CONCOCT Documentation

Release 0.4.1

Johannes Alneberg, Brynjar Smari Bjarnason, Ino de Bruijn, Mela

June 09, 2017

Contents

CONCOCT "bins" metagenomic contigs. Metagenomic binning is the process of clustering sequences into clusters corresponding to operational taxonomic units of some level.

For any known issues with CONCOCT check the issue tracker: https://github.com/BinPro/CONCOCT/issues

CHAPTER 1

Features

CONCOCT does unsupervised binning of metagenomic contigs by using nucleotide composition - kmer frequencies - and coverage data for multiple samples. CONCOCT can accurately (up to species level) bin metagenomic contigs. For optimal performance:

- Map several samples against your assembled contigs.
- Cut longer contigs into 10 - 20 kb pieces prior to mapping.
- Evaluate your bins using single copy genes.

CHAPTER 2

Installation

For a comprehensive guide on how to install CONCOCT and all its dependencies, see Installation.

CHAPTER 3

Contribute

- Issue Tracker: github
- Source Code: github

CHAPTER 4

Support

If you are having issues, please let us know. We have a mailing list located at: concoct-support@lists.sourceforge.net which you can subscribe to here.

CHAPTER 5

Licence

FreeBSD

CHAPTER 6

Contents:

Installation

Dependencies

Fundamental dependencies

```
python v2.7.*
gcc
gsl
```

These items are prerequisities for the installation of concoct as described below. The installation procedure varies on different systems, and described in this README is only how to proceed with a linux (ubuntu) distribution.

The first item, `python v2.7.*`, should be installed on a modern Ubuntu distribution. A c-compiler, e.g. `gcc`, is needed to compile the c parts of concoct that uses the GNU Scientific Library `gsl`. For linux (ubuntu) this is installed through:

```
apt-get install build-essential libgsl0-dev
```

Python packages

```
cython>=0.19.2
numpy>=1.7.1
scipy>=0.12.0
pandas>=0.11.0
biopython>=1.62b
scikit-learn>=0.13.1
```

These are the python packages that need to be installed in order to run concoct. If you follow the installation instructions below, these will be installed automatically, but are listed here for transparency.

Optional dependencies

- For assembly, use your favorite, here is one
  - Velvet
    - In velvet installation directory Makefile, set 'MAXKMERLENGTH=128', if this value is smaller in the default installation.

- To create the input table (containing average coverage per sample and contig)
  - BEDTools version >= 2.15.0 (only genomeCoverageBed)
  - Picard tools version >= 1.110
  - samtools version >= 0.1.18
  - bowtie2 version >= 2.1.0
  - GNU parallel version >= 20130422
  - Python packages: `pysam>=0.6`
- For validation of clustering using single-copy core genes
  - Prodigal >= 2.60
  - Python packages: `bcbio-gff>=0.4`
  - R packages: `gplots, reshape, ggplot2, ellipse, getopt` and `grid`
  - BLAST >= 2.2.28+

If you want to install these dependencies on your own server, you can take a look at doc/Dockerfile.all_dep for ideas on how to install them.

Installation

Here we describe two recommended ways of getting concoct to run on your computer/server. The first option, using Anaconda, should work for any *nix (e.g. Mac OS X or Linux) system even where you do not have 'sudo' rights (e.g. on a common computer cluster). The second option is suitable for a linux computer where you have root privileges and you prefer to use a virtual machine where all dependencies to run concoct are included. Docker does also run on Mac OS X through a virtual machine. For more information check out the Docker documentation.

Using Anaconda

This instruction shows how to install all dependencies (except the 'Fundamental dependencies' and the 'Optional dependencies' listed above) using an Anaconda environment. Anaconda is a tool to isolate your python installation, which allows you to have multiple parallel installations using different versions of different packages, and gives you a very convenient and fast way to install the most common scientific python packages. Anaconda is free but not open source, you can download Anaconda here. Installation instructions can be found here.

