Metagenomics Tutorial Documentation

Release 1

Alex Sczyrba, Christian Henke

Aug 31, 2018

Contents

Welcome to the ECCB18 Metagenomics Tutorial. This tutorial will guide you through the typical steps of metagenome assembly and binning.

You will run the workflow using the Common Workflow Language and all tools will be executed inside Docker containers with the aim to keep the workflow reproducible, portable and its components reusable.

CHAPTER 1

Setting up your de.NBI Cloud instance

As metagenome assemblies require a lot of compute resources, we will run the tutorial on the de.NBI Cloud infrastructure. Each workshop participant will log into a dedicated virtual machine (VM) and run all jobs on this machine.

The VM can be accessed by pointing your web browser to:

```
https://<number>.llil.de
```

Where `<number>` is the number of the VM that has been assigned to you, e.g. `https://123.llil.de`

You will be presented with a web-based text editor for editing files as well as a terminal window for command line access.

The files view on the left should list a folder called `eccb18tutorial` containing all the needed materials for this tutorial.

The contents of the folder are also available here.

CHAPTER 2

The Common Workflow Language

The Common Workflow Language (CWL) is a specification for describing analysis workflows and tools in a way that makes them portable and scalable across a variety of software and hardware environments.

—https://www.commonwl.org/

CHAPTER 3

The Docker Container Engine

CHAPTER 4

Download the Tutorial Data Set

We have prepared a small toy data set for this tutorial. You can download the data set using the following command:

```
~/eccb18tutorial/pipeline/download_input_files.sh
```

This will create a folder named `eccb18tutorial/input` with the following content:

File	Content
genomes/	Directory containing the reference genomes
read1.fq.gz	Read 1 of paired reads (gzipped FASTQ)
read2.fq.gz	Read 2 of paired reads (gzipped FASTQ)

CHAPTER 5

FastQC Quality Control

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 main functions of FastQC are

- Import of data from BAM, SAM or FastQ files (any variant)
- Providing a quick overview to tell you in which areas there may be problems
- Summary graphs and tables to quickly assess your data
- Export of results to an HTML based permanent report
- Offline operation to allow automated generation of reports without running the interactive application

See the FastQC home page for more info.

To run `FastQC` on our data, simply type:

```
cd /vol/spool/tutorial-data
fastqc read1.fq read2.fq
```

After `FastQC` finished running, copy the results to your `public_html` directory:

```
cp -r read?.fq_fastqc ~/public_html/
```

Now you can access the report at:

```
http://<YOUR_AWS_IP_ADDRESS>/~ubuntu/read1.fq_fastqc/fastqc_report.html
http://<YOUR_AWS_IP_ADDRESS>/~ubuntu/read2.fq_fastqc/fastqc_report.html
```

Check out the FastQC home page for examples of reports including bad data.

CHAPTER 6

Assembly

We are going to use different assemblers and compare the results.

6.1 Velvet Assembly

Velvet was one of the first de novo genomic assemblers specially designed for short read sequencing technologies. It was developed by Daniel Zerbino and Ewan Birney at the European Bioinformatics Institute (EMBL-EBI). 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. See the Velvet home page for more info.

6.1.1 Step 1: velveth

`velveth` takes in a number of sequence files, produces a hashtable, then outputs two files in an output directory (creating it if necessary), Sequences and Roadmaps, which are necessary for running `velvetg` in the next step.

Let's create multiple hashtables using kmer-lengths of 31 and 51. We are going to redirect the output into a file *velveth_31.log* and *velveth_51.log*:

```
cd /vol/spool/tutorial-data
velveth velvet_31 31 -shortPaired -fastq -separate read1.fq read2.fq >& velveth_31.
↪log &
velveth velvet_51 51 -shortPaired -fastq -separate read1.fq read2.fq >& velveth_51.
↪log &
```

This will create two output directories for the two different kmer-lengths: *velvet_31* and *velvet_51*.

6.1.2 Step 2: velvetg

Now we have to start the actual assembly using `velvetg`. `velvetg` is the core of Velvet where the de Bruijn graph is built then manipulated. Let's run assemblies for both kmer-lengths. See the Velvet manual for more info about

parameter settings. Again, output will be redirected to log-files *velvetg_31.log* and *velvetg_51.log*:

```
velvetg velvet_31 -cov_cutoff auto -ins_length 270 -min_contig_lgth 500 -exp_cov auto␣
↪>& velvetg_31.log &
velvetg velvet_51 -cov_cutoff auto -ins_length 270 -min_contig_lgth 500 -exp_cov auto␣
↪>& velvetg_51.log &
```

The contig sequences are located in the *velvet_31* and *velvet_51* directories in file *contigs.fa*. Let's get some very basic statistics on the contigs. The script `getN50.pl` reads the contig file and computes the total length of the assembly, number of contigs, N50 and largest contig size. In our example we will exclude contigs shorter than 500bp (option *-s 500*):

```
getN50.pl -s 500 -f velvet_31/contigs.fa
getN50.pl -s 500 -f velvet_51/contigs.fa
```

Note: Most jobs above will be started in the backgroud using the `&` at the end of each command, which allows you to continue working in the shell.

