences, removes errors then produces high quality unique contigs. It then uses paired-end read and long read information, when available, to retrieve the repeated areas between contigs."

1.5.1 Download/install Velvet (might already be installed)

To-do
Download Velvet from the developer website (https://www.ebi.ac.uk/~zerbino/velvet/velvet_1.2.10.tgz).

Note! Emergency download-link here or at http://sschmeier.com/bioinf-workshop/genome-assembly/apps/velvet_1.2.10.tgz.

Attention! Should we use the pre-installed version of velvet you can jump this step. Most likely it was compiled with the standard *31-mer* length though. Discuss what differences you expect as opposed to the version with *63-mers*.

```
# unzip
$ gunzip velvet_1.2.10.tgz
# extract the archive
$ tar xvf velvet_1.2.10.tar
# change directory
$ cd velvet_1.2.10/
# compile program
$ make 'MAXKMERLENGTH=63'
# this will take a few seconds
# we have now two programs: velveth and velvetg


# copy these to your home directory
```

1.5.2 Understanding Velvet

Velvet uses two different programs sequentially to achieve the assembly. You have to use them one at a time. You should familiarize yourself with the structure of the program calls. Once you understand this it is very easy to change things around.

`velveth`

This program is a hashing program. It basically prepares your data for the assembler `velvetg`. An example program call looks like:

```
# Running velveth without parameters gives you a help page
$ ./velveth
# Here a complete example call
$ ./velveth results_directory/ 63 -fastq -shortPaired input_seqs.fastq
```

- `./velveth` the program including the full path where the program is located
- `result_directory/` the directory where you want the results to be saved
- 63 the *k*-mer value
- `-fastq` specifies the input sequence-file type
- `-shortPaired` the type of the reads (here short paired reads)
- `input_seqs.fastq` the file-name of the input data

`velvetg`

This program is the core of velvet and the actual assembler. An example program call looks like this:

```
$ ./velvetg result_directory/ -cov_cutoff 10 -exp_cov 30
      -min_contig_lgth 500 -ins_length 300
```

- `./velvetg` the program including the full path where the program is located
- `result_directory/` the directory where you want the results to be saved
- `-cov_cutoff 10` removal of low coverage nodes AFTER tour bus or allow the system to infer it [do not worry about this]
- `-exp_cov 30` the expected coverage of unique regions or allow the system to infer it [do not worry about this]
- `-min_contig_lgth 500` the minimum size of contig to report back (to exclude tiny ones we only look at contigs >500bp)
- `-ins_length 300` expected distance between two paired-end reads in the respective short-read dataset

1.5.3 Run Velvet with untrimmed data

To-do

1. Make sure you have the untrimmed data and you know where it is.
2. Create a directory for the results –> `velvet_untrimmed/`
3. Run velveth with the data `data_paired.fastq`
4. Run velvetg

Hint! Should you not get it right, try these commands here or at http://sschmeier.com/bioinf-workshop/genome-assembly/code/velvet1.txt.

1.5.4 Run Velvet with trimmed data

To-do

1. Make sure you have the trimmed data and you know where it is.
2. Create a directory for the results –> `velvet_trimmed/`
3. Run velveth with the data `qa_toTrimmed/data_paired.fastq.trimmed`
4. Run velvetg

Hint! Should you not get it right, try these commands here or at http://sschmeier.com/bioinf-workshop/genome-assembly/code/velvet2.txt.

1.5.5 Evaluate assemblies

To-do

1. Look at the Log files from trimmed and untrimmed assemblies.
2. Look at the stat.txt files from trimmed and untrimmed assemblies.
3. Look at the contigs.fa files of the trimmed and untrimmed assemblies.
4. What can we say about the the assemblies?
5. How does untrimmed and trimmed compare?
6. What can you say about the trimming procedure in light of assembling sequences?

1.5.6 Assembly quality assessment

Quast (QUality ASsesment Tool, http://quast.bioinf.spbau.ru/) [Gurevich2013], evaluates genome assemblies by computing various metrics, including:

- N50: length for which the collection of all contigs of that length or longer covers at least 50% of assembly length
- NG50: where length of the reference genome is being covered
- NA50 and NGA50: where aligned blocks instead of contigs are taken
- missassemblies: misassembled and unaligned contigs or contigs bases
- genes and operons covered

The program can be used on their website (http://quast.bioinf.spbau.ru/) or downloaded for local use at this direct link.