After installing Anaconda, create a new environment that will contain the concoct installation:

```
conda create -n concoct_env python=2.7.6
```

After choosing to proceed, run the suggested command:

```
source activate concoct_env
```

then install the concoct dependencies into this environment:

```
conda install cython numpy scipy biopython pandas pip scikit-learn
```

Finally, download the CONCOCT distribution from https://github.com/BinPro/CONCOCT/releases (stable) and extract the files, or clone the repository with github (potentially unstable). Resolve all dependencies, see above and then execute within the CONCOCT directory:

```
python setup.py install
```

Using Docker

If you have root access to a machine where you want to install concoct and you have storage for roughly 2G "virtual machine" then Docker provides a very nice way to get a Docker image with concoct and its dependencies installed. This way the only thing you install on your host system is Docker, the rest is contained in an Docker image. This allows you to install and run programs in that image without it affecting your host system. You should get to know Docker here. You need to get Docker installed and specially if you have Ubuntu. When Docker is installed you need to download and log into the concoct image.

We provide a Docker image:

binpro/concoct_latest contains CONCOCT and all its dependencies for the Complete Example V0.4 with the exception of the SCG evaluation.

The following command will then download the image from the Docker image index, map the Data folder to the image and log you into the docker image.

```
sudo docker run -v /home/USER/Data:/opt/Data -i -t binpro/concoct_latest bash
```

To test concoct you can then do:

```
$ cd /opt/CONCOCT_latest
$ nosetests
```

Which should execute all tests without errors.

Usage

CONCOCT uses several command line options to control the clustering, here is a complete documentation of these. These can also be viewed by typing `concoct -h` on the command line:

```
usage: - [-h] [--coverage_file COVERAGE_FILE]
         [--composition_file COMPOSITION_FILE] [-c CLUSTERS] [-k KMER_LENGTH]
         [-l LENGTH_THRESHOLD] [-r READ_LENGTH]
         [--total_percentage_pca TOTAL_PERCENTAGE_PCA] [-b BASENAME] [-s SEED]
         [-i ITERATIONS] [-e EPSILON] [--no_cov_normalization]
         [--no_total_coverage] [--no_original_data] [-o] [-d] [-v]

optional arguments:
  -h, --help            show this help message and exit
  --coverage_file COVERAGE_FILE
                        specify the coverage file, containing a table where
                        each row correspond to a contig, and each column
                        correspond to a sample. The values are the average
                        coverage for this contig in that sample. All values
                        are separated with tabs.
  --composition_file COMPOSITION_FILE
                        specify the composition file, containing sequences in
                        fasta format. It is named the composition file since
                        it is used to calculate the kmer composition (the
                        genomic signature) of each contig.
  -c CLUSTERS, --clusters CLUSTERS
                        specify maximal number of clusters for VGMM, default
                        400.
  -k KMER_LENGTH, --kmer_length KMER_LENGTH
                        specify kmer length, default 4.
  -l LENGTH_THRESHOLD, --length_threshold LENGTH_THRESHOLD
```

```
                      specify the sequence length threshold, contigs shorter
                      than this value will not be included. Defaults to
                      1000.
-r READ_LENGTH, --read_length READ_LENGTH
                      specify read length for coverage, default 100
--total_percentage_pca TOTAL_PERCENTAGE_PCA
                      The percentage of variance explained by the principal
                      components for the combined data.
-b BASENAME, --basename BASENAME
                      Specify the basename for files or directory where
                      outputwill be placed. Path to existing directory or
                      basenamewith a trailing '/' will be interpreted as a
                      directory.If not provided, current directory will be
                      used.
-s SEED, --seed SEED  Specify an integer to use as seed for clustering. 0
                      gives a random seed, 1 is the default seed and any
                      other positive integer can be used. Other values give
                      ArgumentTypeError.
-i ITERATIONS, --iterations ITERATIONS
                      Specify maximum number of iterations for the VBGMM.
                      Default value is 500
-e EPSILON, --epsilon EPSILON
                      Specify the epsilon for VBGMM. Default value is 1.0e-6
--no_cov_normalization
                      By default the coverage is normalized with regards to
                      samples, then normalized with regards of contigs and
                      finally log transformed. By setting this flag you skip
                      the normalization and only do log transorm of the
                      coverage.
--no_total_coverage   By default, the total coverage is added as a new
                      column in the coverage data matrix, independently of
                      coverage normalization but previous to log
                      transformation. Use this tag to escape this behaviour.
--no_original_data    By default the original data is saved to disk. For big
                      datasets, especially when a large k is used for
                      compositional data, this file can become very large.
                      Use this tag if you don't want to save the original
                      data.
-o, --converge_out    Write convergence info to files.
-d, --debug           Debug parameters.
-v, --version         show program's version number and exit
```