You can watch your running jobs by typing `top` (hit `q` to exit `top`).

You can look into the log-files by typing e.g. `less LOGFILE` (hit `q` to quit) or `tail -f LOGFILE` (hit `^C` to quit).

6.2 MEGAHIT Assembly

MEGAHIT is a single node assembler for large and complex metagenomics NGS reads, such as soil. It makes use of succinct de Bruijn graph (SdBG) to achieve low memory assembly. MEGAHIT can optionally utilize a CUDA-enabled GPU to accelerate its SdBG contstruction. See the MEGAHIT home page for more info.

MEGAHIT can be run by the following command. As our AWS instance has 16 cores, we use the option *-t 16* to tell MEGAHIT it should use 16 parallel threads. The output will be redirected to file *megahit.log*:

```
cd /vol/spool/tutorial-data
megahit -1 read1.fq -2 read2.fq -t 16 -o megahit_out >& megahit.log &
```

The contig sequences are located in the *megahit_out* directory in file *final.contigs.fa*. Again, let's get some basic statistics on the contigs:

```
getN50.pl -s 500 -f megahit_out/final.contigs.fa
```

Note: Most jobs above will be started in the backgroud using the `&` at the end of each command, which allows you to continue working in the shell.

You can watch your running jobs by typing `top` (hit `q` to exit `top`).

You can look into the log-files by typing e.g. `less LOGFILE` (hit `q` to quit) or `tail -f LOGFILE` (hit `^C` to quit).

6.3 IDBA-UD Assembly

IDBA is the basic iterative de Bruijn graph assembler for second-generation sequencing reads. IDBA-UD, an extension of IDBA, is designed to utilize paired-end reads to assemble low-depth regions and use progressive depth on contigs to reduce errors in high-depth regions. It is a generic purpose assembler and epspacially good for single-cell and metagenomic sequencing data. See the IDBA home page for more info.

IDBA-UD can be run by the following command. Note that IDBA-UD requires paired-end reads stored in single FastA file and a pair of reads is in consecutive two lines. You can use *fq2fa* (part of the IDBA repository) to merge two FastQ read files to a single file. We have prepared such a FASTA formatted file called *reads.fas*. As our AWS instance has 16 cores, we use the option *–num_threads 16* to tell IDBA-UD it should use 16 parallel threads. The log output will be redirected to file *idba_ud.log*:

```
cd /vol/spool/tutorial-data
idba_ud -r reads.fas --num_threads 16 -o idba_ud_out >& idba_ud.log &
```

The contig sequences are located in the *idba_ud_out* directory in file *contig.fa*. Again, let's get some basic statistics on the contigs:

```
getN50.pl -s 500 -f idba_ud_out/contig.fa
```

Note: Most jobs above will be started in the backgroud using the `&` at the end of each command, which allows you to continue working in the shell.

You can watch your running jobs by typing `top` (hit `q` to exit `top`).

You can look into the log-files by typing e.g. `less LOGFILE` (hit `q` to quit) or `tail -f LOGFILE` (hit `^C` to quit).

6.4 Ray Assembly

Ray is a parallel software that computes de novo genome assemblies with next-generation sequencing data. Ray is written in C++ and can run in parallel on numerous interconnected computers using the message-passing interface (MPI) standard. See the Ray home page for more info.

Ray can be run by the following command using a kmer-length of 31. As our AWS instance has 16 cores, we specify this in the ‘mpiexec -n 16 ‘ command to let Ray know it should use 16 parallel MPI processes:

```
cd /vol/spool/tutorial-data
mpiexec -n 16 Ray -k 31 -p read1.fq read2.fq -o ray_31 >& ray_31.log &
```

This will create the output directory *ray_31* and the final contigs are located in *ray_31/Contigs.fasta*. Again, let's get some basic statistics on the contigs:

```
getN50.pl -s 500 -f ray_31/Contigs.fasta
```

Now that you have run assemblies using Velvet, MEGAHIT, IDBA-UD and Ray, let's have a quick look at the assembly statistics of all of them:

```
cd /vol/spool/tutorial-data
./get_assembly_stats.sh
```

Note: Most jobs above will be started in the backgroud using the `&` at the end of each command, which allows you to continue working in the shell.

