SqueezeMeta: a fully automated metagenomics pipeline, from reads to bins

Version 1.6.3, September 2023

Javier Tamames and
Fernando Puente-Sánchez

CNB-CSIC, Madrid, Spain

SLU, Uppsala, Sweden

Index

SqueezeMeta: a fully automated metagenomics pipeline, from reads to bins

Version 1.6.3, September 2023

1. What is SqueezeMeta?

SqueezeMeta is a full automatic pipeline for metagenomics/metatranscriptomics, covering all steps of the analysis. SqueezeMeta includes multi-metagenome support allowing the co-assembly of related metagenomes and the retrieval of individual genomes via binning procedures. Thus, SqueezeMeta features several unique characteristics:

1. Co-assembly procedure with read mapping for estimation of the abundances of genes in each metagenome
2. Co-assembly of a large number of metagenomes via merging of individual metagenomes
3. Includes binning and bin checking, for retrieving individual genomes
4. The results are stored in a database, where they can be easily exported and shared, and can be inspected anywhere using a web interface.
5. Internal checks for the assembly and binning steps inform about the consistency of contigs and bins, allowing to spot potential chimeras.
6. Metatranscriptomic support via mapping of cDNA reads against reference metagenomes

SqueezeMeta supports different assembly strategies (co-assembly, sequential, assembly merging, and sequential-merging) and different assemblers (see below for details). SqueezeMeta uses a combination of custom scripts and external software packages for the different steps of the analysis:

1. Assembly
2. RNA prediction and classification
3. ORF (CDS) prediction
4. Homology searching against taxonomic and functional databases

5. Hmmer searching against Pfam database
6. Taxonomic assignment of genes
7. Functional assignment of genes
8. Blastx on parts of the contigs with no gene prediction or no hits (OPTIONAL)
9. Taxonomic assignment of contigs, and check for taxonomic disparities
10. Coverage and abundance estimation for genes and contigs
11. Estimation of taxa abundances
12. Estimation of function abundances
13. Merging of previous results to obtain the ORF table
14. Binning with different methods
15. Binning integration with DAS tool
16. Taxonomic assignment of bins, and check for taxonomic disparities
17. Checking of bins with CheckM
18. Merging of previous results to obtain the bin table
19. Merging of previous results to obtain the contig table
20. Prediction of kegg and metacyc pathways for each bin
21. Final statistics for the run

2. Installation

SqueezeMeta is intended to be run in a x86_64 Linux OS (tested in Ubuntu, Debian10 and CentOS7). The easiest way to install it is by using conda. Conda might however be slow solving the dependencies, so it's better to first get `mamba` into your base environment with

```
conda install -c conda-forge mamba
```

and then use `mamba` to install SqueezeMeta

```
mamba create -n SqueezeMeta -c bioconda -c conda-forge -c anaconda
-c fpusan squeezemeta --no-channel-priority
```

This will create a new conda environment named SqueezeMeta, which must then be activated.

```
conda activate SqueezeMeta
```

When using conda, all the scripts from the SqueezeMeta distribution will be available on `$PATH`.

Alternatively, you can download the latest release from the GitHub repository and uncompress the tarball in a suitable directory. The tarball includes the SqueezeMeta scripts as well as the third-party software redistributed with SqueezeMeta. The INSTALL files contain detailed installation instructions, including all the external libraries required to make SqueezeMeta run in a vanilla Ubuntu 20.04. Note that you may need different libraries and potentially recompiling some binaries from source in order for the manual install to work in other Ubuntu versions or other distributions. The conda method is now the recommended way to install SqueezeMeta, and we may not be able to support issues regarding manual installation.

The test_install.pl script can be run in order to check whether the required dependencies are available in your environment.

```
/path/to/SqueezeMeta/utils/install_utils/test_install.pl
```

3. Downloading or building databases

SqueezeMeta uses several databases. GenBank nr for taxonomic assignment, and eggnog, KEGG and Pfam for functional assignment. The script `download_databases.pl` can be run to download a pre-formatted version of all the databases required by SqueezeMeta.

```
/path/to/SqueezeMeta/utils/install_utils/download_databases.pl
/download/path/
```

, where `/download/path/` is the destination folder. This is the recommended option.

Alternatively, the script *make_databases.pl* can be run to download from source and format the latest version of the databases.

```
/path/to/SqueezeMeta/utils/install_utils/make_databases.pl
/download/path/
```

The databases occupy 200Gb, but we recommend having at least 350Gb free disk space during the building process.

If the SqueezeMeta databases are already built in another location in the system, a different copy of SqueezeMeta can be configured to use them with

```
/path/to/SqueezeMeta/utils/install_utils/configure_nodb.pl
/path/to/db/
```

After configuring the databases, the `test_install.pl` can be run in order to check that SqueezeMeta is ready to work (see previous section). If `download_databases.pl` or `make_databases.pl` can't find our server, you can instead run `make_databases_alt.pl` (same syntax as `make_databases.pl`) which will try to download the data from an alternative site.

4. Choosing an assembly strategy

SqueezeMeta can be run in five different modes, depending of the type of multi-metagenome support. These modes are:

- Sequential mode: All samples are treated individually and analysed sequentially.
- Coassembly mode: Reads from all samples are pooled and a single assembly is performed. Then reads from individual samples are mapped to the coassembly to obtain gene abundances in each sample. Binning methods allow to obtain genome bins.
- Merged mode: if many big samples are available, co-assembly could crash because of memory requirements. This mode allows the co-assembly of an unlimited number of samples, using a procedure inspired by the one used by Benjamin Tully for analysing TARA Oceans data (https://dx.doi.org/10.17504/protocols.io.hfqb3mw). Briefly, samples are assembled individually and the resulting contigs are merged in a single co-assembly. Then the analysis proceeds as in the co-assembly mode. This is not the recommended procedure (use co-assembly if possible) since the possibility of creating chimeric contigs is higher. But it is a viable alternative when standard co-assembly is not possible.
- Seqmerge mode: This is intended to work with more samples than the merged mode. Instead of merging all individual assemblies in a single step, which can be very computationally demanding, seqmerge works sequentially. First, it assembles individually all samples, as in merged mode. But then it will merge the two most similar assemblies. Similarity is measured as Amino Acid Identity values using the wonderful CompareM software by Donovan Parks. After this first merging, it again evaluates similarity and merge, and proceeds this way until all metagenomes have been merged in one. Therefore, for n metagenomes, it will need n-1 merging steps.
- Clustered mode: When having a very big number of samples, it is possible that coassembly or merged modes won't work. This mode is a sequential one, assembling all samples separately and predicting genes for each of them, but then performs a clustering on the predicted protein sequences, using the MMseqs2

software (https://github.com/soedinglab/MMseqs2). This identifies similar proteins in the different samples, and decreases the analysis time because all subsequent steps will be done using the clustered set of proteins, which will be smaller than the original.

Note that the `merged` and `seqmerge` modes work well as a substitute of coassembly for running small datasets in computers with low memory (e.g. 16 Gb) but are very slow for analysing large datasets (>10 samples) even in workstations with plenty of resources. Still, setting `-contiglen` to 1000 or higher can make `seqmerge` a viable strategy even in those cases. Otherwise, we recommend to use either the sequential or the clustered modes.

Regarding the choice of assembler, MEGAHIT and SPAdes work better with short Illumina reads, while Canu and Flye support long reads from PacBio or ONT-Minion. MEGAHIT (the default in SqueezeMeta) is more resource-efficient than SPAdes, consuming less memory, but SPAdes supports more analysis modes and produces slightly better assembly statistics. SqueezeMeta can call SPAdes in three different ways. The option `-a spades` is meant for metagenomic datasets, and will automatically add the flags `--meta -k 21,33,55,77,99,127` to the `spades.py` call. Conversely, `-a rnaspades` will add the flags `--rna -k 21,33,55,77,99,127`. Finally, the option `-a spades_base` will add no additional flags to the `spades.py` call. This can be used in conjunction with `--assembly options` when one wants to fully customize the call to SPAdes, e.g. for assembling single cell genomes.

5. Execution, restart and running scripts

Scripts location

The scripts composing the SqueezeMeta pipeline can be found in the `.../SqueezeMeta/scripts` directory. We recommend adding it to your `$PATH` environment variable.

Execution

The command for running SqueezeMeta has the following syntax:

```
SqueezeMeta.pl -m <mode> -p <project name> -s <equiv file> -f <raw fastq dir> <options>
```

Arguments

Mandatory parameters

- `-m <mode>`: run mode (sequential, coassembly, merged, seqmerge, clustered) (REQUIRED)
- `-p <project>`: project name (REQUIRED in all modes except sequential)
- `-s <samples file>|-samples`: samples file (REQUIRED)
- `-f|-seq <sequence dir>`: fastq read files' directory (REQUIRED)

Restarting

- `--restart: Restarts the given project where it stopped (project must be specified with -p option) (will NOT overwrite previous results, unless --force_overwrite is specified)`
- `-step <step number>: In combination with --restart, restarts the project starting in the given step number (combine with --force_overwrite to regenerate results`

Filtering

- `--cleaning`: filters with Trimmomatic (Off by default)
- `-cleaning_options [options]`: options for Trimmomatic (default if not specified: “LEADING:8 TRAILING:8 SLIDINGWINDOW:10:15 MINLEN:30”). Please provide all options as a single quoted string

Assembly

- `-a [assembler]` (megahit,spades,rnaspades,spades-base,canu,flye): assembler. (default: megahit).
- `-assembly_options [options]`: options for the assembler (refer to manual of the specified assembler). Please provide all the extra options as a single quoted string (e.g. `-assembly_options “--opt1 foo --opt2 bar”`)
- `-c|-contiglen <contig size>`: minimum length of contigs to keep (default: 200)
- `-extassembly <file>`: name of the file containing an external assembly provided by the user. The file must contain contigs in fasta format. This overrides the assembly step of SqueezeMeta.
- `--sg|--singletons`: unassembled will be treated as contigs and included in the contig fasta file resulting from the assembly. This will produce 100% mapping

percentages, and will increase BY A LOT the number of contigs to process. Use with caution (default: no)

- `-contigid <string>`: Nomenclature for contigs (Default: assembler´s name)
- `--norename: Don't rename contigs (Use at your own risk, characters like '_' in contig names will make it crash)`

Mapping

- `-map <mapper>`: mapping (aligner) software [bowtie,bwa,minimap2-ont,minimap2-pb,minimap2-sr] (default: bowtie)
- `-mapping_options [options]`: Extra options for the mapper (refer to the manual of the specific mapper). Please provide all the extra options as a single quoted string (e.g. `-mapping_options "--opt1 foo --opt2 bar"`)

Annotation

- `--nodiamond: Check if Diamond results are already in place, and just in that case skips the Diamond run (Default: no)`
- `--nocog`: skip COG assignment
- `--nokegg`: skip KEGG assignment
- `--nopfam`: skip Pfam assignment
- `--euk`: Drop identity filters for eukaryotic annotation. This is recommended for analyses in which the eukaryotic population is relevant, as it will yield more annotations. See The LCA algorithm for details.
- `-consensus <value>`: Minimum percentage of genes for a taxon needed for contig consensus (Default: 50)
- `-extdb <database file>`: list of user-provided databases for functional annotations.
- `-b|-block-size <block size>`: block size for DIAMOND against the nr database. Lower values reduce RAM memory usage. Set it to 3 or below for running in a desktop computer (default: calculate automatically)
- `--D|--doublepass`: extra-sensitive ORF prediction. first pass looking for genes using gene prediction, second pass using BlastX (Off by default)

Binning

- `--nobins`: skip all binning. Overrides `-binners`

- `-binners`: Specify binning programs to be used (available: `maxbin, metabat2, concoct`) (Default: `metabat2,concoct`)
- `-taxbinmode <s,c,s+c,c+s>`: Source of taxonomy annotation of bins (s: SqueezeMeta; c: CheckM; s+c: SqueezeMeta+CheckM; c+s: CheckM+SqueezeMeta; (Default: s). This will use CheckM to complement/override SqueezeMeta taxa assignments for bins. Modes s and c trust only in SqueezeMeta and CheckM taxonomies respectively. Modes s+c and c+s combine both, giving preference to SqueezeMeta or CheckM assignments, respectively.

Performance

- `-t <threads>`: number of threads (default: 12)
- `-canumem <memory>`: memory for Canu in Gb (default: 32)
- `--lowmem`: run on less than 16 Gb of RAM memory (default:no)

Settings for MinION

- `--minion`: run on MinION reads (use Canu and minimap2)

Other settings

- `-test <step>`: For testing purposes, stops AFTER the given step number
- `--empty`: Creates an empty directory structure and configuration files. It does not run the pipeline
- `--force_overwrite`: It will overwrite existing results

Information

- `-v`: displays version number
- `-h`: help

Example SqueezeMeta call:

```
SqueezeMeta.pl -m coassembly -p test -s test.samples -f mydir
--nopfam
```

This will create a project "`test`" for co-assembling the samples specified in the file "`test.samples`", skipping Pfam annotation. The `-p` parameter indicates the name under which all results and data files will be saved. This is not required for sequential

mode, where the name will be taken from the samples file instead. The `-f` parameter indicates the directory where the read files specified in the sample file are stored.

The samples file

The samples file specifies the samples, the names of their corresponding raw read files and the sequencing pair represented in those files, separated by tabulators.

It has the format: `<Sample> <filename> <pair1|pair2>`

An example could be

```
Sample1 readfileA_1.fastq      pair1
Sample1 readfileA_2.fastq      pair2
Sample1 readfileB_1.fastq      pair1   noassembly
Sample1 readfileB_2.fastq      pair2   noassembly
Sample2 readfileC_1.fastq.gz   pair1
Sample2 readfileC_2.fastq.gz   pair2
Sample3 readfileD_1.fastq      pair1   nobinning,extassembly=/path/to/extassembly
Sample3 readfileD_2.fastq      pair2   nobinning,extassembly=/path/to/extassembly
```

The first column indicates the sample ID (this will be the project name in sequential mode), the second contains the file names of the sequences, and the third specifies the pair number of the reads. A fourth optional column can take the following comma-separated values:

"`noassembly`" indicates that these samples must not be assembled with the rest (but will be mapped against the assembly to get abundances). This is the case for RNAseq reads that can hamper the assembly but we want them mapped to get transcript abundance of the genes in the assembly.

"`nobinning`" value can be included in order to avoid using those samples for binning.

"`extassembly`" indicates the external assemblies for each of the samples, if they are to be used. This allows to specify different external assemblies for each sample when running in sequential mode. For coassembly, merged or seqmerge modes, use the SqueezeMeta *-extassembly* option instead (see Section 5).

Notice that a sample can have more than one set of paired reads. The sequence files can be in either fastq or fasta format, and can be gzipped.

The parameters file

The file `parameters.pl` stored in the `scripts` directory sets several parameters used by SqueezeMeta's scripts. You can change them to adjust the performance of the pipeline.

Restart

Any interrupted SqueezeMeta run can be restarted invoking the option - -restart

```
SqueezeMeta.pl <project name> --restart
```

This command will restart the run of that project by reading the progress.txt file to find out the point where the run stopped.

Alternatively, the user can specify the step in which to restart the analysis, using the `-step` option (refer to the scripts section for the list of steps):

```
SqueezeMeta.pl <project name> --restart -step <step number>
```

By default, this will NOT redo any step that was already done. If you want to override this behavior and redo previous steps, please add the - -force_overwrite option.

In sequential mode, where there is not a single project name because several individual projects are created, you need to restart the interrupted project, rewrite the samples file to eliminate the samples that were already processed, and then rerun SqueezeMeta with the remaining samples.

Running scripts

Also, any individual script of the pipeline can be run using the same syntax:

`script <project name>` (for instance, `04.rundiamond.pl <project name>` to repeat the DIAMOND run for the project)

6. Analyzing an user-supplied assembly

An user-supplied assembly can be passed to SqueezeMeta with the flag *-extassembly <your_assembly.fasta>*. The contigs in that fasta file will be analyzed by the SqueezeMeta pipeline starting from step 2.