Complete Example V0.4

This documentation page aims to be a complete example walk through for the usage of the CONCOCT package version 0.3. It assumes you have successfully gone through the installation description found in the README.

Required software

To run the entire example you need to install all dependencies as stated in the README dependencies. This includes all the optional dependencies. You can also look at doc/Dockerfile to help you install these packages on your server.

Another way to get everything set up is to use our full Docker image (binpro/concoct_latest) as suggested in the README docker.

It is not required to run all steps. The output files for each step are in the test data repository. At the end of this example the results should be the same as the results in the corresponding test data repository: https://github.com/BinPro/CONCOCT-test-data/releases. The version numbers listed above are the ones used to generate the results in that repository. Using newer versions will probably not be a problem, but your results may be different in that case.

Downloading test data

First download the test data repository of CONCOCT corresponding to the version of CONCOCT, that you have installed. The test data repository can be downloaded here. Then extract it in a suitable location.

If you are running the current unstable master branch of concoct, you need to clone the latest version of the test-data-repository as well.

Setting up the test environment

Using Docker

On your HOST machine create the following folder structure below (Data, Data/CONCOCT-complete-example, Data/CONCOCT-test-data):

```
mkdir -p /HOST/path/to/Data
mkdir /HOST/path/to/Data/CONCOCT-complete-example
```

Move the test data that was downloaded and extracted (CONCOCT-test-data) to the Data folder

```
# Move the test data you extracted in the download part into the Data folder
mv /HOST/extracted/test/data/CONCOCT-test-data /HOST/path/to/Data/CONCOCT-test-data
```

Now you want to execute the following command to log into our Docker image and to map the `/HOST/path/to/Data` to your image and the Data folder will be accessable in /opt/Data:

```
sudo docker run -v /HOST/path/to/Data:/opt/Data/ -i -t binpro/concoct_latest bash
```

This will download the 2G image to your machine and then leaves you in a BASH shell. In the Docker image, the following environmental variables have been set. So if you have your folders set up differently in the steps above you need to alter these variables accordingly:

```
CONCOCT=/opt/CONCOCT_latest
CONCOCT_TEST=/opt/Data/CONCOCT-test-data
CONCOCT_EXAMPLE=/opt/Data/CONCOCT-complete-example
```

Your own setup

After obtaining the test data, create a folder where you want all the output from this example to go:

```
mkdir CONCOCT-complete-example
cd CONCOCT-complete-example
```

Set three variables with full paths. One pointing to the root directory of the `CONCOCT` software, one pointing to the test data repository, named `CONCOCT_TEST` and one to the directory we just created. If you now have these in the folder `/home/username/src/`, for instance, then use:

```
CONCOCT=/home/username/src/CONCOCT
CONCOCT_TEST=/home/username/src/CONCOCT-test-data
CONCOCT_EXAMPLE=/home/username/CONCOCT-complete-example
```

You can see the full path of a directory you are located in by running the command `pwd`.

