# CoCoNet

***Release 1.1.0***

**Mar 16, 2021**

# Contents

# CHAPTER 1

# Citation (Work in progress)

Arisdakessian C., Nigro O., Steward G., Poisson G., Belcaid M. CoCoNet: An Efficient Deep Learning Tool for Viral Metagenome Binning

# CHAPTER 2

# Description

CoCoNet (Composition and Coverage Network) is a binning method for viral metagenomes. It leverages deep learning to abstract the modeling of the k-mer composition and the coverage for binning contigs assembled form viral metagenomic data. Specifically, our method uses a neural network to learn from the metagenomic data a flexible function for predicting the probability that any pair of contigs originated from the same genome. These probabilities are subsequently combined to infer bins, or clusters representing the species present in the sequenced samples. Our approach was specifically designed for diverse viral metagenomes, such as those found in environmental samples (e.g., oceans, soil, etc.).

# CHAPTER 3

# Install

## 3.1 Install latest PyPi release (recommended)

```
pip3 install --user numpy
pip3 install --user coconet-binning
```

For more installation options, see the documentation

# CHAPTER 4

# Basic usage

CoCoNet is available as the command line program. For a list of all the options, open a terminal and run:

```
coconet run -h
```

For more details, please see the documentation on ReadTheDocs

# CHAPTER 5

# Checking the installation

A test dataset is provided in this repository in tests/sim_data. To quickly verify the installation worked, you can simply download the repository and run the test command as follows:

```
git clone https://github.com/Puumanamana/CoCoNet
cd CoCoNet
make test
```

# CHAPTER 6

# Contribute

- Issue Tracker: github
- Source Code: github

## 6.1 Installation

### 6.1.1 Pre-requisites

CoCoNet was tested on both MacOS and Ubuntu 18.04. To install and run CoCoNet, you will need:

1. *python* (>=3.5)
2. *pip3*, the python package manager or the *conda* installer.

### 6.1.2 Install the latest release on PyPi

You can install CoCoNet, from the Python Package Index. To install it, you simply need to run the following command (you can omit –user if you're working in a vitualenv):

```
# Install numpy until scikit-bio issue #1690 is resolved
pip install --user numpy
# Install CoCoNet
pip3 install --user coconet-binning
```

### 6.1.3 Install the development version

You can also install the most up to date version with the following command:

```
# Install numpy until scikit-bio issue #1690 is resolved
pip install --user numpy
# Install CoCoNet
git clone https://github.com/Puumanamana/CoCoNet.git && cd CoCoNet
pip install --user .
```

If you encounter any issue with the development version, you should try with the latest release as the development version might not have been thoroughly checked.

### 6.1.4 Using Docker

Alternatively, CoCoNet can be pulled directly from DockerHub. Assuming your contigs are located in /data/contigs.fasta and your indexed bam files are in /data/*.bam and /data/*.bai, then you can run CoCoNet with the following command:

```
docker run -v /data:/workspace nakor/coconet coconet run --fasta contigs.fasta --bam␣
↪*.bam
```

### 6.1.5 Install with bioconda

CoCoNet is available in bioconda. You will need miniconda or anaconda installed on your computer. CoCoNet can be installed using the following command:

```
conda install -c bioconda coconet-binning
```

## 6.2 Usage

### 6.2.1 Inputs

CoCoNet's main running mode bins contigs assembled using multiple samples. The minimum required is:

1. The assembly in *fasta* format, generated from the raw reads with your favorite assembler (e.g. metaspades, megahit, idb-ud, metavelvet, . . . )
2. The alignments in the *bam* format. These files can be generated from the raw *fastq* reads and the assembly using an aligner such as bwa or bowtie2.

### 6.2.2 Running CoCoNet

The CoCoNet package comes with 4 subcommands to run different part of the algorithm:

- `preprocess`: Filter the assembly and coverage based on minimum length, and prevalence. In addition, alignments are filtered based on quality, SAM flag, coverage and template length.
- `learn`: Train the neural network to predict whether two DNA fragments belong to the same genome.
- `cluster`: Cluster the contigs using the previously trained neural network.
- `run`: Runs all the previous steps combined (recommended)

To run CoCoNet with the default parameters, you simply need to provide the assembly and the bam coverage files:

```
coconet --fasta scaffolds.fasta --bam cov/*.bam --output binning_results
```

You can see the usage information for each subcommand by typing `coconet <subcommand> -h`, where <*subcommand*> is either *preprocess*, *learn*, *cluster* or *run*. For more details about the options, see the *hyperparameters* section

You can use the `--continue` flag to resume an interrupted run. However, depending where your run was interrupted, you might have corrupted files, in which case you would need to either re-run from the start or delete the corrupted file.

