Methods in Microbiomics

Release 0.0.1

Sunagawa Group

Apr 19, 2024

DATA PREPROCESSING

Set of guidelines and best practices for robust and reproducible bioinformatics processing and data analysis with the focus on Microbiomics research.

Important: This documentation is currently under construction.

CHAPTER

ONE

TUTORIALS

Tutorials are available for each section listed below. To start, create a directory for the tutorial of your interest:

```
mkdir <section>_tutorial
cd <section>_tutorial
```

For example, for *Data Preprocessing*, run:

```
mkdir preprocessing_tutorial
cd preprocessing_tutorial
```

Note: For the next section, you have to install conda first. Installation instructions can be found here.

For each of the sections we have provided a link to a conda environment file (i.e. a file that specifies which packages to install) and a test dataset that can be used for practice. Download these files and save them to the tutorial directory. Next, you can create and activate the conda environment and extract the test data:

```
conda env create -f <environment name>.yaml
conda activate <environment name>
tar -xvzf Sample1_isolate.tar.gz
```

For *Data Preprocessing*, you need to install mamba. Here are the commands to create the conda environment and unpack the data:

```
mamba env create -f isolate_assembly.yaml
conda activate isolate_assembly
tar -xvpf Sample1_isolate.tar.gz
```

Now, you are ready to run the example commands given in the corresponding section.

CHAPTER

TWO

DATA PREPROCESSING

Warning: Before proceeding to any of the bioinformatics workflows, make sure you have good quality data. See *Data Preprocessing* for more.

Download `data preprocessing conda environment file` and `data preprocessing test dataset`

CHAPTER

THREE

GENOME ASSEMBLY

Best practices for *Genome Assembly* and *Metagenomic Assembly*.

Download `isolate assembly conda environment file` and `isolate assembly test dataset`

CHAPTER

FOUR

TAXONOMIC PROFILING

Best practises for profiling of amplicon and metagenomic data

CHAPTER

FIVE

TRANSCRIPTOMICS ANALYSIS

Best practices for transcriptomic and metatranscriptomic data analysis

Best practices in metagenomic data analysis

CHAPTER

SIX

SUPPORT

- If you have any questions or suggestions leave a comment below!

6.1 Data Preprocessing

Protocol provided by Hans-Joachim Ruscheweyh.

6.1.1 General Considerations

Data quality control is an essential first step in any bioinformatics workflow. Below we discuss recommended preprocessing steps for **short read Illumina** sequencing data. Broadly, these steps involve Illumina adapter removal, contaminant filtering and quality-trimming. Additional preprocessing steps, recommended only for specific workflows, are detailed in *Other Considerations*.

Important: This applies to (standard) Illumina short read data. Long read sequencing data from other technologies, or other library preparations from Illumina (e.g. Nextera Mate Pair Reads data) will require a different preprocessing protocol.

Note: Sample data for this section can be found `here`. The conda environment specifications are `here`. See the *Tutorials* section for instructions on how to unpack the data and create the conda environment. After unpacking the data, you should have a set of forward (Sample1_R1.fq.gz) and reverse (Sample1_R2.fq.gz) reads. Also included are Illumina adapter sequences (adapters.fa) and PhiX genome (phix174_ill.ref.fa.gz).

---

1. **Adapter Trimming**. The adapter sequences contain the sequencing primer binding sites, index sequences, and sequences that allow flow-cell binding. Unless removed, these can interfere with downstream analyses. For this and other preprocessing steps, we use BBTools, a set of tools developed by the Joint Genome Institute. Adapter trimming is performed using BBDuk. In this step, a FASTA file with Illumina adapter sequences is specified as reference, and BBDuk will perform k-mer matching to trim the adapter sequences from the reads. The example command is shown below.

   Example command

   ```
   bbduk.sh -Xmx1G usejni=t in=Sample1_R1.fq.gz in2=Sample1_R2.fq.gz \
   out=Sample1_trimmed_R1.fq.gz out2=Sample1_trimmed_R2.fq.gz \
   outm=Sample1_adapter_matched.fq.gz outs=Sample1_adapter_s.fq.gz  \
   refstats=Sample1.adapter_trim.stats statscolumns=5 overwrite=t ref=adapters.fa \
   ktrim=r k=23 mink=11 hdist=1 2>> preprocessing.log
   ```

Options Explained

`-Xmx`	This will be passed to Java to set memory usage. `Xmx1G` will set it to 1G.
`usejni`	Enable JNI-accelerated version of BBDuk.
`ktrim`	`ktrim = r` trims the adapter as well as all the bases to the right of the adapter sequence.
`k`	Length of the k-mer used for matching.
`mink`	Additionally matches shorter k-mers (with lengths between 23 and 11) to trim partial adapter sequences.
`hdist`	Hamming distance for reference k-mers. The Hamming distance describes the number of bases by which two DNA sequences differ.
`outs`	Write singleton reads whose mate has failed filters to this file.

Note:

Why are adapter sequences trimmed from only the 3' ends of reads?
Why do we choose k-mer length between 23 and 11?

2. **Contaminant removal**. Spike-ins (most commonly PhiX) are usually used for quality control of sequencing runs as well as to ensure nucleotide diversity when sequencing low complexity libraries. We perform this filtering step prior to downstream analysis to be completely sure that these sequences are not be present in your data. Here we use BBDuk and PhiX genome is used as the reference.

   Example command

```
bbduk.sh -Xmx1G usejni=t in=Sample1_trimmed_R1.fq.gz in2=Sample1_trimmed_R2.
↪fq.gz \
out=Sample1_phix_removed_R1.fq.gz out2=Sample1_phix_removed_R2.fq.gz \
outm=Sample1_phix_matched.fq.gz outs=Sample1_phix_s.fq.gz \
ref=phix174_ill.ref.fa.gz k=31 hdist=1 \
refstats=Sample1_phix.stats statscolumns=5 2>> contaminant.log
```

Here, we use a different kmer size `k=31` to remove Spike-ins. This is the recommended length by BBDuk to remove all reads that have a 31-mer match to the PhiX genome.

Note: High nucleotide diversity (i.e. equal relative proportions of A,C,G, and T in each cycle) is critical to the performance of Illumina sequencers. Low diversity (or low complexity) libraries, such as amplicon libraries, will have a large proportion of one nucleotide and small proportions of other nucleotides in a cycle. To compensate for low complexity, a PhiX DNA sequence is often added to the library. Different sequencers use different chemistry and image processing software and require different amounts of PhiX spike-in (anywhere between 5% and 50%). Check the latest information about your sequencing platform.

3. **Quality filtering and trimming**. In this step we use BBDuk to trim low quality bases from the ends of the reads and filter reads based on length, average read quality, and number of Ns present.

   Example command

```
bbduk.sh -Xmx1G usejni=t in=Sample1_phix_removed_R1.fq.gz in2=Sample1_phix_
↪removed_R2.fq.gz  \
out1=Sample1_clean_R1.fq.gz out2=Sample1_clean_R2.fq.gz \
outm=Sample1_qc_failed.fq.gz outs=Sample1_s.fq.gz minlength=45 \
qtrim=rl maq=20 maxns=1  stats=Sample1_qc.stats statscolumns=5 trimq=14 2>>␣
↪qc.log
```

Options Explained

`minlength=45`	Filters out reads that are shorter than 45 bp.
`qtrim=rl`	Trims low quality bases on the right and left ends of the reads.
`trimq=14`	Regions with average quality BELOW 14 will be trimmed.
`maq=20`	Filters out reads with average quality BELOW 20.
`maxns=1`	Filters out reads with more than 1 N.

Note: Base quality scores (i.e. level of confidence for any one base call) are an integral part of many bioinformatics pipelines (i.e. alignment and variant calling). Quality scores are usually expressed on a Phred scale ($Q = -10log_{10}P$, where P is the probability of an error in the base call). Base quality scores normally range somewhere between 2 and 40, where Q40 represents an error probability of 1/10000. More recently, Illumina started using binned quality scores. For example, NovaSeq (with RTA3) only produces 4 Q-scores: 2 is assigned to no-calls, 12 to calls <Q15, 23 to ~Q20 and 37 to >Q30. According to Illumina and in our hands, these binned quality scores did not affect the downstream analyses (i.e. variant calling).

All of the preprocessing commands can be piped together as follows:

```
bbduk.sh -Xmx1G usejni=t in=Sample1_R1.fq.gz in2=Sample1_R2.fq.gz \
out=stdout.fq outm=Sample1_adapter_matched.fq.gz outs=Sample1_adapter_s.fq.gz  \
refstats=Sample1.adapter_trim.stats statscolumns=5 overwrite=t ref=adapters.fa \
ktrim=r k=23 mink=11 hdist=1  2>> preprocessing.log | \
bbduk.sh -Xmx1G usejni=t interleaved=true overwrite=t \
in=stdin.fq out=stdout.fq outm=Sample1_phix_matched.fq.gz outs=Sample1_phix_s.fq.gz \
ref=phix174_ill.ref.fa.gz k=31 hdist=1 refstats=Sample1_phix.stats statscolumns=5 2>>␣
↪preprocessing.log | \
bbduk.sh -Xmx1G usejni=t overwrite=t interleaved=true \
in=stdin.fq out1=Sample1_clean_R1.fq.gz out2=Sample1_clean_R2.fq.gz \
outm=Sample1_qc_failed.fq.gz outs=Sample1_s.fq.gz minlength=45 \
qtrim=rl maq=20 maxns=1  stats=Sample1_qc.stats statscolumns=5 trimq=14  2>>␣
↪preprocessing.log;
```

6.1.2 Other Considerations

Below are some of the other preprocessing steps that are recommended for specific applications only. All of these steps will be performed on the clean reads produced by general preprocessing workflow outlined above.

Preprocessing Step	Recommended for	Tools
Filtering out host reads	Any samples containing host DNA	BBMap
Coverage normalization	Metagenomic assembly (very large samples only)	BBNorm
Paired-read merging	Metagenomic assembly, 16S and mOTUs profiling	BBMerge

Filtering out host reads

Samples containing host DNA can be filtered by mapping the reads to the host genome. This step is performed using BBMap aligner.

Note: As described in this post, simply mapping reads to host genome, might lead to false positives, i.e. reads that are bacterial in origin, but nevertheless mapped to host genome. Removal of these reads might have a negative effect on the quality of the assemblies. In case of human host, this can be avoided by using this masked genome. The masking procedure is described in the post linked above. However, this is not available for other host genomes. Unmasked references can be downloaded from NCBI, Ensembl, UCSC. Be sure to keep track of the genome version you are using. Genomes for commonly analyzed organisms can also be downloaded from Illumina iGenomes.

Example Command

```
bbmap.sh -Xmx23g usejni=t threads=20 overwrite=t qin=33 minid=0.95 maxindel=3␣
↪bwr=0.16 bw=12 quickmatch fast \
minhits=2 path=host_bbmap_ref qtrim=rl trimq=15 untrim in1=in.1.fq.gz in2=in.2.
↪fq.gz outu1=out.1.fq.gz \
outu2=out.2.fq.gz outm=out.host.matched.fq.gz 2>> removeHost.log
```

This step has to be repeated for singleton sequences generated in the QC step:

```
bbmap.sh -Xmx23g usejni=t threads=24 overwrite=t qin=33 minid=0.95 maxindel=3 \
bwr=0.16 bw=12 quickmatch fast    minhits=2 \
path=host_bbmap_ref qtrim=rl trimq=15 untrim in=in.s.fq.gz outu=out.s.fq.gz \
outm=out.s.host.matched.fq.gz 2>> out.rmHost.log
```

`qin`	Set to 33 or 64 to specify input quality value ASCII offset. 33 is Sanger, 64 is old Solexa. Could be left unspecified (default=auto).
`minid`	Approximate minimum alignment identity to look for.
`maxindel`	Don't look for indels longer than this. Lower is faster.
`bwr`	If above zero, restrict alignment band to this fraction of read length. Faster but less accurate.
`bw`	Set the bandwidth directly.
`qickmatc`	Generate cigar strings more quickly.
`fast`	Sets other paramters to run faster, at reduced sensitivity.
`minhits`	Minimum number of seed hits required for candidate sites.
`path`	Specify the location to write the index.
`qtrim`	Quality-trim ends before mapping.
`trimq`	Trim regions with average quality below this.
`untrim`	Undo trimming after mapping.
`in`	Primary reads input.
`outu`	Write only unmapped reads to this file.
`outm`	Write only mapped reads, that fail filters to this file.

Important: This command will **NOT** remove all of the host sequences from your sample. The main puropose of the host removal as described here, is to improve metagenome assemblies, not to eliminate all of the host sequences, i.e. if you're working with humand data, **some human reads might still be present** in your samples.

Normalization

This step normalizes the coverage by down-sampling reads over high-coverage areas. This step is only necessary for very large metagenomic samples in order to make the assembly computationally tractable. An example using BBNorm is shown below. As above this step needs to be repeated for the singletons.

Example Command

```
bbnorm.sh -Xmx{memory_limit}G threads={threads} extra=s.fq.gz in1=r1.fq.gz \
in2=r2.fq.gz out1=output_1.fq.gz out2=output_2.fq.gz target=40 mindepth=0␣
↪hist=output.hist \
peaks=output.peaks &> pe_norm.log; \

bbnorm.sh -Xmx{memory_limit}G threads={threads} extra=r1.fq.gz,r2.fq.gz \
in=s.fq.gz out=output_s.fq.gz target=40 mindepth=0 hist=output.hist2 \
peaks=output.peaks2 &> s_norm.log
```

-Xmx	This will be passed to Java to set memory usage.
threads	Set to number of threads desired.
extra	For the kmer table: Additional files to use for input, but not for output.
in1	Path to the forward reads.
in2	Path to the reverse reads.
out1	Normalized forward reads.
out2	Normalized reverse reads.
target	Target normalization depth.
mindepth	Kmers with depth below this number will not be included when calculating the depth of a read.
hist	Specify a file to write the input kmer depth histogram.
peaks	Write the peaks to this file.

Pair-read Merging

Merging refers to merging two overlapping reads into one. This is recommended for amplicon data, mOTUs profiling and metagenomic assembly. We do not usually merge the reads for isolate genome assembly. This can be done using BBMerge.