You can watch your running jobs by typing `top` (hit `q` to exit `top`).

You can look into the log-files by typing e.g. `less LOGFILE` (hit `q` to quit) or `tail -f LOGFILE` (hit `^C` to quit).

CHAPTER 7

Gene Prediction

Prodigal (Prokaryotic Dynamic Programming Genefinding Algorithm) is a microbial (bacterial and archaeal) gene finding program developed at Oak Ridge National Laboratory and the University of Tennessee. See the Prodigal home page for more info.

To run `prodigal` on our data, simply type:

```
cd /vol/spool/tutorial-data/megahit_out
prodigal -p meta -a final.contigs.genes.faa -d final.contigs.genes.fna -f gff -o␣
↪final.contigs.genes.gff -i final.contigs.fa
```

Output files:

final.contigs.genes.gff	positions of predicted genes in GFF format
final.contigs.genes.faa	protein translations of predicted genes
final.contigs.genes.fna	nucleotide sequences of predicted genes

CHAPTER 8

Assembly Evaluation

We are going to evaluate our assemblies using the reference genomes.

8.1 Read Mapping

In this part of the tutorial we will look at the assemblies by mapping the reads to the assembled contigs. Different tools exists for mapping reads to genomic sequences such as bowtie or bwa. Today, we will use the tool BBMap.

BBMap: Short read aligner for DNA and RNA-seq data. Capable of handling arbitrarily large genomes with millions of scaffolds. Handles Illumina, PacBio, 454, and other reads; very high sensitivity and tolerant of errors and numerous large indels. Very fast. See the BBMap home page for more info.

`bbmap` needs to build an index for the contigs sequences before it can map the reads onto them. Here is an example command line for mapping the reads back to the MEGAHIT assembly:

```
cd /vol/spool/tutorial-data/megahit_out
~/bbmap/bbmap.sh ref=final.contigs.fa
```

Now that we have an index, we can map the reads:

```
~/bbmap/bbmap.sh in=../read1.fq in2=../read2.fq out=megahit.sam bamscript=sam2bam.sh
```

`bbmap` produces output in SAM format by default, usually you want to convert this into a sorted BAM file. `bbmap` creates a shell script which can be used to convert `bbmap`'s output into BAM format:

```
source sam2bam.sh
```

SAM and BAM files can be viewed and manipulated with SAMtools. Let's first build an index for the FASTA file:

```
samtools faidx final.contigs.fa
```

To look at the BAM file use:

```
samtools view megahit_sorted.bam | less
```

We will use a genome browser to look at the mappings. For this, you have to (1) open a terminal window on **your local workstation**, (2) download the BAM file and (3) download and start IGV: Integrative Genomics Viewer:

```
cd ~/mg-tutorial
scp -i bibigrid/MGAssemblyTutorial.pem ubuntu@52.16.173.148:/vol/spool/tutorial-data/
↪megahit_out/final.contigs.fa* .
scp -i bibigrid/MGAssemblyTutorial.pem ubuntu@52.16.173.148:/vol/spool/tutorial-data/
↪megahit_out/*.bam* .
scp -i bibigrid/MGAssemblyTutorial.pem ubuntu@52.16.173.148:/vol/spool/tutorial-data/
↪megahit_out/*.gff .
wget http://data.broadinstitute.org/igv/projects/downloads/IGV_2.3.59.zip
unzip IGV_2.3.59.zip
IGV_2.3.59/igv.sh
```

Now let's look at the mapped reads:

1. Load the contig sequences into IGV. Use the menu `Genomes->Load Genome from File...`
2. Load the BAM file into IGV. Use menu `File->Load from File...`
3. Load the predicted genes as another track. Use menu `File->Load from File...` to load the GFF file.

8.2 MetaQUAST

QUAST stands for QUality ASsessment Tool. The tool evaluates genome assemblies by computing various metrics. You can find all project news and the latest version of the tool at sourceforge. QUAST utilizes MUMmer, GeneMarkS, GeneMark-ES, GlimmerHMM, and GAGE. In addition, MetaQUAST uses MetaGeneMark, Krona tools, BLAST, and SILVA 16S rRNA database. See the QUAST home page for more info.