7. Using user-supplied databases for taxonomic annotation

This feature was included in version 1.7 of SqueezeMeta. Instead of using the default GenBank nr databases, the user can provide a custom database for taxonomic

annotation. This database must be provide as a fasta file, with the only requirement of being compliant with GenBank taxonomy and headers. That is, the sequence headers must follow this format:

```
>MBO6898974.1 [Shimia sp.] [Rhodobacteraceae bacterium]
```

Specifying the taxa between brackets, following an (arbitrary) accession number. As in the example above, any number of taxa can be given to each of the sequences (for instance, if the database is the result of clustering a bigger database, each of the (clustered) sequences can belong to several taxa, corresponding to the members of the cluster).

The script **make_custom_taxdb.pl** must be used to format this database for usage with SqueezeMeta, as follows:

make_custom_taxdb.pl <directory for new database> <db fasta file>

Then, the directory containing the new database must be specified to SqueezeMeta.pl using the -db option. This will produce taxonomic annotations using the new database.

8. Using external databases for functional annotation

Version 1.0 of SqueezeMeta implements the possibility of using one or several external databases (user-provided) for functional annotation. This is invoked using the `--extdb` option. The argument must be a file (external database file) with the following format (tab-separated fields):

```
<Database Name> <Path to database>    <Functional annotation file>
```

For example, we can create the file mydb.list containing information of two databases:

```
DB1   /path/to/my/database1 /path/to/annotations/database1
DB2   /path/to/my/database2 /path/to/annotations/database2
```

and give it to SqueezeMeta using `--extdb mydb.list`.

Each database must be a fasta file of amino acid sequences, in which the sequences must have a header in the format:

```
>ID|...|Function
```

Where ID can be any identifier for the entry, and Function is the associated function that will be used for annotation. For example, a KEGG entry could be something like:

```
>WP_002852319.1|K02835
MKEFILAKNEIKTMLQIMPKEGVVLQGDLASKTSLVQAWVKFLVLGLDRVDSTPTFSTQKYE...
```

You can put anything you want between the first and last pipe, because these are the only fields that matter. For instance, the previous entry could also be:

```
>WP_002852319.1|KEGGDB|27/02/2019|K02835

MKEFILAKNEIKTMLQIMPKEGVVLQGDLASKTSLVQAWVKFLVLGLDRVDSTPTFSTQKYE...
```

Just remember not to put blank spaces, because they act as field separators in the fasta format.

This database must be formatted for DIAMOND usage. For avoiding compatibility issues between different versions of DIAMOND, it is advisable that you use the DIAMOND that is shipped with SqueezeMeta, and is placed in the *bin* directory of SqueezeMeta distribution. You can do the formatting with the command:

```
/path/to/SqueezeMeta/bin/diamond makedb -d
/path/to/ext/db/dbname.dmnd --in /path/to/my/ext/dbname.fasta
```

For each database, you can OPTIONALLY provide a file with functional annotations, such as the name of the enzyme or whatever you want. Its location must be specified in the last field of the external database file. It must have only two columns separated by tabulators, the first with the function, the second with the additional information. For instance:

```
K02835    peptide chain release factor 1
```

The ORF table will show both the database ID and the associated annotation for each external database you provided.

9. Extra-sensitive detection of ORFs

Version 1.0 implements the `-D` option (doublepass), that attempts to provide a more sensitive ORF detection by combining the Prodigal prediction with a BlastX search on parts of the contigs where no ORFs were predicted, or where predicted ORFs did not match anything in the taxonomic and functional databases. The procedure starts after the usual taxonomic and functional annotation. It masks the parts of the contigs in which there is a predicted ORF with some (taxonomic and functional) annotation. The remaining sequence corresponds to zones with no ORF prediction or orphan genes (no hits). The first could correspond to missed ORFs, the second to wrongly predicted ORFs. Then a DIAMOND BlastX is run only on these zones, using the same databases. The resulting hits are added as newly predicted ORFs, and the pipeline continues taking into account these new ORFs.

The pros: This procedure is able to detect missing genes or correct errors in gene prediction (for example, these derived from frameshifts). For prokaryotic metagenomes, we estimate a gain of 2-3% in the number of ORFs. This method is especially useful when

you suspect that gene prediction can underperform, for instance in cases in which eukaryotes and viruses are present. Prodigal is a prokaryotic gene predictor and its behaviour for other kingdoms is uncertain. In these cases, the gain can be higher than for prokaryotes.

The con: Since it has to do an additional DIAMOND run (and using six frame-Blastx) it slows down the analysis, especially in the case of big and/or many metagenomes.

10. Testing SqueezeMeta

The `download_databases.pl` and `make_databases.pl` scripts also download two datasets for testing that the program is running correctly. Assuming either was run with the directory `/download/path` as its target the test run can be executed with

```
cd /download/path/test

/path/to/SqueezeMeta/SqueezeMeta.pl -m coassembly -p Hadza -s
test.mock.samples -f raw
```

Alternatively, `-m sequential` or `-m merged` can be used.

In addition to this mock dataset, we also provide two real metagenomes. A test run on those can be executed with

```
SqueezeMeta.pl -m coassembly -p Hadza -s test.samples -f raw
```

11. Working with Oxford Nanopore MinION and PacBio reads

Since version 0.3.0, SqueezeMeta is able to seamlessly work with single-end reads. In order to obtain better mappings of MinION and PacBio reads against the assembly, we advise to use minimap2 for read counting, by including the `-map minimap2-ont` (MinION) or `-map minimap2-pb` (PacBio) flags when calling SqueezeMeta.

We also include the Canu and Flye assemblers, which are specially tailored to work with long, noisy reads. They can be selected by including the `-a canu` or `-a flye` flag when calling SqueezeMeta. As a shortcut, the `--minion` flag will use both Canu and minimap2 for Oxford Nanopore MinION reads.

As an alternative to assembly, we also provide the `sqm_longreads.pl` script, which will predict and annotate ORFs within individual long reads.

12. Working in a low-memory environment

In our experience, assembly and DIAMOND alignment against the nr database are the most memory-hungry parts of the pipeline. By default SqueezeMeta will set up the right parameters for DIAMOND and the Canu assembler based on the available memory in the system. DIAMOND memory usage can be controlled via the `-b` parameter (DIAMOND will consume ~5**b* Gb of memory according to the documentation, but to be safe we set `-b` to `free_ram/8`). Assembly memory usage is trickier, as memory requirements increase with the number of reads in a sample. We have managed to run SqueezeMeta with as much as 42M 2x100 Illumina HiSeq pairs on a virtual machine with only 16Gb of memory. Conceivably, larger samples could be split an assembled in chunks using the merged mode. We include the shortcut flag `--lowmem`, which will set DIAMOND block size to 3, and Canu memory usage to 15Gb. This is enough to make SqueezeMeta run on 16Gb of memory, and allows the *in situ* analysis of Oxford Nanopore MinION reads. Under such computational limitations, we have been able to coassemble and analyze 10 MinION metagenomes (taken from SRA project SRP163045) in less than 4 hours.

13. Tips for working in a computing cluster

SqueezeMeta will work fine inside a computing cluster, but there are some extra things that must be taken into account. Here is a list in progress based on frequent issues that have been reported.

- Run `test_install.pl` to make sure that everything is properly configured.
- If using the conda environment, make sure that it is properly activated by your batch script-
- If an administrator has set up SqueezeMeta for you (and you have no write privileges in the installation directory), make sure they have run `make_databases.pl`, `download_databases.pl` or `configure_nodb.pl` according to the installation instructions. Once again, `test_install.pl` should tell you whether things seem to be ok.
- Make sure to request enough memory. See the previous section for a rough guide on what is "enough". If you get a crash during the assembly or during the annotation step, it will be likely because you ran out of memory.
- Make sure to manually set the `-b` parameter so that it matches the amount of memory that you requested divided by `8`. Otherwise, SqueezeMeta will assume that it can use all the free memory in the node in which it is running. This is fine if you got a full node for yourself, but will lead to crashes otherwise.

14. Updating SqueezeMeta

Assuming your databases are not inside the SqueezeMeta directory, just remove it, download the new version and configure it with

```
/path/to/SqueezeMeta/utils/install_utils/configure_nodb.pl
/path/to/db/
```

15. Downstream analysis of SqueezeMeta results

SqueezeMeta comes with a variety of options to explore the results and generate different plots. These are fully described further in this manual. Briefly, the three main ways to analyze the output of SqueezeMeta are the following:

1) Integration with R**:** We provide the *SQMtools* R package, which allows to easily load a whole SqueezeMeta project and expose the results into R. The package includes functions to select particular taxa or functions and generate plots. The package also makes the different tables generated by SqueezeMeta easily available for third-party R packages such as *vegan* (for multivariate analysis), *DESeq2* (for differential abundance testing) or for custom analysis pipelines. ***SQMtools* can also be used in Mac or Windows**, meaning that you can run SqueezeMeta in your Linux server and then move the results to your own computer and analyze them there. See advice on how to install *SQMtools* in Mac/Windows here.

2) Integration with the anvi'o analysis pipeline**:** We provide a compatibility layer for loading SqueezeMeta results into the anvi'o analysis and visualization platform (http://merenlab.org/software/anvio/). This includes a built-in query language for selecting the contigs to be visualized in the anvi'o interactive interface.

We also include utility scripts for generating itol and pavian -compatible outputs.

16. Analyzing SqueezeMeta results in your desktop computer

Many users run SqueezeMeta remotely (e.g. in a computing cluster). However it is easier to explore the results interactively from your own computer. Since version 1.6.2, we provide an easy way to achieve this.

1) In the system in which you ran SqueezeMeta, run the utility script `sqm2zip.py /path/to/my_project /output/dir`, where `/path/to/my_project` is the path to the output of SqueezeMeta, and `/output/dir` an arbitrary output directory.

2) This will generate a file in `/output/dir` named `my_project.zip`, which contains the essential files needed to load your project into SQMtools. Transfer this file to your desktop computer.

3) Assuming R is present in your desktop computer, you can install SQMtools with

```
if (!require("BiocManager", quietly = TRUE))
    install.packages("BiocManager")
BiocManager::install("SQMtools")
```

This will work seamlessly in Windows and Mac computers, for Linux you may need to previously install the `libcurl` development library.

4) You can then load the project directly from the zip file (no need for decompressing) with `import(SQMtools); SQM = loadSQM("/path/to/my_project.zip")`.

17. Alternative analysis modes

In addition to the main SqueezeMeta pipeline, we provide extra modes that enable the analysis of individual reads.

1) **sqm_reads.pl**: This script performs taxonomic and functional assignments on individual reads rather than contigs. This can be useful when the assembly quality is low, or when looking for low abundance functions that might not have enough coverage to be assembled.

2) **sqm_longreads.pl**: This script performs taxonomic and functional assignments on individual reads rather than contigs, assuming that more than one ORF can be found in the same read (e.g. as happens in PacBio or MinION reads).

3) **sqm_hmm_reads.pl**: This script provides a wrapper to the Short-Pair software, which allows to screen the reads for particular functions using an ultra-sensitive HMM algorithm.

A thorough description of these programs can be found in the section "Utilities: alternative analysis modes"

18. Adding new binners and assemblers

Version 1.5 of SqueezeMeta allows connecting other binning tools than the shipped ones (maxbin, metabat2 and concoct). To do so, you must create a script for putting the new binning results in a new directory under project/intermediate/binners. For instance, suppose that you have created a script **amazingbinner.py** that runs the new binning tool "amazingbinner", and your project will be named "myrun". Your script must create a directory "myrun/intermediate/binners/amazingbinner and put the fasta files resulting from the binning in that directory (these fasta files MUST have extension .fasta or .fa).

Next, edit the `SqueezeMeta_conf.pl` in the scripts directory of the SqueezeMeta installation. You will see a line like this:

```
%binscripts=('maxbin',"$installpath/lib/SqueezeMeta/bin_maxbin.pl"
,'metabat2',"$installpath/lib/SqueezeMeta/bin_metabat2.pl",'concoc
t',"$installpath/lib/SqueezeMeta/bin_concoct.pl");
```

This tells SqueezeMeta the available scripts for running binners. Add your new script:

```
%binscripts=('maxbin',"$installpath/lib/SqueezeMeta/bin_maxbin.pl"
,'metabat2',"$installpath/lib/SqueezeMeta/bin_metabat2.pl",'concoc
t',"$installpath/lib/SqueezeMeta/bin_concoct.pl",'amazingbinner',"
mylocation/amazingbinner.py");
```

Where "mylocation" is the directory where you put your script (You may want to move it to the lib/SqueezeMeta directory in the SqueezeMeta installation, to have all binning scripts in one place).

Next, give execution permissions to the script (for instance, sudo chmod a+x `mylocation/amazingbinner.py`). Notice that the script can be written in any language (python in this example), as long as it is executable and provides the results in the expected place.

And you are done! Now, for running SqueezeMeta with your new binner, just mention it using the -binners option, for instance:

```
SqueezeMeta.pl -p myrun -s mysamples -f /path/to/sequences -m
coassembly -binners "maxbin,metabat,concoct,amazingbinner"
```

From version 1.6, SqueezeMeta also allows the connection of other assemblers than the ones shipped with the distro (MEGAHIT, SPAdes, Canu and Flye). Here I will teach you a practical example of how to do it, showing the plugging of the IDBA-UD assembler (https://github.com/loneknightpy/idba) (Peng et al, Bioinformatics 2012, 28:111420-1428; https://doi.org/10.1093/bioinformatics/bts174)

I will assume that you already installed the IDBA-UD software and put it somewhere in your system, for instance the /software/idba directory (my choice, but you can put it wherever you want)

What you need to do is to create a script to run the assembler. Your script will be called by the SqueezeMeta pipeline. I named my script `assembly_idba.pl`, and it looks like this:

```
#!/usr/bin/perl
use strict;
print "Running IDBA assembly\n";
`#-- By default, SqueezeMeta will pass the following arguments to your script:`

my $projectdir=$ARGV[0];     # First argument: Directory of the project
my $sample=$ARGV[1];                 # Second argument: Name of the sample (in sequential mode) or project (in the
rest)
my $par1name=$ARGV[2];               # Third argument: Name of the pair1 file
```

```
my $par2name=$ARGV[3];                  # Fourth argument: Name of the pair2 file
my $numthreads=12;                      #-- In addition, we can define other parameters, for instance number of threads
`#-- IDBA wants data as an interlaced fasta file`

`#-- Fortunately, they provide a fq2fa script converting our fastq files to that format`

`#-- But, our fastq files are gzipped. Therefore first thing is to gunzip them`

`#-- We define $g1 and $g2 as variables containing the name of the gunzipped files`
  `#-- Simply remove the ".gz" extension to get the gunzipped name`

my $g1=$par1name; $g1=~s/\.gz$//; my $g2=$par2name; $g2=~s/\.gz$//;
my $fastafile="temp.fasta";    #-- And we define $fastafile as the resulting interlaced fasta file
`#-- Now, we gunzip the files and run the fq2fa script`

my $merge_command="gunzip $par1name; gunzip $par2name; /software/idba/bin/fq2fa --merge $g1 $g2 $fastafile";
system($merge_command);
`#-- And then we can run the IDBA assembler, just providing the input filename ($fastafile)`

`#-- We could the desired assembler options to this command line.`

`#-- For instance, we added the number of threads`

`#-- The results will be stored in a directory we named "tempidba"`

my $assembly_command="/software/idba/bin/idba -r $fastafile --num_threads $numthreads -o tempidba";
system($assembly_command);
`#-- Finally, we have to move the resulting fasta file to the "results" directory of the SqueezeMeta project`

`#-- IDBA names the file "scaffold.fa"`

`#-- Keep in mind that the file must be named "01.project.fasta"`

my $mv_command="mv tempidba/scaffold.fa $projectdir/results/01.$sample.fasta"; system($mv_command);
#-- To finish, we clean up things we don't need anymore (the temporal directory and fasta files)
my $rm_command="rm -r tempidba; rm $g1; rm $g2"; system($rm_command);
print "All done here! Have fun!\n";
```

Take into account that when SqueezeMeta calls your script, it will pass four arguments to it: The project directory, the project name, and the read files (two paired-end, gunzipped fastq or fasta files). This is probably all you need to know to call the assembler.