Example Command

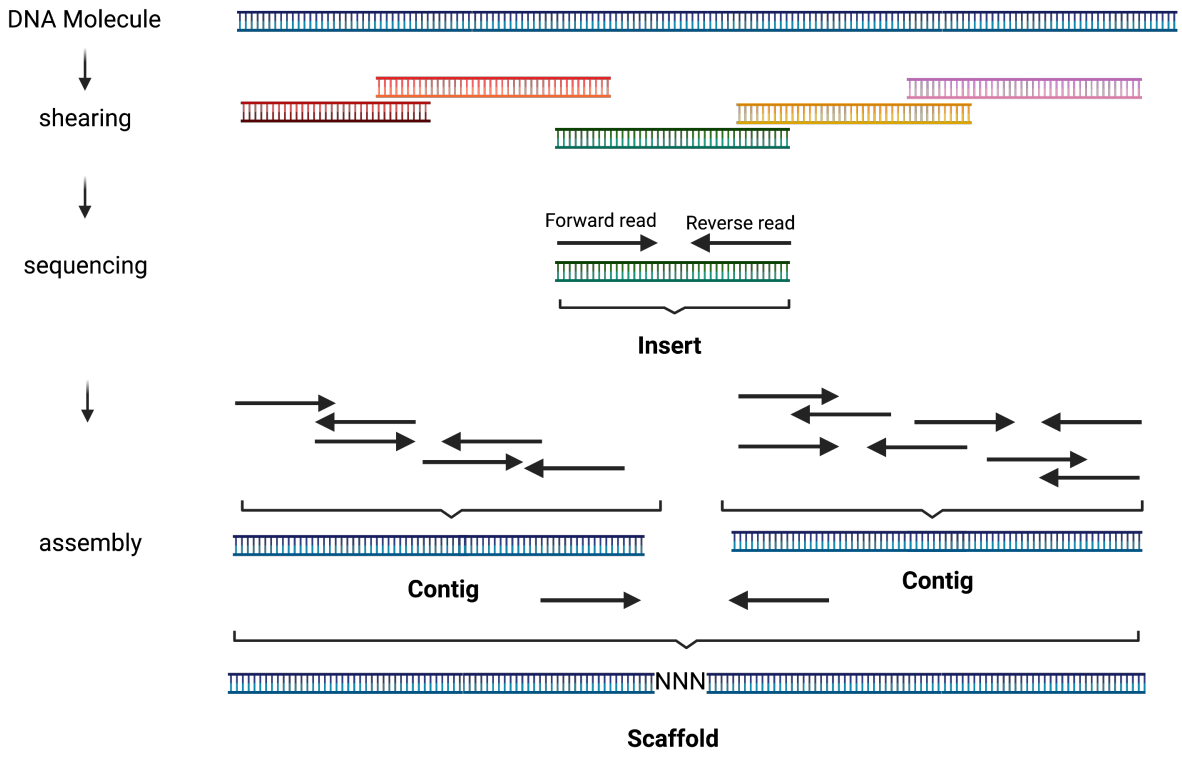
```
bbmerge.sh -Xmx32G threads=32 in1=Sample1_R1.fq.gz in2=Sample1_R2.fq.gz out=Sample1.
↪m.fq.gz \
outu1=Sample1.merge.R1.fq.gz outu2=Sample1.merge.R2.fq.gz minoverlap=16 usejni=t \
ihist=Sample1.merge.hist &> merge.log
```

-Xmx	This will be passed to Java to set memory usage.
threads	Set to number of threads desired.
in1	Path to the forward reads.
in2	Path to the reverse reads.
out	File for merged reads.
outu1	File for forward unmerged reads.
outu2	File for reverse unmerged reads.
minoverlap	Minimum number of overlapping bases to allow merging.
ihist	Insert length histogram output file.
usejni	Do overlapping in C code, which is faster. Requires compiling the C code.

6.2 Genome Assembly

Protocol provided by Anna Sintsova.

Over the recent years, bacterial whole genome sequencing has become an indispensable tool for microbiologists. While powerful, short read sequencing technologies only allow assembly of draft genomes (i.e. assembly consisting of multiple scaffolds). As illustrated below, during whole genome shotgun sequencing, DNA is randomly sheared into inserts of known size distribution and sequenced. If paired-end sequencing is used, two DNA sequences (reads) are generated - one from each end of a DNA fragment). The assemblers look for overlaps between sequencing reads to stitch them together into contigs. The contigs can then sometimes be linked together into longer scaffolds (for example with information from mate-pair reads).



Note: Long read sequencing (with PacBio or Nanopore) offers creation of complete circularized bacterial genomes. However, the bioinformatics methods for this are still in development. They are likely to change as technology develops, and the standard protocols are less well established (See this genome assembly guide for current suggestions).

6.2.1 Isolate genome assembly using short reads

Note: Sample data for this section can be found `here`. The conda environment specifications are `here`. See the *Tutorials* section for instructions on how to unpack the data and create the conda environment. After unpacking the data, you should have a set of forward (Sample1_R1.fq.gz) and reverse (Sample1_R2.fq.gz) reads. These reads have already been through the *Data Preprocessing* workflow and can be used directly for genome assembly. (Note: The included files adapters.fa and phix174_ill.ref.fa.gz are not needed here.)

1. **Data Preprocessing**. Before proceeding to the assembly, it is important to preprocess the raw sequencing data. Standard preprocessing protocols are described in *Data Preprocessing*. In addition to standard quality control and adapter trimming, we also suggest normalization with bbnorm.sh and merging (see *Data Preprocessing* for more details). Besides the common preprocessing steps, we usually run mOTUs on the cleaned sequencing reads, to check for sample contamination or mis-labelling (both occur more frequently than you would expect). For more details please check the *Taxonomic Profiling of Metagenomes* section.
2. **Genome Assembly**. Following data preprocessing, we assemble the cleaned reads using SPAdes. While SPAdes generated scaffolds using paired end data (i.e. no mate-pair libraries), there will be few differences between scaffolds.fasta and contigs.fasta. We use scaffolds for all subsequent analysis.

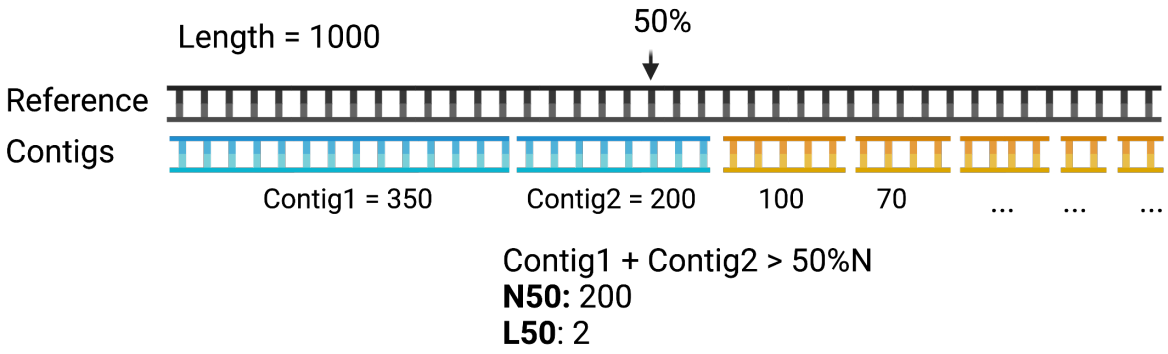
Example Command

```
mkdir sample1_assembly; \
spades.py -t 4 --isolate --pe1-1 Sample1_R1.fq.gz \
--pe1-2 Sample1_R2.fq.gz  -o sample1_assembly
```

`-t`	Number of threads
`--isolate`	Use SPAdes isolate mode
`--pe1-1`	Forward reads
`--pe1-2`	Reverse reads
`-o`	Specify output directory

3. **Assembly Quality Control**. Following assembly, we generate assembly statistics using assembly-stats, and filter out scaffolds that are < 500 bp in length. The script we use for contig/scaffold filtering can be found here: `scaffold_filter.py`. Alternatively, the metrics to evaluate genome quality can be also calculated using QUAST. The output will contain information on the number of contigs, the largest contig, total length of the assembly, GC%, N50, L50 and others. If reference genome assembly is available, QUAST will also assess misassemblies and try to categorize them.

Note: **N50 and L50**: Given a set of contigs sorted by length in descending order, L50 is the smallest number of contigs, whose length adds up to at least 50% of the genome length. N50 is the length of the smallest contig included in L50 (i.e. if L50 is 2, N50 will be length of the 2nd contig).



Example Command for filtering and stats:

```
python scaffold_filter.py Sample1 scaffolds \
sample1_assembly/scaffolds.fasta sample1_assembly ISO;
assembly-stats -l 500 \
-t sample1_assembly/Sample1.scaffolds.min500.fasta > \
sample1_assembly/Sample1.assembly.stats
```

`Sample1`	Sample name
`scaffolds`	Sequence type (can be contigs, scaffolds or transcripts)
`sample1_assembly/scaffolds.fasta`	Input assembly to filter
`sample1_assembly`	Prefix for the output file
`ISO`	Type of assembly (ISO for metagenomics or META for isolate genomes

Example QUAST Command:

```
quast.py sample1_assembly/Sample1.scaffolds.min500.fasta \
-1 Sample1_R1.fq.gz -2 Sample1_R2.fq.gz -o sample1_assembly
```

Options Explained

`-1` (or `--pe1`)	File with forward paired-end reads in FASTQ format (files compressed with gzip are allowed).
`-2` (or `--pe2`)	File with reverse paired-end reads in FASTQ format (files compressed with gzip are allowed).
`-o`	Specify output directory

4. **Gene Calling and Annotation**. Genome annotation is locating of genomic features (i.e. genes, rRNAs, tRNAs, etc) in the newly assembled genomes, and for protein coding genes, describing the putative gene product. The example below shows how this can be accomplished using prokka. More information about prokka can be found here.

Example Command

```
prokka --outdir sample1_assembly --locustag sample1 \
--compliant --prefix sample1 sample1_assembly/Sample1.scaffolds.min500.fasta --force
```

Options Explained

`--outdir`	Output folder
`--locustag`	Locus tag prefix
`--compliant`	Force Genbank/ENA/DDJB compliance: `--addgenes --mincontiglen 200 --centre XXX`
`--addgenes`	Add 'gene' features for each 'CDS' feature
`--mincontiglen`	Minimum contig size [NCBI needs 200]
`--centre`	Sequencing centre ID.
`--prefix`	Filename output prefix
`--force`	Force overwriting existing output folder

6.2.2 Alternative Approach

Alternatively, we had good results building short-read assemblies with Unicycler. However, these are not significantly different from SPAdes assemblies described above (not surprising, since Unicycler runs SPAdes under the hood). In addition, Unicycler is not being actively developed, and does not support the latest version of SPAdes. Please see Ryan Wick's Genome Assembly Guide for example command.

6.3 Metagenomic Assembly

Protocol provided by Hans-Joachim Ruscheweyh.

Technical advances in sequencing technologies in recent decades have allowed detailed investigation of complex microbial communities without the need for cultivation. Sequencing of microbial DNA extracted directly from environmental or host-associated samples have provided key information on microbial community composition. These studies have also allowed gene-level characterization of microbiomes as the first step to understanding the communities' functional potential. Furthermore, algorithmic improvements, as well as increased availability of computational resources, make it now possible to reconstruct whole genomes from metagenomic samples (metagenome-assembled genomes (MAGs)). Methods for microbial community composition analysis are discussed in *Taxonomic Profiling of Metagenomes*. Here, we describe building *Metagenomic Assembly* as well as building *Gene Catalogs* and *MAGs* from metagenomic data.

Note: Sample data and conda environment file for this section can be found `here`. See the *Tutorials* section for instructions on how to unpack the data and create the conda environment. After unpacking the data, run `cd metag_test`, and you should see conda specifications `metag.yaml` and a `reads` directory containing a set of forward (.1.fq.gz) and reverse (.2.fq.gz) reads for 3 metagenomic samples. These reads have already been through the *Data Preprocessing* workflow and can be used directly for metagenomic assembly. There should also be `scaffolds_filter.py`.

6.3.1 Metagenomic Assembly

1. **Data Preprocessing**. Before proceeding to the assembly, it is important to preprocess the raw sequencing data. Standard preprocessing protocols are described in *Data Preprocessing*. In addition to standard quality control and adapter trimming, we also suggest normalization with **bbnorm.sh** and merging of paired-end reads (see *Data Preprocessing* for more details).
2. **Metagenomic Assembly**. Following data preprocessing, we use clean reads to perform a metagenomic assembly using **metaSPAdes**. metaSPAdes is part of the SPAdes assembly toolkit. Following the assembly, we generate some assembly statistics using **assembly-stats**, and filter out contigs that are < 1 kbp in length. The script we use for scaffold filtering can be found here: `scaffold_filter.py`. It is also included in the test dataset for this section.

Assembly:

```
mkdir metag_assembly
for i in 1 2 3
  do
    mkdir metag_assembly/metag$i
    metaspades.py -t 4 -m 10 --only-assembler \
    --pe1-1 reads/metag$i.1.fq.gz \
    --pe1-2 reads/metag$i.2.fq.gz \
    -o metag_assembly/metag$i
  done
```

`-t`	Number of threads
`-m`	Set memory limit in Gb; spades will terminate if that limit is reached
`--only-assembler`	Run assembly module only (spades can also perform read error correction, this step will be skipped)
`--pe1-1`	Forward reads
`--pe1-2`	Reverse reads
`--pe1-s`	Unpaired reads
`--pe1-m`	Merged reads
`-o`	Specify output directory

> **Computational Resources** needed for metagenomic assembly will vary significantly between datasets. In general, metagenomic assembly requires a lot of memory (usually > 100 Gb). You can use multiple threads (16-32) to speed up the assembly. Because test data set provided is very small, merging of the pair-end reads was not necessary (see *Data Preprocessing*). It is helpful when working with real data - don't forget to include the merged and singleton files with `--pe1-m` and `--pe1-s` options.

Filtering:

Assumes `scaffolds_filter.py` is in `metag_test`

```
cd metag_assembly
for i in 1 2 3
  do
    python ../scaffold_filter.py metag$i scaffolds metag$i/scaffolds.fasta␣
↪metag$i META
  done
```

`metag1`	Sample name
`scaffolds`	Sequence type (can be contigs, scaffolds or transcripts)
`metag1/scaffolds.fasta`	Input assembly to filter
`metag1`	Prefix for the output file
`META`	Type of assembly (META for metagenomics or ISO for isolate genomes)

Stats:

```
for i in 1 2 3
  do
    assembly-stats -l 500 -t <(cat metag$i/metag$i.scaffolds.min500.fasta) \
    > metag$i/metag$i.assembly.stats
  done
```

`-l`	Minimum length cutoff for each sequence
`-t`	Print tab-delimited output

3. The metagenomic scaffolds generated in step 2 can now be used to build and/or profile *Gene Catalogs* or to construct *MAGs*.

6.3.2 Gene Catalogs

Gene catalog generation and profiling (i.e. gene abundance estimation) can provide important insights into the community's structure, diversity and functional potential. This analysis could also identify relationships between genetic composition and environmental factors, as well as disease associations.

Note: Integrated catalogs of reference genes have been generated for many ecosystems (e.g. ocean, human gut, and many others) and might be a good starting point for the analysis.

Building

This protocol will allow you to create a de novo gene catalog from your metagenomic samples.

1. **Gene calling**. We use **prodigal** to extract protein-coding genes from metagenomic assemblies (using **scaffolds** >= 500 bp as input). Prodigal has different gene prediction modes with single genome mode as default. To run prodigal on metagenomic data, we add the `-p meta` option. This will produce a fasta file with amino acid sequences (.faa), nucleotide sequences (.fna) for each gene, as well as an annotation file (.gff).

Gene Calling

Assumes you are in the `metag_assembly` directory.

```
for i in 1 2 3
  do
    prodigal -a metag$i/metag$i.faa -d metag$i/metag$i.fna -f gff \
    -o metag$i/metag$i.gff -c -q -p meta \
    -i metag$i/metag$i.scaffolds.min500.fasta
  done
```

`-a`	Specify protein translations file
`-d`	Specify nucleotide sequences file
`-f`	Specify output format: gbk: Genbank-like format (Default); gff: GFF format; sqn: Sequin feature table format; sco: Simple coordinate output
`-o`	Specify output file, default stdout
`-c`	Closed ends, do not allow partial genes at edges of sequence
`-q`	Run quietly (suppress logging output)
`-p`	Specify mode: single or meta
`-i`	Input FASTA or Genbank file

2. **Gene de-replication**. At this point gene-nucleotide sequences from all samples are concatenated together and duplicated sequences are removed from the catalog. For this, genes are clustered at 95% identity and 90% coverage of the shorter gene using CD-HIT. The longest gene sequence from each cluster is then used as a reference sequence for this gene.