### 6.2.3 Outputs

The output data folder contains many files. The ones you might be interested in are:

- The binning outcome, *bins_*.csv* with 2 columns: the first is the contig name, and the second is the bin number.
- The log file, *coconet.log*, that contains all of the runtime information, run parameters and filtering information.
- The run configuration *config.yaml* with the run parameters.
- The filtered alignments (both .bam and .h5 formats) and the filtered assembly (.fasta).
- The list of contigs that were set aside because they didn't pass the prevalence or minimum length filter, *exclude.tsv*. The first column is the contig name, the second is the reason why the contig was excluded, and the remaining field is the values corresponding to the filtering criteria (length or list of coverage values).

## 6.3 Hyperparameters

CoCoNet's parameters were set in order to put an emphasis on bin homogeneity. However, depending on the research goal, the user might want to emphasize on either one. Mainly, three parameters can be adjusted to improve bin completeness at the cost of possibly decreased homogeneity:

1. The fragment length, `--fragment-length`
2. The minimum prevalence `--min-prevalence` (i.e. the number of samples a given contig appear in)
3. The minimum number matches between two contigs connected by an edge in the contig-contig graph, $\theta$, `--theta`
4. The minimum edge density required for a cluster to be considered as a coherent bin, $\gamma$, `--gamma2`

Decreasing the values of $\theta$ or $\gamma$ (respectively 80% and 75% by default) decreases the binning stringency. This can, therefore, improve the completeness of viruses with higher variance in their k-mer or coverage patterns, albeit at the cost of possibly decreased homogeneity. Similarly, increasing the fragment length can minimize the variance in the k-mer and coverage distributions between contigs of the same species and, consequently, improve completeness. Nevertheless, a longer fragment length (or greater prevalence) thresholds can result in more contigs being assigned to singleton bins simply because they were not long enough to be processed. In addition, increasing the minimum prevalence will results in selecting contigs that are more broadly present across samples. Intuitively, a high prevalence provides a more robust information for binning; indeed, two contigs co-occurring in two samples provide a less robust evidence than if they were co-occurring in 5 samples. Naturally, decreasing the values of $\theta$, $\gamma$, the fragment length or the minimum prevalence can result in more homogeneous but less complete bins.

A few other parameters can be worth tuning. However, their effect has not yet been assessed thoroughly and were empirically chosen.

### 6.3.1 Preprocessing

- `--min-ctg-length` (default: 2048): Discard short contigs. Shorter contig's composition can have local pattern that very different than the complete genome. As a result, they might generate more false positives. In CoCoNet, `--min-ctg-length` needs to be longer than the `--fragment-length` since contigs are split into fragments during the clustering phase.
- `--min-prevalence` (default: 2): Contig that appear in few samples are harder to bin since less information can be leveraged from their co-occurrence with other contigs. Therefore increasing the `--min-prevalence` should increase the binning quality (but filter out more contigs).
- `--flag` (default: 3596): Sam flag filtering (corresponds to `-F` flag in samtools). The flag meaning can be explored here
- `--min-mapping-quality` (default: 30): Discards any alignment with a low quality
- `--min-aln-coverage` (default: 50): Discards any alignment with less than x % aligned nucleotides
- `--tlen-range` (default: no filtering): Discards any paired alignments with a template length outside this range