Assembling Metagenomic Reads

The first step in the analysis is to assemble all reads into contigs, here we use the software Velvet for this. This step can be computationaly intensive but for this small data set comprising a synthetic community of four species and 16 samples (100,000 reads per sample) it can be performed in a few minutes. If you do not wish to execute this step, the resulting contigs are already in the test data repository, and you can copy them from there insted. The commands for running Velvet are:

```
cd $CONCOCT_EXAMPLE
cat $CONCOCT_TEST/reads/Sample*_R1.fa > All_R1.fa
cat $CONCOCT_TEST/reads/Sample*_R2.fa > All_R2.fa
velveth velveth_k71 71 -fasta -shortPaired -separate All_R1.fa All_R2.fa
velvetg velveth_k71 -ins_length 400 -exp_cov auto -cov_cutoff auto
```

After the assembly is finished create a directory with the resulting contigs and copy the result of Velvet there (this output is also in `$CONCOCT_TEST/contigs`):

```
mkdir contigs
cp velveth_k71/contigs.fa contigs/velvet_71.fa
rm All_R1.fa
rm All_R2.fa
```

Cutting up contigs

In order to give more weight to larger contigs and mitigate the effect of assembly errors we cut up the contigs into chunks of 10 Kb. The final chunk is appended to the one before it if it is < 10 Kb to prevent generating small contigs. This means that no contig < 20 Kb is cut up. We use the script `cut_up_fasta.py` for this:

```
cd $CONCOCT_EXAMPLE
python $CONCOCT/scripts/cut_up_fasta.py -c 10000 -o 0 -m contigs/velvet_71.fa > contigs/velvet_71_c1(
```

Map the Reads onto the Contigs

After assembly we map the reads of each sample back to the assembly using bowtie2 and remove PCR duplicates with MarkDuplicates. The coverage histogram for each bam file is computed with BEDTools genomeCoverageBed. The script that calls these programs is provided with CONCOCT.

If you are not using the Docker image, then one does need to set an environment variable with the full path to the MarkDuplicates jar file. `$MRKDUP` which should point to the MarkDuplicates jar file e.g.

```
#NOTE not necessary if using the Docker image
export MRKDUP=/home/username/src/picard-tools-1.77/MarkDuplicates.jar
```

It is typically located within your picard-tools installation.

The following command is to be executed in the `$CONCOCT_EXAMPLE` dir you created in the previous part. First create the index on the assembly for bowtie2:

```
cd $CONCOCT_EXAMPLE
bowtie2-build contigs/velvet_71_c10K.fa contigs/velvet_71_c10K.fa
```

Then run this for loop, which for each sample creates a folder and runs `map-bowtie2-markduplicates.sh`:

```
for f in $CONCOCT_TEST/reads/*_R1.fa; do
    mkdir -p map/$(basename $f);
    cd map/$(basename $f);
    bash $CONCOCT/scripts/map-bowtie2-markduplicates.sh -ct 1 -p '-f' $f $(echo $f | sed s/R1/R2/) pa
    cd ../..;
done
```

The parameters used for `map-bowtie2-markduplicates.sh` are:

- `-c` option to compute coverage histogram with genomeCoverageBed
- `-t` option is number of threads
- `-p` option is the extra parameters given to bowtie2. In this case `-f`.

The five arguments are:

- pair1, the fasta/fastq file with the #1 mates
- pair2, the fasta/fastq file with the #2 mates
- pair_name, a name for the pair used to prefix output files
- assembly, a fasta file of the assembly to map the pairs to
- assembly_name, a name for the assembly, used to postfix outputfiles
- outputfolder, the output files will end up in this folder

Generate coverage table

Use the bam files of each sample to create a table with the coverage of each contig per sample.

```
cd $CONCOCT_EXAMPLE/map
python $CONCOCT/scripts/gen_input_table.py --isbedfiles \
    --samplenames <(for s in Sample*; do echo $s | cut -d'_' -f1; done) \
    ../contigs/velvet_71_c10K.fa */bowtie2/asm_pair-smds.coverage \
> concoct_inputtable.tsv
mkdir $CONCOCT_EXAMPLE/concoct-input
mv concoct_inputtable.tsv $CONCOCT_EXAMPLE/concoct-input/
```