Clustering

```
cd ..
mkdir gene_catalog
cat metag_assembly/metag*/metag*fna > gene_catalog/gene_catalog_all.fna
cat metag_assembly/metag*/metag*faa > gene_catalog/gene_catalog_all.faa
cd gene_catalog
mkdir cdhit9590
cd-hit-est -i gene_catalog_all.fna -o cdhit9590/gene_catalog_cdhit9590.fasta \
-c 0.95 -T 64 -M 0 -G 0 -aS 0.9 -g 1 -r 1 -d 0
```

`-i`	Input filename in fasta format, required
`-o`	Output filename, required
`-c`	Sequence identity threshold, default 0.9
`-T`	Number of threads, default 1; with 0, all CPUs will be used
`-M`	Memory limit (in MB) for the program, default 800; 0 for unlimited
`-G`	Use global sequence identity, default 1; if set to 0, then use local sequence identity, don't use -G 0 unless you use alignment coverage controls (e.g. options -aS)
`-aS`	Alignment coverage for the shorter sequence, default 0.0; if set to 0.9, the alignment must cover 90% of the sequence
`-g`	1 or 0, default 0; by cd-hit's default algorithm, a sequence is clustered to the first cluster that meets the threshold (fast cluster); if set to 1, the program will cluster it into the most similar cluster that meets the threshold (accurate but slow mode); either 1 or 0 won't change the representatives of final clusters
`-r`	1 or 0, default 1; by default do both +/+ & +/- alignments; if set to 0, only +/+ strand alignment
`-d`	length of description in .clstr file, default 20; if set to 0, it takes the fasta defline and stops at first space

The fasta file generated by CD-HIT will contain a representative sequence for each gene cluster. To extract protein sequences for each gene in the catalog, we first extract all the sequence identifiers from the CD-HIT output file and use seqtk `subseq` command to extract these sequences from `gene_catalog_all.faa`. This file can be then used for downstream analysis (ex. KEGG annotations).

```
grep "^>" cdhit9590/gene_catalog_cdhit9590.fasta | \
cut -f 2 -d ">" | \
cut -f 1 -d " " > cdhit9590/cdhit9590.headers
seqtk subseq gene_catalog_all.faa cdhit9590/cdhit9590.headers \
> cdhit9590/gene_catalog_cdhit9590.faa
```

Gene Catalog Profiling

Warning: Incomplete protocol

This protocol allows quantification of genes in a gene catalog for each metagenomic sample.

1. **Read alignment**. In the first step, (cleaned) sequencing reads are mapped back to the gene catalog using BWA aligner. Note that forward, reverse, singleton and merged reads are mapped separately and are then filtered and merged in a later step.

Alignment

Make sure you are back in `metag_test` directory. Note that test data do not include merged and singleton files. If you have those, do not forget to align those separately as well.

```
mkdir alignments
bwa index gene_catalog/cdhit9590/gene_catalog_cdhit9590.fasta

bwa mem -a -t 4 gene_catalog/cdhit9590/gene_catalog_cdhit9590.fasta reads/metag1.1.fq.gz␣
↪\
| samtools view -F 4 -bh - > alignments/metag1.r1.bam

bwa mem -a -t 4 gene_catalog/cdhit9590/gene_catalog_cdhit9590.fasta reads/metag1.2.fq.gz␣
↪\
| samtools view -F 4 -bh - > alignments/metag1.r2.bam
```

BWA:

`-a`	Output all found alignments for single-end or unpaired paired-end reads, these alignments will be flagged as secondary alignments
`-t`	Number of threads

samtools:

`-F *FLAG*`	Do not output alignments with any bits set in *FLAG* present in the FLAG field. When *FLAG* is 4, do not output unmapped reads.
`-b`	Output in the BAM format
`-h`	Include the header in the output

2. **Filtering the alignment files**. To make sure that quantification of gene abundance relies only on high confidence alignments, the alignment files are first filtered to only include alignments with length > 45 nt and percent identity > 95%.

3. **Counting gene abundance**. This step counts the number of reads aligned to each gene for each of the samples.

Important: We're currently working on a tool that can merge and filter alignment files, as well as quantify gene abundances. Stay tuned! In the meanwhile, please contact us to learn more.

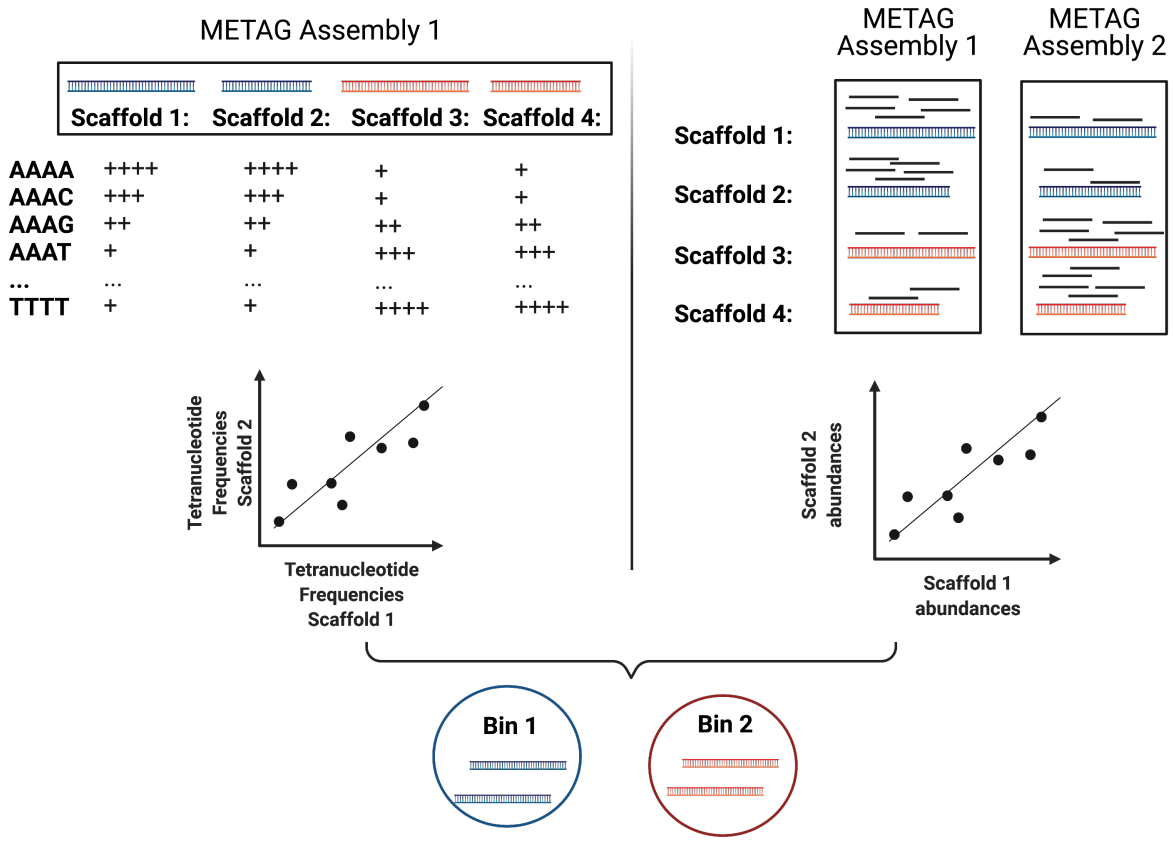
Note: Gene catalogs and collections of MAGs are often used to infer abundance of microorganisms in metagenomic samples, however none are comprehensive and will miss some members (or the majority) of the microbial community. It is important to estimate what percentage of the microbial community is represented in a gene catalog or a collection of MAGs. This is evaluated using mapping rates: number of mapped reads (after alignment and filtering, as described in *Gene Catalog Profiling*) divided by total number of quality-control reads.

Important: Per-cell normalization. Metagenomic profiles should be normalized to relative cell numbers in the sample. This can be achieved by dividing the gene abundances by the median abundance of 10 universal single-copy phylogenetic marker genes (MGs).

6.3.3 MAGs

The Holy Grail of metagenomics is to be able to assemble individual microbial genomes from complex community samples. However, short-read assemblers fail to reconstruct complete genomes. For that reason, binning approaches have been developed to facilitate creation of Metagenome Assembled Genomes (MAGs).

MAG reconstruction algorithms have to decipher which of the scaffolds generated during *Metagenomic Assembly* belong to the same organism (referred to as bin). While different binning approaches have been described, here we use MetaBAT2 for MAG reconstruction. As shown in the figure below, MetaBAT2 uses scaffolds' tetranucleotide frequencies and abundances to group scaffolds into bins.



In this (very) simplified example, the blue scaffolds show similar tetranucleotide frequencies and similar abundances (across multiple samples), and consequently end up binned together, and separately from the red scaffolds.

MAG Building

This workflow starts with size-filtered metaSPAdes assembled scaffolds (resulted from *Metagenomic Assembly*). Note that for MAG building we are using >= 1000 bp **scaffolds**.

1. **All-to-all alignment**. In this step, quality controlled reads for each of the metagenomic samples are mapped to each of the metagenomic assemblies using BWA. Here, we use `-a` to allow mapping to secondary sites. Note that merged, singleton, forward and reverse reads are all aligned separately, and are later merged into a single `bam` file.

Important: For MAG construction, the generated alignment files are filtered to only include alignments that are at least 45 nucleotides long, with an identity of >= 95 and covering 80% of the read sequence. The alignment filtering

was done with a tool we are building in the lab and is not included in the example command below. Please contact us to learn more.

Create BWA index for each assembly:

Make sure you are in the `metag_test` directory and have run metagenomic assembly steps described above.

```
for i in 1 2 3
  do
    bwa index metag_assembly/metag$i/metag$i.scaffolds.min1000.fasta
  done
```

Mapping every sample to every assembly:

```
mkdir -p alignments
for i in 1 2 3
  do
    for j in 1 2 3
      do
        bwa mem -a -t 16 metag_assembly/metag$i/metag$i.scaffolds.min1000.
↪fasta reads/metag$j.1.fq.gz \
        | samtools view -F 4 -bh - | samtools sort -O bam -@ 4 -m 4G >␣
↪alignments/metag"$j"_to_metag"$i".1.bam
        bwa mem -a -t 16 metag_assembly/metag$i/metag$i.scaffolds.min1000.
↪fasta reads/metag$j.2.fq.gz \
        | samtools view -F 4 -bh - |samtools sort -O bam -@ 4 -m 4G >␣
↪alignments/metag"$j"_to_metag"$i".2.bam
        samtools merge alignments/metag"$j"_to_metag"$i".bam \
        alignments/metag"$j"_to_metag"$i".1.bam alignments/metag"$j"_to_metag"
↪$i".2.bam
      done
  done
```

BWA:

`-a`	Output all found alignments for single-end or unpaired paired-end reads, these alignments will be flagged as secondary alignments
`-t`	Number of threads

samtools:

`-F *FLAG*`	Do not output alignments with any bits set in *FLAG* present in the FLAG field. When *FLAG* is 4, do not output unmapped reads.
`-b`	Output in the BAM format
`-h`	Include the header in the output

Important: Computational Resources: Depending on the size of the dataset, this step would require significant computational resources.

2. **Within- and between-sample abundance correlation for each contig**. MetaBAT2 provides *jgi_summarize_bam_contig_depth* script that allows quantification of within- and between-sample abun-

dances for each scaffold. Here, we generate an abundance (depth) file for each metagenomic assembly by providing the alignment files generated using this assembly. This depth file will be used by MetaBAT2 in the next step for scaffold binning.

Depth calculation:

```
for i in 1 2 3
  do
    jgi_summarize_bam_contig_depths --outputDepth alignments/metag$i.depth \
    alignments/metag*_to_metag"$i".bam
  done
```

Note: Binning with MetaBAT2 can also be accomplished without between-sample abundance correlation, however this step significantly improves the quality of reconstructed MAGs, and, in our opinion, is worth the computational burden of all-to-all alignment.

3. **Metagenomic Binning**. Finally, we run MetaBAT2 to bin the metagenomic assemblies using depth files generated in the previous step.

```
mkdir mags
for i in 1 2 3
  do
    metabat2 -i metag_assembly/metag$i/metag$i.scaffolds.min1000.fasta -a␣
↪alignments/metag$i.depth \
    -o mags/metag$i --minContig 2000 \
    --maxEdges 500 -x 1 --minClsSize 200000 --saveCls -v
  done
```

4. **Quality Control**. After MAG reconstruction, it is important to estimate how well the binning perform. CheckM places each bin on a reference phylogenetic tree and evaluates genome quality by looking at a set of clade-specific marker genes. CheckM outputs completeness (estimation of fraction of genome present), contamination (percentage of foreign scaffolds), and strain heterogeneity (high strain heterogeneity would suggest that contamination is due to presence of closely related strains in your sample).

Warning: Linux only!

```
checkm lineage_wf mags mags -x fa \
-f mags/checkm_summary.txt --tab_table
```

MAG building: low abundance metagenome/pooled assembly

Warning: Under construction

6.4 Amplicon Sequencing

Protocol provided by the Sunagawa group.

Contact: Anna Sintsova and Hans-Joachim Ruscheweyh

Important: We use DADA2 pipeline for 16S Amplicon analysis. DADA2 tutorial provides a good introduction. If using Illumina data with binned quality scores, please checkout this discussion.

16S amplicon sequencing remains one of the most widely used methods for the analysis of microbiome community composition. Taxonomic composition, as well as alpha and beta diversity metrics, can provide novel biological insight and show association with environmental conditions or clinical variables. 16s rRNA gene is about 1500 bp long and encodes rRNA component of the small subunit of prokaryotic ribosome. The gene is composed of highly conserved and 9 variable regions. This allows us to use the conserved regions as primer binding sites, and variable regions for taxonomic classification.

Although 16S is a cost-effective and powerful technique, a number of factors can influence the outcomes of the analysis, hindering comparisons between studies. The outcomes can be influenced by:

- sampling and sample storage strategy
- primer annealing efficiency
- which variable region is targeted
- library preparation and sequencing protocols
- bioinformatic processing pipelines (OTUs vs ASVs)
- database used for taxonomic annotation

6.4.1 16S and 18S Sequencing primers

Below you can find a table listing commonly used primers for 16S analysis.

As described in Walters et al, 515f-806r bacterial/archaeal primer pair, traditionally used by the Earth Microbiome Project, has been shown to be biased against specific archeal and bacterial clades. Parada et al. and Apprill et al. have modified the 515f/806r 16S rRNA gene primer pair to reduce these biases.

V-Region	Primer Names	Primer Sequences	Specificity	Size	Reference
V1-V3	27F/534R	AGAGTTTGATY ATTACCGCGGC	Bacteria & Archaea	507	Walker et al.
V3-V4	341F/785R	CCTACGGGNGC GACTACHVGGC	Bacteria & Archaea	465	Klindworth et al.
V4	515F/806R	GTGCCAGCMG GGACTACHVGC	Bacteria & Archaea	291	Caporaso et al.
V4	515F-modified/806R	GTGYCAGCMG GGACTACHVGC	Bacteria & Archaea	291	Parada et al.
V4	515F/806R-modified	GTGCCAGCMG GGACTACNVGC	Bacteria & Archaea	291	Apprill et al.
V4-V5	515F/926R	GTGCCAGCMG CCGYCAATTYM	Bacteria & Archaea & Eukaryotes	411	Parada et al.