### 6.3.2 Learning

- `-k` (default 4, recommended range is between 3-6): kmer size. It will impact the composition feature. The longer the kmer and the more specific it will be to a particular genome. However, for short fragments, it might generate very sparse vectors that might prevent the network to learn appropriately. As such, for kmer of size 5 or greater, we recommend increasing the fragment length as well.
- `--wsize` (default: 64) and `--wstep` (default: 32) control the smoothing of the coverage input. Coverage is smoothed using an averaging window of length `wsize` and step `wstep`.
- `--n-train` (default: 4M): Number of training examples. Training examples are generated by randomly pairing contigs' fragments. Because there are many potential pairs (depending on the number and length of the contigs), we usually can generates millions of example. In our tests, the training usually starts to plateau after a few hundred thousands examples. Therefore, we set the number of training examples to 1 million.
- The neural network learning hyperparameters: `--batch-size` (default: 256) and `--learning-rate` (default: 1e-4).
- The neural network architecture: These values were empiriclly chosen and should not require any tuning. They are the number of neurons in the composition layer `--compo-neurons` (default: [64, 32]), in the coverage layer `--cover-neurons` (default: [64, 32]) and in the merging layer `--merge-neurons` (default: 32). The convolution in the coverage network is controlled by the number of filters `--cover-filters` (default: 32), the kernel size `--cover-kernel` (default: 7) and stride `--cover-stride` (default: 3).
- `--load-batch` (default: 100): does not affect the accuracy but simply controls how many examples are loaded at once in the memory. It should be used if the memory is very limited.
- `--n-frags` (default: 30): Number of fragments to split the contigs in for the clustering phase. More fragments should make the contig-contig comparisons more trustworthy. The number of non-redundant fixed-size fragments in a contig is however limited, and the value of this variable will likely plateau or even be detrimental is this value becomes to high.
- `--features` (default: coverage composition): The features to use for binning. We recommend to use both coverage and composition since the network can learn from two separate sources to group DNA fragments. However, if the dataset contains very few samples (<3) and if the coverage information is of low quality, composition-only binning might be a good alternative.

### 6.3.3 Clustering

- `--max-neighbors` (default: 250): The maximum number of neighbors to consider to compute the adjacency matrix. Increasing it should make the results more accurate but might also significantly increase computing time.
- `--vote-threshold` (default: None): When set, contigs are compared against each other using a voting scheme. Instead of summing the network probabilities across all fragments pairs between the two contigs, a hard threshold is used to set each comparion to 0 or 1.
- `--theta` (default: 0.8): Minimum percent of edges between two contigs to form an edge between them.
- `--algorithm` (default: leiden): Community detection or clustering algorithm to perform the binning. If "spectral" is chosen, then `--n-clusters` needs to be set.
- `--gamma1` (default: 0.3): CPM optimization value for the first run of the Leiden clustering.
- `--gamma2` (default: 0.4): CPM optimization value for the final run of the Leiden clustering.
- `--n-clusters` (no default): When spectral clustering is used, it corresponds to the maximum bins in the data.

## 6.4 Analysis workflow

Here we present a full pipeline example for running CoCoNet from the raw reads to final bins. We assume that you have paired-end illumina reads available in the `data/` directory. Each sample is named with the convention `<sample>_R{1,2}.fastq.gz`.

### 6.4.1 Trimming

The first step consists in quality trimming and filtering your reads. Many tools can be used for this step, such as fastp or trimmomatic. Below is a trimming example with fastp. Note that the parameters need to be tuned to your specific case. FastQC and MultiQC can be very useful in that regard.

```
#!/usr/bin/env bash

# SAMPLES is an environment variable containing the sample names
# (e.g.: export SAMPLES="sample1 sample2")

for sample in $SAMPLES; do
    fastp \
    -i data/${sample}_R1.fastq.gz -I data/${sample}_R2.fastq.gz \
    -o ${sample}-trimmed_R1.fastq.gz -O ${sample}-trimmed_R2.fastq.gz
done
```

### 6.4.2 Assembly

To assemble your reads, you have many choices. One of the most accurate for metagenomics is metaSPAdes. However if you have a lot of samples and/or high coverage, metaSPAdes will require a significant amount of time and memory, in which case Megahit can be a good alternative.

```
# combine all samples
cat data/*-trimmed_R1.fastq.gz > forward.fastq.gz
cat data/*-trimmed_R2.fastq.gz > reverse.fastq.gz

```

(continues on next page)

(continued from previous page)

```
# run metaspades
metaspades.py \
-1 forward.fastq.gz -2 reverse.fastq.gz \
-o assembly-outputs
```

### 6.4.3 Coverage

To compute contig coverage, we can align the trimmed reads on the assembled contigs with tools such as bwa or bowtie2:

```
mkdir coverage

# Build index
bwa index -p coverage/index assembly-outputs/scaffolds.fasta

for sample in $SAMPLES; do
    bwa mem -p coverage/index data/${sample}-trimmed_R*.fastq.gz \
    | samtools view -bh \
    | samtools sort -o coverage/${sample}.bam
    samtools index coverage/${sample}.bam coverage/${sample}.bai
done
```

### 6.4.4 Binning

You can finally use CoCoNet and bin your contigs:

```
coconet run --fasta assembly-outputs/scaffolds.fasta --bam coverage/*.bam --output␣
↪binning
```

The binning assignment are then available in the file *binning/bins-*.csv*.

# CHAPTER 7

# Indices and tables

- genindex
- modindex
- search