Generate linkage table

The same bam files can be used to give linkage per sample between contigs:

```
cd $CONCOCT_EXAMPLE/map
python $CONCOCT/scripts/bam_to_linkage.py -m 8 \
    --regionlength 500 --fullsearch \
    --samplenames <(for s in Sample*; do echo $s | cut -d'_' -f1; done) \
    ../contigs/velvet_71_c10K.fa Sample*/bowtie2/asm_pair-smds.bam \
> concoct_linkage.tsv
mv concoct_linkage.tsv $CONCOCT_EXAMPLE/concoct-input/
```

Run concoct

To see possible parameter settings with a description run

```
$CONCOCT/bin/concoct --help
```

We will only run concoct for some standard settings here. First we need to parse the input table to just contain the mean coverage for each contig in each sample:

```
cd $CONCOCT_EXAMPLE
cut -f1,3- concoct-input/concoct_inputtable.tsv > concoct-input/concoct_inputtableR.tsv
```

Then run concoct with 40 as the maximum number of cluster `-c 40`, that we guess is appropriate for this data set:

```
cd $CONCOCT_EXAMPLE
concoct -c 40 --coverage_file concoct-input/concoct_inputtableR.tsv --composition_file contigs/velvet
```

When concoct has finished the message "CONCOCT Finished, the log shows how it went." is piped to stdout. The program generates a number of files in the output directory that can be set with the `-b` parameter and will be the present working directory by default.

Evaluate output

This will require that you have Rscript with the R packages gplots, reshape, ggplot2, ellipse, getopt and grid installed. The package grid does not have to be installed for R version > 1.8.0

First we can visualise the clusters in the first two PCA dimensions:

```
cd $CONCOCT_EXAMPLE
mkdir evaluation-output
Rscript $CONCOCT/scripts/ClusterPlot.R -c concoct-output/clustering_gt1000.csv -p concoct-output/PCA_
```

https://github.com/BinPro/CONCOCT-test-data/tree/master/evaluation-output/ClusterPlot.pdf

We can also compare the clustering to species labels. For this test data set we know these labels, they are given in the file `clustering_gt1000_s.csv`. For real data labels may be obtained through taxonomic classification, e.g. using:

https://github.com/umerijaz/TAXAassign

In either case we provide a script Validate.pl for computing basic metrics on the cluster quality:

```
cd $CONCOCT_EXAMPLE
cp $CONCOCT_TEST/evaluation-output/clustering_gt1000_s.csv evaluation-output/
$CONCOCT/scripts/Validate.pl --cfile=concoct-output/clustering_gt1000.csv --sfile=evaluation-output/c
```

This script requires the clustering output by concoct `concoct-output/clustering_gt1000.csv` these have a simple format of a comma separated file listing each contig id followed by the cluster index and the species labels that have the same format but with a text label rather than a cluster index. The script should output:

```
N   M   TL  S   K   Rec.    Prec.   NMI Rand    AdjRand
684 684 6.8023e+06  5   4   0.897224    0.999604    0.841911    0.911563    0.823200
```

This gives the no. of contigs N clustered, the number with labels M, the number of unique labels S, the number of clusters K, the recall, the precision, the normalised mutual information (NMI), the Rand index, and the adjusted Rand index. It also generates a file called a `confusion matrix` with the frequencies of each species in each cluster. We provide a further script for visualising this as a heatmap:

```
$CONCOCT/scripts/ConfPlot.R  -c evaluation-output/clustering_gt1000_conf.csv -o  evaluation-output/cl
```

This generates a file with normalised frequencies of contigs from each cluster across species:

https://github.com/BinPro/CONCOCT-test-data/tree/master/evaluation-output/clustering_gt1000_conf.pdf