6.4.2 16S Data Analysis

1. **Removing adapters and splitting reads in forward/reverse orientation**. It is essential to remove adapter sequences for DADA2 pipeline to work properly. For this purpose we use cutadapt. We run cutadapt multiple times to remove adapters that were added to the sequence multiple times. While this rarely happens, this step saves some work in the downstream analysis. We also split forward and reverse inserts (e.g. 515-926 inserts from 926-515 inserts), as the sequencing protocol produces both orientations.

Example command:

```
cutadapt -O 12 --discard-untrimmed -g {fwd_primer} -G {rev_primer} -o {output.
↪r1tmp} -p {output.r2tmp} {input.r1} {input.r2} -j {threads} --pair-adapters -
```

(continues on next page)

(continued from previous page)

```
↪-minimum-length 75
    cutadapt -O 12 --times 5 -g {fwd_primer} -o {output.r1tmp2} -j {threads}
↪{output.r1tmp}
    cutadapt -O 12 --times 5 -g {rev_primer} -o {output.r2tmp2} -j {threads}
↪{output.r2tmp}
    cutadapt -o {output.r1} -p {output.r2} {output.r1tmp2} {output.r2tmp2} -j
↪{threads} --minimum-length {minlength}
```

2. **Filter and trim the reads**. We next trim low quality bases, this is important for DADA2 merging to work. This can be accomplished with DADA2 *filterAndTrim* function.

Important: How much do I truncate? It is recommended to look at the quality profile of your data, and, while ensuring that you have enough sequence that your forward/reverse reads still overlap enough to merge (leave at least 10 nt overlap for merging), truncate off as many of the nucleotides that come after quality crashes as you can. The quality of the reverse reads usually deteriorates faster, thus reverse reads usually need more trimming than the forward reads.

Example command:

```
library(dada2);
packageVersion("dada2")

filterAndTrim(fwd={infqgz1}, filt={outfqgz1}, rev={infqgz2}, filt.rev=
↪{outfqgz2}, matchIDs=TRUE, maxEE={maxee}, truncQ={truncq}, maxN=0, rm.
↪phix=TRUE, compress=compress, verbose=TRUE, multithread={threads}, minLen=
↪{minlen}, truncLen = c({trunclen_r1}, {trunclen_r2}))
```

3. **Learning Error Rates**. DADA2 algorithm needs to first estimate error rates from the data. This should be done separately for samples sequenced on different lanes.

Warning: New Illumina sequencing data (e.g. NovaSeq) provides only binned quality scores (see *Data Preprocessing* for more details). This created a problem for DADA2 error learning step. This is an ongoing issue, and is discussed in detailed here and in this tutorial. Below is our current solution to the problem, the best solution might be dataset specific.

Here we define a modified error function that maintains monotonicity even with binned quality reads:

```
loessErrfun_mod <- function (trans) {
   qq <- as.numeric(colnames(trans))
   est <- matrix(0, nrow = 0, ncol = length(qq))
   for (nti in c("A", "C", "G", "T")) {
     for (ntj in c("A", "C", "G", "T")) {
       if (nti != ntj) {
         errs <- trans[paste0(nti, "2", ntj), ]
         tot <- colSums(trans[paste0(nti, "2", c("A","C", "G", "T")), ])
         rlogp <- log10((errs + 1)/tot)
         rlogp[is.infinite(rlogp)] <- NA
         df <- data.frame(q = qq, errs = errs, tot = tot,
                          rlogp = rlogp)
         mod.lo <- loess(rlogp ~ q, df, weights = log10(tot),span = 2)
         pred <- predict(mod.lo, qq)
```

(continues on next page)

(continued from previous page)

```
        maxrli <- max(which(!is.na(pred)))
        minrli <- min(which(!is.na(pred)))
        pred[seq_along(pred) > maxrli] <- pred[[maxrli]]
        pred[seq_along(pred) < minrli] <- pred[[minrli]]
        est <- rbind(est, 10^pred)
       } }
       }
  MAX_ERROR_RATE <- 0.25
  MIN_ERROR_RATE <- 1e-07
  est[est > MAX_ERROR_RATE] <- MAX_ERROR_RATE
  est[est < MIN_ERROR_RATE] <- MIN_ERROR_RATE
  err <- rbind(1 - colSums(est[1:3, ]), est[1:3, ], est[4,
                                          ], 1 - colSums(est[4:6, ])
  colSums(est[7:9, ]), est[9, ], est[10:12, ], 1 - colSums(est[10:1
   , est[5:6, ], est[7:8, ], 1 -
   2,
  rownames(err) <- paste0(rep(c("A", "C", "G", "T"), each = 4),
                             "2", c("A", "C", "G", "T"))
  colnames(err) <- colnames(trans)
  return(err)
   }
```

The error rates can than be modeled as follows:

```
samplefile <- "samplefile_r1_fw"
outfile <- "samplefile_r1_fw.errors.rds"
outfile.plot <- paste(outfile, '.pdf', sep = '')
threads <- 8
nbases <- 1e8
]))
sample.files <- read.csv(samplefile, header=FALSE, sep='\t', stringsAsFactors␣
↪= FA
LSE)[2]
s.f <- sample.files$V2
err <- learnErrors(s.f, nbases=nbases, multithread=threads, randomize=TRUE,␣
↪verbos
e = 1, errorEstimationFunction = loessErrfun_mod)
saveRDS(err, file = outfile)
plot <- plotErrors(err,nominalQ=TRUE)
ggsave(outfile.plot, plot = plot)
```

4. **Sample Inference**. This is the core function of DADA2. Each read, its abundance and its quality is tested to determine whether it is an actual, error-free ASV or a spurious sequence with errors. The error function from the previous step is reused. DADA2 is using the error model to infer unique ASVs in each sample. This is also done separately for samples from different lanes. You can read more about the core sample inference algorithm in the DADA2 paper.

Example command:

```
library(dada2); packageVersion("dada2")

sample.files <- read.csv({samplefile}, header=FALSE, sep='\t',␣
↪stringsAsFactors = FALSE)[2]
```

(continues on next page)

(continued from previous page)

```
s.f <- sort(sample.files$V2)
sample.names <- sapply(strsplit(basename(s.f), "_R"), `[`, 1)
#if(!identical(sample.names.r1, sample.names.r2)) stop("Forward and reverse␣
↪files do not match.")
names(s.f) <- sample.names
err <- readRDS({err.rds})
dd <- dada(s.f, err=err, pool='pseudo', multithread = threads,␣
↪errorEstimationFunction = loessErrfun_mod)

seqtab <- makeSequenceTable(dd)
saveRDS(seqtab, file = {outfile.tab})
saveRDS(dd, file = {outfile.dd})
```

5. **Read Merging**. Now reads can be merged into inserts. The forward subsample is merged in standard orientation. The reverse subsample is merged in inverse orientation. That way, all inserts will have the same orientation after this step.

Example command:

```
library(dada2); packageVersion("dada2")
sample.files.r1 <- read.csv({samplefile.r1}, header=FALSE, sep='\t',␣
↪stringsAsFactors = FALSE)[2]
sample.files.r2 <- read.csv({samplefile.r2}, header=FALSE, sep='\t',␣
↪stringsAsFactors = FALSE)[2]
s.f.r1 <- sort(sample.files.r1$V2)
s.f.r2 <- sort(sample.files.r2$V2)
sample.names.r1 <- sapply(strsplit(basename(s.f.r1), "_R1"), `[`, 1)
sample.names.r2 <- sapply(strsplit(basename(s.f.r2), "_R2"), `[`, 1)
if(!identical(sample.names.r1, sample.names.r2)) stop("Forward and reverse␣
↪files do not match.")
names(s.f.r1) <- sample.names.r1
names(s.f.r2) <- sample.names.r2
dd.r1 <- readRDS({infile.r1})
dd.r2 <- readRDS({infile.r2})
mergers <- mergePairs(dd.r1, s.f.r1, dd.r2, s.f.r2, verbose = TRUE)
seqtab.m <- makeSequenceTable(mergers)
saveRDS(mergers, file = {outfile.dd.m})
saveRDS(seqtab.m, file = {outfile.seqtab.m})
```

6. **Chimera Removal**. Chimeras/Bimeras are removed from each sample individually. Remember that each sample consists of 2 subsamples, forward and reverse.

> **Warning:** You should not be losing a lot of reads during the merging and chimera removal steps.

Example command:

```
library(dada2); packageVersion("dada2")
nobim.tab <- removeBimeraDenovo({wbim.tab}, method="pooled", multithread=
↪{threads}, verbose=TRUE)
saveRDS(nobim.tab, file = {nobim.file})
```

Note: **Optional**: remove spurious ASVs. In the next step we merge the individual tables into one big ASV table. Most of the ASVs are spurious (appear in low counts and in only 1 sample). We remove all ASVs that appear < 5 times.

7. **Taxonomic annotation**. Taxonomic annotation is performed using IDTAXA with the training set corresponding to the SILVA database v.138 and a confidence threshold of 40.

Example command:

```
#!/usr/bin/env Rscript
suppressMessages(library(optparse))

# Define arguments
option_list = list(
  make_option(c("-i", "--path_to_seqtab"), type="character", default=NULL,help=
↪"Path to the sequence table file (RDS file containing a matrix with␣
↪sequences as columns and samples as rows)", metavar="character"),
  make_option(c("-s", "--path_to_training_set"), type="character",␣
↪default=NULL,help="Path to the SILVA training set (will be downloaded if it
↪'s not provided)", metavar="character"),
  make_option(c("-c", "--threshold"), type="integer", default=40,help="IdTaxa␣
↪threshold (default = 40)", metavar="integer"),
  make_option(c("-t", "--threads"), type="integer", default=1,help="Number of␣
↪threads (default = 1)", metavar="integer"),
  make_option(c("-o", "--out_path"), type="character", default=NULL,help="Path␣
↪to the output file (table with taxonomy as a tab-delimitted file)", metavar=
↪"character")
);

description<-paste("The program loads an RDS file containing a sequence table␣
↪and assigns the taxonomy of ASVs/OTUs using IDTAXA\n\n")


opt_parser = OptionParser(option_list=option_list,description = description);
opt = parse_args(opt_parser);

if (is.null(opt$path_to_seqtab) | is.null(opt$out_path)){
  print_help(opt_parser)
  stop("At least one argument must be supplied for -i and -o", call.=FALSE)
}


library(DECIPHER)
library(data.table)
library(tidyverse)

path_to_seqtab<-opt$path_to_seqtab
path_to_training_set<-opt$path_to_training_set
threads<-opt$threads
out_path<-opt$out_path
threshold<-opt$threshold

# Check if the training set exists or download it and load it
```

(continues on next page)

(continued from previous page)

```
if (is.null(path_to_training_set)){
  cat("Training set not provided. It will be downloaded\n")
  system(paste("wget --content-disposition -P ./ http://www2.decipher.codes/
↪Classification/TrainingSets/SILVA_SSU_r138_2019.RData",sep=""))
  path_to_training_set<-"SILVA_SSU_r138_2019.RData"
} else{
  cat("Training set already exists. Using local copy\n")
}
load(path_to_training_set)

# Read the RDS file
seqtab<-readRDS(path_to_seqtab)
seqs_fasta<-DNAStringSet(x=as.character(colnames(seqtab)))
names(seqs_fasta)<-as.character(colnames(seqtab))


# Run IDTAXA and parse
annot <- IdTaxa(seqs_fasta, trainingSet=trainingSet, strand="top",␣
↪processors=threads,threshold=threshold)

annot_df<-sapply(annot,function(x){as.data.frame(x) %>%␣
↪mutate(annot=paste(rank,taxon,round(confidence,2),sep=";")) %>%␣
↪summarise(tax=paste(annot,collapse="|"))}) %>%
  unlist() %>%
  as.data.frame() %>%
  rename(tax=".") %>%
  rownames_to_column(var="seq") %>%
  mutate(seq=gsub(".tax$","",seq))

seqtab_annot<-t(seqtab) %>%
  as.data.frame() %>%
  rownames_to_column(var="seq") %>%
  left_join(annot_df,by="seq") %>%
  select(seq,tax,everything())

# Save file
fwrite(seqtab_annot,file=out_path,sep="\t")
```

6.5 Taxonomic Profiling of Metagenomes

Protocol provided by Anna Sintsova.

Taxonomic profiling of complex microbial communities is an essential first step in the investigation of relationship between community composition and environmental and/or health factors. The most common approach to community profiling is amplification and classification the 16S rRNA gene. Methods related to 16S rRNA analysis are discussed in detail in *Amplicon Sequencing*. Recently shotgun metagenomic sequencing has started to replace the amplicon based approaches, as it provides higher resolution information about the microbial community, and resolves some of the biases associated with 16S approach. A number of software tools have been developed to taxonomically profile metagenomic samples. These tools have been benchmarked in recent studies. Here we're going to talk about the use of *mOTUs* and *mTAGs* for taxonomic profiling.

6.5.1 mOTUs

mOTUs determines the composition of metagenomic samples using 10 single copy phylogenetic marker genes and an extensive database consisting of reference genomes, metagenomes and metagenome assembled genomes from 23 different environments. Different use cases and applications are discussed in detail in a recent publication and on mOTUs website. Here we provide a quick reference guide to basic mOTUs functionality.

Note: Please download `sample data` and `conda environment file` for this section if you want to follow along. See the *Tutorials* section for instructions on how to unpack the data and create the conda environment. mOTUs installation requires database download, so expect it to take a little bit of time.

---

1. **Data Preprocessing**. Before taxonomic profiling, it is important to preprocess the raw sequencing data. Standard preprocessing protocols are described in *Data Preprocessing*.

Important: In addition to standard quality control and adapter trimming, we also suggest merging of paired-end reads (see *Data Preprocessing* for more details). Using merged reads increases speed and accuracy.

---

2. **Profile**. Taxonomic profiles for each sample can be generated using mOTUs *profile* command. The output profile will consist of identified mOTUs and their abundance.

```
mkdir motus_profiles
motus profile -f  reads/ERR479298_sub1_R1.fq.gz \
     -r reads/ERR479298_sub1_R2.fq.gz \
     -n ERR479298_sub1 -o motus_profiles/ERR479298_sub1.motus -c -k mOTU -q -p
motus profile -f  reads/ERR479298_sub2_R1.fq.gz \
     -r reads/ERR479298_sub2_R2.fq.gz \
     -n ERR479298_sub2 -o motus_profiles/ERR479298_sub2.motus -c -k mOTU -q -p
```

`-f`	input file(s) for reads in forward orientation
`-r`	input file(s) for reads in reverse orientation
`-s`	input file(s) for unpaired reads (singletons or merged pair end reads)
`-n`	sample name
`-o`	output file name
`-c`	print result as counts instead of relative abundances
`-k`	taxonomic level (kingdom, phylum, class, order, family, genus, mOTU)
`-q`	print the full rank taxonomy
`-p`	print NCBI taxonomy identifiers

Important: Expect mOTU counts (when run with `-c` option) to be relatively small (compared to total number of reads in your sample). The counts are proportional to the library size, and you can expect ~600 mOTU counts for 5,000,000 reads. If you still think you should be getting higher counts, please see FAQ for common issues.

Note: The unassigned at the end of the profile file represents the fraction of unmapped reads. This represents species that we know to be present in the sample, but we are not able to quantify individually; hence we group them together into an unassigned fraction. For almost all the analysis, it is better to remove this value, since it does not represent a single species/clade. Please see FAQ for more information.