In the script I run a formatting script `fq2fa` provided by `IDBA-UD`, to put the runs in the format it wants them. Then I run the assembler, and finally I move the resulting contig file to the results directory in the SqueezeMeta project. This is very important because the rest of the pipeline will look for the contig file there. Also, take into account that the name of the contig file must be `01.project.fasta` (where project is your project name).

To plug this into SqueezeMeta, the first thing to do is to move your script to the place where all other assembly scripts are, which is the `installpath/lib/SqueezeMeta` directory (where `installpath` is the installation directory of SqueezeMeta. You will see there other scripts for running assemblers, like `assembly_megahit.pl`, `assembly_spades.pl`, etc). Then, edit the `SqueezeMeta_conf.pl` file in the scripts directory of the SqueezeMeta installation. You will see a line like this:

```
%assemblers = ("megahit","assembly_megahit.pl","spades",
"assembly_spades.pl","canu","assembly_canu.pl","flye",
"assembly_flye.pl");
```

This line is a hash (equivalent to a dict in python), telling SqueezeMeta the names of the available assemblers and the associated scripts for running them. Just add yours. Remember that the name you specify will be the one to run the assembler:

```
%assemblers = ("megahit","assembly_megahit.pl","spades",
"assembly_spades.pl","canu","assembly_canu.pl","flye",
"assembly_flye.pl",”idba”,”assembly_idba.pl”);
```

Save it, and you are done. Now you can run a SqueezeMeta project using your new “idba” assembler:

```
SqueezeMeta.pl -m coassembly -f mydir -s mysamples.samples -p
idba_test -a idba
```

19. License and third-party software

SqueezeMeta is distributed under a GPL-3 license. Additionally, SqueezeMeta redistributes the following third-party software:

- trimmomatic
- Megahit
- Spades
- canu
- prinseq
- kmer-db
- CD-HIT
- amos
- mummer

- hmmer
- aragorn
- DIAMOND
- bwa
- minimap2
- bowtie2
- barrnap
- MaxBin
- MetaBAT
- CONCOCT
- DAS tool
- checkm
- comparem
- MinPath
- RDP classifier
- pullseq
- Short-Pair
- SAMtools

20. About

SqueezeMeta is developed by Javier Tamames and Fernando Puente-Sánchez. Feel free to contact us for support (jtamames@cnb.csic.es, fpuente@cnb.csic.es).

Scripts, output files and file format

Files marked in blue are placed in the "`results`" directory; in green, files in "`intermediate`" directory; orange, in "`ext_tables`" directory:

Step 1: Assembly

Script: 01.run_assembly.pl (or 01.run_assembly_merged.pl)

Files produced:

- 01.<project>.fasta: Fasta file containing the contigs resulting from the assembly
- 01.<project>.lon: Length of the contigs

(Merged mode will also produce a .fasta and a .lon file for each sample)

- 01.<project>.stats: Some statistics on the assembly (N50, N90, number of reads, etc)
- 01.<project>.mappingstat: mapping percentage of reads to samples

  Format of the file:

  - Column 1: sample name
  - Column 2: total reads for the sample
  - Column 3: mapped reads
  - Column 4: percentage of mapped reads
  - Column 5: total bases for the sample

Step 2: RNA finding

Script: 02.run_barrnap.pl

Files produced:

- 02.<project>.rnas: Fasta file containing all RNAs found
- 02.<project>.16S: Assignment (RDP classifier) for the 16S rRNAs sequences found.

- 02.<project>.maskedrna.fasta: Fasta file containing the contigs resulting from the assembly, masking the positions where a RNA was found.

Step 3: Gene prediction

Script: 03.run_prodigal.pl

Files produced:

- 03.<project>.fna: Nucleotide sequences for predicted ORFs
- 03.<project>.faa: Aminoacid sequences for predicted ORFs
- 03.<project>.gff: Features and position in contigs for each of the predicted genes (moves to intermediate if the `-D` option is selected)

Step 4: Homology searching against taxonomic (nr) and functional (COG, KEGG) databases

Script: 04.rundiamond.pl

Files produced:

- 04.<project>.nr.diamond: result of the homology search for nr
- 04.<project>.eggnog.diamond: result of the homology search for COG
- 04.<project>.kegg.diamond: result of the homology search for KEGG

(the nr is updated regularly. You can check the date nr was downloaded by looking at the DB_BUILD_DATE file in the database directory. The KEGG database requires licensing, therefore we use the last publicly available version. For the COG/eggNOG database we currently use version 4.5)

- 04.<project>.optdb.diamond: result of the homology search for the optional database provided

Step 5: HMM search for Pfam database

Script: 05.run_hmmer.pl

Files produced:

- 05.<project>.pfam.hmm: results of the HMM search

(The Pfam database is updated regularly)

Step 6: Taxonomic assignment

Script: 06.lca.pl

Files produced:

- 06.<project>.fun3.tax.wranks: taxonomic assignments for each ORF, including taxonomic ranks (moves to intermediate if the `-D` option is selected)
- 06.<project>.fun3.tax.noidfilter.wranks: same as above, but assignment was done not considering identity filters (refer to the explanation of the LCA algorithm in the algorithm section)

Step 7: Functional assignment

Script: 07.fun3assign.pl

Files produced:

- 07.<project>.fun3.cog: COG functional assignment for each ORF (moves to intermediate if `-D` option is selected)
- 07.<project>.fun3.kegg: KEGG functional assignment for each ORF (moves to intermediate if `-D` option is selected)
- 07.<project>.fun3.optdb: functional assignment for each ORF for the optional database provided (moves to intermediate if the `-D` option is selected)

  Format of these files:

  - Column 1: Name of the ORF
  - Column 2: Best hit assignment
  - Column 3: Best average assignment (refer to the explanation of the fun3 algorithm)
- 07.<project>.pfam: PFAM functional assignment for each ORF.

Step 8: Blastx on parts of the contigs without gene prediction or without hits

Script: 08.blastx.pl

Files produced:

- 08.<project>.gff: features and position in contigs for each of the prodigal and Blastx ORFs.
- 08.<project>.blastx.fna: nucleotide sequences for blastx ORFs
- 08.<project>.fun3.tax.wranks: taxonomic assignment for the mix of prodigal and Blastx ORFs, including taxonomic ranks
- 08.<project>.fun3.cog: COG functional assignment for the mix of prodigal and Blastx ORFs
- 08.<project>.fun3.kegg: KEGG functional assignment for the mix of prodigal and Blastx ORFs
- 08.<project>.fun3.opt_db: functional assignment for the mix of prodigal and Blastx ORFs, for the optional database provided.

  The format of these last three files is the same as above (step 7)

Step 9: Contig assignment

Script: 09.summarycontigs3.pl

Files produced:

- 09.<project>.contiglog: assignment of contigs based on ORFs annotations

  Format of the file:

  - Column 1: name of the contig
  - Column 2: taxonomic assignment, with ranks
  - Column 3: lower rank of the assignment
  - Column 4: disparity value (refer to the algorithm section)
  - Column 5: number of genes in the contig

For detailed information on the algorithm, please refer to algorithm's section at the end of this manual.

Step 10: Mapping of reads to contigs and calculation of abundance measures

Script: 10.mapsamples.pl

Files produced:

- 10.<project>.mapcount: several measures regarding mapping of reads to ORFs

  Format of the file:

  - Column 1: ORF name
  - Column 2: ORF length (nucleotides)
  - Column 3: number of reads mapped to that ORF
  - Column 4: number of bases mapped to that ORF
  - Column 5: RPKM value for the ORF
  - Column 6: coverage value for the ORF (Bases mapped / ORF length)
  - Column 7: TPM value for the ORF
  - Column 8: sample to which these abundance values correspond

- 10.<project>.contigcov: several measurements regarding mapping of reads to contigs

  Format of the file:

  - Column 1: ORF name
  - Column 2: coverage value for the contig
  - Column 3: RPKM value for the contig
  - Column 4: TPM value for the contig
  - Column 5: contig length (nucleotides)
  - Column 6: number of reads mapped to that contig
  - Column 7: number of bases mapped to that contig
  - Column 8: sample to which these abundance values correspond

Step 11: Calculation of the abundance of all taxa

Script: 11.mcount.pl

Files produced:

- 11.<project>.mcount

  Format of the file:

  - Column 1: taxonomic rank for the taxon

- Column 2: taxon
- Column 3: accumulated contig size: Sum of the length of all contigs for that taxon
- Column 4 (and all even columns from this one): number of reads mapping to the taxon in the corresponding sample
- Column 5 (and all odd columns from this one): number of bases mapping to the taxon in the corresponding sample

Step 12: Calculation of the abundance of all functions

Script: 12.funcover.pl

Files produced:

- 12.<project>.cog.stamp: COG function table for STAMP (http://kiwi.cs.dal.ca/Software/STAMP).

  Format of the file:

  - Column 1: functional class for the COG
  - Column 2: COG ID and function name
  - Column 3 and above: abundance of reads for that COG in the corresponding sample

- 12.<project>.kegg.stamp: KEGG function table for STAMP

  Format of the file:

  - Column 1: KEGG ID and function name
  - Column 2 and above: abundance of reads for that KEGG in the corresponding sample

- 12.<project>.cog.funcover: Several measurements of the abundance and distribution of each COG

  Format of the file:

  - Column 1: COG ID
  - Column 2: sample name
  - Column 3: number of different ORFs of this function in the corresponding sample (copy number)

- Column 4: sum of the length of all ORFs of this function in the corresponding sample (Total length)
- Column 5: sum of the bases mapped to all ORFs of this function in the corresponding sample (Total bases)
- Column 6: coverage of the function (Total bases / Total length)
- Column 7: TPM value for the function
- Column 9: number of the different taxa per rank (k: kingdom, p: phylum; c: class; o: order; f: family; g: genus; s: species) in which this COG has been found
- Column 10: function of the COG

- 12.<project>.kegg.funcover: several measurements of the abundance and distribution of each KEGG

  Format of the file: Same format than previous one but replacing COGs by KEGGs. Additionally, the function of the KEGG will be present in column 11, while column 10 will contain the name of the KEGG.

In addition, .funcover and .stamp files will be created for the user-provided databases specified via the `--extdb` argument.

Step 13: Creation of the ORF table

Script: 13.mergeannot2.pl

File produced:

- 13.<project>.orftable

  Format of the file:

  - Column 1: ORF name
  - Column 2: Contig name
  - Column 3: molecule (CDS or RNA)
  - Column 4: method of ORF prediction (prodigal, barrnap, blastx)
  - Column 5: ORF length (nucleotides)
  - Column 6: ORF length (amino acids)
  - Column 7: GC percentage for the ORF

- Column 8: Gene name
- Column 9: Taxonomy for the ORF
- Column 10: KEGG ID for the ORF (If a * sign is shown here, it means that the functional assignment was done by both best hit and best average scores, therefore is more reliable. Otherwise, the assignment was done using just the best hit, but there is evidence of a conflicting annotation)
- Column 11: KEGG function
- Column 12: KEGG functional class
- Column 13: COG ID for the ORF (If a * sign is shown here, it means that the functional assignment was done by both best hit and best average scores, therefore is more reliable. Otherwise, the assignment was done using just the best hit, but there is evidence of a conflicting annotation)
- Column 14: COG function
- Column 15: COG functional class
- Column 16: function in the external database provided
- Column 17: functional class or associated information provided for the external database.
- (If there is more than one external databases, all of them will be shown here)
- Column 18: Pfam annotation
- Column 19 and beyond: TPM, coverage, read count and base count for the ORF in the different samples

Step 14: Binning methods

Script: 14.runbinning.pl

Results produced:

- Directory intermediate/binners/maxbin containing fasta files of contigs corresponding to each bin (if maxbin was selected)
- Directory intermediate/binners/metabat containing fasta files of contigs corresponding to each bin (if metabat was selected)
- Directory intermediate/binners/concoct containing fasta files of contigs corresponding to each bin (if concoct was selected)

Step 15: Merging bins with DAS Tool

Script: 15.dastool.pl

Results produced:

- Directory results/DAS containing fasta files of contigs corresponding to each bin, corresponding to the merge of the previous binning methods.

Step 16: Taxonomic assignment of bins

Script: 16.addtax2.pl

Files produced:

- tax files for each fasta in the directory DAS (or the binning directory)
- 16.<project>.bintax:

  Format of the file:

  - Column 1: binning method
  - Column 2: name of the bin
  - Column 3: taxonomic assignment for the bin, with ranks
  - Column 4: size of the bin (accumulated sum of contig lengths)
  - Column 5: disparity of the bin (refer to the algorithm section)

For detailed information on the algorithm, please refer to algorithm's section at the end of this manual.

Step 17: Bin assessment with CheckM

Script: 17.checkM_batch.pl

File produced:

- 17.<project>.checkM

  Format of the file: concatenated CheckM output for each bin

Step 18: Creation of the bin table

Script: 18.getbins.pl

Files produced:

- 18.<project>.bincov: coverage and TPM values for each bin

  Format of the file:

  - Column 1: bin name
  - Column 2: binning method
  - Column 3: coverage of the bin in the corresponding sample (Sum of bases from reads in the sample mapped to contigs in the bin / Sum of length of contigs in the bin)
  - Column 4: TPM for the bin in the corresponding sample (Sum of reads from the corresponding sample mapping to contigs in the bin x 10^6 / Sum of length of contigs in the bin x Total number of reads)
  - Column 5: sample name

- 18.<project>.bintable: compilation of all data for bins

  Format of the file:

  - Column 1: bin name
  - Column 2: binning method
  - Column 3: taxonomic annotation (from the annotations of the contigs)
  - Column 4: taxonomy for the 16S rRNAs if the bin (if any)
  - Column 5: bin size (sum of length of the contigs)
  - Column 6: GC percentage for the bin
  - Column 7: number of contigs in the bin
  - Column 8: disparity of the bin
  - Column 9: completeness of the bin (checkM)
  - Column 10: contamination of the bin (checkM)
  - Column 11: strain heterogeneity of the bin (checkM)
  - Column 12 and beyond: coverage and TPM values for the bin in each sample.

Step 19: Creation of the contig table

Script: 19.getcontigs.pl

Files produced:

- 19.<project>.contigsinbins: list of contigs and corresponding bins
- 19.<project>.contigtable: compilation of data for contigs

  Format of the file:

  - Column 1: contig name
  - Column 2: taxonomic annotation for the contig (from the annotations of the ORFs)
  - Column 3: disparity of the contig
  - Column 4: GC percentage for the contig
  - Column 5: contig length
  - Column 6: number of genes in the contig
  - Column 7: bin to which the contig belong (if any)
  - Column 8 and beyond: values of coverage, TPM and number of mapped reads for the contig in each sample

Step 20: Prediction of pathway presence in bins using MinPath

Script: 20. minpath.pl

Files produced:

- 20.<project>.kegg.pathways: prediction of KEGG pathways in bins

  Format of the file:

  - Column 1: bin name
  - Column 2: taxonomic annotation for the bin
  - Column 3: number of KEGG pathways found
  - Column 4 and beyond: NF indicates that the pathway was not predicted. A number shows that the pathway was predicted to be present, and correspond to the number of enzymes of that pathway that were found.

- 20.<project>.metacyc.pathways: prediction of Metacyc pathways in bins

  Format of the file: same as for KEGG, but using MetaCyc pathways

Step 21: Final statistics for the run

Script: 21.stats.pl

File produced:

- 21.<project>.stats: several statistics regarding ORFs, contigs and bins.

Utilities: alternative analysis modes (Working with reads)

SQM_reads.pl

This procedure performs taxonomic and functional assignments directly on the reads. This is useful when the assembly is not good, usually because of low sequencing depth, high diversity of the microbiome, or both. One indication that this is happening can be found in the mappingstat file. Should you find there low mapping percentages (below 50%), it means that most of your reads are not represented in the assembly and can we can try to classify the reads instead of the genes/contigs. It will probably provide an increment in the number of annotations. But on the other hand, the annotations could be less precise (we are working with a smaller sequence) and you lose the capacity to map reads onto an assembly and thus comparing metagenomes using a common reference. Use this for Illumina reads. This method is less suited to analyze long MinION reads where more than one gene can be represented (See SQM_longreads.pl in that case). This script can be found in the `/path/to/SqueezeMeta/utils/` directory.