Validation using single-copy core genes

We can also evaluate the clustering based on single-copy core genes. You first need to find genes on the contigs and functionally annotate these. Here we used prodigal (https://github.com/hyattpd/Prodigal) for gene prediction and annotation, but you can use anything you want:

```
cd $CONCOCT_EXAMPLE
mkdir -p $CONCOCT_EXAMPLE/annotations/proteins
prodigal -a annotations/proteins/velvet_71_c10K.faa \
         -i contigs/velvet_71_c10K.fa \
         -f gff -p meta  > annotations/proteins/velvet_71_c10K.gff
```

We used RPS-Blast to COG annotate the protein sequences using the script `RSBLAST.sh`. You need to set the evironmental variable `COGSDB_DIR`:

```
export COGSDB_DIR=/proj/b2010008/nobackup/database/cog_le/
```

The script furthermore requires GNU parallel and rpsblast. Here we run it on eight cores:

```
$CONCOCT/scripts/RPSBLAST.sh -f annotations/proteins/velvet_71_c10K.faa -p -c 8 -r 1
mkdir $CONCOCT_EXAMPLE/annotations/cog-annotations
mv velvet_71_c10K.out annotations/cog-annotations/
```

The blast output has been placed in:

```
$CONCOCT_TEST/annotations/cog-annotations/velvet_71_c10K.out
```

Finally, we filtered for COGs representing a majority of the subject to ensure fragmented genes are not over-counted and generated a table of counts of single-copy core genes in each cluster generated by CONCOCT. Remember to use a real email adress, this is supplied since information is fetched from ncbi using their service eutils, and the email is required to let them know who you are.

```
cd $CONCOCT_EXAMPLE
$CONCOCT/scripts/COG_table.py -b annotations/cog-annotations/velvet_71_c10K.out \
-m $CONCOCT/scgs/scg_cogs_min0.97_max1.03_unique_genera.txt \
-c concoct-output/clustering_gt1000.csv \
--cdd_cog_file $CONCOCT/scgs/cdd_to_cog.tsv > evaluation-output/clustering_gt1000_scg.tab
```

The script requires the clustering output by concoct `concoct-output/clustering_gt1000.csv`, a file listing a set of SCGs (e.g. a set of COG ids) to use `scgs/scg_cogs_min0.97_max1.03_unique_genera.txt` and a mapping of Conserved Domain Database ids (https://www.ncbi.nlm.nih.gov/Structure/cdd/cdd.shtml) to COG ids `$ONCOCT/scgs/cdd_to_cog.tsv`. If these protein sequences were generated by Prokka, the names of the contig ids needed to be recovered from the gff file. Since prodigal has been used, the contig ids instead are recovered from the protein ids using a separator character, in which case only the string before (the last instance of) the separator will be used as contig id in the annotation file. In the case of prodigal the separator that should be used is _ and this is the default value, but other characters can be given through the '–separator' argument.

The output file is a tab-separated file with basic information about the clusters (cluster id, ids of contigs in cluster and number of contigs in cluster) in the first three columns, and counts of the different SCGs in the following columns.

This can also be visualised graphically using the R script:

```
cd $CONCOCT_EXAMPLE
$CONCOCT/scripts/COGPlot.R -s evaluation-output/clustering_gt1000_scg.tab -o evaluation-output/cluste
```

The plot is downloadable here:

https://github.com/BinPro/CONCOCT-test-data/tree/master/evaluation-output/clustering_gt1000_scg.pdf

Incorporating linkage information

To perform a hierarchical clustering of the clusters based on linkage we simply run:

```
$CONCOCT/scripts/ClusterLinkNOverlap.pl --cfile=concoct-output/clustering_gt1000.csv --lfile=concoct-
```

The output indicates that the clusters have been reduced from four to three. The new clustering is given by `concoct-output/clustering_gt1000_l.csv`. This is a significant improvement in recall:

```
$CONCOCT/scripts/Validate.pl --cfile=concoct-output/clustering_gt1000_l.csv --sfile=evaluation-output
N   M   TL  S   K   Rec.    Prec.   NMI Rand    AdjRand
684 684 6.8400e+02  5   3   1.000000    0.997076    0.995805    0.999979    0.999957
```

The algorithm is explained in more depth in the paper on arXiv

CONCOCT Scripts

The scripts in the `CONCOCT/scripts` directory are not fully maintained. They implement methods that we apply after binning with CONCOCT. Eventually some of these methods might make it to a package of their own.