---

3. **Merge**. Individual taxonomic profiles can be merged together using mOTUs *merge* command to facilitate downstream analysis.

```
motus merge -i motus_profiles/ERR479298_sub1.motus,motus_profiles/ERR479298_sub1.motus -
↪o motus_profiles/merged.motus
```

-i	list of mOTU profiles to merge
-o	output file name

6.5.2 mTAGs

mTAGs generates taxonomic profiles from short-read metagenomic sequencing data using small subunit of the ribosomal RNA (SSU-rRNA). The mTAGs tool uses a reference database built by clustering sequences within each genus defined in SILVA 138 into OTUs at 97% identity. Each OTU is represented in the database as a degenerate consensus sequence (generated using the IUPAC DNA code). mTAGs detects sequencing reads belonging to SSU-rRNA and annotates them through the alignment to consensus reference sequences. For more information about the methods please see the mTAGs paper

1. **Data Preprocessing**. As always, it is important to preprocess the raw sequencing data. Standard preprocessing protocols are described in *Data Preprocessing*. As with mOTUs, we also suggest merging of paired-end reads (see *Data Preprocessing* for more details).

2. **Download mTAGs_ database**.

```
mtags download
```

2. **Profile**. Taxonomic profiles for each sample can be generated using mTAGs *profile* command. The tool produces profiles at 8 different taxonomic levels (root, domain, phylum, class, order, family, genus, and otu). Root level combines all domains, the otu level was generated by clustering of sequences within each genus. Each profile will have an 'Unaligned' and 'Unassigned' entry, these represent sequences that could not be aligned or could not be assigned at a given taxonomic level. These need to be taken into account when calculating relative abundances, but should be removed for most of downstream analyses.

```
mkdir mtags_profiles
mtags profile -f  reads/ERR479298_sub1_R1.fq.gz \
     -r reads/ERR479298_sub1_R2.fq.gz \
     -n ERR479298_sub1 -o mtags_profiles
mtags profile -f  reads/ERR479298_sub2_R1.fq.gz \
     -r reads/ERR479298_sub2_R2.fq.gz \
     -n ERR479298_sub2 -o mtags_profiles
```

-f	input file(s) for reads in forward orientation
-r	input file(s) for reads in reverse orientation
-s	input file(s) for unpaired reads (singletons or merged pair end reads)
-n	sample name
-o	output directory

3. **Merge**. Individual taxonomic profiles can be merged together using mTAGs *merge* on **.bins* files produced by *mtags profile*.

```
mtags merge -i mtags_profiles/*bins -o mtags_profiles/merged.mtags
```

`-i`	list of mOTU profiles to merge
`-o`	output file name

Choosing between mOTUs and mTAGs

mOTUs and mTAGs both generate taxonomic profiles from shotgun metagenomic data, however they differ in their approaches. The choice of the tool will depend on the specific dataset and question at hand.

Here are a few considerations to keep in mind:

1. mTAGs and mOTUs rely on different methodologies for classification. mTAGs uses rRNA sequences clustered at 97% identity, while mOTUs relies on 10 universal single-copy marker genes.
2. If you would like to compare your data to rRNA-based studies (for example 16S rRNA amplicon), mTAGs would be a better choice.
3. Since mOTUs does not rely on rRNA genes (unlike mTAGs), it avoids the potential problem of copy number variation.
4. mTAGs relies on SILVA database, which in general has a better coverage of diversity. The % of not profiled reads is usually much lower in mTAGs compared to mOTUs. However, this is highly dependent on the environment being studied.
5. Very often the resolution of the mOTUs clusters is higher than that of rRNA OTUs. As a consequence, a single 16S sequence can correspond to multiple mOTUs.
6. The general patterns found in alpha and beta diversity correlate well between these two methods.
7. mOTUs profiles can provide additional information beyond the taxonomic annotation: ref-mOTUs are directly linked to genomes (through specIs defined in ProGenomes2) and ext-mOTUs are obtained from MAGs. This allows to explore the gene content of the profiled mOTUs, which is not possible for mTAGs profiles, which are defined based on 16S rRNA sequences.

6.5.3 MAPseq

MAPseq is a fast and accurate taxonomic classification tool. Since it relies on rRNA sequences for profiling, it can be applied to both amplicon and metagenomic data.

Important: Workflow coming soon!

6.6 zAMP: Amplicon-based Metagenomics Pipeline for reproducible and scalable Microbiota Profiling

Protocol provided by Sedreh Nassirnia.

Microbiota profiling is a versatile and powerful tool with a wide range of applications in healthcare, research, environmental studies, agriculture, and industry. 16S amplicon sequencing helps us to understand the composition and functionality of microbial communities across various contexts, and can identify specific microbial patterns or biomarkers. For more on 16S sequencing see *Amplicon Sequencing*.

6.6.1 Applications

- **Disease diagnosis and prognosis**: By profiling the microbiome, researchers can identify microbial signatures that are associated with certain diseases. These biomarkers can be used to diagnose or predict the course of a disease, enhancing the understanding and management of health conditions.
- **Treatment response prediction**: Certain microbial taxa within the microbiome can serve as indicators of how a patient might respond to a particular treatment. This aspect of microbiota profiling is particularly valuable in personalizing medical treatments and the development of new therapeutic strategies.
- **Health and disease research**: Beyond clinical applications, microbiota profiling is instrumental in broadening the scientific understanding of the role of the microbiome in health and disease. This research can lead to new insights into disease mechanisms and the development of novel preventive strategies.
- **Environmental and ecological studies**: Microbiota profiling is not limited to human health. It's also used in environmental and ecological research to understand the role of microbial communities in various ecosystems, contributing to conservation efforts and the study of biodiversity.
- **Agricultural and industrial applications**: In agriculture, microbiota profiling can help to improve soil health, plant growth, and disease resistance. It can aid in processes like waste treatment and bioenergy production in industrial contexts.

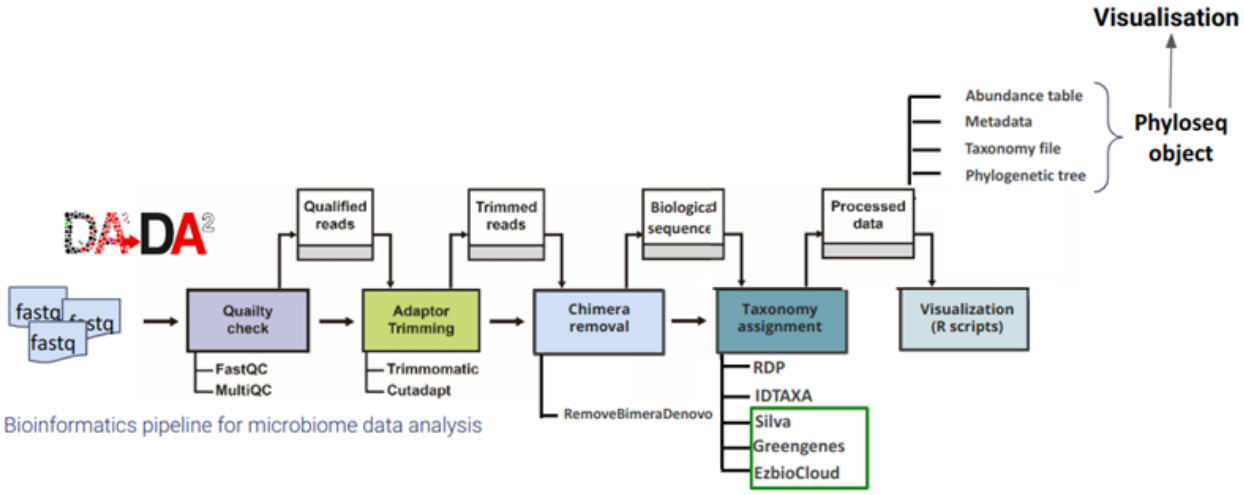
To effectively carry out microbiota profiling for these diverse purposes, we need standard protocols and tools that are compatible with many types of samples and able to provide accurate answers to the very diverse questions of the NCCR community. Therefore, we developed zAMP, an in-house customized, standardized, and automated DADA2-based data analysis pipeline, which includes a wide variety of command-line tools and R packages.

6.6.2 Implementation of zAMP

zAMP offers flexibility, allowing users to customize it depending on the research questions and sample type. To ensure reproducibility, zAMP integrates command-line tools and R packages as well as their dependencies through the Singularity container and allowed users to run and share microbiota profiling workflow.

6.6.3 Input Files for zAMP

The workflow accepts absolute paths to fastq files as input, paired-end sequencing reads from local storage, and raw reads from the Sequence Reads Archive (SRA), which will be downloaded with the SRA Toolkit.



6.6.4 Some of the Features

- **Read classification and taxonomy assignment**: After the preprocessing steps, like merging paired reads, zAMP classifies reads using the *assignTaxonomy* function across multiple taxonomic ranks, from Kingdom to Species. This is facilitated by integrated classifiers like the original *RDP*, *RDP* as integrated into *QIIME*, and *Decipher IDTAXA*, all requiring a reference taxonomic database.
- **Reference taxonomic database preprocessing**: The tool includes a secondary Snakemake workflow for preparing various reference taxonomic databases, such as *Greengenes*, *SILVA*, and *EzBioCloud*, enhancing the accuracy of taxonomic classification.
- **Dedicated scripts for taxonomic clarity**: To avoid confusion between similar taxa, zAMP features a specific script that identifies and fuses taxa with identical sequences, providing clearer taxonomic information to the user.
- **In silico validation tool**: Embedded within zAMP, this tool allows for accurate identification and classification of specific taxa or pathogens in clinical samples. Using a user-provided list of accession numbers for specific taxa, zAMP downloads genomes of interest from NCBI, performs in silico PCR using *Simulate*, and classifies amplicons using *EzBioCloud* or other user-specified databases.
- **Normalization of microbiome data**: The pipeline offers various normalization methods, including rarefying, TSS, CLR, CSS, TM, and transformations like log, Logit, and Arc-Sine Square Root, ensuring robust downstream analysis.
- **Diverse output formats**: zAMP generates outputs in multiple formats, including tab-delimited ASV tables, melted phyloseq table, and BIOM files. These are also combined into a single phyloseq object for easy manipulation and visualization in R.
- **Detailed visualization and statistics**: Users benefit from a statistic table and barplot for raw, processed, and taxonomically filtered reads. The interactive Krona plot allows for detailed investigation of sample compositions, while rarefaction curves offer insights into species richness.
- **User-friendly execution**: Designed for convenience, zAMP can be executed with a single command line, making it accessible for users to transform their microbiome data into meaningful insights.

For additional details about the tool and guidance on installation and tuning the parameters in the config file, please refer to the zAMP documentation and the GitHub repository.

6.7 SNV Analysis on Metagenomic Data

Protocol provided by Aiswarya Prasad.

Single genomes and metagenome-assembled genomes (MAGs) provide a single snapshot in time and space of a community of bacteria. They can also be thought of as an average representation of a group of very closely related microbes (strains of a species) found in that sample or pool of samples. Profiling single-nucleotide variants (SNVs) provides deeper insight into the sub-populations of the community. With sequencing becoming cheaper and faster, it is now possible to sequence metagenomic samples deep enough to recover MAGs and use those to detect SNVs in those samples.

SNV profiling is cheaper and more straightforward for simple microbial communities with few species or genera. A crude way of assessing the minimum depth (e.g., 10x) needed a priori is to consider the number of species/genomes you are interested in. Suppose you are interested in **10** genomes where the length of these genomes is **2** Mb with reads of **150** bp, you would calculate the following:

$$2\ Mb * 10\ genomes * 10\ coverage\ /\ 150\ read\ length \sim 1.3\ million\ reads\ (\sim 200\ Mb\ of\ data)$$

In this case, the above calculation underestimates the coverage, because it assumes that the ten genomes have similar abundance, which is rarely true. As a result, more abundant genomes will have more than 10x coverage, while the low-abundance genomes of interest will have little to none. Therefore, it is best to sequence deeper than this calculation

suggests. In addition, if you have pooled samples, you must sequence deeper than you would sequence individual samples, and you would also lose information about strain-level variation between individuals.

Note: What are bacterial species?

While the boundaries specifying units of diversity in microbial communities are not easily defined, bacterial genomes can still be categorized into units based on their genome sequence. One popular method to do this is to define clusters of genomes that share >95% average nucleotide identity (ANI). These clusters in most cases correspond to genomes of the same species. This approach is becoming more prevalent, especially due to cheaper sequencing methods that allow us to recover MAGs from complex samples, as well as tools that allow for pairwise comparisons of genomes.

In this protocol, we describe an approach to analyze population-level diversity using inStrain, which is documented in detail here.

6.7.1 Aim

The goal of this analysis is to profile SNVs in a metagenomic sample in a database-independent manner, starting from bam files obtained by mapping trimmed reads from each sample to a database of MAGs recovered from all the samples de-replicated into 95% clusters (recommended). For guidelines on how to preprocess sequencing data and assemble MAGs, please refer to *Data Preprocessing* and *Metagenomic Assembly*.

6.7.2 Overview

We must complete a few steps to prepare the data for *inStrain*. Including recovering the set of MAGs from the samples of interest, dereplicating the MAGs, and gene calling or annotation of the MAGs (e.g., using Prokka). In addition, it is helpful to create a metadata file containing information about the MAGs, such as which dereplicated group they belong to, their quality score, etc. The tutorial *Metagenomic Assembly* covers building metagenomic assembly.

inStrain outlines the considerations to be made when preparing the input in their tutorial on establishing and evaluating genome databases. This is a very useful place to gain an understanding of some important concepts and the reasons behind choosing this approach. This tutorial includes examples of code that can be used to achieve this.

The tutorials on the inStrain documentation page are also very helpful. This tutorial presents an alternate scenario parallel to Tutorial #2 on that page.

Create a Database of Metagenome-assembled Genomes (MAGs)

In this step, we create a database of MAGs containing one MAG to represent each species to infer SNVs. This can be done by choosing the highest quality MAG from each cluster of MAGs that are on average, e.g. 95% similar to each other. The MAGs can be clustered using the tool dRep. Once you have collected and renamed all medium and good quality (>50% completeness and <5% contamination - as a rule of thumb) MAGs, make a tab-separated file containing the columns named:

- `Bin Id` - genome name with the extension
- `Completeness` - from *checkM*
- `Contamination` - from *checkM*

You can do this using a script that parses the checkM output for the list of MAGs of at least medium quality.

```
dRep dereplicate /your/output/directory -g ./folder/file.fa -comp 0 -con 1000 --
↪clusterAlg average \
        --genomeInfo drep_genome_info.tsv -sa 0.95 -nc 0.2 -p 1 --debug
```

The result that we are interested in will be found in `/your/output/directory/data_tables/`

- `Sdb.csv` lists each MAG and its score.
- `Cdb.csv` contains the information about which cluster each MAG ends up in.

Using this information, collect the best scoring MAG for each dRep cluster (on the second column of `Cdb.csv` named `secondary_cluster`) and concatenate them into the file `MAG_rep_database.fa`. Ensure that the scaffolds have unique names across MAGs. In the rep database fasta, you will simply put all the scaffolds with each having its own header and not necessarily the information about which MAG it is from. This information will be separately summarized in another file for later steps.