The usage of SQM_reads is very similar to that of SqueezeMeta:

```
sqm_reads.pl -p <project name> -s <equiv file> -f <raw fastq dir>
[options]
```

Arguments

Mandatory parameters

- `-p`: project name (REQUIRED)
- `-s|-samples`: samples file (REQUIRED)
- `-f|-seq`: fastq read files' directory (REQUIRED)

Options

- `--nocog`: skip COG assignment
- `--nokegg`: skip KEGG assignment
- `--nodiamond`: Assumes that Diamond output files are already in place (for instance, if you are redoing the analysis) and skips all Diamond runs.
- `--euk`: Drop identity filters for eukaryotes, to improve eukaryotic annotation (Defalut: no)
- `-extdb`: List of external databases to use in the functional annotation. See 6. Using external databases for functional annotation for details.
- `-t`: number of threads (default: 12)

- `-b|-block-size`: block size for DIAMOND against the nr database. Lower values reduce RAM memory usage. Set it to 3 or below for running in a desktop computer (default: 8)
- `-e|-evalue`: max e-value for discarding hits in the DIAMOND run (default: 1e-03)
- `-miniden`: identity percentage for discarding hits in DIAMOND run (default: 50)

The method will do a DIAMOND Blastx alignment of reads with nr, COG and KEGG databases, and will assign taxa as functions using the lca and fun3 methods, as SqueezeMeta does.

Output

It produces the following files:

- `<project>.out.allreads`: taxonomic and functional assignments for each read

  Format of the file:

  - Column 1: sample name
  - Column 2: read name
  - Column 3: corresponding taxon
  - Column 4 and beyond Functional assignments (COG, KEGG)

- `<project>.out.mcount`: abundance of all taxa

  Format of the file:

  - Column 1: taxonomic rank for the taxon
  - Column 2: taxon
  - Column 3: accumulated read number (number of reads for that taxon in all samples)
  - Column 4 and beyond: number of reads for the taxon in the corresponding sample

- `<project>.out.funcog`: abundance of all COG functions

  Format of the file:

  - Column 1: COG ID
  - Column 2: accumulated read number: Number of reads for that COG in all samples
  - Column 3 and beyond: number of reads for the COG in the corresponding sample
  - Next to last column: COG function
  - Last column: COG functional class

- `<project>.out.funkegg`: abundance of all KEGG functions

  Format of the file:

  - Column 1: KEGG ID
  - Column 2: accumulated read number (number of reads for that KEGG in all samples)
  - Column 3 and beyond (number of reads for the KEGG in the corresponding sample)
  - Next to last column: KEGG function
  - Last column: KEGG functional class

SQM_longreads.pl

This script works in the same way as SQM_reads.pl, that is, it attempts to produce taxonomic and functional assignments directly on the raw reads, not using an assembly. The difference is that this script assumes that more than one ORF can be found in the read. It performs Diamond Blastx searches against taxonomic and functional databases, and then identifies ORFs by collapsing the hits in the same region of the read. The --range-culling option of Diamond makes this possible, since it limits the number of hits to the same region of the sequence, making it possible to recover hits for all parts of the read.

The method assigns taxa as functions to each ORF using the lca and fun3 methods, as SqueezeMeta does. In addition, it calculates the consensus of the taxonomic assignment for the read (see Explanation of SqueezeMeta algorithms). The taxon provided for the read is that consensus annotation.

The usage of SQM_longreads.pl is the same than that of SQM_reads.pl:

```
sqm_longreads.pl -p <project name> -s <equiv file> -f <raw fastq dir> [options]
```

Arguments

Mandatory parameters

- `-p`: project name (REQUIRED)
- `-s|-samples`: samples file (REQUIRED)
- `-f|-seq`: fastq read files' directory (REQUIRED)

Options

- `--nocog`: skip COG assignment
- `--nokegg`: skip KEGG assignment
- `--nodiamond`: Assumes that Diamond output files are already in place (for instance, if you are redoing the analysis) and skips all Diamond runs.
- `--euk`: Drop identity filters for eukaryotes, to improve eukaryotic annotation (Defalut: no)
- `-extdb`: List of external databases to use in the functional annotation. See 6. Using external databases for functional annotation for details.
- `-t`: number of threads (default: 12)
- `-b|-block-size`: block size for DIAMOND against the nr database. Lower values reduce RAM memory usage. Set it to 3 or below for running in a desktop computer (default: 8)
- `-e|-evalue`: max e-value for discarding hits in the DIAMOND run (default: 1e-03)
- `-i|-miniden`: identity percentage for discarding hits in DIAMOND run (default: 50)
- `-n|-nopartialhits`: ignore partial hits if they occur at the middle of a long read (default: no)

- `--force_overwrite`: Overwrite previous results

Output

The output is the same than for sqm_reads.pl, please refer to it. In addition, sqm_longreads.pl provides information about the consensus in the readconsensus.txt files in each of the samples directory.

Ignoring or not partial hits

A truncated hit (one missing to find one, or both, extremes) often happens in the extremes of the long read (because the read is ending and so is the hit), but it is unexpected to find it in the middle of a long read. There you would expect to see a complete hit. Whatever the reasons for this, the hit is suspicious and can be excluded using the -n option. But beware, this probably will decrease significantly the number of resulting ORFs.

sqm_hmm_reads.pl

This script does functional assignment of the raw reads, using an ultra-sensitive Hidden Markov Model (HMM) search implemented in the third-party software Short-Pair (https://sourceforge.net/projects/short-pair). By using an approximate Bayesian approach employing distribution of fragment lengths and alignment scores, Short-Pair can retrieve the missing end and determine true domains for short paired-end reads (Techa-Angkoon *et al.*, BMC Bioinformatics 18, 414, 2017). This is intended to give an answer to the question "Is my function of interest present in the metagenome?", avoiding assembly biases where low-abundance genes may be not assembled and therefore will not be represented in the metagenome. This is also expected to be more sensitive than Diamond assignment of reads done by SQM_reads.pl above.

As HMM searches are slower than short-read alignment, it is not practical to do this for all functions. Instead, the user must specify one or several Pfam ID and the search will be done just for these. The script will connect to Pfam database (https://pfam.xfam.org) to download the corresponding hmm and seed files. This script can be found in the `/path/to/SqueezeMeta/utils/` directory.

Usage

```
sqm_hmm_reads.pl -pfam <PFAM list> -pair1 <pair1 fasta file>
-pair2 <pair2 fasta file> [options]
```

Arguments

Mandatory parameters

- `-pfam`: list of Pfam IDs to retrieve, comma-separated (eg: -pfam PF00069,PF00070) (REQUIRED)
- `-pair1`: fasta file for pair 1 (REQUIRED)
- `-pair2`: fasta file for pair 2 (REQUIRED)
- NOTE THAT pair1 AND pair2 MUST BE UNCOMPRESSED FASTA FILES

Options

- `-t`: number of threads (default: 12)
- `-output`: name of the output file (default: SQM_pfam.out)

Output

The output file follows the `Short-Pair` output format:

- First column: read name (.1 for first pair, .2 for second pair)
- Second column: Pfam domain family
- Third column: alignment score
- Fourth column: e-value
- Fifth column: start position of alignment on the pfam domain model
- Sixth column: end position of alignment on the pfam domain model
- Seventh column: start position of alignment on the read
- Eighth column: end position of alignment on the read
- Ninth column: strand (+ for forward, - for reverse)

Utilities: Read Mapping

sqm_mapper.pl

This script maps reads to a given reference using one of the included sequence aligners (Bowtie2, BWA), and provides estimation of the abundance of the contigs and ORFs in the reference. In addition to the reads and reference files, it also needs a gff file specifying the positions of the genes in the contigs. It works in the same way than the mapping step of the main pipeline, and provides values for the coverage, TPM and RPKM of genes and contigs.

A file including functional annotations for the genes can also be given. If so, the script will provide abundance estimations for functions as well.

Usage

```
sqm_mapper.pl -r <reference> -s <sample file> -f <reads directory>
-g <gff file> -o <output directory> [options]
```

Arguments

Mandatory parameters

- `-r`: Reference sequence, the one reads will be mapped to. This can be a fasta file containing contigs, or even a single sequence coming from a complete genome (REQUIRED)
- `-s`: Samples file (refer to Section 4 in this manual for format specification. (REQUIRED)
- `-f`: Directory in which read files (the ones specified in the samples file) are located (REQUIRED)
- `-g`: GFF file specifying the genomic features in the reference. This can be downloaded for genomes, or created using a gene predictor (REQUIRED unless `--filter`). See below to know about the proper definition of this file
- `-o`: Output directory for storing results (REQUIRED)

Options

- `-t`: number of threads (default: 12)

- `-m`: Aligner to use (Bowtie, BWA) (Default: Bowtie)
- `--filter`: Use to remove reads mapping to a reference genome
- `-n|-name`: Prefix name for the results (Default: sqm)
- `-fun`: File containing functional annotations for the genes in the reference

  This is a two-column file. First column indicate the name of the gene, Second column corresponds to the function (or gene name). For instance:

  ```
  gene1	COG0735
  gene2	recA
  ```

Output

The script will produce:

- A `mappingstat` file, indicating the number of reads and percentage of alignment
- A `contigcov` file, with the abundance measures for each of the contigs in the reference
- A `mapcount` file, with the abundance measures for each ORF in the gff file corresponding to the reference.
- If a functional file was specified with the `-fun` option, it will also produce a `mapcount.fun` file, with the abundance measures for each of the functions.

The gff file (tab separated), should contain a tag "ID" in its ninth field, with the id being the contigname and, separated by "_", the initial and final positions of the gene (separated by "-"), and a final semicolon. Something like:

"ID=contig1_1-580;"

and the full line in the gff should read:

```
contig1 samplename	CDS	1	580	.	+	1	ID=contig1_1-580;
```

Utilities: Functional & Taxonomic annotation of genes and genomes

sqm_annot.pl

This script performs functional and taxonomic annotation for a set of genes or genomes. Genomes must be nucleotide sequences, while gene sequences can be either nucleotides or amino acids. All sequence files must be in fasta format.

For a genome, the script will call SqueezeMeta to predict RNAs and CDS, and then proceeds to run Diamond searches against the usual taxonomic (GenBank nr) and functional (COGs and KEGG) databases and annotate the genes according the same procedures used in the main SqueezeMeta pipeline (LCA for taxa, best hit/best average for functions. Please refer to the “Explanation of SqueezeMeta Algorithms" section for details). For gene sequences, it is assumed that each sequence corresponds to an already identified ORF, and then RNA and CDS prediction is skipped.

Diamond searches are automatically set to “blastp” for amino acids, and “blastx” for nucleotides.

The scripts needs a sample file following the format:

<sample name> <fasta file name> <genome|aa|nt>

The first field corresponds to the project name. The script will create a project directory with that name, where all results will be placed. The second field is the name of the genome, amino acid or nucleotide fasta file containing the sequences. And the third field specifies the type of data: genome, aa or nt. As explained above, genome will trigger gene prediction and run Diamond blastp on the predicted peptides, aa will run Diamond blastp for the provided sequences, and nt will run Diamond blastx for the provided sequences.

Usage

```
sqm_annot.pl -s <samples file> -f <sequence file directory>
[options]
```

Arguments

Mandatory parameters

- `-f`: Directory in which the sequence files specified in the samples file are located (REQUIRED). The sequence files MUST be in FASTA format.

- `-s`: Samples file (REQUIRED)

  Options

- `-t`: number of threads (default: 12)
- `--notax: Skips taxonomic annotation`
- `--nocog: Skips COGs annotation`
- `--nokegg: Skips KEGG annotation`
- `-extdb <database file>`: list of user-provided databases for functional annotations (see section 8 in this manual).
- `-b <block size>`: block size for DIAMOND against the nr database. Lower values reduce RAM memory usage. Set it to 3 or below for running in a desktop computer (default: 8)

Output

This scripts takes advantage of the standard SqueezeMeta machinery, therefore the output files are these obtained in the steps 6 and 7 of the pipeline:

- 06.<project>.fun3.tax.wranks: taxonomic assignments for each ORF, including taxonomic ranks
- 06.<project>.fun3.tax.noidfilter.wranks: same as above, but assignment was done not considering identity filters (refer to the explanation of the LCA algorithm in the algorithm section)

  Refer to description of step 6 for information on the format of these files

- 07.<sample>.fun3.cog: COG functional assignment for each ORF
  07.<sample>.fun3.kegg: KEGG functional assignment for each ORF

  Refer to description of step 7 for information on the format of these files

- COG.summary: Counts and functions for each COG

  KEGG.summary: Counts and functions for each KEGG

  Format of these files:

  - Column 1: COG/KEGG ID
  - Column 2: Abundance (number of assignments)
  - Column 3: Name of the gene

- Column 4: Function of the gene
- Column 5: Functional class or pathway
- Column 6: ORFs belonging to current COG/KEGG

Utilities: Estimation of sequencing depth needed

cover.pl

COVER intends to help in the experimental design of metagenomics by addressing the unavoidable question: How much should I sequence to get good results? Or the other way around: I can expend this much money, would it be worth to use it in sequencing the metagenome?

To answer these questions, COVER allows the estimation of the amount of sequencing needed to achieve a particular objective, being this the coverage attained for the most abundant N members of the microbiome. For instance, how much sequence is needed to reach 5x coverage for the four most abundant members (from now on, OTUs). COVER was first published in 2012 (Tamames et al 2012, Environ Microbiol Rep. 4:335-41), but we are using a different version of the algorithm described there.

Usage: `cover.pl -i <input file> [options]`

Arguments

Mandatory parameters

- `-i <input file>`: Fasta file containing 16S sequences (REQUIRED)

Options

- `-t`: Number of threads (Default: 4)
- `-idcluster`: Identity threshold for collapsing OTUs (Default: 0.98)
- `-c|-coverage`: Target coverage (Default: 5)
- `-r|-rank`: Rank of target OTU (Default: 4)

(Default values imply looking for 5 x coverage for the 4 th most abundant OTU)

- `-cl|-classifier`: Classifier to use (RDP or Mothur) (Default: mothur)

- `-d|-dir`: Output directory (Default: cover)
- `-h|-help`: This help

COVER needs information on the composition of the microbiome, and that must be provided as a file containing 16S rRNA sequences obtained by amplicon sequencing of the target microbiome. If you don't have that, you can look for a similar sample already sequenced (for instance, in NCBI's SRA, see below).

The first step is clustering the sequences at the desired identity level (default: 98%) to produce OTUs. COVER uses cd-hit (Schmieder et al 2011, Bioinformatics 27:863-4) for doing this. The abundance of each OTU is also obtained in this step (the number of sequences in each OTU). Then, a taxonomic annotation step must be done for inferring genomic size and 16S rRNA copy number for each of the OTUs. This annotation can be done using the RDP classifier (Wang et al 2007, Appl Environ Microbiol 73, 5261-7), or Mothur (Schloss et al, Appl Environ Microbiol, 2009. 75:7537-41) alignment against the SILVA database. The latter is the default option. It is slower but provides more accurate results.

The taxonomic annotation allows to infer the approximate genomic size by comparison with the size of already sequenced genomes from the same taxon (we've got this information from NCBI's genome database). In the same way, we inferred the expected copy number by comparison to the rrnDB database (Stoddard et al 2014, *Nucleic Acids Research* doi: 10.1093/nar/gku1201; https://rrndb.umms.med.umich.edu). Obviously, the most accurate the annotation, the most precise this estimation will be. In case that the OTU could not be annotated, COVER uses default values of 4 Mb genomic size and 1 for copy number. These values can be greatly inaccurate and affect the results. Therefore, it is strongly advised that the taxonomic annotation is as good as possible.