To test all scripts that have tests one could do:

```
cd CONCOCT/scripts/tests
nosetests
```

Before using a script it would be good to check if its test (in case it has one) is working for you:

```
cd CONCOCT/scripts/tests
nosetests -s test_script_name
```

Contents:

dnadiff_dist_matrix.py

Usage

The usage and help documentation of `dnadiff_dist_matrix.py` can be seen by running `pyhton dnadiff_dist_matrix -h`:

```
usage: - [-h] [--min_coverage MIN_COVERAGE] [--fasta_names FASTA_NAMES]
         [--plot_image_extension PLOT_IMAGE_EXTENSION] [--skip_dnadiff]
         [--skip_matrix] [--skip_plot]
         output_folder fasta_files [fasta_files ...]

Output distance matrix between fasta files using dnadiff from MUMmer. Generates
dnadiff output files in folders:

output_folder/fastaname1_vs_fastaname2/
```

```
output_folder/fastaname1_vs_fastaname3/

etc

where fastaname for each fasta file can be supplied as an option to the script.
Otherwise they are just counted from 0 to len(fastafiles)

The distance between each bin is computed using the 1-to-1 alignments of the
report files (not M-to-M):

1 - AvgIdentity if min(AlignedBases) >= min_coverage. Otherwise distance is 1.
Or 0 to itself.

Resulting matrix is printed to stdout and to output_folder/dist_matrix.tsv. The
rows and columns of the matrix follow the order of the supplied fasta files. The
names given to each fasta file are also outputted to the file
output_folder/fasta_names.tsv

A hierarchical clustering of the distance using euclidean average linkage
clustering is plotted. This can be deactivated by using --skip_plot. The
resulting heatmap is in output_folder/hclust_heatmap.pdf or
output_folder/hclust_dendrogram.pdf. The image extension can be changed.

positional arguments:
  output_folder         Output folder
  fasta_files           fasta files to compare pairwise using MUMmer's dnadiff

optional arguments:
  -h, --help            show this help message and exit
  --min_coverage MIN_COVERAGE
                        Minimum coverage of bin in percentage to calculate
                        distance otherwise distance is 1. Default is 50.
  --fasta_names FASTA_NAMES
                        File with names for fasta file, one line each. Could
                        be sample names, bin names, genome names, whatever you
                        want. The names are used when storing the MUMmer
                        dnadiff results as in
                        output_folder/fastaname1_vs_fastaname2/. The names are
                        also used for the plots.
  --plot_image_extension PLOT_IMAGE_EXTENSION
                        Type of image to plotted e.g. pdf, png, svg.
  --skip_dnadiff        Skips running MUMmer and uses output_folder as given
                        input to calculate the distance matrix. Expects
                        dnadiff output as
                        output_folder/fastaname1_vs_fastaname2/out.report
  --skip_matrix         Skips Calculating the distance matrix.
  --skip_plot           Skips plotting the distance matrix. By default the
                        distance matrix is clustered hierarchically using
                        euclidean average linkage clustering. This step
                        requires seaborn and scipy.
```

Example

An example of how to run `dnadiff_dist_matrix` on the test data:

```
cd CONCOCT/scripts
python dnadiff_dist_matrix.py test_dnadiff_out tests/test_data/bins/sample*.fa
```

This results in the following output files in the folder `test_dnadiff_out/`:

- `dist_matrix.stv` The distance matrix
- `fasta_names.tsv` The names given to each bin (or fasta file)
- `hcust_dendrogram.pdf` Dendrogram of the clustering (click for example)
- `hcust_heatmap.pdf` Heatmap of the clustering (click for example)

Then there is also for each pairwise `dnadiff` alignment the following output files in a subfolder `fastaname1_vs_fastaname2/`:

```
out.1coords
out.1delta
out.cmd
out.delta
out.mcoords
out.mdelta
out.qdiff
out.rdiff
out.report
out.snps
out.unqry
out.unref
```

See MUMmer's own manual for an explanation of each file with `dnadiff --help`.

extract_scg_bins.py

Usage

The usage and help documentation of `extract_scg_bins.py` can be seen by running `pyhton extract_scg_bins -h`:

```
usage: - [-h] --output_folder OUTPUT_FOLDER --scg_tsvs SCG_TSVS [SCG_TSVS ...]
         --fasta_files FASTA_FILES [FASTA_FILES ...] --names NAMES [NAMES ...]
         [--groups GROUPS [GROUPS ...]] [--max_missing_scg MAX_MISSING_SCG]
         [--max_multicopy_scg MAX_MULTICOPY_SCG]

Extract bins with given SCG (Single Copy genes) criteria. Criteria can be set
as a combination of the maximum number of missing SCGs and the maximum number
of multicopy SCGs. By default the script selects from pairs of scg_tsvs and
fasta_files, the pair that has the highest number of approved bins. In case
there are multiple with the max amount of approved bins, it takes the one that
has the highest sum of bases in those bins. If that is the same, it selects the
first one passed as argument.

One can also group the pairs of scg_tsvs and fasta_files with the --groups
option so one can for instance find the best binning per sample.

optional arguments:
  -h, --help            show this help message and exit
  --output_folder OUTPUT_FOLDER
                        Output folder
  --scg_tsvs SCG_TSVS [SCG_TSVS ...]
                        Single Copy Genes (SCG) tsvs as outpututted by
                        COG_table.py. Should have the same ordering as
                        fasta_files.
```

```
  --fasta_files FASTA_FILES [FASTA_FILES ...]
                        Fasta files. Should have the same ordering as scg_tsvs
  --names NAMES [NAMES ...]
                        Names for each scg_tsv and fasta_file pair. This is
                        used as the prefix for the outputted bins.
  --groups GROUPS [GROUPS ...]
                        Select the best candidate for each group of scg_tsv
                        and fasta_file pairs. Number of group names given
                        should be equal to the number of scg_tsv and
                        fasta_file pairs. Identical group names indicate same
                        groups.
  --max_missing_scg MAX_MISSING_SCG
  --max_multicopy_scg MAX_MULTICOPY_SCG
```

Example

An example of how to run `extract_scg_bins` on the test data:

```
cd CONCOCT/scripts/tests/test_data
python extract_scg_bins.py \
    --output_folder test_extract_scg_bins_out \
    --scg_tsvs tests/test_data/scg_bins/sample0_gt300_scg.tsv \
               tests/test_data/scg_bins/sample0_gt500_scg.tsv \
    --fasta_files tests/test_data/scg_bins/sample0_gt300.fa \
                  tests/test_data/scg_bins/sample0_gt500.fa \
    --names sample0_gt300 sample0_gt500 \
    --max_missing_scg 2 --max_multicopy_scg 4 \
    --groups gt300 gt500
```

This results in the following output files in the folder `test_extraxt_scg_bins_out/`:

```
$ ls test_extract_scg_bins_out/
sample0_gt300_bin2.fa  sample0_gt500_bin2.fa
```

Only bin2 satisfies the given criteria for both binnings. If we want to get the best binning of the two, one can remove the `--groups` parameter (or give them the same group id). That would only output `sample0_gt500_bin2.fa`, because the sum of bases in the approved bins of `sample0_gt500` is higher than that of `sample0_gt300`.