Annotate the representative Database (optional)

If you would like to have gene-level information in inStrain you can include a file specifying the positions of genes in the MAGs obtained using prodigal (this is optional for *inStrain*).

```
prodigal -i MAG_rep_database.fa -d MAG_rep_database.fna -a MAG_rep_database.faa -p meta &
↪> my_log_file.log
```

Make Scaffolds to bin File

```
with open(output.scaffold_to_bin_file, "w") as f:
    for mag in input.rep_mags:
        with open(mag, "r") as m:
            for line in m:
                mag_name = os.path.basename(mag).split(".")[0]
                if line.startswith(">"):
                    scaffold = line.strip().split(">")[1]
                    f.write(f"{scaffold}\t{mag_name}\n")
```

Map reads to MAG Database

Ensure that you use *bowtie2* for this, as recommended by *inStrain*. Avoid BWA (even though it might be your favorite aligner) as *inStrain* may have issues handling the way that it calculates insert size, and the BWA documentation is unclear about how this is performed.

```
bowtie2-build mag_rep_database.fa mag_rep_database.fa &> bowtie2_build.log # bowtie index
# map to rep MAGs
bowtie2 -X 1000 -x mag_rep_database.fa -1 sample_R1_repaired.fastq.gz -2 sample_R2_
↪repaired.fastq.gz | samtools view -bh - | samtools sort - > sample_bowtie.bam
samtools flagstat sample_bowtie.bam > sample_bowtie_flagstat.tsv
```

Make the inStrain Profile

Output and parameter information is well-documented in inStrain - run using db_mode if you wish to run *inStrain compare* later. This makes it much faster.

```
inStrain profile sample_bowtie.bam mag_rep_database.bam -o /your/output/directory/ -p 8 -
↪g mag_rep_database_genes.fna \
    --max_insert_relative 5 -s scaffold_to_bin_file.tsv
inStrain plot -i inStrain_profile_object -pl a -p 16
inStrain profile sample_bowtie.bam mag_rep_database.fa -o /path/to/output/folder/sample -
↪p 8 \
    -g mag_rep_database_genes.fna --max_insert_relative 5 --database_mode -s scaffold_to_
↪bin_file.tsv
```

Run *inStrain compare*

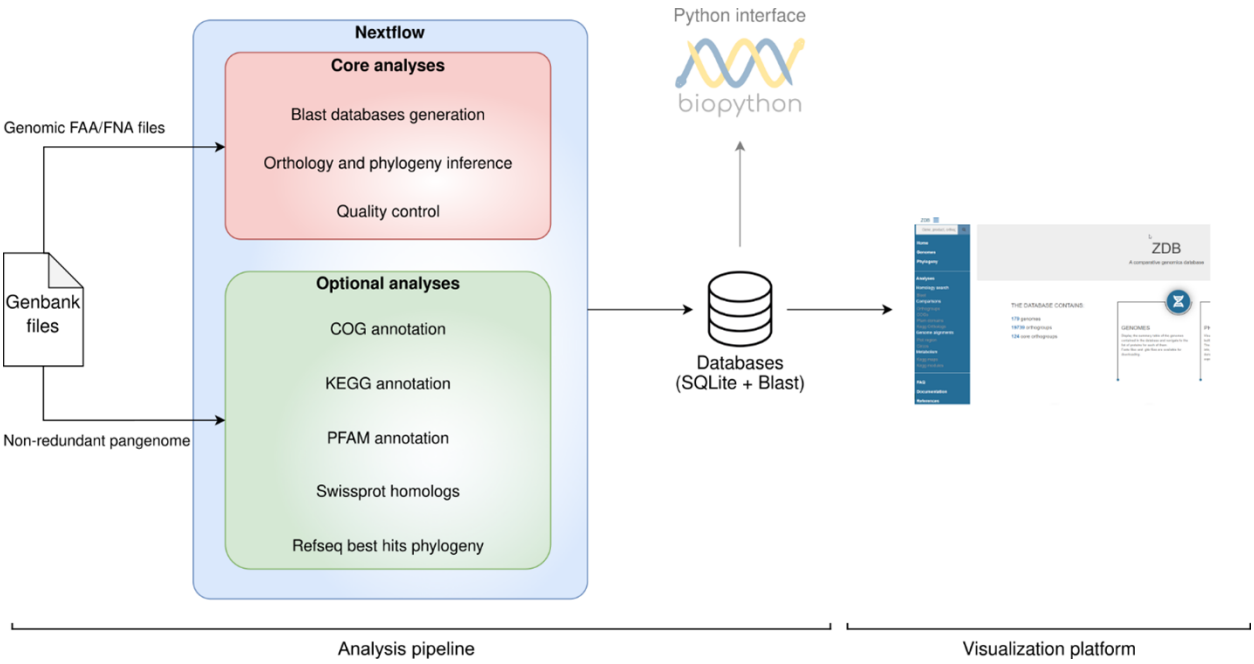
This can be run on all profiles together, especially if you did not have a lot of samples, but for datasets including a large number of samples, it will be more efficient to run this in parallel for each species at a time by using the `--genome` option to specify one genome at a time.

```
inStrain compare -i inStrain_profile -s scaffold_to_bin_file.tsv -p 8 -o /your/output/
↪directory/ \
            --database_mode --genome mag.stb
```

6.8 zDB: Comparative Genomics Analysis

Protocol provided by Alessia Carrara.

Comparative genomic analyses allow researchers to study and compare the genetic information of different organisms, providing valuable insights into their evolution, biology, and function. However, it is a tedious and time-consuming process that requires specialized knowledge since it relies on the integration of results obtained from multiple tools, further combined for visualization purposes. Therefore, to facilitate genome analysis and comparison, we developed zDB, an application that integrates an **analysis pipeline** and a **visualization platform**.



6.8.1 Target Audience

zDB has been developed for both newbies in genomics, who don't have the experience with bioinformatics tools, and trained bioinformaticians, who would like to invest in a reproducible tool. Therefore, don't be scared and don't hesitate to use it!

6.8.2 zDB: Supported Analyses

The analysis pipeline built in zDB can support different types of analysis according to the user's needs. Indeed, a core set of analyses is run by default, while optional analyses can be separately added.

- **Orthology inference**: identification of orthogroups by OrthoFinder (an orthogroup is the group of genes descended from a single gene in the last common ancestor (LCA) of a group of species).
- **Phylogenetic reconstructions**: phylogeny built based on concatenated alignments of single copy orthologs with FastTree.
- **Blast search**: generate blast databases out of your input genome
- **COG annotation**: label orthologs with the 25 functional categories provided by the COG database (Clusters of Orthologous Groups of proteins) and previously identified based on shared evolutionary origins and functional roles of genes.
- **KEGG orthologs annotation and pathway completion analysis**: label orthologs with molecular interaction pathways and networks provided by the KEGG database (Kyoto Encyclopedia of Genes and Genomes). Obtain insights about the completeness of the identified pathways.
- **PFAM domains annotation**: assign protein domains to orthologs
- **Swissprot homologs search**: look for homologous proteins for each ortholog in the Swiss-Prot database which contains manually curated protein sequences and provides a high level of annotation.
- **RefSeq homologs search**: look for homologous proteins for each ortholog in a comprehensive collection of curated and annotated sequences of a wide range of organisms.

For more info about the tools and parameters please refer to the methods section of the zDB documentation or the materials and methods section of the manuscript.

6.8.3 Input Files for zDB

- Input csv file containing the path to *.gbk* files.
- The requirement of annotated genomes (*.gbk* files) implies that the user is responsible for the annotation of the genomes. Here are some examples of annotation tools you can use: *bakta*, *prokka*, *PGAP*.
- The input genomes (minimum 2) can be publicly available and/or newly sequenced ones.

6.8.4 How to install and run zDB

In the GitHub repository, you can have detailed instructions on how to install and run zDB. Please refer to if you have any problem or you want to get more details. In the meantime, here below is a brief list of all the key steps:

1. Install zDB and tools to run the analysis and/or web app in containers (recommended).

```
conda install nextflow=22.10 -c bioconda # install nextflow
conda install zdb -c bioconda # zDB installation
conda install singularity=3.8.4 -c conda-forge # singularity installation
```

For the installation of docker, please see the Docker documentation.

2. Download the database(s) needed for the annotation steps (optional).

Select the databases needed for your analysis. The RefSeq database must be downloaded and prepared autonomously.

```
zdb setup --pfam --swissprot --cog --ko --conda
```

3. Run the pipeline.

- Add flags according to the annotation you want to get.
- RefSeq homologs search significantly slows down the analysis.
- All the results will be stored in an SQL database.

```
zdb run --input=input.csv --ko -cog
```

4. Initiate and access the web server.

The terminal will output an IP address where a customized web-based interface built from the SQL database is available. To access the interface, follow this example: If the output looks like this: `@155.105.138.249 172.17.0.1 on port 8080`, type either `155.105.138.249:8080` or `172.17.0.1:8080` on your web browser to visualize.

6.8.5 Accessibility of the Output

If you want to share your newly generated SQL database you can export it using *zdb export* command and transfer it to other machines. You may want to transfer it to i) a personal machine for personal usage, ii) a lab machine to host it on an intranet domain and make it accessible to other lab mates, or iii) host it on an internet domain to make it accessible to everyone.

6.8.6 Useful Links to explore zDB by yourself!

- This example of the web interface generated via zDB on a dataset of 41 *Rickettsiales* genomes.
- Tips on how to navigate the web interface and interpret your data.

6.9 Metatranscriptomics without Metagenomics (defined Community)

Protocol provided by Anna Sintsova.

Important: This documentation is currently under construction.

Metatranscriptomic data arising from defined communities (i.e. community, whose composition is known) can be analysed in a way that's similar to traditional RNASeq with a few key differences. In this case, we first map the quality-controlled reads to the bacterial genomes, and then count number of reads mapping to each feature. The statistical analysis to identify differentially expressed features can be performed using DESeq2.

Once the metagenome is ready, you are read to proceed with transcript quantification workflow

6.9.1 Transcript quantification

Preprocessing the genomes

1. First we create a *metagenome*, i.e. a concatenation of genome sequences for all of the organisms present in the community. We also need to create a combined annotation file (*gff*). This will be needed later on to count how many reads mapped to each gene.

Note: Why is competitive mapping important? It properly accounts for sequences that potentially map to multiple targets/species (multi-mappers, count as fractions). If the sample was mapped to each species individually, these reads will be counted towards each genome, overestimating the counts.

```
cat species1.fasta species2.fasta > metagenome.fasta
```

2. Here, we build the genome index using bowtie2. This is an essential step before any read alignment step regardless of aligner you choose.

```
bowtie2-build metagenome.fasta metagenome
```

Transcript profiling

3. (Optional) Depending on the library preparation strategy, metatranscriptomic samples can contain large amounts of rRNA. You can use *fastqc_screen* to assess the amount of rRNA in your samples, and sortmerna[] to filter it out.

3. Next, we align reads from each sample to our indexed metagenome.

```
bowtie2 ...
```

Note: Different alignment and counting tools can be used for this step. We have tested *BWA + sushiCounter*, *salmon* as well as *bowtie2 + featureCounts*. In our hands, all of these pipelines produce very similar results. It is always best to test and see what works for your data!

---

4. Next, we count number of inserts aligned to each feature of interest (i.e. gene). For this we use featureCounts and we use *–fraction* to assign multi-mapped reads ...

```
featureCounts ...
```

Warning: Be careful when combining different aligners and counting methods - not all of them are perfectly compatible. For example, featureCounts cannot recognize multi-mapped reads in alignment files generated by BWA, and smth about STAR and featureCounts as well.

5. Statistical analysis. To effectively analyse metatranscriptomic data, you need to account for variation in taxonomic composition across samples. Above, we used matching metagenomic data for this purpose. While we cannot do this here, we can still perform taxon-specific scaling, since we know the taxonomic composition of the community. This is dissected in detail in this paper [], which also provides template code for the analysis. This analysis ends up being equivalent to analysing the dataset as if it were a composition of N traditional RNAseq datasets, where N is the number of species in the community.

6.10 Metatranscriptomics with Metagenomics

Protocol adapted from Guillem Salazar.

Metatranscriptomics is the analysis of all of the transcriptomes present in a sample and is an effective method to assess the activity of a microbial community. Unlike *metagenomic data*, metatranscriptomics can help decipher the metabolic functions actively expressed by the community at any given time.

- Advantages:
  - Best way to get information about community activity.
- Disadvantages:
  - Sample preparation might be difficult and expensive.
  - Data analysis methods are not well established.

Metatranscriptomic experiments can be broadly summarised into 2 different types: experiments that include matching metagenomic samples and those that do not. The analysis of these two datatypes are different. You can find a tutorial of how to analyze metatranscriptomic data without metagenomics on our *Metatranscriptomics without Metagenomics (defined Community)* website.

6.10.1 Dataset

Here, we will combine metatranscriptomics with metagenomics by using metagenomic data to normalize the transcript abundance. In order to do this, metagenomic data was processed as described in *gene catalog creation* to create a gene catalog. Gene functional annotation into orthologous groups was then performed using the KEGG database. The **metagenomic data** was then mapped back to the gene catalog to determine **gene abundance**. And the **metatranscriptomic data** from each sample was then mapped to the gene catalog to determine **transcript abundance**. To determine **gene expression**, we will look at the ratios of transcript abundance to gene abundance (after some normalisations of course).

Note: Transcript abundance depends on both gene abundance (number of genes) and expression level (number of mRNAs per gene). See Figure S1 for more information.

Note: The -1 fraction are unannotated genes, which we must account for, for proper normalisation. You can read more on this in the *Gene length normalisation* section.

The dataset used in this tutorial is from the article Gene Expression Changes and Community Turnover Differentially Shape the Global Ocean Metatranscriptome, Salazar et al. The data can be downloaded `here`. These files are just a subset of the full dataset and are only meant to be used for this tutorial. To learn how to download and unpack the data follow *these instructions*.

This will contain the following files:

- *MGS_K03040_K03043_tara.tsv.gz* contains a sample of raw counts for metagenomic and metatranscriptomic samples.
- *MGS_K03040_K03043_tara_lengthnorm.tsv.gz* contains length normalised counts for metagenomic and metatranscriptomic samples.
- *K03704_tara_lengthnorm_percell.tsv.gz* contains length and per-cell normalised counts for K03704 for metagenomic and metatranscriptomic samples.
- *sample_info.csv* contains some metadata about the samples.

6.10.2 Data Normalisation

For both gene abundance and transcript abundance data, we must remove the following sources of bias:

- Differences in gene length between genes.
- Differences in sequencing depth between samples.
- Differences in genome size distribution between samples.
- Compositionality: The number of inserts for a given gene in a given sample can only be interpreted relative to the rest of the genes in the sample.

Note: Genome size differences can lead to biases and challenges in accurately comparing gene expression levels between different samples. To account for the genome size differences we will normalize the gene and transcript abundance data by abundances of 10 marker genes. This is explained in further detail in the section *Sequencing depth, per cell normalisation and compositionality*.