In the next step, COVER calculates the probability of sequencing a base for each of the OTUs. First, the abundance of each OTU is divided by its copy number:

$$Abundance_n = Raw\ abundance_n / Copy\ number_n$$

Then, all abundances are summed, and individual abundances are normalized by this total abundance.

$$Corr\ abundance_n = Abundance_n / \Sigma_n Abundance$$

The fraction of the microbiome occupied by each OTU, f, is the product of its abundance by its genomic size

$$f_n = Corr\ abundance_n * Size_n$$

and the total size of the microbiome is the sum of all individual fractions.

$$F = \Sigma_n f_n$$

Then, the probability of sequencing one base of a particular OTU is the ratio between its fraction and the total size:

$$p_n = f_n / F$$

And the amount of sequence needed (S) to attain coverage *C* for genome n is then:

$$S = C * Size_n / p_n$$

COVER calculates this value of S for the n-th OTU, as specified by the user. Then, coverages for all other OTUs are also calculated using the last equation and this value of S:

$$C_n = S * p_n / Size_n$$

In the previous calculation, we have assumed that we can calculate abundances for all members of the microbiome. Obviously this is not true, because there will be a fraction of unobserved (rare) OTUs that were not sequenced in our 16S. The size of that fraction

will depend on the completeness of our 16S sequencing, which is influenced by the diversity of the microbiome and by the sequencing depth. This unobserved fraction can bias greatly the results. Luckily, there is a way to estimate it by means of the Good's estimator of sample coverage (Chao & Shen 2003 Environ Ecol Stat 10: 429–443), that supposes that the fraction of sequence reads corresponding to unobserved OTUs is approximately equal to the fraction of observed singletons (OTUs with just one sequence):

$$U = f_1 / N_{OTUs}$$

Both f_1 and N_{OTUs} are obtained in the OTU clustering step. Then, we just need to correct the value of S by this value:

$$S_{corrected} = S / (1-U)$$

The output is a table that first lists the amount of sequencing needed, both uncorrected and corrected by the Good's estimator:

Needed 4775627706 bases, uncorrected

Correcting by unobserved: 6693800053 bases

And then lists the information and coverages for each OTU, with the following format:

OTU	Size	Raw ab	copy num		Corr ab	Pi	%Genom		Cover	Taxon
OTU1	2928853	278	1.4		0.010	0.0056	99.99	9.2		family:Nitrososphaeraceae
OTU2	8379229	224	1.1		0.010	0.0159	99.99	9.1		genus:Bradyrhizobium
OTU3	3462770	138	1.0		0.007	0.0046	99.83	6.4		order:Haplotaxida
OTU4	5142220	279	2.6		0.005	0.0054	99.33	5.0		order:Rhizobiales
OTU5	5862923	126	1.2		0.005	0.0060	99.27	4.9		phylum:Acidobacteria

The columns correspond to:

- OTU: Name of the OTU
- Size: Inferred genomic size of the OTU
- Raw abundance: Number of sequences in the OTU
- Copy number: Inferred 16S rRNA copy number
- Corrected abundance: Abundance $_n$ / Σ_n Abundance
- P_i : Probability of sequencing a base of this OTU
- %Genome sequenced: Percentage of the genome that will be sequenced for that OTU
- Coverage: Coverage that will be obtained for that OTU
- Taxon: Deepest taxonomic annotation for the OTU

We are aware that often you will not have a 16S amplicon sequencing of the microbiome. In that case, you can use the fabulous collection of 16S sampling stored in the NCBI's SRA archive (https://www.ncbi.nlm.nih.gov/sra). You can, for instance, locate BioProjects involving 16S sequencing (https://www.ncbi.nlm.nih.gov/bioproject/?term=16S), then restrict to "environmental", and look for similar microbiomes. The sequencing estimate that you will get in this way would be just approximate, but you could find it useful to have an initial idea on the range of sequencing that you need.

Utilities: integration with iTOL

sqm2itol.pl

This script generates the files for creating a radial plot of abundances using iTOL (https://itol.embl.de/), such as the ones below, taken from the SqueezeMeta paper. This script can be found in the `/path/to/SqueezeMeta/utils/` directory.

Figure 1: Taxonomic plot. Abundance of bins in the diverse samples. Bins were compared with the CompareM software (https://github.com/dparks1134/CompareM) to estimate their reciprocal similarities. The distances calculated between the bins were used to create a phylogenetic tree illustrating their relationships. The tree is shown in the inner part of the Figure. Branches in the tree corresponding to the four more abundant phyla in the tree (*Firmicutes*, *Bacteriodetes*, *Proteobacteria* and *Spirochaetes*) were colored. Bins were named with their ID number and original genera, and labels for the most abundant genera were also colored. Outer circles correspond to: the completeness of the bins (green-colored, most internal circle), and the abundance of each bin in each sample (red-colored). Each circle corresponds to a different sample, and the red color intensities correspond to the bin's abundance in the sample.

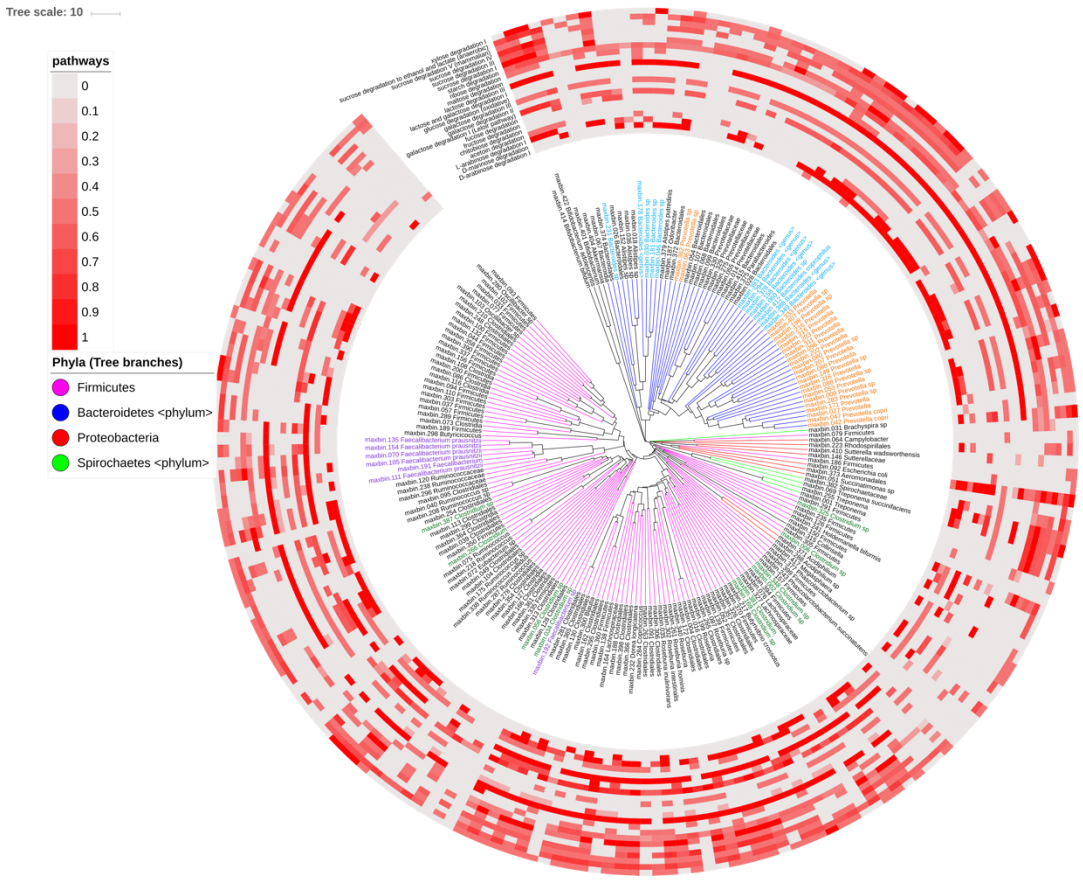


Figure 2: Functional plot. Presence of several carbohydrate degradation pathways in bins.

The outer circles indicate the percentage of genes from a pathway present in each of the bins. According to that gene profile, MinPath estimates whether or not the pathway is present. Only pathways inferred to be present are colored. As in Figure 1, the bins tree is performed from a distance matrix of the orthologous genes' amino acid identity, using the compareM software. The four most abundant phyla are colored (branches in the tree), as well as the most abundant genera (bin labels). The picture was elaborated using the iTOL software.

Usage:

```
sqm2itol.pl <options> project name
```

Arguments

Mandatory parameters

- project name. Directory containing a valid SqueezeMeta analysis

Options

- `-completion` [percentage]: select only bins with completion above that threshold (default: 30)
- `-contamination` [percentage]: select only bins with contamination below that threshold (default: 100)
- `-classification` [metacyc|kegg]: functional classification to use (default: metacyc)
- `-functions` [file]: file containing the name of the functions to be considered (for functional plots). For example:

```
arabinose degradation
galactose degradation
glucose degradation
```

Output

The program will generate several datafiles that you must upload to iTOL to produce the figure.

Utilities: integration with iPath

sqm2ipath.pl

This script creates data on the existence of enzymatic reactions that can be plotted in the interactive pathway mapper iPath (http://pathways.embl.de)

Usage:

```
sqm2ipath.pl <options> project name
```

Arguments

Mandatory parameters

- project name. Directory containing a valid SqueezeMeta analysis

Options

- `-taxon: Selects the taxon to be plotted` (default: all)
- `-color: RGB color to be used in the plot (default: red)`
- `-c|classification` [cog|kegg]: functional classification to use (default: kegg)

- `-functions` [file]: file containing the COG/KEGG identifiers of the functions to be considered. For example:

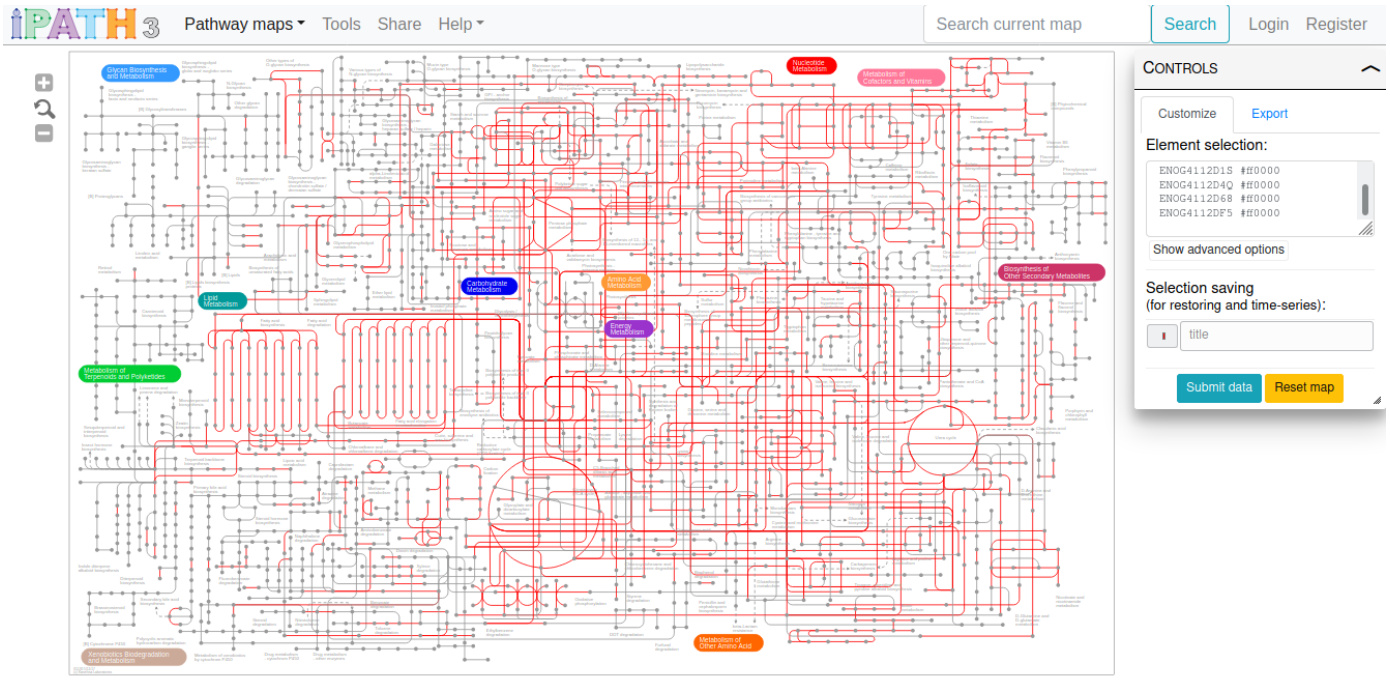
```
K00036
K00038
K00040
K00052  #ff0000
K00053
```

  A second argument following the identifier selects the RGB color to be associated to that ID in the plot

- `-o|out`: Name of the output file (default: ipath.out)

The plotting colors can be specified by the -color option, or by associating values to each of the IDs in the functions file. In that case, several colors can be used in the same plot. If no color is specified, default is red.

The results in the output file must be copied in the iPath interface. Several output files can be combined, for instance using different colors for different taxa.



Utilities: integration with pavian

sqm2pavian.pl

This script produces output files containing abundance of taxa that can be plotted using the Pavian tool (https://github.com/fbreitwieser/pavian).It translates the results in the SqueezeMeta mapcount file to the kraken report format wanted by Pavian. This script can be found in the `/path/to/SqueezeMeta/utils/` directory.

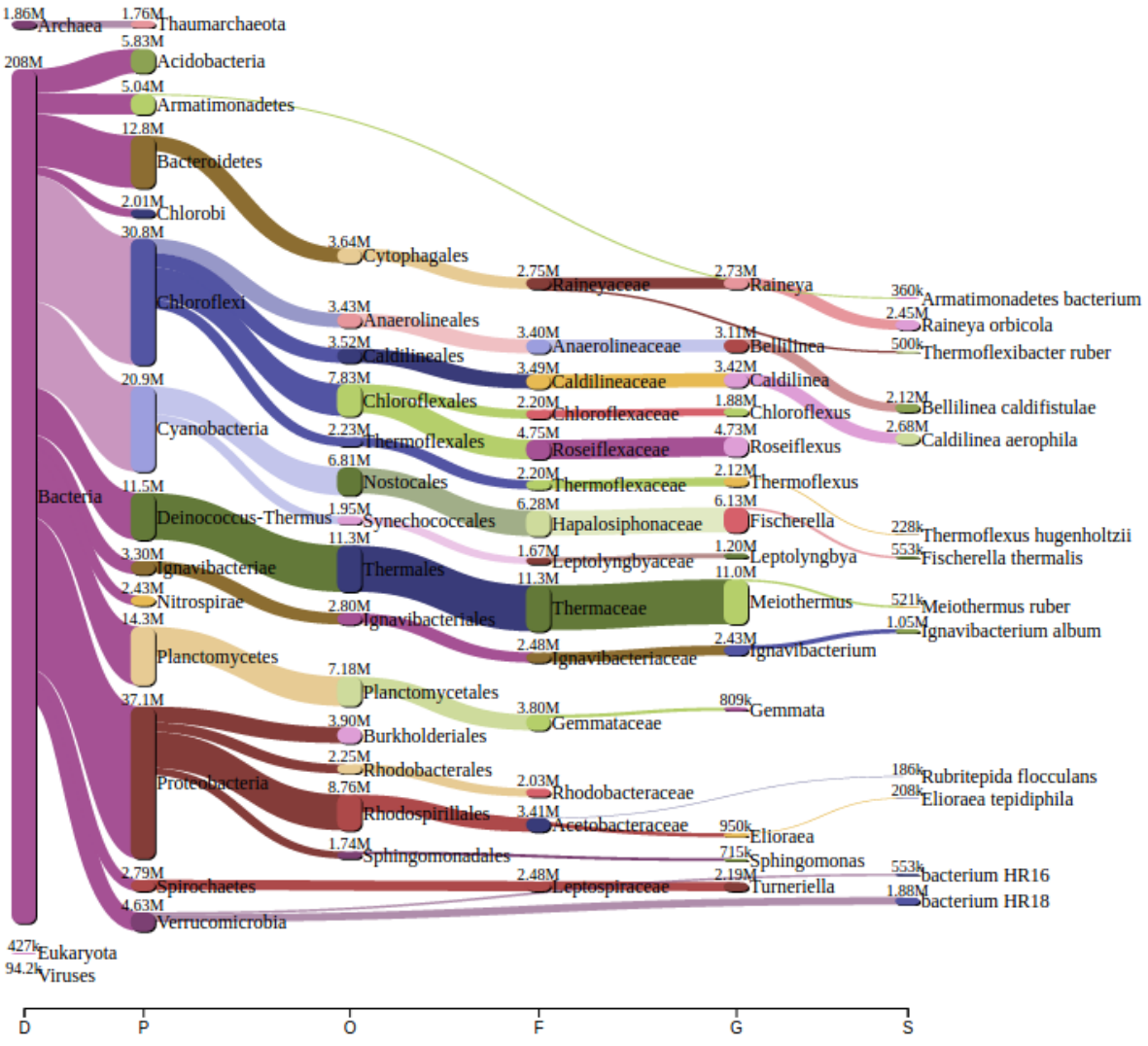


Figure 3: SqueezeMeta analysis of a thermal microbial mat and plotted by Pavian. Numbers correspond to the amount of reads belonging to each of the phylogenetic nodes.