To call `metaquast.py` we have to provide reference genomes which are used to calculate a number of different metrics for evaluation of the assembly. In real-world metagenomics, these references are usually not available, of course:

```
cd /vol/spool/tutorial-data
python ~/quast-3.1/metaquast.py --threads 16 --gene-finding --meta \
-R /vol/spool/tutorial-data/genomes/Aquifex_aeolicus_VF5.fna,\
/vol/spool/tutorial-data/genomes/Bdellovibrio_bacteriovorus_HD100.fna,\
/vol/spool/tutorial-data/genomes/Chlamydia_psittaci_MN.fna,\
/vol/spool/tutorial-data/genomes/Chlamydophila_pneumoniae_CWL029.fna,\
/vol/spool/tutorial-data/genomes/Chlamydophila_pneumoniae_J138.fna,\
/vol/spool/tutorial-data/genomes/Chlamydophila_pneumoniae_LPCoLN.fna,\
/vol/spool/tutorial-data/genomes/Chlamydophila_pneumoniae_TW_183.fna,\
/vol/spool/tutorial-data/genomes/Chlamydophila_psittaci_C19_98.fna,\
/vol/spool/tutorial-data/genomes/Finegoldia_magna_ATCC_29328.fna,\
/vol/spool/tutorial-data/genomes/Fusobacterium_nucleatum_ATCC_25586.fna,\
/vol/spool/tutorial-data/genomes/Helicobacter_pylori_26695.fna,\
/vol/spool/tutorial-data/genomes/Lawsonia_intracellularis_PHE_MN1_00.fna,\
/vol/spool/tutorial-data/genomes/Mycobacterium_leprae_TN.fna,\
/vol/spool/tutorial-data/genomes/Porphyromonas_gingivalis_W83.fna,\
/vol/spool/tutorial-data/genomes/Wigglesworthia_glossinidia.fna \
-o quast \
-l MegaHit,Ray_31,velvet_31,velvet_51,idba_ud \
megahit_out/final.contigs.fa \
ray_31/Contigs.fasta \
```

(continues on next page)

(continued from previous page)

```
velvet_31/contigs.fa \
velvet_51/contigs.fa \
idba_ud_out/contig.fa
```

QUAST generates HTML reports including a number of interactive graphics. To access these reports, copy the quast directory to your *public_html* folder:

```
cp -r quast ~/public_html
```

After that, you can load the reports in your web browser:

```
http://YOUR_AWS_IP/~ubuntu/quast/summary/report.html
http://YOUR_AWS_IP/~ubuntu/quast/combined_quast_output/report.html
```

CHAPTER 9

Binning

After the assembly of metagenomic sequencing reads into contigs, binning algorithms try to recover individual genomes to allow access to uncultivated microbial populations that may have important roles in the samples community.

9.1 MaxBin Binning

MaxBin is a software that is capable of clustering metagenomic contigs into different bins, each consists of contigs from one species. MaxBin uses the nucleotide composition information and contig abundance information to do achieve binning through an Expectation-Maximization algorithm. For users' convenience MaxBin will report genome-related statistics, including estimated completeness, GC content and genome size in the binning summary page. See the MaxBin home page for more info.

Let's run a MaxBin binning on the MEGAHIT assembly. First, we need to generate an abundance file from the mappes reads:

```
/vol/spool/tutorial-data/megahit_out
~/bbmap/pileup.sh in=megahit.sam  out=cov.txt
awk '{print $1"\t"$5}' cov.txt | grep -v '^#' > abundance.txt
```

Next, we can run MaxBin:

```
~/MaxBin-2.1/run_MaxBin.pl -thread 16 -contig final.contigs.fa -out maxbin -abund␣
↪abundance.txt
```

Assume your output file prefix is (out). MaxBin will generate information using this file header as follows.

(out).0XX.fasta	the XX bin. XX are numbers, e.g. out.001.fasta
(out).summary	summary file describing which contigs are being classified into which bin.
(out).log	log file recording the core steps of MaxBin algorithm
(out).marker	marker gene presence numbers for each bin. This table is ready to be plotted by R or other 3rd-party software.
(out).marker.pdf	visualization of the marker gene presence numbers using R
(out).noclass	all sequences that pass the minimum length threshold but are not classified successfully.
(out).tooshort	all sequences that do not meet the minimum length threshold.

Now you can run a gene prediction on each genome bin and BLAST one sequence for each bin for a (very crude!) classification:

```
for i in max*fasta; do prodigal -p meta -a $i.genes.faa -d $i.genes.fna -f gff -o $i.
↪genes.gff -i $i& done
```

Does the abundance of the bins match the 16S profile of the community?