1.5.7 Visualize assemblies

Bandage (Bioinformatics Application for Navigating De novo Assembly Graphs Easily, https://rrwick.github.io/Bandage/) is a program that visualizes an assembly as a graph [Wick2015].

1.6 Genome annotation

Genome annotation can be facilitated for example with these programs:

- Genome annotation pipeline RAST (http://rast.nmpdr.org/) [Aziz2008]
- Phage-specific annotation pipeline PHAST (http://phast.wishartlab.com/) [Zhou2011]

1.7 What's next?

Next steps could include:

- Map all reads to the "new" genome
- Look at the aligned reads in a genome viewer
- Predict genes in the "new" genome
- Overlay gene information with aligned reads in the genome viewer
- etc.

1.8 Further reading

Background on Genome Assemblies:

- Compeau2011
- Nagarajan2013

Evaluation of Genome Assembly tools:

- Salzberg2012

1.9 References

Aziz RK, Bartels D, Best AA, DeJongh M, Disz T, Edwards RA, Formsma K, Gerdes S, Glass EM, Kubal M, Meyer F, Olsen GJ, Olson R, Osterman AL, Overbeek RA, McNeil LK, Paarmann D, Paczian T, Parrello B, Pusch GD, Reich C, Stevens R, Vassieva O, Vonstein V, Wilke A, Zagnitko O. **The RAST Server: rapid annotations using subsystems technology.** *BMC Genomics 2008 Feb 8;9:75*

Compeau PE, Pevzner PA, Tesler G. **How to apply de Bruijn graphs to genome assembly.** *Nat Biotechnol. 2011 Nov 8;29(11):987-91*

Cox MP, Peterson DA and Biggs PJ. **SolexaQA: At-a-glance quality assessment of Illumina second-generation sequencing data.** *BMC Bioinformatics 2010, 11:485*

Gurevich A, Saveliev V, Vyahhi N and Tesler G. **QUAST: quality assessment tool for genome assemblies.** *Bioinformatics 2013, 29(8), 1072-1075*

Nagarajan N, Pop M. **Sequence assembly demystified.** *Nat Rev Genet. 2013 Mar;14(3):157-67*

Salzberg SL, Phillippy AM, Zimin A, Puiu D, Magoc T, Koren S, Treangen TJ, Schatz MC, Delcher AL, Roberts M, Marçais G, Pop M, Yorke JA. **GAGE: A critical evaluation of genome assemblies and assembly algorithms.** *Genome Res. 2012 Mar;22(3):557-67*

Wick RR, Schultz MB, Zobel J and Holt KE. **Bandage: interactive visualization of de novo genome assemblies.** *Bioinformatics 2015, 10.1093/bioinformatics/btv383*

Zerbino DR and Birney E. **Velvet: algorithms for de novo short read assembly using de Bruijn graphs.** *Genome Res. 2008, 18:821-829.*

Zhou Y, Liang Y, Lynch KH, Dennis JJ, Wishart DS. **PHAST: a fast phage search tool.** *Nucleic Acids Res. 2011 Jul;39(Web Server issue):W347-52.*

2.0 Web links

This tutorial: http://sschmeier.com/bioinf-workshop/genome-assembly/

Lectures for this topic: **Next-generation sequencing and quality control: An introduction**, figshare, https://dx.doi.org/10.6084/m9.figshare.2972320.v1; **Genome Assembly: An Introduction**, figshare, https://dx.doi.org/10.6084/m9.figshare.2972323.v1

Illumina paired-end sequencing: http://www.illumina.com/technology/next-generation-sequencing/paired-end-sequencing_assay.html

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

SolexaQA: http://solexaqa.sourceforge.net/

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

Quast: http://quast.bioinf.spbau.ru/

Bandage: https://rrwick.github.io/Bandage/

PHAST: http://phast.wishartlab.com/

RAST: http://rast.nmpdr.org/

File: index.md - PDF-version: link1 | link2 *- Sebastian Schmeier - Last update: 2016-03-16*