This script produces output files containing abundance of taxa that can be plotted using the Pavian tool (https://github.com/fbreitwieser/pavian).It translates the results in the SqueezeMeta mapcount file to the kraken report format wanted by Pavian.

This also works with the output of sqm_reads.pl and sqm_longreads.pl

Usage

```
sqm2pavian.pl <project name> [reads|bases]
```

The abundances can be counted either in reads or in bases, as specified when calling the script. By default, reads are used. The script will produce a file named `<project>.pavian` that can be uploaded in the pavian app (https://fbreitwieser.shinyapps.io/pavian) or in the pavian R package.

Utilities: adding new databases to an existing project

add_database.pl

This script can be found in the `/path/to/SqueezeMeta/utils/` directory. It adds one or several new databases to the results of an existing project. The list of databases must be provided in an external database file as specified in section 6. It must be a tab-delimited file with the following format:

```
<Database Name> <Path to database>    <Functional annotation file>
```

The databases to add must also be formatted in Diamond format. See section 6 for details. If the external database file already exists (because you already used some external databases when running SqueezeMeta), DO NOT create a new one. Instead add the new entries to the existing database file.

Usage

```
add_database.pl <project name> <database file>
```

The script will run Diamond searches for the new databases, and then will re-run several SqueezeMeta scripts to include the new database(s) to the existing results. The following scripts will be invoked:

07.fun3assign.pl: For functional assignment to the new database(s)

12.funcover.pl: For estimating abundances of the new functions

13.mergeannot2.pl. For redoing the ORF table, including the new annotations

21.stats.pl: For including the number of hits to new database(s) in the final stats.

Outpust of these programs will be regenerated (but all files corresponding to other databases will remain untouched).

Utilities: integration with R and other analysis environments

The sqm2tables.py script will produce tabular outputs suitable for analysis in environments such as R, matlab or python/pandas.

For convenience, we also provide the SQMtools R package. This package provides an easy way to expose the different results of SqueezeMeta (orfs, contigs, bins, taxonomy, functions…) into R, as well as a set of utility functions to filter and display SqueezeMeta results.

sqm2zip.py

This script generates a compressed zip file with all the essential information needed to load a project into SQMtools. If the directory `project_path/results/tables` is not present, it will also run `sqm2tables.py` to generate the required tables (see below).

Usage

```
sqm2tables.py [options] <project_path> <output_dir>
```

Mandatory parameters

- `project_path`: path to the SqueezeMeta run
- `output_dir`: output directory

Options

- `--trusted-functions`: include only ORFs with highly trusted KEGG and COG assignments in aggregated functional tables. This will be ignored if the `project_path/results/tables` directory already exists
- `--ignore-unclassified`: ignore reads with no functional classification when aggregating abundances for functional categories (KO, COG, PFAM). This will be ignored if the `project_path/results/tables` directory already exists
- `--force-overwrite`: write results even if the output file already exists
- `--doc`: print the documentation

sqm2tables.py

This script generates tabular outputs from a SqueezeMeta run. It will aggregate the abundances of the ORFs assigned to the same feature (be it a given taxon or a given function) and produce tables with features in rows and samples in columns. This script can be found in the `/path/to/SqueezeMeta/utils/` directory. Note that if you want to create tables coming from a `sqm_reads.pl` or `sqm_longreads.pl` run you will need to use the `sqmreads2tables.py` script.

Usage

```
sqm2tables.py [options] <project_path> <output_dir>
```

Arguments

Mandatory parameters

- `project_path`: path to the SqueezeMeta run
- `output_dir`: output directory

Options

- `--trusted-functions`: include only ORFs with highly trusted KEGG and COG assignments in aggregated functional tables
- `--ignore-unclassified`: ignore reads with no functional classification when aggregating abundances for functional categories (KO, COG, PFAM)
- `--force-overwrite`: write results even if the output directory already exists
- `--doc`: print the documentation

Output

- For each functional classification system (KO, COG, PFAM, and any external database provided by the user) the script will produce the following files:
  - `<project_name>.<classification>.abunds.tsv`: raw read counts of each functional category in the different samples
  - `<project_name>.<classification>.bases.tsv`: raw base counts of each functional category in the different samples
  - `<project_name>.<classification>.tpm.tsv`: normalized (TPM) abundances of each functional category in the different samples. This normalization takes into account both sequencing depth and gene length
    - The `--ignore_unclassified` flag can be used to control whether unclassified ORFs are counted towards the total for normalization
  - `<project_name>.<classification>.copyNumber.tsv`: average copy numbers per genome of each functional category in the different samples. Copy numbers are obtained by dividing the aggregate coverage of each function in each sample by the coverage of RecA (COG0468) in each sample.
  - `<project_name>.<classification>.names.tsv`: extended description of the functional categories in that classification system. For KO and COG the file will contain three fields: ID, Name and Path within the functional hierarchy. For external databases, it will contain only ID and Name.
- `<project_name>.RecA.tsv`: coverage of RecA (COG0468) in the different samples.
- `<project_name>.orf.tax.allfilter.tsv`: taxonomy of each ORF at the different taxonomic levels. Minimum identity filters for taxonomic assignment are applied to all taxa.
- `<project_name>.orf.tax.prokfilter.tsv`: taxonomy of each ORF at the different taxonomic levels. Minimum identity filters for taxonomic assignment are applied to bacteria and archaea, but not to eukaryotes.

- `<project_name>.contig.tax.allfilter.tsv`: consensus taxonomy of each contig at the different taxonomic levels, based on the taxonomy of their constituent ORFs (applying minimum identity filters).
- `<project_name>.contig.tax.prokfilter.tsv`: consensus taxonomy of each contig at the different taxonomic levels, based on the taxonomy of their constituent ORFs. Minimum identity filters for taxonomic assignment are applied to bacteria and archaea, but not to eukaryotes).
- `<project_name>.bin.tax.tsv`: consensus taxonomy of each bin at the different taxonomic levels, based on the taxonomy of their constituent contigs.
- `<project_name>.orfs.sequences.tsv`: ORF sequences.
- `<project_name>.orfs.sequences.tsv`: contig sequences.
- For each taxonomic rank (superkingdom, phylum, class, order, family, genus, species) the script will produce the following files:
  - `<project_name>.<rank>.allfilter.abund.tsv`: raw abundances of each taxon for that taxonomic rank in the different samples, applying the identity filters for taxonomic assignment (see explanation of the LCA algorithm below).
  - `<project_name>.<rank>.prokfilter.abund.tsv`: raw abundances of each taxon for that taxonomic rank in the different samples. Identity filters for taxonomic assignment are applied to prokaryotic (bacteria + archaea) ORFs but not to Eukaryotes (see below).

Details

- By default, SqueezeMeta applies Luo *et al.* (2014) identity cutoffs in order to assign an ORF to a given taxonomic rank (see explanation of the LCA algorithm). In our tests, these cutoffs resulted in a very low percentage of annotation for eukaryotic ORFs. To circumvent this issue, the `*.prokfilter.*` files generated by this script contain the aggregated taxonomic abundances obtained by applying Luo's filter only to Bacteria and Archaea, but not to Eukaryotes.
- SqueezeMeta uses NCBI's *nr* database for taxonomic annotation, and reports the superkingdom, phylum, class, order, family, genus and species ranks. In some cases, the NCBI taxonomy is missing some intermediate ranks. For example, the NCBI taxonomy for the order *Trichomonadida* is:
  - superkingdom: `Eukaryota`
  - no rank: `Parabasalia`
  - order: `Trichomonadida`

NCBI does not assign *Trichomonadida* to any taxa in the class and phylum ranks. For clarity, the `sqm2tables.py` will indicate this by recycling the highest available taxonomy and adding the "`(no <rank> in NCBI)`" string after it. For example, ORFs that can be classified down to the *Trichomonadida* order (but are unclassified at the family level) will be reported as:

  - superkingdom: `Eukaryota`
  - phylum: `Trichomonadida (no phylum in NCBI)`
  - class: `Trichomonadida (no class in NCBI)`
  - order: `Trichomonadida`
  - family: `Unclassified Trichomonadida`
  - genus: `Unclassified Trichomonadida`
  - species: `Unclassified Trichomonadida`

- Some ORFs will have multiple KEGG/COG annotations in the `13.*.orftable` file. This is due to their best hit in the KEGG/COG databases actually being annotated with more than one function. The script will split the abundances of those ORFs between the different functions they have been assigned to.
- The "Unclassified" category represents features that were classifiable with our method (i.e. contained a protein-coding sequence). In addition to the normal taxon names and the "Unclassified" category, the results will contain 2 extra categories.
  - "Unmapped": reads not mapping to any contigs.
  - "No CDS": features (or reads mapping to features) that contained no protein-coding sequences (e.g. rRNAs).

sqmreads2tables.py

This script generates tabular outputs from a `sqm_reads.pl` or `sqm_longreads.pl` run. It will aggregate the abundances of the ORFs assigned to the same feature (be it a given taxon or a given function) and produce tables with features in rows and samples in columns. It can optionally accept a query argument to generate tables containing only certain taxa and functions. This script can be found in the `/path/to/SqueezeMeta/utils/` directory.

Usage

```
sqmreads2tables.py [options] <project_path> <output_dir>
```

Arguments

Mandatory parameters

- `project_path`: path to the SqueezeMeta run
- `output_dir`: output directory

Options

- `--trusted-functions`: include only ORFs with highly trusted KEGG and COG assignments in aggregated functional tables
- `--force-overwrite`: write results even if the output directory already exists
- `-q/—query`: Filter the results based on the provided query in order to create tables containing only certain taxa or functions. Query syntax is the same as in the `anvi-filter-sqm.py` script.
- `--doc`: print the documentation

Output

- For each functional classification system (KO, COG, PFAM, and any external database provided by the user) the script will produce the following files:
  - `<project_name>.<classification>.abunds.tsv`: raw abundances of each functional category in the different samples
  - `<project_name>.<classification>.names.tsv`: extended description of the functions in that classification system. For KO and COG the file will contain three fields: ID, Name and Path within the functional hierarchy. For external databases, it will contain only ID and Name.
- For each taxonomic rank (superkingdom, phylum, class, order, family, genus, species) the script will produce the following files:
  - `<project_name>.<rank>.allfilter.abund.tsv`: raw abundances of each taxon for that taxonomic rank in the different samples, applying the identity filters for taxonomic assignment (see explanation of the LCA algorithm below).
  - `<project_name>.<rank>.prokfilter.abund.tsv`: raw abundances of each taxon for that taxonomic rank in the different samples. Identity filters for taxonomic assignment are applied to prokaryotic (bacteria + archaea) ORFs but not to Eukaryotes (see sqm2tables.py for details).

sqm2stamp.pl

This script generates tables for the statistical package STAMP (https://beikolab.cs.dal.ca/software/STAMP) from the results a `sqm_reads.pl` or `sqm_longreads.pl` project. This script can be found in the `/path/to/SqueezeMeta/utils/` directory.

Usage

```
sqm2stamp.pl <project_path>
```

combine-sqm-tables.py

Combine tabular outputs (as generated by sqm2tables.py or sqmreads2tables.py) from different SqueezeMeta or SQM_reads projects. This script can combine the results of different samples run using the sequential mode, in which each sample is run separately, and also the results of different coassembly or merged SqueezeMeta runs. *NOTE: since SqueezeMeta version 1.3, this is deprecated in favor of the* `combineSQMlite` *function from the SQMtools package.*

Usage

```
combine-sqm-tables.py [options] <project_paths>
```

Arguments

Positional arguments

- `project_paths:` a space-separated list of paths

Options

- `-f/--paths-file:` file containing the paths of the SqueezeMeta projects to combine, one path per line
- `-o/--output-dir:` name of the output directory (default: “combined”)
- `-p/--output-prefix:` Prefix for the output files (default: “combined”)
- `--trusted-functions`: include only ORFs with highly trusted KEGG and COG assignments in aggregated functional tables
- `--ignore-unclassified`: ignore ORFs with no functional classification when aggregating abundances for functional categories (KO, COG, PFAM)
- `--sqm-reads:` projects were generated using sqm_reads.pl
- `--force-overwrite`: write results even if the output directory already exists
- `--doc`: print the documentation

Examples

- Combine projects `/path/to/proj1` and `/path/to/proj2` and store output in a directory named `outputDir`

    - `combine-sqm-tables.py /path/to/proj1 /path/to/proj2 -o output_dir`
- Combine a list of projects contained in a file, use default output dir
    - `combine-sqm-tables.py -f project_list.txt`

Output

- Tables containing aggregated counts and feature names for the different functional hierarchies and taxonomic levels for each sample contained in the different projects that were combined. Tables with the TPM and copy number of functions will also be generated for SqueezeMeta runs, but not for SQM reads runs.

The SQMtools R package

This package provides an easy way to expose the different results of SqueezeMeta (orfs, contigs, bins, taxonomy, functions…) into R, as well as a set of utility functions to filter and display SqueezeMeta results.