Setting up R environment and loading the data

We perform all of the normalisation steps in R. To run this analysis you will need *tidyverse* and *data.table* libraries.

```
library(data.table)
library(tidyverse)
# To read compressed files data.table needs R.utils library
library(R.utils)
library(patchwork)
library(GGally)
```

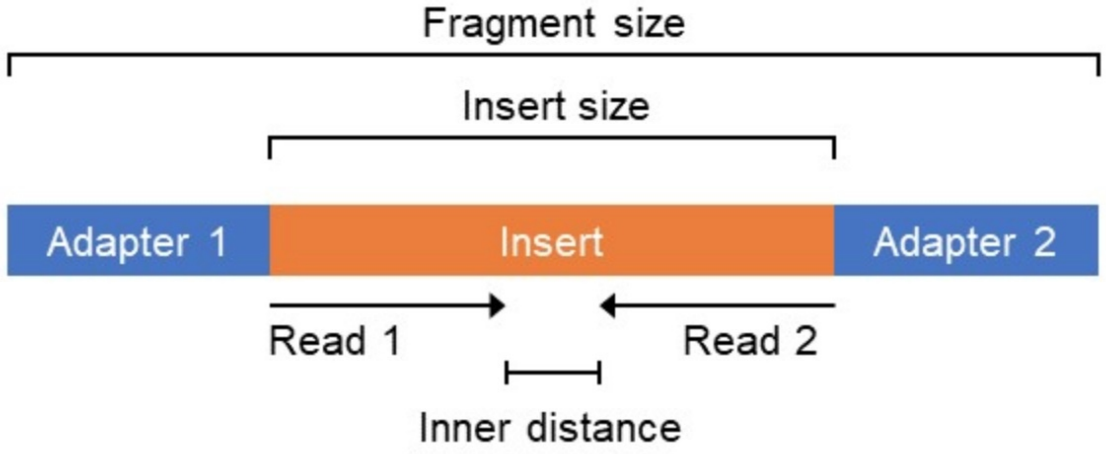
Next we are going to load gene and transcript abundances and metadata (i.e. temperature, location, depth, etc.).

```
# Load the gene and transcript abundances
profile <- fread("datasets/part1/MGS_K03040_K03043_tara.tsv.gz",sep="\t",
             header=T,data.table = F,tmpdir=".")
sample_info <- fread("datasets/part1/sample_info.csv",sep=",",
            header=T, data.table = F, tmpdir=".")
```

Gene length normalisation

The first step in the normalisation process is to divide the insert counts by the gene length for each gene in each sample. Since the **unmapped (-1)** fraction does not have a length, we assign it the median gene length.

Note: During short read sequencing, DNA is randomly sheared into **inserts** of known size distribution and sequenced. If paired-end sequencing is used, two DNA sequences (reads) are generated - one from each end of a DNA fragment. Here, we count inserts, not reads.




```
# Assigns median gene length to -1 fraction
# Example file does not contain -1 fraction, so this will have no effect for us
if (length(which(profile$length < 0)) > 0){
  med_length = median(profile$length[which(profile$length > 0)])
  profile$length[which(profile$length < 0)] <- med_length
}
```

We now build a gene-length normalized profile

```
profile_lengthnorm <- profile[, 1:4]
for (i in 5:ncol(profile)){
  cat("Normalizing by gene length: sample", colnames(profile)[i], "\n")
  tmp <- profile[, i]/profile$length %>%
    as.data.frame()
  colnames(tmp) <- colnames(profile)[i]
  profile_lengthnorm <- profile_lengthnorm %>%
    bind_cols(tmp)
}
```

Sequencing depth, per cell normalisation and compositionality

To account for differences in sequencing depth, as well as for differences in genome sizes between different samples,

---

Note: What are marker genes (MGs)?

- Universal: present in "all" prokaryotes
- Single-copy: always present once per cell (genome)
- Are housekeeping genes

Because of these characteristics, the abundance of marker genes (MGs) correlates well with the sequencing depth. In addition, the median abundance of MGs is a good proxy for the number of cells captured in a given metagenomic/metatranscriptomic sample. The per-cell normalization accounts for differences in genome sizes between samples and also controls for compositionality. The result of this normalisation is a biologically meaningful unit: **gene copies per total cell in the community**.

To normalize by abundance of 10 MGs, we first compute their total insert count in each sample (i.e. sum the counts for each of the 10 KOs). We then compute the median of the 10 MGs in each sample. Finally, we divide the gene-length normalized abundances by this median for each sample.

In this example we use the following marker genes:

```
K06942  ychF; ribosome-binding ATPase
K01889  FARSA, pheS; phenylalanyl-tRNA synthetase alpha chain [EC:6.1.1.20]
K01887  RARS, argS; arginyl-tRNA synthetase [EC:6.1.1.19]
K01875  SARS, serS; seryl-tRNA synthetase [EC:6.1.1.11]
K01883  CARS, cysS; cysteinyl-tRNA synthetase [EC:6.1.1.16]
K01869  LARS, leuS; leucyl-tRNA synthetase [EC:6.1.1.4]
K01873  VARS, valS; valyl-tRNA synthetase [EC:6.1.1.9]
K01409  OSGEP, KAE1, QRI7; N6-L-threonylcarbamoyladenine synthase [EC:2.3.1.234]
K03106  SRP54, ffh; signal recognition particle subunit SRP54 [EC:3.6.5.4]
K03110  ftsY; fused signal recognition particle receptor
```

```
# Define the KOs corresponding to the 10 MGs
mgs <- c("K06942", "K01889", "K01887", "K01875", "K01883",
         "K01869", "K01873", "K01409", "K03106", "K03110")


# Build a MGs normalized profile
profile_lengthnorm_mgnorm <- profile_lengthnorm[, 1:4]

for (i in 5:ncol(profile_lengthnorm)){
```

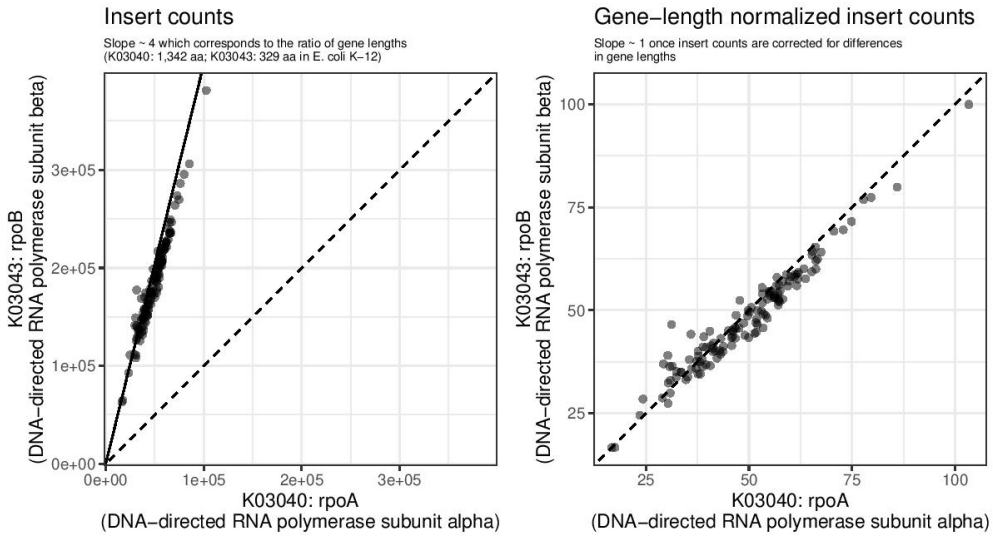
(continues on next page)

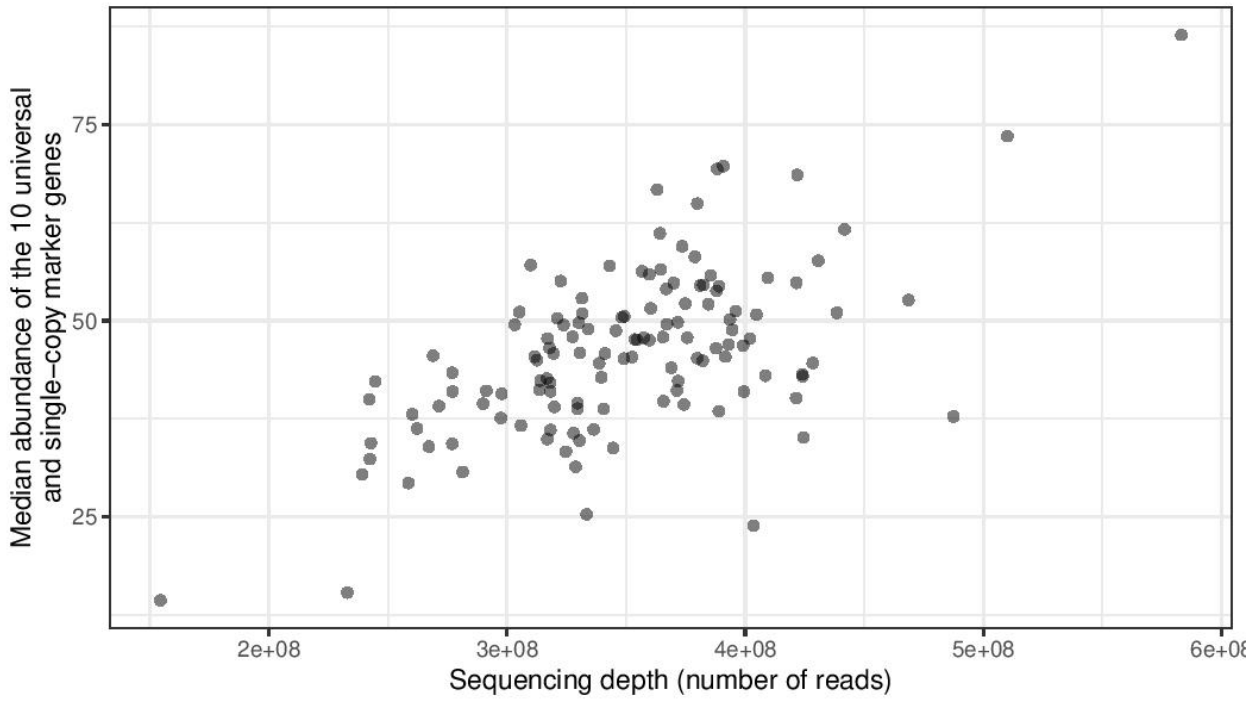
(continued from previous page)

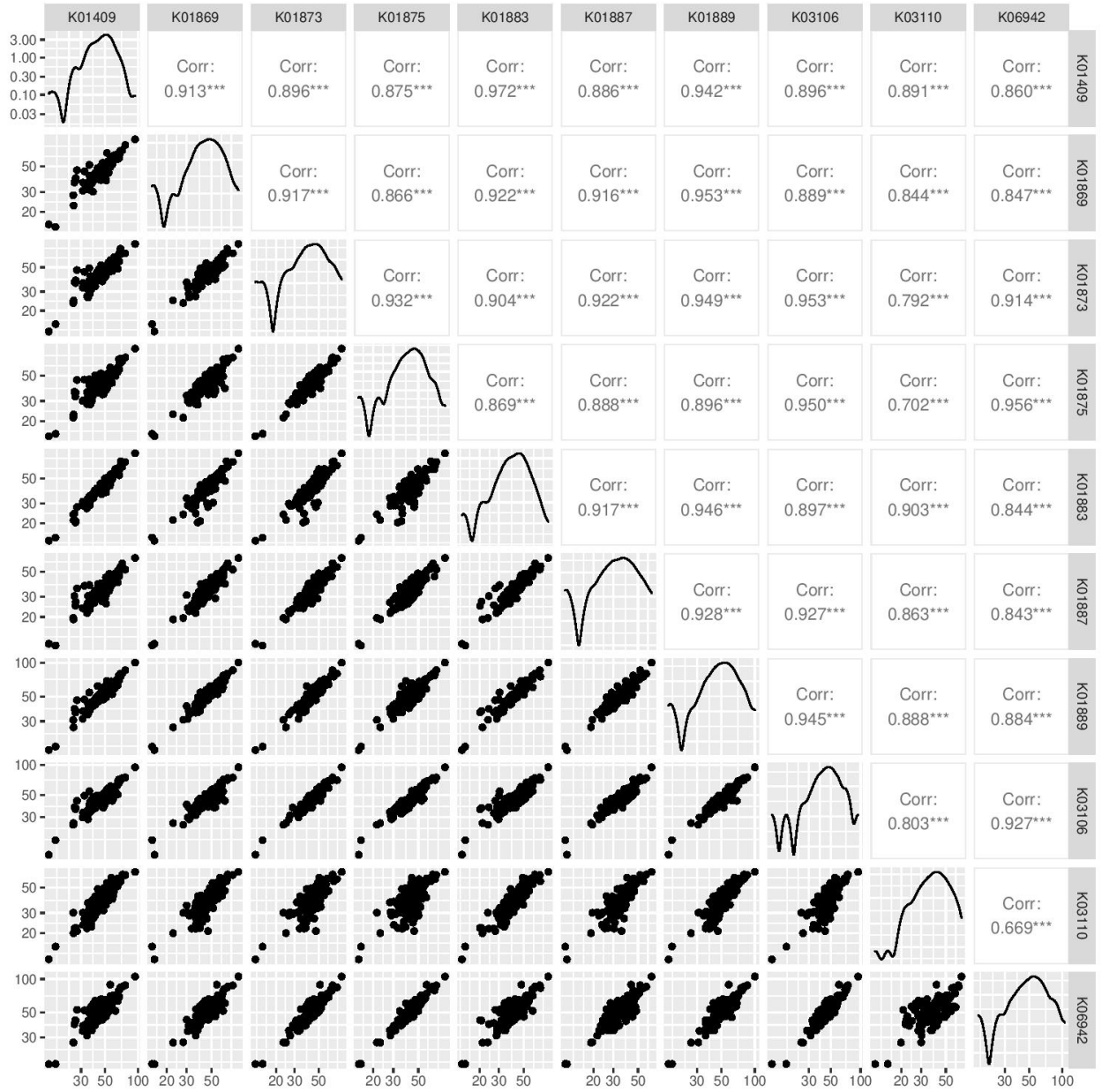
```
  cat("Normalizing by 10 MGs: sample", colnames(profile_lengthnorm)[i], "\n")
  mg_median <- profile_lengthnorm %>%
    select(KO, abundance = all_of(colnames(profile_lengthnorm)[i])) %>%
    filter(KO %in% mgs) %>%
    group_by(KO) %>% summarise(abundance = sum(abundance)) %>%
    ungroup() %>% summarise(mg_median = median(abundance)) %>%
    pull()
  tmp <- profile_lengthnorm[,i]/mg_median
  tmp <- tmp %>% as.data.frame()
  colnames(tmp) <- colnames(profile_lengthnorm)[i]
  profile_lengthnorm_mgnorm <- profile_lengthnorm_mgnorm %>%
    bind_cols(tmp)
}
```

Showing the effect of the normalization

Here, we visualize the effect of the normalization based on length and abundance of marker genes. Using this script we create the following plots:




Median abundance of the 10 universal and single−copy marker genes
75
50
25
2e+08
3e+08
4e+08
5e+08
6e+08
Sequencing depth (number of reads)



We will first look at 2 KOs: K03040 and K03043. These encode for 2 subunits of RNA polymerase. We first subset the raw metagenomic counts for these 2 genes (*rp_ab*) and then do the same with length normalised counts (*rp_ab_lengthnorm*), and finally visualize the relationship.

```
# Compute the abundance of K03040 and K03043 with and without gene-length normalization
rp_ab <- profile %>%
  select(-reference, -length, -Description) %>%
  filter(KO %in% c("K03040", "K03043")) %>%
  pivot_longer(-KO, names_to = "sample", values_to = "inserts") %>%
  filter(grepl('METAG', sample)) %>%
  group_by(KO, sample) %>% summarize(inserts = sum(inserts)) %>%
  pivot_wider(names_from = "KO", values_from = "inserts")

rp_ab_lengthnorm <- profile_lengthnorm %>%
```

(continues on next page)