Once SqueezeMeta has finished running, just go into R and load the project.

```
library(SQMtools)
project = loadSQM("<project_directory>")
```

This will also work with a zip file produced by `sqm2zip.py`.

```
project = loadSQM("<project.zip>")
```

The resulting SQM object contains all the relevant information, distributed in an R list (see Figure 5). For example, a matrix with the taxonomic composition of the different samples at the phylum level in percentages can be obtained with

```
project$taxa$phylum$percent
```

while a matrix with the average copy number per genome of the different PFAMs across samples can be obtained with

```
project$functions$PFAM$copy_number
```

The `SQMtools` package also provides functions for selecting subsets of your data and plotting/exporting results. The basic workflow is illustrated in Figure 6. For example, we can make a plot with the taxonomic distribution of all the genes related to vitamin metabolism.

```
vit = subsetFun(project, "Metabolism of cofactors and vitamins")
```

```
plotTaxonomy(vit)
```

As an alternative to running a full SqueezeMeta project, you can just load the taxonomic and functional aggregate tables. This will work with the output of sqm2tables.py, sqmreads2tables.py and combine-sqm-tables.py, so you can analyze the ouput of sqm_reads.pl, or the combined results of several SqueezeMeta or sqm_reads projects. To do so, you can use the `loadSQMlite` function from SQMtools.

```
project = loadSQMlite("<tables_directory>")
```

	lvl1	lvl2	lvl3	type	rows/names	columns	data
SQM	$orfs	$table		dataframe	orfs	misc. data	misc. data
		$abund		numeric matrix	orfs	samples	abundances (reads)
		$bases		numeric matrix	orfs	samples	abundances (bases)
		$cov		numeric matrix	orfs	samples	coverages
		$cpm		numeric matrix	orfs	samples	coverages per million of reads
		$tpm		numeric matrix	orfs	samples	tpm
		$seqs		character vector	orfs	(n/a)	sequences
		$tax		character matrix	orfs	tax. ranks	taxonomy
	$contigs	$table		dataframe	contigs	misc. data	misc. data
		$abund		numeric matrix	contigs	samples	abundances (reads)
		$bases		numeric matrix	contigs	samples	abundances (bases)
		$cov		numeric matrix	contigs	samples	coverages
		$cpm		numeric matrix	contigs	samples	coverages per million of reads
		$tpm		numeric matrix	contigs	samples	tpm
		$seqs		character vector	contigs	(n/a)	sequences
		$tax		character matrix	contigs	tax. ranks	taxonomies
		$bins		character matrix	contigs	bin. methods	bins
	$bins	$table		dataframe	bins	misc. data	misc. data
		$length		numeric vector	bins	(n/a)	length
		$abund		numeric matrix	bins	samples	abundances (reads)
		$percent		numeric matrix	bins	samples	percentages (reads)
		$bases		numeric matrix	bins	samples	abundances (bases)
		$cov		numeric matrix	bins	samples	coverages
		$cpm		numeric matrix	bins	samples	coverages per million of reads
		$tax		character matrix	bins	tax. ranks	taxonomy
	$taxa	$superkingdom	$abund	numeric matrix	superkingdoms	samples	abundances (reads)
			$percent	numeric matrix	superkingdoms	samples	percentages (reads)
		$phylum	$abund	numeric matrix	phyla	samples	abundances (reads)
			$percent	numeric matrix	phyla	samples	percentages (reads)
		$class	$abund	numeric matrix	classes	samples	abundances (reads)
			$percent	numeric matrix	classes	samples	percentages (reads)
		$order	$abund	numeric matrix	orders	samples	abundances (reads)
			$percent	numeric matrix	orders	samples	percentages (reads)
		$family	$abund	numeric matrix	families	samples	abundances (reads)
			$percent	numeric matrix	families	samples	percentages (reads)
		$genus	$abund	numeric matrix	genera	samples	abundances (reads)
			$percent	numeric matrix	genera	samples	percentages (reads)
		$species	$abund	numeric matrix	species	samples	abundances (reads)
			$percent	numeric matrix	species	samples	percentages (reads)
	$functions	$KEGG	$abund	numeric matrix	KEGG ids	samples	abundances (reads)
			$bases	numeric matrix	KEGG ids	samples	abundances (bases)
			$cov	numeric matrix	KEGG ids	samples	coverages
			$cpm	numeric matrix	KEGG ids	samples	coverages per million of reads
			$tpm	numeric matrix	KEGG ids	samples	tpm
			$copy_number	numeric matrix	KEGG ids	samples	avg. copies
		$COG	$abund	numeric matrix	COG ids	samples	abundances (reads)
			$bases	numeric matrix	COG ids	samples	abundances (bases)
			$cov	numeric matrix	COG ids	samples	coverages
			$cpm	numeric matrix	COG ids	samples	coverages per million of reads
			$tpm	numeric matrix	COG ids	samples	tpm
			$copy_number	numeric matrix	COG ids	samples	avg. copies
		$PFAM	$abund	numeric matrix	PFAM ids	samples	abundances (reads)
			$bases	numeric matrix	PFAM ids	samples	abundances (bases)
			$cov	numeric matrix	PFAM ids	samples	coverages
			$cpm	numeric matrix	PFAM ids	samples	coverages per million of reads
			$tpm	numeric matrix	PFAM ids	samples	tpm
			$copy_number	numeric matrix	PFAM ids	samples	avg. copies
	$total_reads			numeric vector	samples	(n/a)	total reads
	$misc	$project_name		character vector	(empty)	(n/a)	project name
		$samples		character vector	(empty)	(n/a)	samples
		$tax_names_long	$superkingdom	character vector	short taxa names	(n/a)	long taxa names
			$phylum	character vector	short taxa names	(n/a)	long taxa names
			$class	character vector	short taxa names	(n/a)	long taxa names
			$order	character vector	short taxa names	(n/a)	long taxa names
			$family	character vector	short taxa names	(n/a)	long taxa names
			$genus	character vector	short taxa names	(n/a)	long taxa names
			$species	character vector	short taxa names	(n/a)	long taxa names
		$tax_names_short		character vector	long taxa names	(n/a)	short taxa names
		$KEGG_names		character vector	KEGG ids	(n/a)	KEGG names
		$KEGG_paths		character vector	KEGG ids	(n/a)	KEGG hierarchy
		$COG_names		character vector	COG ids	(n/a)	COG names
		$COG_paths		character vector	COG ids	(n/a)	COG hierarchy
		$ext_annot_sources		character vector	(empty)	(n/a)	External databases

Figure 5: Structure of the SQM R object. If external databases for functional classification were provided to SqueezeMeta via the `-extdb` argument, the corresponding abundance (reads and bases), tpm and copy number profiles will be present in `SQM$functions` (e.g. results for the CAZy database would be present in `SQM$functions$CAZy`. Additionally, the extended names of the features present in the external database will be present in `SQM$misc` (e.g. `SQM$misc$CAZy_names`). The SQMlite object will have a similar structure, but will lack the `SQM$orfs`, `SQM$contigs` and `SQM$bins` section. Additionally, if the

results come from a `sqm_reads.pl` run, the SQMlite object will also be missing TPM, bases and copy numbers for the different functional classification methods.

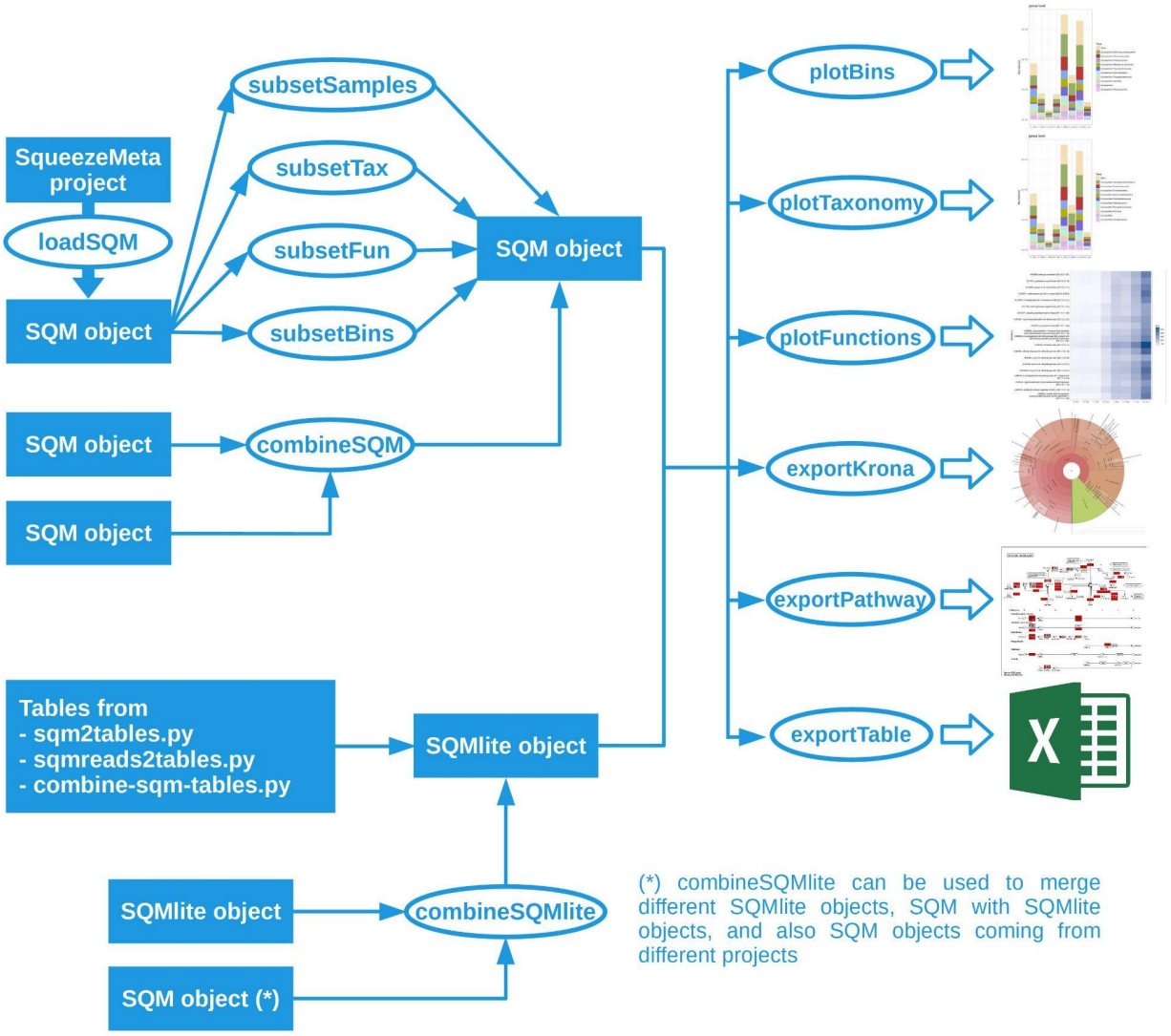


Figure 6: Basic workflow of the SQMtools package. The basic unit used in the package is the SQM object. This object can contain a full SqueezeMeta project or a subset of genes, contigs or bins. The data in the SQM object can be accessed directly (e.g. for using it with other R packages such as *vegan* for ordination analyses or *DESeq2* for differential abundance analysis) but we also provide some utility functions for exploring the most abundant functions or taxa in a SQM object. Alternatively, aggregate tables can be loaded into a SQMlite objects, which supports plot and export functionality. SQMlite objects can not be subsetted, but can be combined.

Utilities: integration with Anvi'o

The following scripts allow a seamless integration between the SqueezeMeta pipeline and the anvi'o visualization tool (http://merenlab.org/software/anvio). Anvi'o is a powerful analysis tool, but due to its nature visual analysis is better suited for relatively small datasets. The `anvi-filter-SQM.py` script allows to easily select small subsets of your metagenome (e.g. a particular taxon or a particular function or functional

category) using a simple yet powerful query language, and to visualize them using anvi'o.

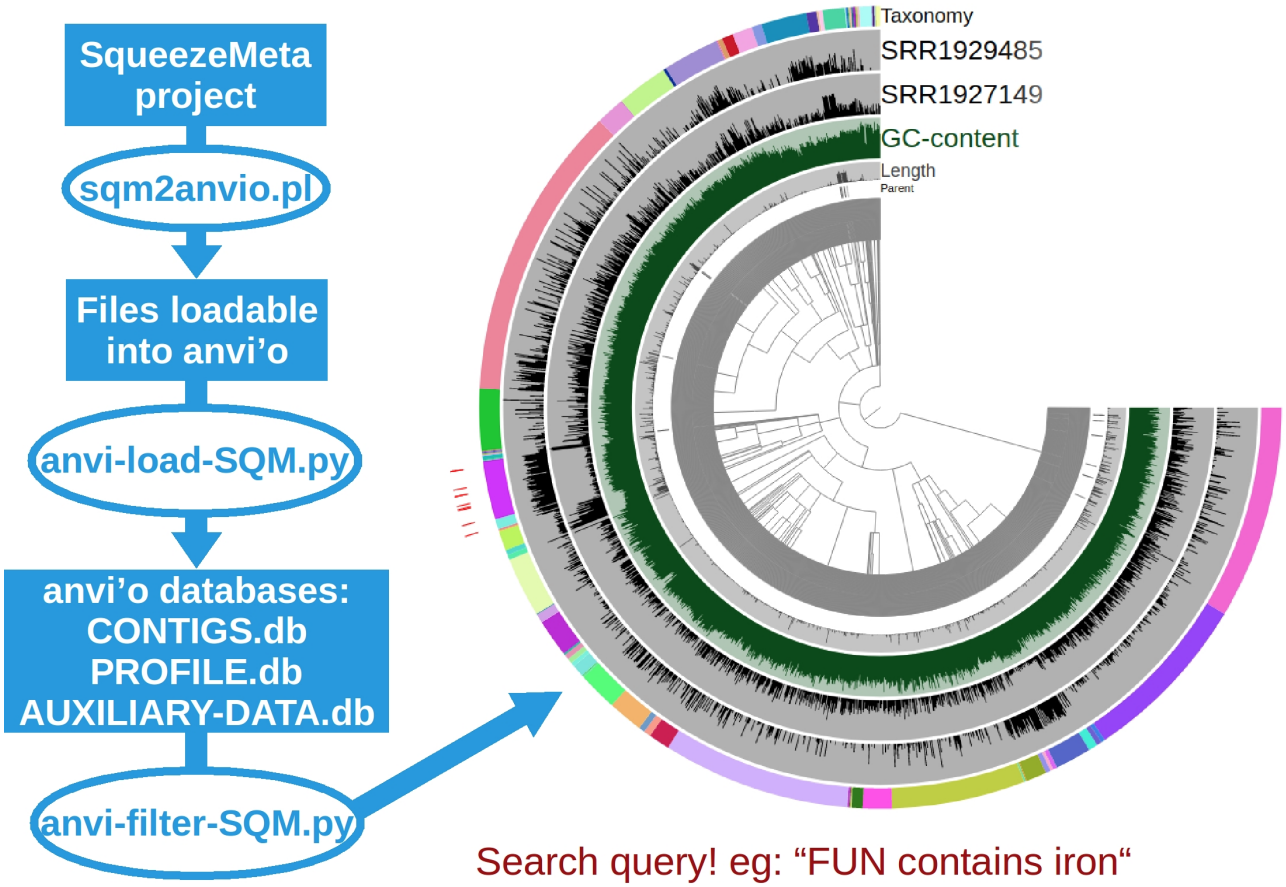


Figure 4: Analysis of the Hadza hunter-gatherer test dataset included with SqueezeMeta. After running the test analysis as described in Section 7, the `sqm2anvio.pl` and `anvi-load-SQM.py` scripts were applied sequentially in order to generate an anvi'o database. Finally, the `anvi-filter-SQM.py` was used to select all the contigs containing genes related to iron metabolism and visualize them in the anvi'o platform. This platform allow further visual exploration of the results: in this example genes related to the enterobactin iron transporter were searched for and marked (red ticks at the left, outside the taxonomy circle) using the anvi'o interface. We can see that the use of enterobactin for iron acquisition occurs on sample SRR1929485, but not in sample SRR1927149 (abundance bars below the taxonomy circle).

sqm2anvio.pl

This script generates the files required for loading SqueezeMeta results into anvi'o. It can be found in the `/path/to/SqueezeMeta/utils/anvio_utils` directory. The direct use of this script has been deprecated in v1.1.0. Instead, `anvi-load-sqm.py` will make an anvi'o database from a SqueezeMeta project in a single step.

Usage

```
sqm2anvio.pl <project name> <output directory> <anvio version>
```

Output

The script will produce a directory named "`output directory`" with all the required files.

Notes

Currently we support anvio versions 6 and 7. Support is only for released versions, the master/develop branches of anvi'o might (and will likely) not work.

anvi-load-sqm.py

This script creates an anvi'o database from a SqueezeMeta project. The database can then be filtered and visually explored using the `anvi-filter-sqm.py` script. This script can be found in the `/path/to/ SqueezeMeta/utils/anvio_utils` directory. For this script to work, anvi'o must be installed and present in your `PATH`.

Usage

```
anvi-load-sqm.py -p <project> -o <output> [options]
```

Arguments

Mandatory parameters

- `-p/--project`: project name
- `-o/--output`: output directory

Options

- `--num-threads`: number of threads (default: 12)
- `--run-HMMS`: run the `anvi-run-hmms` command from anvi'o for identifying single-copy core genes
- `--run-scg-taxonomy`: run the `anvi-run-scg-taxonomy` command from anvi'o for assigning taxonomy based on single-copy core genes
- `--min-contigs-length`: minimum length of contigs (default: 0)

- `--min-mean-coverage`: minimum mean coverage for contigs (default: 0)
- `--skip-SNV-profiling`: skip the profiling of single nucleotide variants
- `--profile-SCVs`: perform characterization of codon frequencies in genes
- `--force-overwrite`: force overwrite if the output directory already exists
- `--doc`: print the documentation

Output

- `CONTIGS.db, PROFILE.db, AUXILIARY-DATA.db`: anvi'o databases
- `<project_name>_anvio_contig_taxonomy.txt`: contig taxonomy to be loaded by `anvi.filter-sqm.py`

anvi-filter-sqm.py

This script filters the results of a SqueezeMeta project (previously loaded into to an anvi'o database by the `sqm2anvio.pl` and `anvi-load-sqm.py` scripts) and opens an anvi'o interactive interface to examine them. Filtering criteria can be specified by using a simple query syntax. This script can be found in the `/path/to/SqueezeMeta/utils/anvio_utils` directory. For this script to work, anvi'o must be installed and present in your `PATH`.

```
anvi-filter-sqm.py -p <profile db> -c <contigs db> -t <contigs taxonomy file> -q <query> [options]
```

Mandatory parameters

- `-p/--profile-db`: anvi'o profile db, as generated by `anvi-load-sqm.py`
- `-c/--contigs-db`: anvi'o contigs db, as generated by `anvi-load-sqm.py`
- `-t/--taxonomy`: contigs taxonomy, as generated by `anvi-load-sqm.py`
- `-q/--query`: query

Options

- `-o/--output_dir`: output directory for the filtered anvi'o databases (default: `"filteredDB"`)

- `-m/--max-splits`: maximum number of splits to be loaded into anvi'o. If the provided query returns a higher number of splits, the program will stop. By default it is set to 25,000, larger values may make the anvi'o interface to respond slowly. Setting `--max-splits` to 0 will allow an arbitrarily large number of splits to be loaded
- `--enforce-clustering`: make anvi'o perform an additional clustering based on abundances across samples and sequence composition
- `--extra-anvio-args`: extra arguments for `anvi-interactive`, surrounded by quotes.. e.g. `--extra-anvio-args "--taxonomic-level t_phylum --title Parrot"`
- By default, the script uses an in-house method to subset the anvi'o databases. It's ~5x quicker than using anvi-split in anvio5, and works well for us. However, the night is dark and full of bugs, so if you feel that your anvi'o view is missing some information, you can call the script with `"-s safe"` parameter. This will call `anvi-split` which should be much safer than our hacky solution.
- `--doc` will print the documentation

Query syntax

- Please enclose query strings within double brackets.
- Queries are combinations of relational operations in the form of `<SUBJECT> <OPERATOR> <VALUE>` (e.g. `"PHYLUM == Bacteroidetes"`) joined by logical operators (`AND`, `OR`).
- Parentheses can be used to group operations together.
- The `"AND"` and `"OR"` logical operators can't appear together in the same expression. Parentheses must be used to separate them into different expressions. e.g:
    - `"GENUS == Escherichia OR GENUS == Prevotella AND FUN CONTAINS iron"` would not be valid. Parentheses must be used to write either of the following expressions:
    - `"(GENUS == Escherichia OR GENUS == Prevotella)" AND FUN CONTAINS iron"`
        - splits from either *Escherichia* or *Prevotella* which contain ORFs related to iron.
    - `"GENUS == Escherichia OR (GENUS == Prevotella AND FUN CONTAINS iron)"`

    - all splits from *Escherichia*, and splits of *Prevotella* which contains ORFs related to iron.

- Another example query would be:
    - `"(PHYLUM == Bacteroidetes OR CLASS IN [Alphaproteobacteria, Gammaproteobacteria]) AND FUN CONTAINS iron AND Sample1 > 1"`
    - This would select all the anvi'o splits assigned to either the *Bacteroidetes* phylum or the *Alphaproteobacteria* or *Gammaproteobacteria* classes, that also contain the substring `"iron"` in the functional annotations of any of their ORFs, and whose anvi'o abundance (mean coverage of a split divided by the overall sample mean coverage) in `Sample1` is higher than 1.
- Possible subjects are:
    - `FUN`: search within all the combined databases used for functional annotation.
    - `FUNH`: search within the KEGG BRITE and COG functional hierarchies (e.g. `"FUNH CONTAINS Carbohydrate metabolism"` will select all the splits containing a gene associated with the broad `"Carbohydrate metabolism"` category)
    - `SUPERKINGDOM`, `PHYLUM`, `CLASS`, `ORDER`, `FAMILY`, `GENUS`, `SPECIES`: search within the taxonomic annotation at the requested taxonomic rank.
    - `<SAMPLE_NAME>`: search within the abundances in the sample named `<SAMPLE_NAME>` (e.g. if you have two samples named `"Sample1"` and `"Sample2"`, the query string `"Sample1 > 0.5 AND Sample2 > 0.5"` would return the anvi'o splits with an anvi'o abundance higher than 0.5 in both samples)
- Posible relational operators are `"=="`, `"!="`, `">="`, `"<="`, `">"`, `"<"`, `"IN"`, `"NOT IN"`, `"CONTAINS"`, `"DOES NOT CONTAIN"`

Utilities: binning refinement

Directory utils contains some tools for binning refinement, intended to work on the binning results provided by SqueezeMeta. They rely on checkM analysis to add or remove contigs from the bins.

remove_duplicate_markers.pl

This script attempts to reduce the contamination of bins by identifying duplicated markers (conserved genes for the given taxa that are expected to be single copy but are found to have more than one) in them. Then, it optimizes the removal of contigs containing these duplicated markers so that only one copy of the gene is left, and no other markers are removed.

This script can be found in the `/path/to/SqueezeMeta/utils/binning_utils` directory.

Usage

```
remove_duplicate_markers.pl <project name> [bin name]
```

If no bin name is provided, the script will run the analysis for ALL bins in the project.

Output

The scripts produces a new fasta file for the bin with the name "refined" in the binning directory (usually in <project>/results/DAS/<project>_DASTool_bins). It also runs checkM again to redo the statistics for the bin(s). The result of that checkM run is stored in <project>/temp/checkm_nodupl.txt

find_missing_markers.pl

This script intends to improve the completeness of the bin, using the checkM analysis to find contigs from the same taxa of the bin that contain missing markers (those that were not found in any contig of the bin). The user can then decide whether or not including these contigs in the bin.

This script can be found in the `/path/to/SqueezeMeta/utils/binning_utils/` directory.

Usage

```
find_missing_markers.pl <project name> [bin name]
```

If no bin name is provided, the script will run the analysis for ALL bins in the project.

The script also sets the variable $mode that affects the selection of contigs. Mode "relaxed" will consider contigs from all taxa not contradicting the taxonomy of the bin, including these that belong to higher-rank taxa (for instance, if the bin is annotated as "Escherichia" (genus), the script will consider also contigs classified as "Enterobacteriaceae" (family), "gamma-Proteobacteria" (class), or even "Bacteria" (superkingdom), since these assignments are not incompatible with the one of the bin). Mode "strict" will only consider contigs belonging to the same taxa of the bin (in the example above, only these classified as genus Escherichia).

Output

The script produces a list of contigs containing missing markers for the bin, sorted by the abundance of markers.

Explanation of SqueezeMeta algorithms

The LCA algorithm

We use a Last Common Ancestor (LCA) algorithm to assign taxa to genes.

For the amino acid sequence of each gene, DIAMOND (blastp) homology searches are done against the GenBank nr database. A e-value cutoff of 1e-03 is set by default. The best hit is obtained, and then we select a range of hits (valid hits) having at least 80% of the bitscore of the best hit and differing in less than 10% identity also with the best hit (these values can be set). The LCA of all these hits is obtained, that is, the taxon common to all hits. This LCA can be found at diverse taxonomic ranks (from phylum to species). We allow some flexibility in the definition of LCA: a small number of hits belonging to other taxa than the LCA can be allowed. In this way we deal with putative transfer events, or incorrect annotations in the database. This value is by default 10% of the total number of valid hits, but can be set by the user. Also, the minimum number of hits to the LCA taxa can be set.

An example is shown in the next table:

GenID	Hit ID	Hit taxonomy	Identity	e-value
Gen1	Hit1	Genus:Polaribacter Family: Flavobacteriaceae Order:Flavobacteriales	75.2	1e-94
Gen1	Hit2	Genus:Polaribacter Family: Flavobacteriaceae Order:Flavobacteriales	71.3	6e-88
Gen1	Hit3	Family: Flavobacteriaceae Order:Flavobacteriales	70.4	2e-87
Gen1	Hit4	Genus:Algibacter Family: Flavobacteriaceae Order:Flavobacteriales	68.0	2e-83

Gen1	Hit5	Genus:Rhodospirillum Family: Rhodospirillaceae Order:Rhodospirillales	60.2	6e-68

In this case, the four first hits are the valid ones. Hit 5 does not make the identity and e-value thresholds. The LCA for the four valid hits is Family: Flavobacteriaceae, that would be the reported result.

Our LCA algorithm includes strict cut-off identity values for different taxonomic ranks, according to Luo *et al.*, Nucleic Acids Research 2014, 42, e73. This means that hits must pass a minimum (aminoacid) identity level in order to be used for assigning particular taxonomic ranks. These thresholds are 85, 60, 55, 50, 46, 42 and 40% for species, genus, family, order, class, phylum and superkingdom ranks, respectively. Hits below these levels cannot be used to make assignments for the corresponding rank. For instance, a protein will not be assigned to species level if it has no hits above 85% identity. Also, a protein will remain unclassified if it has no hits above 40% identity. The inclusion of these thresholds guarantees that no assignments are done based on weak, inconclusive hits.

Overall, this results in highly conservative, but also very trustworthy taxonomic annotations. If SqueezeMeta says that something is annotated at the genus or species level it really means it. However, these filters do not work so well for eukaryotes, resulting in too few annotations. We believe that the main reasons are the following:

- Those filters were calculated using prokaryotic genomes, and might not be valid in eukaryotes.
- Eukaryotes are poorly represented in the databases, leading to lower similarities on average.

Adding the `--euk` flag when running SqueezeMeta will apply Luo *et al.*'s cutoffs only to prokaryotic genes, and use a normal LCA algorithm with no cutoff for the eukaryotes. This still gives a very robust taxonomy for prokaryotes, while also providing some information on what kind of eukaryotes might be in the sample.

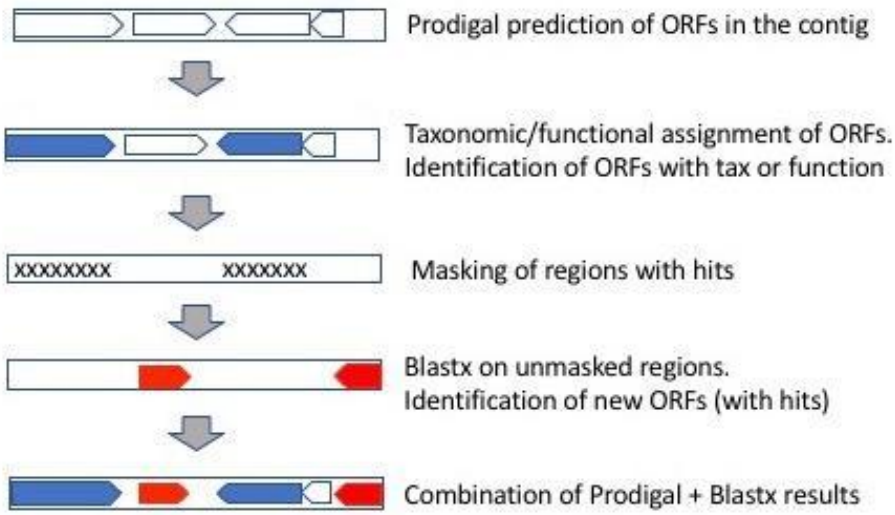
The fun3 algorithm

Fun3 is the algorithm that produces functional assignments (for COGs, KEGG and external databases). It reads the DIAMOND Blastx output of the homology search of the metagenomic genes for these databases. The homology search has been done with the defined parameters of e-value and identity, so that no hits below above the minimum e-value or below the minimum identity are found. Also, partial hits (where query and hits align in less than the percentage given by the user, 30% by default) are discarded. The hits that pass the filters can correspond to more than one functional ID (for instance, COG or KEGG ID). Fun3 provides two types of classification: Best hit is just the functional ID of the highest scoring hit. Best average tries to evaluate also if that functional ID is significantly better than the rest. For that, it takes the first *n* hits corresponding to each functional ID (*n* set by the user, default is 5) and calculates their average bitscore. The gene is assigned to the functional ID with the highest average bitscore that exceeds in a given percentage (given by the user, by default 10%) the score of the second one. This method reports less assignments but it is also more precise, avoiding confusions between closely related protein families.

A unique functional assignment, the best hit, is shown in the gene table. There, the functional ID is shown with a * symbol to indicate that the assignment is supported also by the best average method.

Doublepass: Blastx on contig gaps

The -D option activates the doublepass procedure, where regions of the contigs where no ORFs where predicted, or where these ORFs could not be assigned taxonomically and functionally, are queried against the databases using blastx. This method allows to recover putative ORFs missed by Prodigal, or to correct wrongly predicted ORFs. The following figure illustrates the steps of the doublepass procedure:



Consensus taxonomic annotation for contigs and bins

The consensus algorithm attempts to obtain a consensus taxonomic annotation for the contigs according to the annotations of each of its genes. The consensus taxon is the one fulfilling:

-50% of the genes of the contig belong to (are annotated to) this taxon, and

-70% of the annotated genes belong to (are annotated to) this taxon.

Notice that the first criterion refers to all genes in the contig, regardless if they have been annotated or not, while the second refers exclusively to annotated genes.

As the assignment can be done at different taxonomic ranks, the consensus is the deepest taxon fulfilling the criteria above.

For instance, consider the following example for a contig with 6 genes:

```
Gen1:
k_Bacteria;p_Proteobacteria;c_Gamma-Proteobacteria;o_Enterobacteri
ales;f_ Enterobacteriaceae;g_Escherichia;s_Escherichia coli

Gen2:
k_Bacteria;p_Proteobacteria;c_Gamma-Proteobacteria;o_Enterobacteri
ales;f_ Enterobacteriaceae;g_Escherichia
```

```
Gen3:
k_Bacteria;p_Proteobacteria;c_Gamma-Proteobacteria;o_Enterobacteri
ales;f_ Enterobacteriaceae;g_Escherichia

Gen4:
k_Bacteria;p_Proteobacteria;c_Gamma-Proteobacteria;o_Enterobacteri
ales;f_ Enterobacteriaceae

Gen5: No hits

Gen6: k_Bacteria;p_Firmicutes
```

In this case, the contig will be assigned to k_Bacteria;p_Proteobacteria;c_Gamma-Proteobacteria;o_ Enterobacteriales;f_ Enterobacteriaceae, which is the deepest taxon fulfilling 50% of all the genes belonging to that taxon (4/6=66%), and having 70% of the annotated genes (4/5=80%). The assignment to genus Escherichia was not done since just 3/5=60% of the annotated genes belong to it, which is below the cutoff threshold.

For annotating the consensus of bins, the procedure is the same, but using the annotations of the corresponding contigs instead.

Disparity calculation

Notice that in the example above, the end part of the contig seems to depart from the common taxonomic origin of the rest. This can be due to misassembly resulting in chimerism, or other causes such as a recent LCA transfer or a wrong annotation for the gene. The disparity index attempts to measure this effect, so that the contigs can be flagged accordingly (for instance, we could decide not trusting contigs with high disparity).

Disparity index is calculated for the taxonomic rank assigned by consensus algorithm (in the previous example, family). We compare the assignments at that level for every pair of genes in the contig, and count the number of agreements and disagreements. If one of the taxa has no annotation at that level, is not counted for agreement but it is counted for disagreements if previous ranks do not coincide (we assume that if higher ranks do not agree, lower ranks will not either). That is:

```
Gen1-Gen2: Agree

Gen1-Gen3: Agree
```

```
Gen1-Gen4: Agree

Gen1-Gen5: Unknown

Gen1-Gen6: Disagree (at phylum level)

Gen2-Gen3: Agree

Gen2-Gen4: Agree

Gen2-Gen5: Unknown

Gen2-Gen6: Disagree (at phylum level)

Gen3-Gen4: Agree

Gen3-Gen5: Unknown

Gen3-Gen6: Disagree (at phylum level)

Gen4-Gen5: Unknown

Gen4-Gen6: Disagree (at phylum level)

Gen5-Gen6: Unknown
```

Disparity index is the ratio between the number of disagreements and the total number of comparisons, in this case 4/15=0.26

For calculating the disparity of bins, the procedure is the same, just using the annotations for the contigs belonging to the bin instead.