(continued from previous page)

```
  select(-reference, -length, -Description) %>%
  filter(KO %in% c("K03040", "K03043")) %>%
  pivot_longer(-KO, names_to = "sample", values_to = "inserts_lengthnorm") %>%
  filter(grepl('METAG', sample)) %>%
  group_by(KO, sample) %>% summarise(inserts_lengthnorm = sum(inserts_lengthnorm)) %>%
  pivot_wider(names_from = "KO", values_from = "inserts_lengthnorm")



g1 <- ggplot(data = rp_ab, aes(x = K03040, y = K03043)) +
  geom_point(alpha = 0.5) +
  geom_abline(slope = (1342/329)) +
  geom_abline(linetype = 2) +
  xlim(range(rp_ab$K03040, rp_ab$K03043)) +
  ylim(range(rp_ab$K03040, rp_ab$K03043)) +
  xlab("K03040: rpoA\n(DNA-directed RNA polymerase subunit alpha)") +
  ylab("K03043: rpoB\n(DNA-directed RNA polymerase subunit beta)") +
  labs(title = "Insert counts", subtitle = "Slope ~ 4 which corresponds to the ratio of␣
↪gene lengths\n(K03040: 1,342 aa; K03043: 329 aa in E. coli K-12)") +
  coord_fixed() +
  theme_bw() +
  theme(plot.subtitle = element_text(size = 7))


g2 <- ggplot(data = rp_ab_lengthnorm, aes(x = K03040, y = K03043)) +
  geom_point(alpha = 0.5) +
  geom_abline(linetype = 2) +
  xlim(range(rp_ab_lengthnorm$K03040, rp_ab_lengthnorm$K03043)) +
  ylim(range(rp_ab_lengthnorm$K03040, rp_ab_lengthnorm$K03043)) +
  xlab("K03040: rpoA\n(DNA-directed RNA polymerase subunit alpha)") +
  ylab("K03043: rpoB\n(DNA-directed RNA polymerase subunit beta)") +
  labs(title = "Gene-length normalized insert counts", subtitle = "Slope ~ 1 once insert␣
↪counts are corrected for differences\nin gene lengths") +
  coord_fixed() +
  theme_bw() +
  theme(plot.subtitle = element_text(size = 7))

g1|g2
```

Now, we're going to look at correlation of marker gene abundance with sequencing depth and the correlation in abundance between different marker genes.

```
# Compute the abundance of the 10MGs and correlate to sequencing depth
mgs_ab_lengthnorm <- profile_lengthnorm %>%
  select(-reference, -Description, -length) %>%
  filter(KO %in% mgs) %>%
  pivot_longer(-KO, names_to = "sample", values_to = "inserts_lengthnorm") %>%
  group_by(KO, sample) %>% summarise(inserts_lengthnorm = sum(inserts_lengthnorm)) %>%
  ungroup() %>% group_by(sample) %>% summarise(median_mgs = median(inserts_lengthnorm))
↪%>%
  inner_join(sample_info, by = c("sample" = "sample_metag"))
```

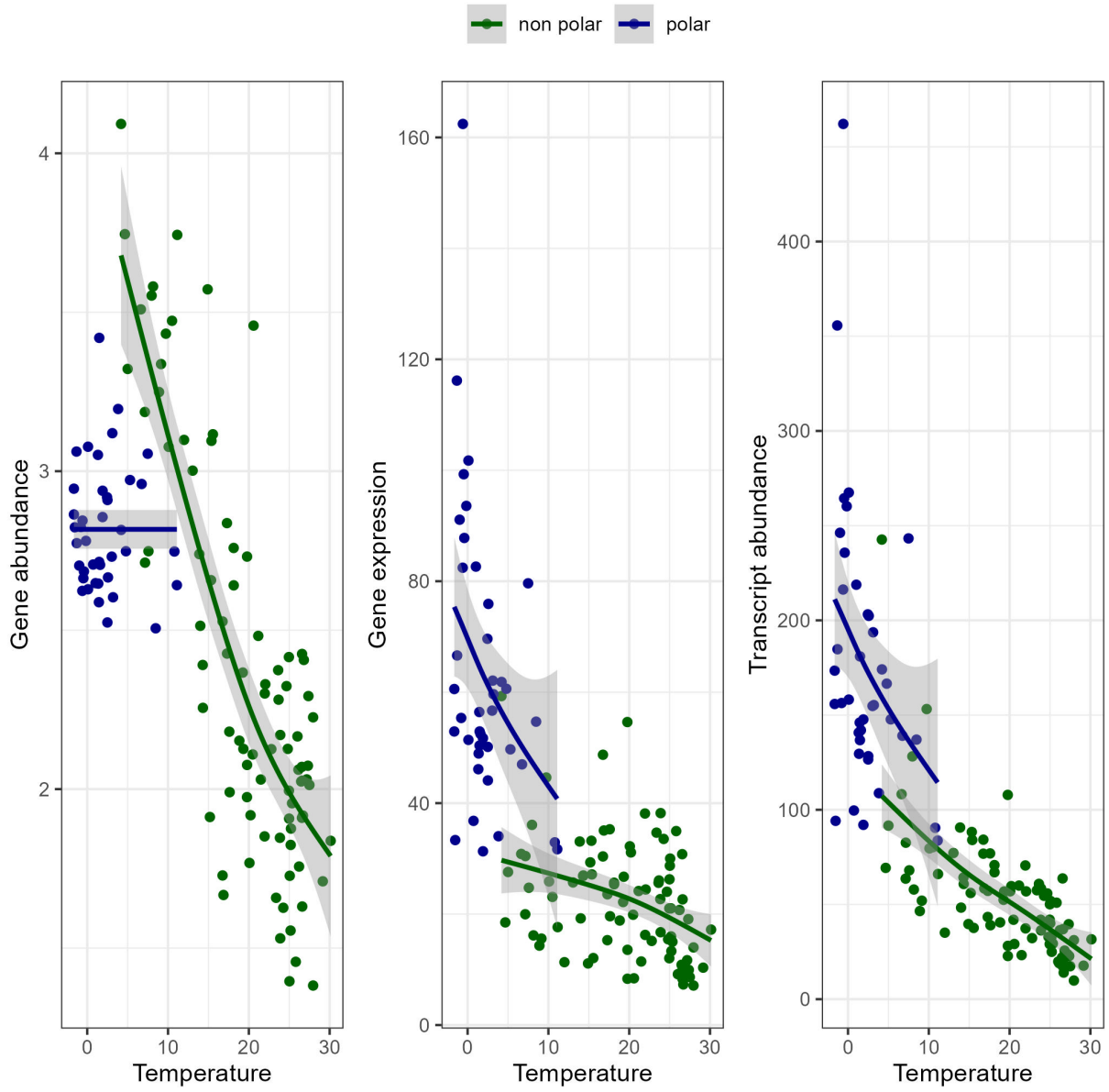
(continues on next page)

(continued from previous page)

```
g3 <- ggplot(data = mgs_ab_lengthnorm, aes(x = sample_metag_nreads, y = median_mgs)) +
  geom_point(alpha = 0.5) +
  #geom_smooth(method = "lm") +
  #scale_x_log10() +
  #scale_y_log10() +
  xlab("Sequencing depth (number of reads)") +
  ylab("Median abundance of the 10 universal\nand single-copy marker genes") +
  theme_bw() +
  theme(legend.title = element_blank())


g3

# Compute the abundance of the 10MGs and their autocorrelation

mgs_ab_lengthnorm <- profile_lengthnorm %>%
  select(-reference, -Description, -length) %>%
  filter(KO %in% mgs) %>%
  pivot_longer(-KO, names_to = "sample", values_to = "inserts_lengthnorm") %>%
  group_by(KO, sample) %>% summarise(inserts_lengthnorm = sum(inserts_lengthnorm)) %>%
  inner_join(sample_info, by = c("sample" = "sample_metag")) %>%
  select(KO, sample, inserts_lengthnorm) %>%
  pivot_wider(names_from = "KO", values_from = "inserts_lengthnorm")

g4 <- ggpairs(data = mgs_ab_lengthnorm %>% column_to_rownames("sample")) +
  scale_x_log10() +
  scale_y_log10()

g4
```

Combining Metatranscriptomic and Metagenomic Data

In this section we combine metatranscriptomic and metagenomic data and create the following plot:




```
# Load normalized profile
gc_profile <- fread("datasets/part1/K03704_tara_lengthnorm_percell.tsv.gz", sep = "\t",␣
↪header = T, data.table = F, tmpdir = ".")
sample_info <- fread("datasets/part1/sample_info.csv",sep=",",
            header=T, data.table = F, tmpdir=".")


ko_profile <- gc_profile %>%
  group_by(KO) %>% summarise(across(starts_with("TARA"), sum)) %>%
  as.data.frame()


# Compute the gene abundance, transcript abundance and expression for the pairs of metaG-
```

(continues on next page)

(continued from previous page)

```
↪metaT samples
# The expression is just the ratio of transcript_abundance to gene_abundance
tmp_sample_info <- sample_info %>%
  select(sample_metag, sample_metat) %>%
  mutate(sample_pair = paste(sample_metag, sample_metat, sep = "-"))
tmp_metag <- ko_profile %>%
  select(KO, all_of(tmp_sample_info$sample_metag)) %>%
  pivot_longer(-KO, names_to = "sample_metag", values_to = "gene_abundance")
tmp_metat <- ko_profile %>%
  select(KO, all_of(tmp_sample_info$sample_metat)) %>%
  pivot_longer(-KO, names_to = "sample_metat", values_to = "transcript_abundance")
final_profile <- tmp_sample_info %>%
  left_join(tmp_metag, by = "sample_metag") %>%
  left_join(tmp_metat, by = c("KO", "sample_metat")) %>%
  mutate(expression = transcript_abundance/gene_abundance)

toplot <- final_profile %>%
  filter(KO == "K03704") %>%
  left_join(sample_info, by = c("sample_metag","sample_metat"))

g_metat <- ggplot(data = toplot, aes(y = transcript_abundance, x = Temperature, color = ␣
↪polar)) +
  geom_point() +
  geom_smooth(method = "gam", se = T, formula = y ~ s(x, bs = "cs", k=5)) +
  scale_color_manual(values = c("darkgreen", "darkblue"))+
  ylab("Transcript abundance") +
  theme_bw() +
  theme(legend.position = "none")
g_metag <- ggplot(data = toplot, aes(y = gene_abundance, x = Temperature, color =␣
↪polar)) +
  geom_point() +
  geom_smooth(method = "gam", se = T, formula = y ~ s(x, bs = "cs", k=5)) +
  #scale_y_log10() +
  #coord_flip() +
  scale_color_manual(values = c("darkgreen", "darkblue")) +
  ylab("Gene abundance") +
  theme_bw() +
  theme(legend.position = "none")
g_exp <- ggplot(data = toplot, aes(y = expression, x = Temperature, color = polar)) +
  geom_point() +
  geom_smooth(method = "gam", se = T, formula = y ~ s(x, bs = "cs", k=5)) +
  #scale_y_log10() +
  #coord_flip() +
  scale_color_manual(values = c("darkgreen", "darkblue")) +
  ylab("Gene expression") +
  theme_bw() +
  theme(legend.position = "top", legend.title = element_blank())
g <- g_metag | g_exp | g_metat
g
```

6.11 TN-Seq Protocol

Protocol provided by the van der Meer group.

As described by Tim van Opijnen et al., Tn-seq is a method to determine quantitative genetic interactions on a genome-wide scale in microorganisms.

6.11.1 Sequence Mapping

In the first step of the data processing, we use the reads as they come from the Illumina machine, we identify in the reads if they carry a Tn5-end; and select only those reads using cutadapt. Then we blast the reads to the transposon sequence to remove those that are only transposon. The remaining reads are mapped to the combined genome using bowtie, counted per position and grouped per gene in a perl script.

6.11.2 Combined clean Reads

In the second step of the data processing, we combine all reads from all samples per library, remove outliers, normalize and save the files.

6.11.3 PCA Analysis

In the third step, we compare all data in a PCA. For this, replace all NaNs by 0, calculate the pca and plot the first two coefficients.

6.11.4 Essentiality Analysis

In the third step, we try to define the essential genes at time t=0. We do this by comparing to a randomized virtual insertion library.

To create the random insertion library, we take the combined genomes and use the actual gene coordinates to outline starts and stops. We produce a random genome file and choose four times randomly unique values within this range. This is then summed for the gene positions to create fake random insertion libraries.

To get the t=0 dataset, select the t=0 samples in the normalized files with outliers removed. Replace the NaN by zeros and estimate whether the mean read number is significantly lower in T0 datasets.

Next, we attribute the group of essentials to their Clusters of Orthologous Genes Category (COG) and compare this attribution to a random simulation of genes drawn. This we do 10 times, to get an average random COG attribution of genes from the genome. From there, we calculate the p-value for the difference of the observed to the simulated COG attributions and plot this as a histogram.

6.11.5 Paired Analysis

In the next step, we define the genes with conditional fitness effects at any of the three sampling time points compared to time t=0. We do this by taking the means across all replicates of the T1-T3 samples, and calculating the ratio to the mean of the T0 values. We do this for each library individually and for each condition. From this comparison, we also calculate a p-value and an FDR, from the function mattest and mafdr using BHFDR correction for multiple hypothesis testing.

As conditional fitness we put a threshold of a ratio change of >2 (‘down’) or <0.5 (‘up’), plus an FDR<0.05, and the initial ratio to the random insertion library >0.25 (to avoid taking previously allocated essential genes).

Finally, we compare gene lists to find the overlap and unique differences per library and condition. This is used to plot the Venn diagram.

For defined sets of gene groups we plot the ratio of mean counts at each time point compared to T0 in colour, overlaid on all ratios; to highlight time trends.

6.11.6 Gene Group Plots

In this step, we use free text searching and defined gene groups for biological systems such as 'flagella' to select the gene insertion count ratios and plot those. We use both heatmap representation for individual genes, as well as a summed representation in which all counts for the group is combined and compared.

6.11.7 COG KEGG Analysis

In this step of the analysis, we focus analysis on attribution of genes from the selected UP and DOWN gene lists to COG or KEGG groups, and calculate statistical enrichment or depletion of such group attribution compared to random models. Finally, we plot the data in a variety of ways.

COG attribution

We start with the paired comparison data of the ratios T0-replicates/T1-3 replicates and the corresponding p-values and FDR. We also load the annotation files of the genomes, the manually curated COG list, the gene name list and the gene number list. For the attribution we use the selected gene lists and the total DOWN and UP files for the two soils.

We now first build a set of random simulations from the length of the selected set. For example, if a DOWN file has 820 items, we simulate random picking for 820 genes from the genome. For each of the randomly picked gene lists, we find matches to the COG and retrieve the corresponding distribution of COG classes. This is done 10 times. From here we calculate the means, standard deviation and variation and we compare to the actual COG attributed list in the DOWN file. Finally, we attribute a confidence interval on the difference to the mean from the random data set by bootstrapping and a p-value of<0.01, and we calculate a p-value and FDR by mafdr of the random to real list comparison. The results are saved to a separate Excel for each comparison, to which we also add the individual gene annotations. The whole procedure and results is also saved in a file.

Polar Plots of COG class distributions

All the produced COG class distributions and enrichment factors are now plotted as a polar plot. Essentially, we plot the mean random relative count of class attribution and the actual relative count, as a log2 transformed bar in a radial output. We add a star if the FDR value is below 0.05.

KEGG class distributions

Essentially we follow the same idea as for the COG attribution, but now we focus on prediction of metabolic pathways. Again we make 10 random distributions to compare the observed data to.

Plotting KEGG class enrichment as Volcano plot

We plot a figure with four panels for each of the exclusive DOWN categories side by side that shows on the x-axis the log-FC of the observation compared to the mean of the random picked/attributed pathways, and on the y-axis the -log10 of the calculated p-value for the variation. We also plot a legend with the precise color attribution for the categories.

Plotting metabolic pathways on the KEGG map

Finally, we plot all and exclusive attributed KEGG metabolic pathways on top of the general map in iPATH3, per library and per soil or liquid category. By gene number or gene name we recover the reaction numbers from the genome-scale model file, copy that list to the iPATH3 website and save the map as .svg.